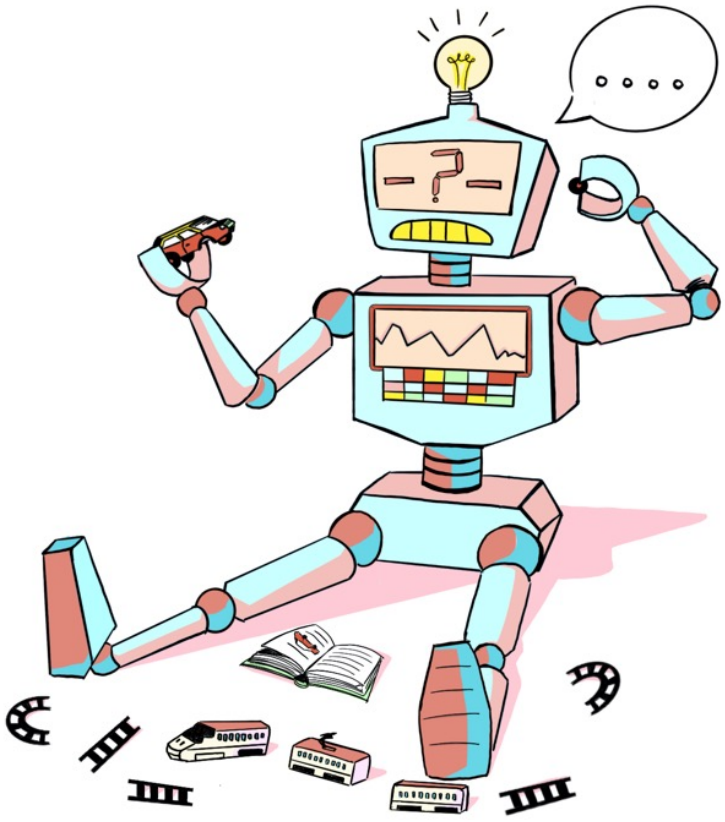
Pengenalan Konsep Pembelajaran Mesin dan Deep Learning

Edisi 1.4 (17 Agustus 2020)



Jan Wira Gotama Putra

Untuk Tuhan, Bangsa, dan Almamater

Kata Pengantar

Buku ini ditujukan sebagai bahan pengantar (atau penunjang) mata kuliah *machine learning* untuk mahasiswa di Indonesia, khususnya tingkat sarjana (tidak menutup kemungkinan digunakan untuk tingkat pascasarjana). Buku ini hanya merupakan komplemen, bukan sumber informasi utama. Buku ini memuat materi dasar *machine learning*, yang ditulis sedemikian rupa sehingga pembaca mampu mendapatkan **intuisi**. Materi pada buku ini tidaklah dalam (tapi tidak dangkal); artinya, pembaca masih harus membaca buku-buku lainnya untuk mendapatkan pemahaman lebih dalam.

Walaupun tidak sempurna, mudah-mudahan buku ini mampu memberi inspirasi. Anggap saja membaca buku ini seperti sedang membaca "*light novel*". Penulis ingin buku ini bisa menjadi *pointer*; i.e. dengan membaca buku ini, diharapkan kawan-kawan juga mengetahui harus belajar apa secara lebih jauh. Setelah membaca buku ini, pembaca diharapkan mampu membaca literatur *machine learning* yang dijelaskan secara lebih matematis ataupun mendalam (kami memberi rekomendasi bacaan lanjutan).

Di Indonesia, penulis banyak mendengar baik dari teman, junior, senior, dll; suatu pernyataan "kuliah mengajari teori saja, praktiknya kurang, dan tidak relevan dengan industri." Tentu saja pernyataan ini cukup benar, tetapi karena permikiran semacam ini terkadang kita tidak benar-benar mengerti permasalahan. Ketika mengalami kendala, kita buntu saat mencari solusi karena fondasi yang tidak kokoh. Banyak orang terburu-buru "menggunakan *tools*" karena lebih praktikal. Penulis ingin mengajak saudara/i untuk memahami konsep *machine learning* secara utuh sebelum memanfaatkan. Ada perbedaan yang mendasar antara orang yang hanya mampu menggunakan *tools* dan mengerti konsep secara utuh.

Buku ini menjelaskan algoritma *machine learning* dari sudut pandang **agak matematis**. Pembaca disarankan sudah memahami/mengambil setidaknya mata kuliah statistika, kalkulus, aljabar linear, pengenalan kecerdasan buatan, dan logika fuzzy. Penulis merasa banyak esensi yang hilang ketika materi *machine learning* hanya dijelaskan secara deskriptif karena itu buku ini ditulis dengan bahasa agak matematis. Walaupun demikian,

penulis berusaha menggunakan notasi matematis seminimal dan sesederhana mungkin, secukupnya sehingga pembaca mampu mendapatkan intuisi. Saat membaca buku ini, disarankan membaca secara runtun. Gaya penulisan buku ini **santai/semiformal** agar lebih mudah dipahami, mudah-mudahan tanpa mengurangi esensi materi.

Buku ini ditulis menggunakan template monograph (LaTeX) dari Springer yang dimodifikasi. Dengan demikian, mungkin ada kesalahan pemenggalan kata. Tentunya, buku tidak lepas dari kekurangan, misalnya kesalahan tipografi. Kami sarankan pembaca untuk membaca secara seksama, termasuk menginterpretasikan variabel pada persamaan.

Petunjuk Penggunaan
Struktur penyajian buku ini dapat dijadikan acuan sebagai struktur kuliah *machine learning* yang berdurasi satu semester (bab 1 untuk sesi pertama, dst). Agar dapat memahami materi per bab, bacalah keseluruhan isi bab secara utuh sebelum mempertanyakan isi materi. Penulis sangat menyarankan untuk membahas soal latihan sebagai tambahan materi (bisa juga sebagai PR). Soal latihan ditujukan untuk mengarahkan apa yang harus dibaca/dipahami lebih lanjut.

Pembaca dipersilahkan menyebarkan (*share*) buku ini untuk alasan **NON KOMERSIAL** (pendidikan), tetapi **dimohon kesadarannya untuk tidak menyalin atau meniru isi buku ini**. Bila ingin memuat konten diktat ini pada media yang pembaca kelola, dimohon untuk mengontak pengarang terlebih dahulu. Tidak semua istilah bahasa asing diterjemahkan ke Bahasa Indonesia supaya makna sebenarnya tidak hilang (atau penulis tidak tahu versi Bahasa Indonesia yang baku).

Bab lebih awal memuat materi yang relatif lebih "mudah" dipahami dibanding bab berikutnya. Buku ini memberikan contoh dimulai dari contoh sederhana (beserta contoh data). Semakin menuju akhir buku, notasi yang digunakan akan semakin simbolik, beserta contoh yang lebih abstrak. Penulis sangat menyarankan untuk **membaca buku ini secara sekuensial**.

Kutipan
Buku ini tergolong *self-published work* (atau mungkin lebih tepat dikatakan sebagai *draft*), tetapi sudah di-*review* oleh beberapa orang. Kami yakin para *reviewer* adalah orang yang berkompeten. Silahkan merujuk buku ini sesuai dengan paduan cara merujuk *self-published work*, apabila memang diperbolehkan untuk merujuk *self-published work* pada pekerjaan pembaca.

Notasi Penting
Karakter *bold* kapital merepresentasikan matriks ($\mathbf{X}, \mathbf{Y}, \mathbf{Z}$). Dimensi matriks ditulis dengan notasi $N \times M$ dimana N merepresentasikan banyaknya baris dan M merepresentasikan banyaknya kolom. Elemen matriks direpresentasikan oleh $\mathbf{X}_{i,j}$, $\mathbf{X}_{[i,j]}$, atau $x_{i,j}$ untuk baris ke-i kolom ke-j (penggunaan akan menyesuaikan konteks pembahasan agar tidak ambigu). Karakter di-

bold merepresentasikan vektor ($\mathbf{x}$). Elemen vektor ke-i direpresentasikan oleh x_i atau $\mathbf{x}_{[i]}$ tergantung konteks. Ketika penulis menyebutkan vektor, yang dimaksud adalah **vektor baris** (*row vector*, memiliki dimensi $1 \times N$, mengadopsi notasi Goldberg [1]). Perhatikan, literatur *machine learning* lainnya mungkin tidak menggunakan notasi *row vector* tetapi *column vector*. Kami harap pembaca mampu beradaptasi. Simbol " $\cdot$ " digunakan untuk melambangkan operator *dot-product*.

Kumpulan data (atau himpunan) direpresentasikan dengan karakter kapital (C, Z), dan anggotanya (*data point*, *data entry*) ke-i direpresentasikan dengan karakter c_i. Perhatikan, elemen vektor dan anggota himpunan bisa memiliki notasi yang sama (himpunan dapat direpresentasikan di komputer sebagai *array*, jadi penggunaan notasi vektor untuk himpunan pada konteks pembicaraan kita tidaklah salah). Penulis akan menggunakan simbol $\mathbf{x}_{[i]}$ sebagai elemen vektor apabila ambigu. Fungsi dapat direpresentasikan dengan huruf kapital maupun non-kapital $f(\dots), E(\dots), G(\dots)$. Ciri fungsi adalah memiliki parameter! Pada suatu koleksi vektor (himpunan vektor) $\mathbf{D}$, vektor ke-i direpresentasikan dengan $\mathbf{d}_i$, dan elemen ke-j dari vektor ke-i direpresentaiskan dengan $\mathbf{d}_{i[j]}$, $\mathbf{D}_{i,j}$, atau $\mathbf{D}_{[i,j]}$ (karena sekumpulan vektor dapat disusun sebagai matriks).

Karakter non-kapital tanpa *bold* dan tanpa indeks, seperti (a, b, c, x, y, z), merepresentasikan *random variable* (statistik) atau variabel (matematik). Secara umum, saat *random variable* memiliki nilai tertentu, dinotasikan dengan $x = X$ (nilai tertentu dinotasikan dengan huruf kapital), kecuali disebutkan secara khusus saat pembahasan. Probabilitas direpresentasikan dengan karakter kapital (P), dengan karakter non-kapital merepresentasikan *probability density* (p). Penulis yakin pembaca dapat menyesuaikan interpretasi simbol berdasarkan konteks pembahasan. Untuk menginterpretasikan notasi lain, selain yang diberikan pada paduan ini, mohon menyesuaikan dengan ceritera pembahasan.

Ucapan Terima Kasih
Terima kasih pada Bapak/Ibu/Saudara/i atas kontribusi pada penulisan buku ini: Adhiguna Surya Kuncoro, Arief Yudha Satria, Candy Olivia Mawalim, Chairuni Aulia Nusapati, Genta Indra Winata, Hayyu Luthfi Hanifah, I Gede Mahendra Darmawiguna, dan Tifani Warnita. Terima kasih pada Natasha Christabelle Santosa atas desain cover.

Catatan lain
Buku ini adalah *ongoing project*. Versi terakhir dan terakurat dapat diakses pada https://wiragotama.github.io/. Buku ini lebih baik dibaca versi full pdf-nya agar pranala bisa di-klik dan gambar memiliki kualitas terbaik.

Tokyo, Jepang Jan Wira Gotama Putra

Daftar Isi

Bagian I

Pengetahuan Dasar

1

Pengenalan

> "People worry that computers will get too smart and take over the world, but the real problem is that they're too stupid and they've already taken over the world."
>
> Pedro Domingos

Penulis yakin istilah *machine learning* atau *deep learning* sudah tidak asing di telinga pembaca. *Machine learning* dan *deep learning* adalah salah satu materi kuliah pada jurusan Teknik Informatika atau Ilmu Komputer. Selain mengenal kedua istilah tersebut dari perkuliahan, pembaca mungkin mengenal istilah tersebut karena digunakan untuk pemasaran (*marketing*). Sebagai permulaan, *machine learning* dan *deep learning* bukanlah kedua hal yang berbeda.[1] Perlu diingat, *deep learning* adalah bagian dari *machine learning*. *Machine learning* sudah diaplikasikan pada banyak hal, baik untuk klasifikasi gambar, mobil tanpa pengemudi, klasifikasi berita, dsb. Bab ini menjelaskan konsep paling dasar dan utama *machine learning*.

1.1 Kecerdasan Buatan

Pada bagian pembukaan (*kata pengantar*) telah dijelaskan bahwa kami menganggap kamu sudah memiliki pengetahuan dasar tentang *artificial intelligence* (kecerdasan buatan), kami akan memberikan sedikit ikhtisar apa hubungan kecerdasan buatan dan *machine learning*. Saat pertama kali kamu mendengar istilah "kecerdasan buatan", mungkin kamu akan terpikir robot yang memiliki raga fisik. Tetapi, kecerdasan buatan tidak hanya terbatas pada sesuatu yang memiliki raga fisik. Raga fisik berguna untuk interaksi

[1] Walau istilah *deep learning* belakangan ini lebih populer.

yang ramah bagi manusia. Tidak mesti memiliki raga fisik, kecerdasan buatan sesungguhnya adalah program[2] yang memiliki bentuk matematis (instruksi); kita sebut sebagai **agen**. Berbeda dengan program biasa yang menghasilkan aksi berdasarkan instruksi, tujuan kecerdasan buatan adalah menciptakan program yang mampu mem-program (*output* program adalah sebuah program). Secara teori, program adalah ***automaton***[3] yang menjalankan suatu instruksi. Sama halnya dengan program pada umumnya, agen kecerdasan buatan juga menjalankan suatu instruksi. Yang menjadikanya beda dengan program biasa adalah **kemampuan untuk belajar**.[4] "Belajar" yang dimaksud tidaklah sama dengan proses manusia belajar. Mesin mampu belajar apabila ia mampu meng-*update* parameter (dijelaskan lebih detil kemudian), dimana parameter tersebut kurang-lebih merepresentasikan "pengetahuan" mesin.

Pada bidang keilmuan kecerdasan buatan, kita ingin menciptakan agen yang mampu melakukan pekerjaan yang membutuhkan kecerdasan manusia. Perhatikan, disini disebut kecerdasan manusia; hewan pun cerdas, tapi kecerdasan manusia dan hewan berbeda; yang kita ingin aproksimasi adalah kecerdasan manusia. Akan tetapi, kecerdasan manusia susah didefinisikan karena memiliki banyak aspek misalnya nalar (logika), kemampuan berbahasa, seni, dsb. Karena kecerdasan manusia memiliki banyak dimensi, kita dapat mencoba menyelesaikan masalah pada sub bidang lebih kecil–spesifik domain (*divide and conquer*). Sampai saat ini pun, peneliti belum juga mengetahui secara pasti apa yang membuat manusia cerdas, apa itu sesungguhnya cerdas, dan bagaimana manusia dapat menjadi cerdas. Dengan demikian, keilmuan kecerdasan buatan adalah interdisiplin, memuat: psikologis, linguistik, ilmu komputer, biologi, dsb. Bila kamu bertanya apakah program deterministik dapat disebut *kecerdasan buatan*, jawabannya "iya", *to some extent* (sampai pada level tertentu) karena memenuhi dimensi *acting rationally* (dijelaskan pada subbab 1.2).

Permasalahan utama bidang kecerdasan buatan terdiri dari (dari klasik sampai lebih modern),[5] yaitu:

1. ***Planning***. Diberikan *start state* dan *goal state*, agen harus merencanakan sekuens aksi untuk merubah *start state* menjadi *goal state*. Contoh permasalahan *planning* adalah merencanakan rute perjalanan dari kota A ke kota B. Bisa jadi, saat merencanakan sekuens aksi, ada kendala (*constraints*) yang harus dioptimisasi.

2. ***Representasi pengetahuan***, yaitu merepresentasikan pengetahuan dalam bentuk formal. Dengan representasi formal tersebut, kita dapat melakukan inferensi dengan operasi logika berbentuk simbolik, misal logika preposisi,

[2] Secara sederhana, program adalah kumpulan atau sekuens instruksi.
[3] Kami sarankan untuk membaca buku [2] untuk materi automata.
[4] Perlu diperhatikan, definisi ini adalah pandangan modern.
[5] Silahkan merujuk Association for the Advancement of Artificial Intelligence (AAAI).

logika orde pertama (*first-order logic*), teori Fuzzy, *abductive reasoning*, ontologi, maupun jaringan semantik (*semantic web*) [3].

3. ***Machine learning***, yaitu teknik untuk melakukan inferensi terhadap data dengan pendekatan matematis. Inti *machine learning* adalah untuk membuat model (matematis) yang merefleksikan pola-pola data (seiring kamu membaca buku ini, kamu akan lebih mengerti). Ini adalah bahasan utama buku ini. Pada abad ke-21 ini, *machine learning* banyak memanfaatkan statistika dan aljabar linier.

4. ***Multi-agent system***, yaitu sistem yang memiliki banyak agen berinteraksi satu sama lain untuk menyelesaikan permasalahan. Agen satu mengerjakan suatu hal tertentu, kemudian bekerja bersama untuk menyelesaikan masalah yang lebih besar (tidak dapat diselesaikan sendiri).

5. Dan lain sebagainya, silahkan mengacu pada topik konferensi *Association for the Advancement of Artificial Intelligence* (AAAI) .

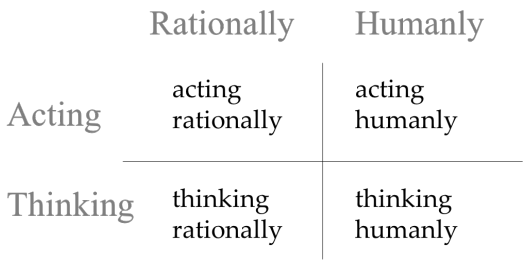
Perhatikan, sub keilmuan representasi pengetahuan dan *machine learning* sama-sama melakukan inferensi, tetapi pada representasi yang berbeda. Inferensi pada bidang keilmuan representasi pengetahuan mencakup tentang bagaimana cara (langkah dan proses) mendapatkan sebuah keputusan, diberikan premis. Sebagai contoh, a adalah anak b, dan c adalah anak b, maka apakah hubungan a dan c? Jawab: c adalah cucu a. Pada *machine learning*, inferensi yang dimaksud lebih menitikberatkan ranah hubungan variabel. Misalnya, *apakah penjualan akan meningkat apabila kita meningkatkan biaya marketing.* Bila kamu ingat dengan mata pelajaran matematika SMA (logika preposisi), kamu sadar bahwa membuat sistem cerdas menggunakan representasi pengetahuan simbolik itu susah. Kita harus mendefinisikan *term*, aturan logika, dsb. Belum lagi kita harus mendefinisikan aturan-aturan secara manual. Disamping itu, pengetahuan manusia sangatlah kompleks, dan translasi pengetahuan menjadi aturan-aturan formal tidaklah mudah. Representasi pengetahuan secara tradisional dianggap relatif kurang *scalable*. Artinya, kita tidak dapat merubah basis pengetahuan dengan mudah (meng-*update* "parameter"). Sementara itu, *machine learning* berada pada daerah representasi data/ilmu/pengetahuan dalam bentuk matematis karena keilmuan *machine learning* diturunkan dari matematika dan statistika. Teknik *machine learning* juga menjadi semakin populer akibat kemampuan komputasi yang terus meningkat.

Pada masa sekarang, kita dianugrahi dengan data yang banyak (bahkan tidak terbatas), teknik *machine learning* menjadi intuitif untuk melakukan inferensi pada data yang besar. Hal ini yang menyebabkan *machine learning* menjadi populer karena konstruksi model inferensi dapat dilakukan secara otomatis. *Machine learning* ibarat sebuah "alat", sama seperti rumus matematika. Bagaimana cara menggunakannya tergantung pada domain per-

masalahan. Dengan demikian, kamu harus paham betul bahwa memahami teknik-teknik *machine learning* saja tidak cukup. Kamu juga harus mengetahui domain aplikasi yang bersesuaian karena pemanfaatan teknik-teknik *machine learning* dapat berbeda pada domain yang berbeda. Sedikit cerita, sub keilmuan *data science* mempelajari banyak domain, misalnya data pada domain sosial, ekonomi, bahasa, maupun visual. Seiring kamu membaca buku ini, kami harap kamu semakin mengerti hal ini.

1.2 Intelligent Agent

Agen cerdas memiliki empat kategori berdasarkan kombinasi dimensi cara inferensi (*reasoning*) dan tipe kelakuan (*behaviour*) [4, 5]. Kategori agen dapat dilihat pada Gambar 1.1 dengan penjelasan sebagai berikut:



Gambar 1.1: Dimensi kecerdasan.

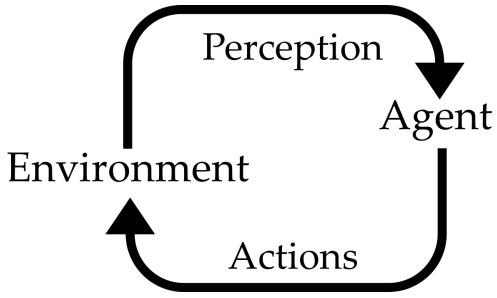
1. ***Acting Humanly***. Pada dimensi ini, agen mampu bertingkah dan berinteraksi layaknya seperti manusia. Contoh terkenal untuk hal ini adalah *turing test*. Tujuan dari *turing test* adalah untuk mengevaluasi apakah suatu sistem mampu "menipu" manusia. Disediakan seorang juri, kemudian juri berinteraksi dengan sesuatu di balik layar. Sesuatu di balik layar ini bisa jadi manusia atau program. Program dianggap mampu bertingkah (berinteraksi) seperti layaknya manusia apabila juri tidak dapat membedakan ia sedang berkomunikasi dengan manusia atau program.

2. ***Acting Rationally***. Pada dimensi ini, agen mampu bertingkah dengan optimal. Tindakan optimal belum tentu menyerupai tindakan manusia, karena tindakan manusia belum tentu optimal. Misalnya, agen yang mampu memiliki rute terpendek dari suatu kota A ke kota B untuk mengoptimalkan penggunaan sumber daya. Sebagai manusia, bisa saja kita mencari jalan sesuka hati.

3. ***Thinking Humanly***. Pada dimensi ini, agen mampu berpikir seperti manusia dalam segi kognitif (e.g. mampu mengerti apa itu kesedihan atau kesenangan). Dapat dibilang, meniru bagaima proses berpikir di otak terjadi (pemodelan otak).

4. ***Thinking Rationally***. Pada dimensi ini, agen mampu berpikir secara rasional. Sederhananya sesuai dengan konsep logika matematika. *Thinking Humanly* lebih cenderung pada pemodelan kognitif secara umum, sementara dimensi *thinking rationally* cenderung pada pemodelan proses berpikir dengan prinsip optimisasi (apa yang harus dilakukan agar hasil optimal).

Perlu dicatat, "*acting*" berarti agen mampu melakukan aksi. Sementara "*thinking*" adalah pemodelan proses. Untuk mewujudkan interaksi manusia-komputer seperti manusia-manusia, tentunya kita ingin *intelligent agent* bisa mewujudkan dimensi *acting humanly* dan *thinking humanly*. Sayangnya, manusia tidak konsisten [6]. Sampai saat ini, konsep kecerdasan buatan adalah meniru manusia; apabila manusia tidak konsisten, peneliti susah untuk memodelkan cara berpikir atau tingkah laku manusia. Berhubung agen dilatih menggunakan contoh-contoh data buatan manusia, ketidak-konsistensi-an agen semata-mata mencerminkan ketidak-konsistensi-an pembuat data. Dengan hal itu, saat ini kita paling mungkin menciptakan agen yang mempunyai dimensi *acting rationally*.



Gambar 1.2: *Agent* vs *environment* [7].

Perhatikan Gambar 1.2! Agen mengumpulkan informasi dari lingkungannya, kemudian memberikan respon berupa aksi. Lingkungan (*environment*) yang dimaksud bisa jadi macam-macam, misal: rumah, papan catur, agen lain, dsb. Kita ingin agen melakukan aksi yang benar. Tentu saja kita perlu mendefinisikan secara detail, teliti, tepat (*precise*), apa arti "aksi yang benar." Dengan demikian, lebih baik apabila kita mengukur kinerja agen, menggunakan ukuran kinerja yang objektif (disebut *performance measure*). Misal-

nya untuk robot pembersih rumah, *performance measure*-nya adalah seberapa persen debu yang dapat ia bersihkan. *Performance measure* adalah ukuran bagaimana model pembelajaran mesin dievaluasi secara eksternal. Di sisi internal, model harus mengoptimalkan suatu fungsi utiiltas (***utility function***), yaitu fungsi apa yang harus dimaksimalkan atau diminimalkan oleh agen tersebut khususnya pada tahap latihan (*training*). Misalnya, robot pembersih rumah memiliki *utility function* untuk mengukur kotoran di tempat terbatas tempat ia berada atau rute paling efisien untuk membersihkan rumah. Setiap tindakan yang dilakukan agen rasional harus mengoptimalkan baik nilai *utility function* (internal) dan *performance measure* (eksternal). Pada buku ini, istilah *performance measure* dan *utility function* merujuk pada hal yang berbeda. Tetapi, pada beberapa kasus, *utility function* dan *performance measure* dapat diukur dengan fungsi yang sama. Mungkin kamu merasa penjelasan ini agak abstrak pada saat ini. Tetapi, kamu akan semakin mengerti perbedaan keduanya setelah membaca buku ini.

1.3 Konsep Belajar

Bayangkan kamu berada di suatu negara asing! Kamu tidak tahu norma yang ada di negara tersebut. Apa yang kamu lakukan agar bisa menjadi orang "normal" di negara tersebut? Tentunya kamu harus **belajar**! Kamu mengamati bagaimana orang bertingkah laku di negara tersebut dan perlahan-lahan mengerti norma yang berlaku. Belajar adalah usaha memperoleh kepandaian atau ilmu; berlatih; berubah tingkah laku atau tanggapan yang disebabkan oleh pengalaman.[6] Pembelajaran adalah proses, cara, perbuatan atau menjadikan orang atau makhluk hidup belajar. Akan tetapi, pada *machine learning*, yang menjadi siswa bukanlah makhluk hidup, tapi mesin.

Definsi sebelumnya mungkin sedikit "abstrak", kita harus mengkonversi definisi tersebut sebagai definsi operasional (bentuk komputasi). Secara operasional, belajar adalah perubahan tingkah laku berdasarkan pengalaman (*event*/data) untuk menjadi lebih baik. Pada konteks *machine learning*, belajar adalah menyesuaikan konfigurasi parameter (tingkah laku) terhadap *utility function* sesuai dengan data (lingkungan).

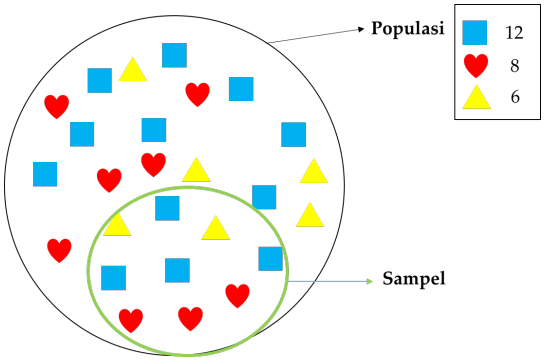
1.4 Statistical Learning Theory

Pada masa sekarang ini data bertebaran sangat banyak dimana-mana. Pemrosesan data secara manual tentu adalah hal yang kurang bijaksana. Beberapa pemrosesan data yang dilakukan seperti kategorisasi (kategorisasi teks berita), peringkasan dokumen, ekstraksi informasi (mencari subjek, objek, dan relasi diatara keduanya pada teks), rekomendasi produk berdasarkan

[6] KBBI Web, diakses pada 10 Oktober 2016

catatan transaksi, dll [7]. Tujuan *machine learning* minimal ada dua: **memprediksi masa depan** (*unobserved event*); dan/atau **memperoleh ilmu pengetahuan** (*knowledge discovery/discovering unknown structure*). Kedua hal ini berkaitan sangat erat. Sebagai contoh, manusia tahu bahwa cara menggunakan pensil dan pulpen sama, walaupun saat kita belum pernah menggunakan pulpen (penulis berasumsi kamu belajar menulis menggunakan pensil). Memprediksi masa depan berarti kita tahu bahwa pulpen adalah alat tulis. *Knowledge discovery* berarti kita tahu bahwa cara menggunakan pulpen dan pensil itu sama, walaupun belum pernah menggunakan pulpen sebelumnya.[7]

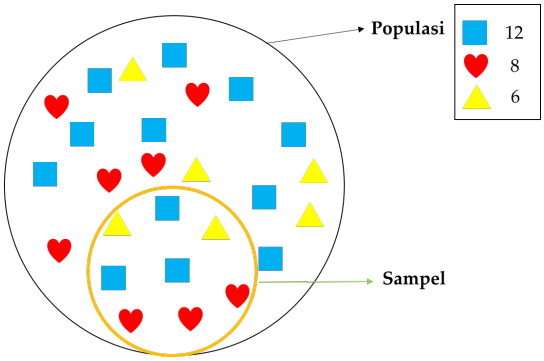
Untuk mencapai tujuan tersebut, kita menggunakan data (sampel), kemudian membuat model untuk menggeneralisasi "aturan" atau "pola" data sehingga kita dapat menggunakannya untuk mendapatkan informasi/membuat keputusan [8, 9]. *Statistical learning theory* (yang diaplikasikan pada *machine learning*) adalah teknik untuk memprediksi masa depan dan/atau menyimpulkan/mendapatkan pengetahuan dari data **secara rasional dan non-paranormal**. Hal ini sesuai dengan konsep *intelligent agent*, yaitu bertingkah berdasarkan lingkungan. Dalam hal ini, yang bertindak sebagai lingkungan adalah data. *Performance measure*-nya adalah seberapa akurat prediksi agen tersebut atau seberapa mirip "pola" data yang ditemukan terhadap data asli. Disebut *statistical* karena basis pembelajarannya memanfaatkan banyak teori statistik untuk melakukan inferensi (misal memprediksi *unobserved event*).[8]



Gambar 1.3: Ilustrasi makanan pesta 1.

Perhatikan Gambar 1.3 (permasalahan yang disederhanakan). Misalkan kamu diundang ke suatu pesta. Pada pesta tersebut ada 3 jenis kue yang disajikan. Kamu ingin mengetahui berapa rasio kue yang disajikan dibandingkan masing-masing jenisnya (seluruh populasi). Tetapi, karena susah untuk menganalisis seluruh data atau keseluruhan data tidak tersedia, kamu mengam-

[7] Baca *zero-shot learning*
[8] Selain itu, *machine learning* juga banyak memanfaatkan teori aljabar linier.



Gambar 1.4: Ilustrasi makanan pesta 2.

bil beberapa sampel. Dari sampel tersebut, kamu mendapati bahwa ada 4 buah kue segi empat, 3 buah kue hati dan 2 buah kue segitiga. Lalu kamu menyimpulkan (model) bahwa perbandingan kuenya adalah 4:3:2 (segiempat:hati:segitiga). Perbandingan tersebut hampir menyerupai kenyataan seluruh kue yaitu 4:2.67:2. Cerita ini adalah kondisi ideal.

Perhatikan Gambar 1.4, temanmu Haryanto datang juga ke pesta yang sama dan ingin melakukan hal yang sama (rasio kue). Kemudian ia mengambil beberapa sampel kue. Dari sampel tersebut ia mendapati bahwa ada 3 buah segiempat, 3 buah hati dan 2 buah segitiga, sehingga perbandingannya adalah 3:3:2. Tentunya hal ini sangat melenceng dari populasi.

Dari dua sampel yang berbeda, kita menyimpulkan, menginferensi (*infer*) atau mengeneralisasi dengan berbeda. Kesimpulan yang kita buat berdasarkan sampel tersebut, kita anggap merefleksikan populasi, kemudian kita menganggap populasi memiliki aturan/pola seperti kesimpulan yang telah kita ciptakan [10]. Baik pada statistika maupun *statistical machine learning*, pemilihan sampel (selanjutnya disebut ***training data***) adalah hal yang sangat penting. Apabila *training data* tidak mampu merepresentasikan populasi, maka model yang dihasilkan pembelajaran (*training*) tidak bagus. Untuk itu, biasanya terdapat juga ***validation data*** dan ***test data***. Mesin dilatih menggunakan *training data*, kemudian diuji kinerjanya menggunakan *validation data*[9] dan *test data*. Seiring dengan membaca buku ini, konsep *training data*, *validation data*, dan *test data* akan menjadi lebih jelas. Cara mengevaluasi apakah sampel (data) yang kita miliki cukup "bagus" untuk merepresentasikan populasi sangat bergantung pada domain dan aplikasi.

Seperti halnya contoh sederhana ini, persoalan *machine learning* sesungguhnya menyerupai persoalan *statistical inference* [10]. Kita berusaha mencari tahu populasi dengan cara menyelidiki fitur (*features* atau sifat-sifat) yang dimiliki sampel. Kemudian, menginferensi aksi yang harus dilakukan

[9] Pada umumnya bertindak sebagai *stopping criterion* saat proses *training*.

terhadap *unobserved data* berdasarkan kecocokan fitur-fitur *unobserved data* dengan model/aturan yang sudah ada.

Dari sisi metode pembelajaran, algoritma *machine learning* dapat dikategorikan sebagai: *supervised learning* (subbab 1.6), *semi-supervised learning* (subbab 1.8), *unsupervised learning* (subbab 1.9), dan *reinforcement learning*. Masing-masing metode akan dibahas pada subbab berikutnya (kecuali *reinforcement learning*, karena diluar cakupan buku ini).

1.5 Training, Validation, Testing Set

Terdapat dua istilah penting dalam pembangunan model *machine learning* yaitu: ***training*** dan ***testing***. *Training* adalah proses konstruksi model dan *testing* adalah proses menguji kinerja model pembelajaran. *Dataset* adalah kumpulan data (sampel dalam statistik). Sampel ini adalah data yang kita gunakan untuk membuat model maupun mengevaluasi model *machine learning*. Umumnya, *dataset* dibagi menjadi tiga jenis yang tidak beririsan (suatu sampel pada himpunan tertentu tidak muncul pada himpunan lainnya):

1. ***Training set*** adalah himpunan data yang digunakan untuk melatih atau membangun model. Pada buku ini, istilah *training data(set)* mengacu pada *training set*.
2. ***Development set*** atau ***validation set*** adalah himpunan data yang digunakan untuk mengoptimisasi saat melatih model. Model dilatih menggunakan *training set* dan pada umumnya kinerja **saat latihan** diuji dengan *validation set*. Hal ini berguna untuk generalisasi (agar model mampu mengenali pola secara generik). Pada buku ini, istilah *development* dan *validation data(set)* mengacu pada hal yang sama.
3. ***Testing set*** adalah himpunan data yang digunakan untuk menguji model setelah **proses latihan selesai**. Pada buku ini, istilah *testing data(set)* atau *test set* mengacu pada *testing set*. Perlu kami tekankan, *testing set* adalah *unseen data*. Artinya, model dan manusia tidak boleh melihat sampel ini saat proses latihan. Banyak orang yang tergoda untuk melihat *testing set* saat proses latihan walaupun itu adalah tingkah laku yang buruk karena menyebabkan *bias*.

Suatu sampel pada himpunan data kita sebut sebagai ***data point*** atau ***instance*** yang merepresentasikan suatu kejadian statistik (*event*). Perlu diingat, *training*, *validation*, dan *testing data* secara ideal diambil (*sampled*) dari distribusi yang sama dan memiliki karakteristik yang sama (*independently and identically distributed*). Distribusi pada masing-masing dataset ini juga sebaiknya seimbang (*balanced*) dan memuat seluruh kasus. Misal, sebuah dataset *binary classification* sebaiknya memuat 50% kasus positif dan 50% kasus negatif.

Pada umumnya, rasio pembagian *dataset* (*training*: *validation*: *testing*) adalah (80% : 10% : 10%) atau (90% : 5% : 5%). *Validation set* pada umumnya bisa tidak digunakan apabila *dataset* berukuran kecil (hanya dibagi menjadi *training* dan *testing set* saja). Dalam kasus ini, pembagian *dataset* menjadi *training* dan *testing* set pada umumnya memiliki rasio (90% : 10%), (80% : 20%), (70% : 30%), atau (50% : 50%). Pada kasus ini, kinerja saat *training* diuji menggunakan *training set* (dikenal sebagai ***closed testing***).

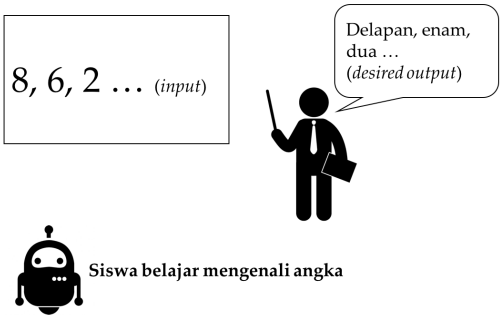
Saat tidak menggunakan *validation set* (hanya ada *training* dan *testing set*), kita juga memiliki opsi untuk mengevaluasi model dengan metode *K-cross-validation*.[10] Artinya, kita membagi *training dataset* (atau keseluruhan *dataset*) menjadi K bagian. Kita menggunakan $K-1$ bagian untuk *training*, kemudian menguji kinerja model saat latihan (*validation*) menggunakan satu bagian. Hal ini diulangi sebanyak K kali dimana sebuah bagian data digunakan sebagai *testing set* sebanyak sekali (bergilir). Mungkin sekarang kamu merasa pusing karena membaca banyak istilah. Jangan khawatir! Kamu akan lebih meresapi arti istilah-istilah tersebut seiring membaca buku ini.

1.6 Supervised Learning

Jika diterjemahkan secara literal, *supervised learning* adalah pembelajaran terarah/terawasi. Artinya, pada pembelajaran ini, ada guru yang mengajar (mengarahkan) dan siswa yang diajar. Kita disini berperan sebagai guru, kemudian mesin berperan sebagai siswa. Perhatikan Gambar 1.5 sebagai ilustrasi! Pada Gambar 1.5, seorang guru menuliskan angka di papan "8, 6, 2" sebagai contoh untuk siswanya, kemudian gurunya memberikan cara membaca yang benar untuk masing-masing angka. Contoh angka melambangkan ***input***, kemudian cara membaca melambangkan ***desired output*** (sering disebut "***gold standard***"). Pasangan ***input–desired output*** ini disebut sebagai *instance* (untuk kasus *supervised learning*). Pembelajaran metode ini disebut *supervised* karena ada yang memberikan contoh jawaban (*desired output*).

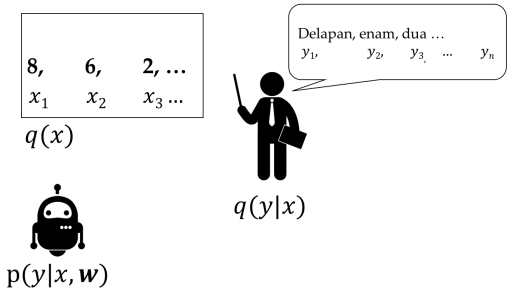
Perhatikan Gambar 1.6 dan Gambar 1.7, x adalah kejadian (*event–random variable*), untuk *event* tertentu dapat dinotasikan sebagai $\{x_1, x_2, x_3, \cdots, x_N\}$. x dapat berupa vektor, teks, gambar, dan lain sebagainya (perhatikan konteks pembahasan buku). Demi pembahasan yang cukup generik, pada bab ini kita membicarakan x yang merepresentasikan *event*, *data point*, atau *input*. Seorang guru sudah mempunyai jawaban yang benar untuk masing-masing contoh dengan suatu fungsi distribusi probabilitas kondisional (***conditional probability density function***) $q(y \mid x)$ baca: *function q for y given x*, melambangkan hasil yang benar/diharapkan untuk suatu event. Siswa (mesin) **mempelajari tiap pasang pasangan *input-desired output (training data)*** dengan mengoptimalkan *conditional probability density function* $p(y \mid x, \mathbf{w})$, dimana y adalah target (*output*), x adalah input dan vektor $\mathbf{w}$ adalah

[10] https://www.openml.org/a/estimation-procedures/1



Gambar 1.5: *Supervised learning.*

learning parameters. Proses belajar, yaitu mengoptimalkan **w** disebut sebagai training. Semakin kamu membaca buku ini, konsep ini akan menjadi semakin jelas. Proses *training* bertujuan untuk mengaproksimasi $q(y \mid x)$ melalui $p(y \mid x, \mathbf{w})$. Proses pembelajaran bertujuan untuk mengubah-ubah **w** sedemikian sehingga $p(y \mid, x, \mathbf{w})$ mirip dengan $q(y \mid x)$.

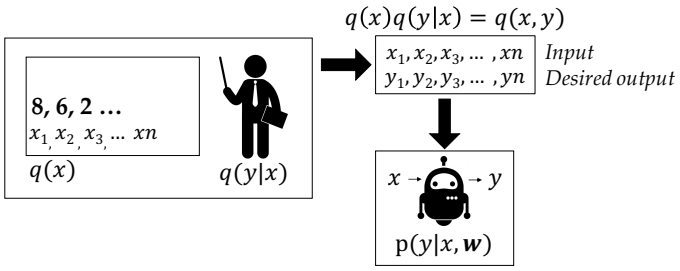


Gambar 1.6: *Supervised learning - mathematical explanation.*

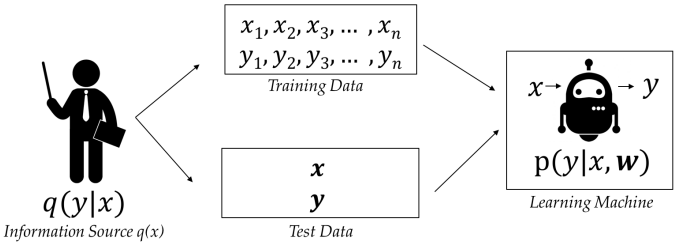
Perhatikan Gambar 1.8! model memiliki panah ke *training data* dan *test data*, artinya model hasil *training* sangat bergantung pada **data dan guru**. Model yang dihasilkan *training* (hasil pembelajaran kemampuan siswa) untuk data yang sama bisa berbeda untuk guru yang berbeda.[11]

Tujuan *supervised learning*, secara umum untuk melakukan klasifikasi (*classification*). Misalkan mengklasifikasikan gambar buah (apa nama buah pada gambar), diilsutrasikan pada Gambar 1.9. Apabila hanya ada dua kategori, disebut ***binary classification***. Sedangkan bila terdapat lebih dari dua kategori, disebut ***multi-class classification***. Contoh *multi-class classifica-*

[11] Penulis rasa hal ini sangat intuitif berhubung hal serupa terjadi pada manusia.



Gambar 1.7: *Supervised learning - mathematical explanation* 2.



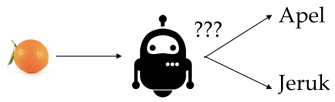
Gambar 1.8: *Supervised learning framework.*

tion adalah mengklasifikasikan gambar buah ke dalam himpunan kelas: *apel*, *mangga* atau *sirsak*.

Ada tipe klasifikasi lain disebut ***multi-label classification*** yaitu ketika kita ingin mengklasifikasikan suatu sampel ke dalam suatu himpunan kelas. Perhatikan! Perbedaan *multi-class* dan *multi-label classification* agak *tricky*. Pada *multi-class classification*, suatu sampel hanya bisa berkorespondensi dengan satu kelas. Sedangkan pada *multi-label classification*, satu sampel dapat berkorespondensi dengan lebih dari satu kelas. Misalnya, suatu berita dapat masuk ke kategori *agama* dan *politik* pada waktu bersamaan. Artinya, label pada *multi-class classification* bersifat *mutually exclusive*, sedangkan label tidak bersifat *mutually exclusive* pada *multi-label classification*.[12] Perhatikan Gambar 1.1 sebagai ilustrasi, dimana setiap baris merepresentasikan kelas yang berkorespondensi dengan setiap *input*, nilai "1" melambangkan TRUE dan nilai "0" melambangkan FALSE. *Multi-label classification* dapat didekomposisi menjadi beberapa *Binary classification*, yaitu mengklasifikasikan apakah *input* dapat ditetapkan ke suatu kelas atau tidak (dijelaskan lebih lanjut pada bab-bab berikutnya).

Pemahaman *supervised learning* adalah mengingat persamaan 1.1. Ada tiga hal penting pada *supervised learning* yaitu *input*, *desired output*, dan

[12] https://scikit-learn.org/stable/modules/multiclass.html



Gambar 1.9: Ilustrasi *binary classification*.

Instance	**Apel**	**Mangga**	**Sirsak**
Gambar-1	1	0	0
Gambar-2	0	1	0
Gambar-3	0	0	1

(a) Multi-class classification.

Instance	**Apel**	**Mangga**	**Sirsak**
Gambar-1	1	1	0
Gambar-2	0	1	1
Gambar-3	1	0	1

(b) Multi-label classification.

Tabel 1.1: Ilustrasi *multi-label* dan *multi-class classification*. Nilai "1" melambangkan TRUE dan "0" melambangkan FALSE (*class assignment*).

learning parameters. Perlu ditekankan *learning parameters* berjumlah lebih dari satu, dan sering direpresentasikan dengan vektor (*bold*) atau matriks. Berdasarkan model yang dibuat, kita dapat melakukan klasifikasi (misal simbol yang ditulis di papan adalah angka berapa). Secara konseptual, klasifikasi didefinisikan sebagai persamaan 1.2 yaitu memilih label (kelas/kategori y) paling optimal dari sekumpulan label C, diberikan (*given*) suatu sampel (*instance*) data tertentu.

$$p(y \mid x, \mathbf{w}) \tag{1.1}$$

$$\hat{y_i} = \arg\max_{y_i \in C} p(y_i \mid x_i, \mathbf{w}) \tag{1.2}$$

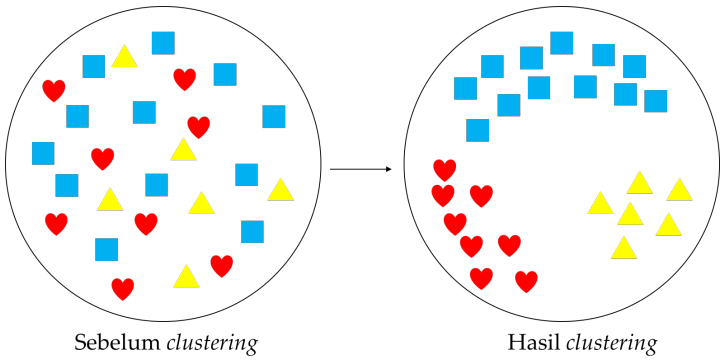
1.7 Regresi

Pada persoalan regresi, kita ingin memprediksi *output* berupa bilangan kontinu. Misalnya pada regresi suatu fungsi polinomial, kita ingin mencari tahu fungsi $f(x)$ diberikan data $\{(x_1, y_1), \cdots, (x_N, y_N)\}$. Setelah itu, kita gunakan fungsi aproksimasi untuk mencari tahu nilai y_{N+1} dari data baru x_{N+1}. Perbedaan regresi dan klasifikasi adalah pada tipe *output*. Untuk regresi, tipe *output* adalah nilai kontinu; sementara tipe *output* pada persoalan klasifikasi adalah suatu objek pada himpunan (i.e., memilih opsi pada himpunan jawaban). Tetapi, kita dapat mengkonversi fungsi regresi menjadi fungsi klasifikasi (dijelaskan pada bab 5).

1.8 Semi-supervised Learning

Semi-supervised learning mirip dengan *supervised learning*, bedanya pada proses pelabelan data. Pada *supervised learning*, ada "guru" yang harus membuat "kunci jawaban" *input-output*. Sedangkan pada *semi-supervised learning* tidak ada "kunci jawaban" eksplisit yang harus dibuat guru. Kunci jawaban ini dapat diperoleh secara otomatis (misal dari hasil *clustering*). Pada kategori pembelajaran ini, umumnya kita hanya memiliki sedikit data. Kita kemudian menciptakan data tambahan baik menggunakan *supervised* ataupun *unsupervised learning*, kemudian membuat model belajar dari data tambahan tersebut.

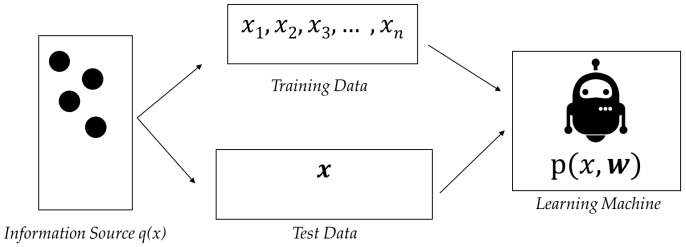
1.9 Unsupervised Learning



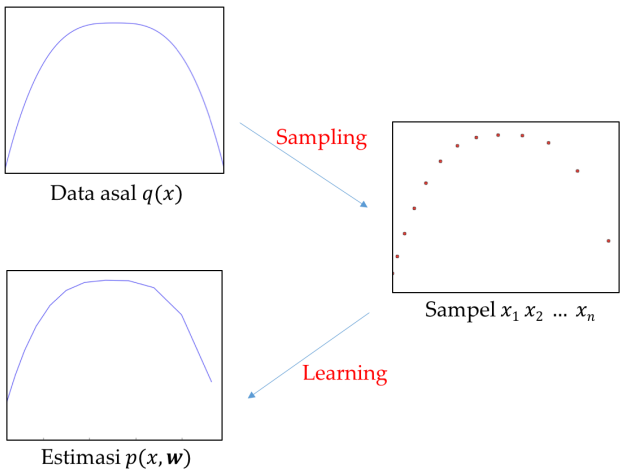
Gambar 1.10: Ilustrasi *clustering*

Jika pada *supervised learning* ada guru yang mengajar, maka pada *unsupervised learning* tidak ada guru yang mengajar. Contoh permasalahan unsupervised learning adalah *clustering*. Mengingat contoh kue sebelumnya, kita ingin mengelompokkan kue-kue yang sama, diilustrasikan oleh Gambar 1.10. Yang kamu lakukan adalah membuat kelompok-kelompok berdasarkan karakteristik kue, misal kelompok kue biru, kelompok kue kuning, atau kelompok kue merah. Teknik-teknik mengelompokkan ini akan dibahas pada bab-bab berikutnya. Contoh algoritma *unsupervised learning* sederhana adalah *K-means* (bab 4).

Perhatikan Gambar 1.11 dan Gambar 1.12! Berbeda dengan supervised learning yang memiliki *desired output*, pada *unsupervised learning* tidak ada *desired output* (jelas, tidak ada gurunya, tidak ada yang memberi contoh).



Gambar 1.11: *Unsupervised learning framework*



Gambar 1.12: *Generalization error of unsupervised learning*

Kita ingin mencari tahu distribusi asli data $q(x)$, berdasarkan beberapa sampel data. *Learning* dilakukan dengan mengoptimalkan $p(x \mid \mathbf{w})$ yang mengoptimasi parameter $\mathbf{w}$. Perbedaan antara estimasi dan fungsi asli disebut sebagai ***generalization loss*** (atau ***loss*** saja – dijelaskan pada bab 5). Kunci pemahaman *unsupervised learning* adalah mengingat persamaan 1.3, yaitu ada *input* dan parameter.

$$p(x \mid \mathbf{w}) \tag{1.3}$$

Perlu kami tekankan, *unsupervised learning* $\neq$ *clustering*! *Clustering* adalah salah satu bentuk *unsupervised learning*; yaitu salah satu hasil inferensi persamaan 1.3. *Unsupervised learning* adalah mencari sifat-sifat (*properties*) data. Kita ingin aproksimasi $p(x \mid \mathbf{w})$ semirip mungkin dengan $q(x)$, dimana $q(x)$ adalah distribusi data yang asli. Dataset di-sampel dari distribusi $q(x)$, kemudian kita ingin mencari tahu $q(x)$ tersebut.

1.10 Proses Belajar

Seperti yang sudah dijelaskan pada subbab sebelumnya, pada *supervised* maupun *unsupervised learning*, kita ingin mengestimasi sesuatu dengan teknik *machine learning*. Kinerja model pembelajaran berubah-ubah sesuai dengan parameter $\mathbf{w}$ (parameter pembelajaran). Kinerja model diukur oleh fungsi tujuan (*utility function*–saat latihan, *performance measure*–saat melakukan prediksi), yaitu mengoptimalkan nilai fungsi tertentu; misalnya meminimalkan nilai *error* (dijelaskan kemudian). Secara intuitif, *learning machine* mirip seperti saat manusia belajar. Kita awalnya membuat banyak kesalahan, tetapi kita mengetahui atau diberi tahu mana yang benar. Untuk itu kita menyesuaikan diri secara perlahan agar menjadi benar (iteratif). Inilah yang juga dilakukan *learning machine*, yaitu mengubah-ubah parameter $\mathbf{w}$ untuk mengoptimalkan suatu fungsi tujuan. Akan tetapi, *machine learning* membutuhkan sangat banyak data. Sementara, manusia dapat belajar dengan contoh yang sedikit. Perhatikan, dengan adanya parameter $\mathbf{w}$, kamu mungkin akan menganggap bahwa teknik *machine learning* adalah fungsi-parametrik. Sebenarnya, ada juga algoritma *machine learning* non-parametrik, tetapi algoritham parametrik yang lebih sering digunakan di masa buku ini ditulis.

Secara bahasa lebih matematis, kami beri contoh *supervised learning*. Kita mempunyai distribusi klasifikasi asli $q(y \mid x)$. Dari distribusi tersebut, kita diberikan beberapa sampel pasangan *input-output* $\{z_1, z_2, z_3, \ldots, z_n\}; z_i = (x_i, y_i)$. Kita membuat model $p(y \mid x, \mathbf{w})$. Awalnya diberi (x_1, y_1), model mengestimasi fungsi asli dengan mengoptimalkan parameter $\mathbf{w}$ sesuai dengan data yang ada. Seiring berjalannya waktu, ia diberikan data observasi lainnya, sehingga *learning machine* menyesuaikan dirinya (konvergen) terhadap observasi yang baru $(x_2, y_2), (x_3, y_3), \ldots$. Semakin lama, kita jadi makin percaya bahwa model semakin optimal (mampu memprediksi fungsi aslinya). Apabila kita diberikan lebih banyak data data sampai sejumlah tak hingga, aproksimasi kita akan semakin mirip dengan distribusi aslinya.

1.11 Tips

Jujur, pengarang sendiri belum menguasai bidang ini secara penuh, tetapi berdasarkan pengalaman pribadi (+membaca) dan beberapa rekan; ada beberapa materi wajib yang harus dipahami untuk mengerti bidang *machine learning*. Sederhananya, kamu harus menguasai banyak teori matematika dan probabilitas agar dapat mengerti *machine learning* sampai tulang dan jeroannya. Kami tidak menyebutkan bahwa mengerti *machine learning* secara intuitif (atau belajar dengan pendekatan deskriptif) itu buruk, tetapi untuk mengerti sampai dalam memang perlu mengerti matematika (menurut pengalaman kami). Disarankan untuk belajar materi berikut:

1. Matematika Diskrit dan Teori Bilangan

2. Aljabar Linier dan Geometri (vektor, matriks, skalar, dekomposisi, transformasi, tensor, dsb)
3. Kalkulus (diferensial dan integral)
4. Teori Optimasi (*Lagrange multiplier, Convex Iptimization, Gradient Descent, Integer Linear Problem,* dsb)
5. Probabilitas dan Statistika (probabilitas, *probability densities*, *hypothesis testing*, *inter-rater agreement*, Bayesian, *sampling*)
6. Teori Fuzzy

Mungkin kamu sudah tahu, tetapi penulis ingin mengingatkan ada dua buku yang sangat terkenal ("kitab") sebagai materi belalajar *machine learning* dan *deep learning*:

1. Patten Recognition and Machine Learning, oleh Christopher M. Bishop [8]
2. Deep Learning, oleh Ian Goodfellow, Yoshua Bengio, dan Aaron Courville [11]

Apabila pembaca memiliki kesempatan, penulis sangat menyarankan membaca kedua buku tersebut.

1.12 Contoh Aplikasi

Sebenarnya, aplikasi pemanfaatan *machine learning* sudah terasa dalam kehidupan sehari-hari. Contoh mudahnya adalah produk-produk Google, misalnya google translate (machine translation, handwritten recognition, speech recognition, Alpha Go). Berikut adalah beberapa artikel menarik:

1. techcrunch google AI beats go world champion
2. http://www-formal.stanford.edu/jmc/whatisai/node3.html
3. https://www.google.com/selfdrivingcar/
4. http://www.osnews.com/story/26838/Palm_I_m_ready_to_wallow_now/page2/

Soal Latihan

1.1. Aplikasi

(a) Carilah contoh-contoh penerapan *machine learning* pada kehidupan sehari-hari selain yang telah disebutkan!
(b) Mengapa mereka menggunakan teknik *machine learning* untuk menyelesaikan permasalahan tersebut?
(c) Apakah tidak ada opsi teknik lainnya? Jelaskan bila ada!
(d) Apa kelebihan dan kekurangan teknik *machine learning* daripada teknik lainnya (yang kamu jelaskan pada soal (c))?

1.2. Kecerdasan

Jelaskan tahapan perkembangan kecerdasan manusia berdasarkan kategori usia! Dari hal ini, kamu akan mengerti kenapa beberapa peneliti membuat agen cerdas berdasarkan kategori usia tertentu.

2

Fondasi Matematis

> "He uses statistics as a drunken man uses lamp posts – for support rather than for illumination."
>
> Andrew Lang

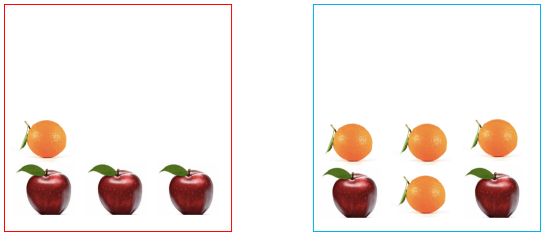
Mungkin saat pertama kali membaca bab ini, kamu merasa bab ini tidak masuk akal atau kurang dibutuhkan. Seiring membaca buku ini, mungkin bab ini akan sering dikunjungi kembali. Bab ini hanyalah pengingat materi yang sudah kamu pernah pelajari saja (semacam *cheatsheet*). Kamu boleh melewati bab ini apabila sudah familiar dengan materi probabilitas, statistika, serta aljabar linier. Seharusnya, bab ini memang diberikan paling awal. Tetapi penulis khawatir pembaca mungkin merasa buku ini tidak menarik apabila bab ini diberikan paling awal.

Bab ini memuat sekilas tentang probabilitas, statistika, dan operasi matriks. Tentunya untuk mengerti materi tersebut sebaiknya kamu mengambil kuliah khusus berkaitan karena kamu diharapkan sudah memiliki cukup latar pengetahuan, bab ini sebenarnya hanyalah sekilas pengingat. Kami akan banyak memakai contoh-contoh dari buku Bishop [8] untuk materi probabilitas. Penulis merekomendasikan untuk menonton kuliah *statistical inference* (coursera) oleh Professor Brian Caffo.

2.1 Probabilitas

Di dunia ini, ada banyak hal yang tidak pasti (*uncertain*). Ssungguhnya, *machine learning* berurusan dengan ketidakpastian (*uncertainty*). Dengan begitu, *machine learning* memiliki kaitan yang sangat erat dengan statistika. Probabilitas menyediakan *framework* untuk kuantifikasi dan manipulasi ketidakpastian [8]. Mari kita lihat contoh sederhana. Terdapat dua buah kotak

berwarna merah dan berwarna biru. Pada kotak merah terdapat 3 *apel* dan 1 *jeruk*. Pada kotak biru, terdapat 2 *apel* dan 4 *jeruk*, kita ingin mengambil buah dari salah satu kotak tersebut. Ilustrasi persoalan dapat dilihat pada Gambar 2.1. Dalam hal ini, kotak adalah *random variable*. *Random variable* k (melambangkan kotak) dapat bernilai *merah* atau *biru*. Begitu pula dengan buah, dilambangkan dengan variabel b, dapat bernilai *apel* atau *jeruk*.



Gambar 2.1: Kotak apel dan jeruk.

Saat kita mengambil buah dari kotak biru, peluang untuk memilih *apel* bernilai $\frac{2}{6}$, sedangkan peluang untuk memilih *jeruk* bernilai $\frac{4}{6}$; kita tulis probabilitas ini sebagai $P(b = apel) = \frac{2}{6}$; dan $P(b = jeruk) = \frac{4}{6}$. Artinya, jika kita mengambil buah dari kotak biru, mungkin kita mendapatkan jeruk. Dengan mengkuantifikasi ketidakpastian, kita bisa mengatur ekspektasi. Nilai suatu probabilitas harus lebih besar sama dengan nol sampai kurang dari atau sama dengan satu ($0 \leq P \leq 1$). Nilai nol berarti suatu kejadian tidak mungkin muncul, sementara nilai satu berarti suatu kejadian pasti terjadi.

Lalu sekarang ada pertanyaan baru; pada suatu percobaan, berapakah probabilitas mengambil sebuah *apel* dari kotak biru **atau** sebuah *jeruk* dari kotak merah. Hal ini dituliskan sebagai $P((k = biru, b = apel)$ **atau** $(k = merah, b = jeruk))$. Nilai probabilitas tersebut dapat dihitung dengan

$$\begin{aligned} &P((k = biru, b = apel) \vee (k = merah, b = jeruk)) \\ &= P(k = biru, b = apel) + P(k = merah, b = jeruk) \end{aligned} \tag{2.1}$$

- $P(k = biru, b = apel)$ disebut *joint probability*, yaitu probabilitas kejadian yang dipengaruhi oleh beberapa variabel (kondisi untuk kedua variabel terpenuhi).
- $P(k = biru, b = apel) + P(k = merah, b = jeruk)$ disebut **aturan tambah**.

Penting untuk diingat bahwa hasil operasi apapun terhadap probabilitas (baik tambah, kurang, kali, atau bagi) haruslah lebih besar sama dengan nol sampai kurang dari atau sama dengan satu ($0 \leq P \leq 1$).

Misalkan terdapat percobaan lain, kali ini kamu mengambil 1 buah. Kamu ingin mengetahui berapakah probabilitas untuk mengambil buah *apel* kotak mana saja. Hal ini dihitung dengan persamaan 2.2.

$$P(b = apel) = \sum_{i=1}^{I} P(k = k_i, b = apel) \tag{2.2}$$

Aturan tambah seperti ini disebut ***marginal probability*** karena hasilnya didapat dengan menjumlahkan probabilitas seluruh kemungkinan nilai pada variabel tertentu (buah) dengan mengontrol variabel lainnya (kotak).

Kemudian, kamu ingin melakukan percobaan lain. Kali ini kamu mengambil 2 buah sekaligus dari kedua kotak. Kamu ingin mengetahui berapakah probabilitas mengambil buah *apel* yang berasal dari kotak biru **dan** buah *jeruk* yang berasal dari kotak merah. Dalam kasus ini, kejadiannya adalah saling bebas (*independent*), artinya mengambil buah dari kotak biru tidak akan mempengaruhi hasil pengambilan dari kotak merah (dan sebaliknya). Apabila kedua *random variable* x dan y bersifat ***independent*** (tidak bergantung satu sama lain), maka $P(x = X, y = Y) = P(X) \times P(Y)$. Permasalahan mengambil buah dapat dihitung dengan persamaan 2.3.

$$\begin{aligned} &P((k = biru, b = apel) \wedge (k = merah, b = jeruk)) \\ &= P(k = biru, b = apel) \times P(k = merah, b = jeruk) \end{aligned} \tag{2.3}$$

Aturan ini disebut **aturan kali**.

Untuk *joint probability*, secara umum dapat ditulis sebagai $P(x, y)$, yaitu peluang x dan y muncul bersamaan. Apabila kedua variabel x dan y tidak saling bebas, maka keduanya disebut ***dependent***. Artinya x dan y saling mempengaruhi. Apabila suatu variabel x dikondisikan (*conditioned*) oleh variabel lain (misal y). Maka probabilitas x adalah *conditional probability function*, ditulis $P(x \mid y)$. Artinya probabilitas x yang dikondisikan oleh y. $P(x \mid y)$ dapat dihitung dengan persamaan 2.4, yaitu peluang kejadian x dan y muncul bersamaan dibagi dengan peluang kejadian y. Apabila x ternyata tidak dikondisikan oleh variabel y, maka $P(x \mid y) = P(x)$.

$$P(x \mid y) = \frac{P(x, y)}{P(y)} \tag{2.4}$$

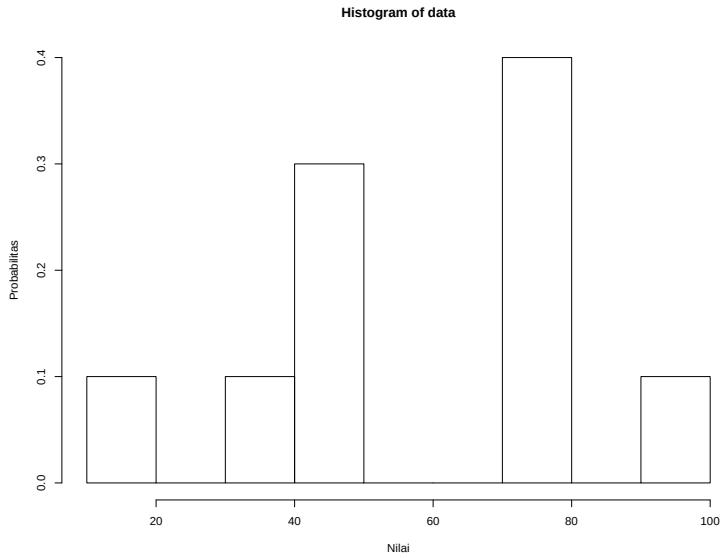
2.2 Probability Density Function

Probability density function dikenal juga dengan istilah distribusi, yaitu tentang persebaran nilai. Sebagai contoh, penulis menceritakan pelajaran di sekolah. Terdapat ujian mata pelajaran di kelas yang beranggotakan 10 siswa, diberikan pada Tabel 2.1. Terdapat 3 orang anak mendapatkan nilai 50, 2 orang anak mendapatkan nilai 75 dan 80, 1 orang anak mendapatkan nilai

id	nilai
1	50
2	75
3	80
4	100
5	50
6	50
7	75
8	80
9	40
10	10

Tabel 2.1: Contoh daftar nilai siswa.

100, 1 orang anak mendapat nilai 40, serta 1 orang anak mendapatkan nilai 10.



Gambar 2.2: Persebaran probabilitas nilai siswa Tabel 2.1.

Guru ingin mengetahui persebaran (distribusi) nilai ujian untuk menentukan batas kelas nilai menggunakan *quantile*. Grafik persebaran nilai mengkuantifikasi seberapa mungkin suatu siswa mendapat nilai tertentu, dapat dilihat pada Gambar 2.2. Grafik ini disebut sebagai distribusi. Fungsi yang menghasilkan distribusi tersebut disebut *probability density function*. Apabila kita

menjumlahkan probabilitas (probabilitas siswa mendapat nilai 0–100) nilainya adalah 1.

Ini adalah contoh untuk data diskrit, tetapi sering kali kita berurusan dengan data kontinu. Untuk mengetahui nilai probabilitas dari himpunan *event*/kejadian, kita dapat mengintegralkan kurva distribusi kejadian pada interval tertentu. Ciri *probability density function*, nilai dibawah kurva pada interval $-\infty$ sampai ∞ adalah 1, i.e., $p(x) \geq 0; \int_{-\infty}^{\infty} p(x)dx = 1$.

2.3 Expectation dan Variance

Salah satu operasi paling penting dalam probabilitas adalah menemukan nilai rata-rata (*average*) sebuah fungsi [8]. Hal ini disebut menghitung ekspektasi (*expectation*). Untuk sebuah fungsi $f(x)$ dengan distribusi probabilitas *random variable* adalah $p(x)$, nilai expectation diberikan pada persamaan 2.5.

$$E(f) = \begin{cases} \sum_x p(x)f(x); & diskrit \\ \int p(x)f(x)dx; & kontinu \end{cases} \tag{2.5}$$

Dalam kasus nyata, misalkan diberikan N buah sampel, *random variable* x dan fungsi $f(x)$, dimana sampel tersebut diambil dengan distribusi tertentu yang kita tidak ketahui, maka fungsi untuk menghitung nilai *expectation* menjadi persamaan 2.6, dimana x_i merepresentasikan data ke-i (*point*).

$$E(f) \simeq \frac{1}{N} \sum_{i=1}^{N} f(x_i) \tag{2.6}$$

Perhatikan, persamaan ini sama dengan persamaan untuk menghitung rata-rata (*mean* atau μ) seperti yang sudah kamu pelajari di SMA. Untuk mengetahui seberapa variasi nilai $f(x)$ di sekitar nilai rata-ratanya, kita menghitungnya mengunakan *variance*, disimbolkan dengan $var(f)$ atau σ^2 (persamaan 2.7).

$$\sigma^2 = var(f) = E\left((f(x) - E(f))^2\right) \tag{2.7}$$

Bila nilai *variance* tinggi, secara umum banyak variabel yang nilainya jauh dari nilai rata-rata. Interpretasi secara "geometris" mata, berarti distribusinya semakin "lebar" seperti pada Gambar 2.3. Untuk fungsi dengan lebih dari satu *random variable*, kita menghitung *covariance*. *Covariance* adalah *variance* untuk kombinasi variabel.

Akar dari *variance*, yaitu $\sqrt{\sigma^2} = \sigma$ disebut *standard deviation* (sering disingkat "SD"). SD melambangkan seberapa jauh jarak dari masing-masing data point x_i dengan rata-rata μ. SD cukup penting karena merupakan ukuran "persebaran" (*spread out*) data.

2.4 Bayesian Probability

Pada subbab sebelumnya, kita menghitung probabilitas dengan frekuensi kejadian yang dapat diulang. Pada pandangan Bayesian, kita ingin menguantifikasi ketidakpastian untuk kejadian yang mungkin sulit untuk diulang. Misalkan kita ingin tahu, seberapa peluang Mars dapat dihuni. Ini adalah sesuatu yang tidak dapat dihitung dengan frekuensi, maupun sebuah kejadian yang dapat diulangi (pergi ke mars, lihat berapa orang yang hidup). Akan tetapi, tentunya kita memiliki sebuah asumsi awal (*prior*). Dengan sebuah alat canggih baru, kita dapat mengumpulkan data baru tentang Mars. Dengan data tersebut, kita mengoreksi pendapat kita tentang Mars (*posterior*). Hal ini menyebabkan perubahan dalam pengambilan keputusan.

Pada keadaan ini, kita ingin mampu menguantifikasi ekspresi ketidakpastian; dan membuat revisi tentang ketidakpastian menggunakan bukti baru [8]. Dalam Bayesian, nilai probabilitas digunakan untuk merepresentasikan derajat kepercayaan/ketidakpastian.

$$P(x \mid y) = \frac{P(y \mid x)P(x)}{P(y)} \tag{2.8}$$

$P(x)$ disebut *prior*, yaitu pengetahuan/asumsi awal kita. Setelah kita mengobservasi fakta baru y (dapat berupa sekumpulan data atau satu *data point*/*event*), kita mengubah asumsi kita. $P(y \mid x)$ disebut ***likelihood function***. *Likelihood function* mendeskripsikan peluang data, untuk asumsi/ pengetahuan tentang x yang berubah-ubah (x sebagai parameter yang dapat diatur). Dengan *likelihood function* tersebut, kita mengoreksi pendapat akhir kita yang dapat digunakan untuk mengambil keputusan (*posterior*). Secara umum probabilitas Bayesian mengubah *prior* menjadi *posterior* akibat adanya kepercayaan baru (*likelihood*).

$$posterior \propto likelihood \times prior \tag{2.9}$$

Teori ini hebat karena kita dapat mentransformasi $P(x \mid y)$ dimana x dependen terhadap y menjadi bentuk $P(y \mid x)$ yang mana y dependen terhadap x. Transformasi ini sangat berguna pada berbagai macam persoalan.

Pada umumnya, untuk mengestimasi *likelihood*, digunakan *maximum likelihood estimator*; yang berarti mengatur nilai x untuk memaksimalkan nilai $P(y \mid x)$. Dalam literatur *machine learning*, banyak menggunakan *negative log of likelihood function* [8]. Ingat kembali nilai probabilitas $0 \leq P \leq 1$. Kadangkala, nilai dibelakang koma (0.xxxx) sangatlah panjang, sehingga dapat terjadi $underflow$ pada komputer. Kita menggunakan nilai logaritma probabilitas untuk menghindari $underflow$. Nilai probabilitas $0 \leq P \leq 1$ membuat nilai logaritmanya sebagaian besar negatif, secara monotonik bertambah, maka memaksimalkan nilai *likelihood* ekuivalen dengan meminimalkan negatif logaritma probabilitas.

Perhatikan kembali persamaan 2.8, secara intuitif, *posterior* dipengaruhi *prior*, artinya bergantung pada sampel yang kita punya, karena *prior* didapatkan atau diestimasi berdasarkan sampel. Hal ini berlaku pada *machine learning*, kualitas model yang dihasilkan bergantung pada kualitas training data.

Pada umumnya, kita tidak mengetahui seluruh informasi tentang situasi tertentu dan tidak mengetahui seluruh informasi probabilitas. Sebagai contoh, probabilitas $P(x \mid y)$ dapat dihitung dengan $\frac{P(x,y)}{P(x)}$. Tetapi, kita tidak tahu seberapa banyak kejadian (x, y) pada saat bersamaan. Oleh sebab itu, kita bisa menggunakan teori bayes untuk menghitung probabilitas dengan informasi lain yang kita tahu.

2.5 Gaussian Distribution

Distribusi adalah fenomena acak atau deskripsi matematis suatu *random variable*. Distribusi paling terkenal (mungkin juga terpenting) adalah *bell curve* atau distribusi normal. Distribusi normal adalah bentuk khusus dari *Gaussian distribution*. Ada beberapa macam distribusi yang akan dibahas pada bab ini, yaitu: *Univariate Gaussian*, *Multivariate Gaussian*, dan *Gaussian Mixture Model*. Pertama kita bahas ***Univariate Gaussian*** terlebih dahulu.

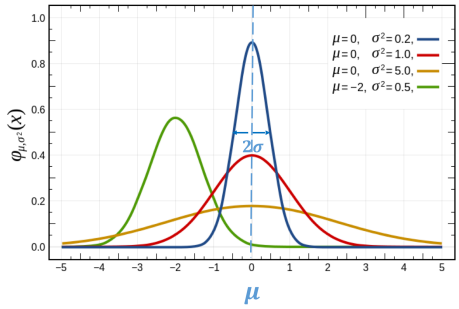
Disebut *univariate* karena distribusinya bergantung pada **satu** input variabel, misalkan x. Distribusi univariate Gaussian dikarakteristikkan oleh variabel x, *mean* (μ) dan *variance* (σ^2) diberikan pada persamaan 2.10. μ dan σ^2 adalah rata-rata dan *variance* untuk kumpulan data. Karena nilai μ dan σ^2 bergantung pada x, maka kita dapat menganggap bahwa *univariate gaussian* bergantung pada satu *random variable* saja yaitu x.

$$N(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)} \tag{2.10}$$

Perhatikan Gambar 2.3, x adalah absis dan nilai N untuk x tertentu (persamaan 2.10) adalah ordinat pada kurva ini. Bentuk distribusi berubah-ubah sesuai dengan nilai rata-rata (*mean*), serta *variance*. Semakin besar *variance*-nya, maka kurva distribusi semakin lebar (seperti yang dijelaskan sebelumnya). Untuk menggeser-geser kurva ke kiri maupun ke kanan, dapat dilakukan dengan menggeser nilai *mean*. Untuk mencari nilai pada suatu interval tertentu, cukup mengintegralkan fungsi pada interval tersebut. Nilai integral fungsi dari $-\infty$, hingga ∞ adalah satu.

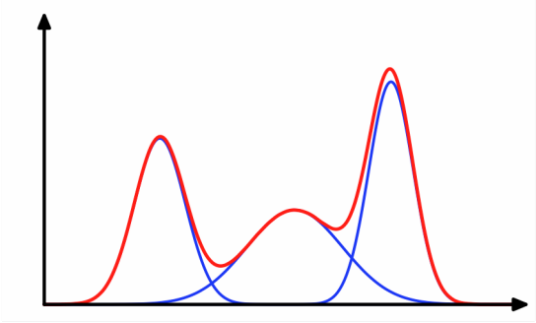
Multivariate Gaussian adalah distribusi gaussian yang bergantung pada lebih dari satu *random variable*. Sedangkan *Gaussian Mixture Model* (GMM) adalah gabungan dari satu atau lebih distribusi Gaussian. Masing-masing distribusi Gaussian memiliki bobot yang berbeda di GMM. Konon katanya,

[1] source: `wikimedia.org by Inductiveload`



Gambar 2.3: Univariate Gaussian.[1]

GMM dapat memodelkan fungsi apapun [12]. Ilustrasinya diberikan pada Gambar 2.4 yang tersusun dari 3 buah *Univariate gaussian*. Distribusi populasi berwarna merah, sedangkan GMM berwarna biru.

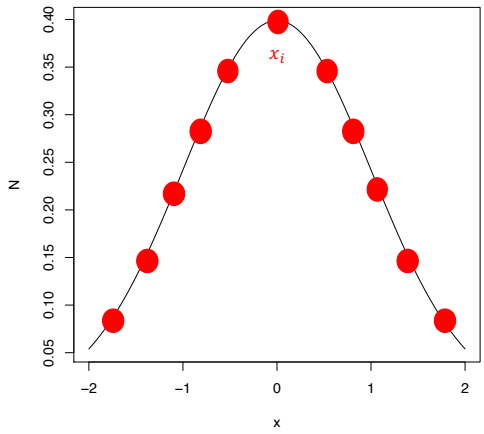


Gambar 2.4: Gaussian Mixture Model.[2]

2.6 Apakah Karakteristik Sampel Mencerminkan Populasi?

Sekarang bayangkan kita diberikan M buah data (diskrit) hasil observasi. Diasumsikan observasi dihasilkan oleh distribusi univariate Gaussian N dengan rata-rata μ dan *variance* σ^2. Ilustrasi diberikan pada Gambar 2.5. Setiap data diambil secara independen dari distribusi yang sama, disebut ***independent and identically distributed*** (iid). Kita tahu bahwa data yang in-

[2] http://dirichletprocess.weebly.com/clustering.html



Gambar 2.5: Contoh *sampling* dari distribusi normal. Titik berwarna merah melambangkan sampel (*instance*).

dependen, apabila dihitung probabilitasnya maka tersusun atas probabilitas masing-masing data, seperti pada persamaan 2.11.

$$p(x \mid \mu, \sigma^2) = \prod_{i=1}^{M} N(x_i \mid \mu, \sigma^2) \tag{2.11}$$

Kita ingin mencari tahu bagaimana distribusi yang sebenarnya. Untuk itu, kita harus mencari nilai μ dan σ^2 pada $p(x \mid \mu, \sigma^2)$. Kita dapat menggunakan teknik *maximum likelihood estimation*, untuk mengestimasi parameter yang memberikan distribusi (silahkan dieksplorasi lebih lanjut). Secara empiris, ternyata *mean* dari sampel dan *variance* dari sampel (diberikan sampel dengan M yang cukup banyak) cukup baik untuk mengestimasi *mean* dan *variance* dari distribusi sebenarnya (persamaan 2.12).

$$\mu \approx \mu_{sample} = \frac{1}{M}\sum_{i=1}^{M} x_i; \qquad \sigma^2 \approx \sigma^2_{sample} = \frac{1}{M}\sum_{i=1}^{M}(x_i - \mu_{sample})^2 \tag{2.12}$$

Pada statistik, apabila kita mengetahui (atau mengasumsikan) bahwa suatu kumpulan sampel diambil dari suatu distribusi q, maka kita bisa mengestimasi distribusi q menggunakan sampel yang ada sebagai p. Apabila sampel yang diberikan berjumlah sangat banyak ($M \to \infty$), maka p akan semakin mirip dengan q (konvergen). Kami menyarankan pembaca untuk menonton kuliah *statistical inference* oleh Professor Brian Caffo, dimana konsep mengestimasi *mean* dan *variance* populasi diestimasi menggunakan *mean* dan *variance* sampel dijelaskan dengan sangat baik (dengan banyak ilustrasi).

Hal ini memiliki interpretasi (secara kasar) yang cukup penting pada *machine learning*. Masih ingat materi bab 1? Pada *machine learning*, kita juga mengestimasi sesuatu yang kita tidak ketahui (populasi) dengan sampel data yang kita miliki. Kita anggap populasi memiliki karakteristik yang sama dengan sampel data. Semakin banyak sampel data yang kita miliki, maka semakin bagus estimasi kita terhadap populasi (jumlah dataset menjadi sangat penting).

Dengan bahasa lebih manusiawi, misal kamu diminta untuk membuat model klasifikasi berita otomatis untuk outlet berita tertentu. Populasi data yang ada yaitu: (1) berita yang sudah di-*publish* selama ini dan (2) berita di masa akan datang. Sampel yang kita punya adalah (1), yaitu berita masa lalu. Kita asumsikan bahwa sampel ini dapat mencerminkan karakteristik berita yang akan datang. Sehingga, model yang dilatih dengan data (1) dapat digunakan untuk memproses data (2).

Walau demikian, asumsi bahwa sampel berita yang dimiliki memiliki karakteristik yang sama dengan berita yang akan muncul di masa akan datang belum tentu tepat di dunia nyata. Sebagai contoh, kata *data science* belum digunakan pada tahun 1990an. Kosa kata selalu berkembang sesuai jaman. Bagaimana jika editor outlet berita bersangkutan mengundurkan diri dan diganti dengan editor lainnya? Gaya penulisan bisa saja berubah. Artinya, karaketeristik data sampel yang kita miliki hanya dapat mencerminkan data masa depan apabila banyak kondisi tetap sama, i.e., (1) dan (2) diambil dari distribusi yang sama. Sedangkan pada kenyataan, (1) dan (2) bisa jadi diambil dari distribusi yang berbeda. Pada kasus ini, model pembelajaran mesin yang dihasilkan menjadi *obsolete*, dan kita harus membuat model yang baru. Melatih atau membuat model baru menjadi peling untuk merefleksikan perubahan pada dunia.

2.7 Teori Keputusan

Diberikan himpunan pasangan data *input-output* $(x_i, y_i); x = input, y = output/target$; walaupun tidak pasti, kita ingin mengestimasi hubungan antara *input* dan *output*. Untuk itu kita melakukan estimasi $p(y \mid x, \mathbf{w})$, dimana $\mathbf{w}$ adalah *learning parameters*. Pada bab pertama, kamu telah mempelajari bahwa kita mampu melakukan hal ini dengan teknik *machine learning*. Lebih jauh lagi, kita juga harus mampu untuk membuat keputusan berbasiskan perkiraan nilai y, aspek ini disebut *decision theory* [8].

Dalam *machine learning* kita dapat membangun model dengan tujuan untuk meminimalkan *error* (secara umum meminimalkan *loss*); konsep meminimalkan *error* dijelaskan pada materi model linear (bab 5). Ibaratnya untuk sebuah robot, kita ingin robot tersebut tidak melakukan tindakan yang salah. Tetapi, kadang kala meminimalkan *error* belum tentu membuat model menjadi lebih baik. Kami ilustrasikan menggunakan contoh dari Bishop [8].

Misalkan kita diminta untuk membuat model klasifikasi kanker. Kita dapat mengklasifikasikan pasien menjadi dua kelas $\{kanker, normal\}$.

Apabila kita ingin meminimalkan *error* saja maka kita ingin mengklasifikasikan secara tepat orang yang kanker dianggap memiliki kanker dan yang tidak dianggap sebagai tidak. Akan tetapi, terdapat *tradeoff* yang berbeda saat salah klasifikasi. Apabila kita mengklasifikasikan orang yang normal sebagai kanker, konsekuensi yang mungkin adalah membuat pasien menjadi stres atau perlu melakukan pemeriksaan ulang. Tetapi bayangkan, apabila kita mengklasifikasikan orang kanker sebagai normal, konsekuensinya adalah penanganan medis yang salah. Kedua kasus ini memiliki beban yang berbeda. Secara sederhana, kesalahan klasifikasi bisa memiliki bobot berbeda untuk tiap kelasnya. Untuk penyederhanaan pembahasan, pada buku ini, kita anggap kesalahan klasifikasi memiliki bobot yang sama.

Fungsi tujuan pembelajaran (secara umum untuk merepresentasikan *error* atau *loss*) dituangkan dalam ***utility function***. Sekali lagi kami tekankan, tujuan *machine learning* adalah memaksimalkan kinerja. Kinerja diukur berdasarkan *utility function*. ***Loss* adalah ukuran seberapa dekat atau berbeda model yang dihasilkan dengan konsep asli**, sementara *error* adalah salah satu fungsi untuk mengukur *loss*.

Untuk mengukur nilai *loss*; dapat diekspresikan dengan *loss function*. Secara umum, ada dua macam *loss*, yaitu ***generalization loss/error*** dan ***training loss/error***. *Generalization loss/error* adalah ukuran sejauh mana algoritma mampu **memprediksi** *unobserved data* dengan tepat, karena kita hanya membangun model dengan data yang terbatas, tentunya bisa saja terdapat ketidakcocokan dengan data yang asli. Sedangkan *training loss/error* seperti namanya, ukuran *loss* saat *training*. Misalkan $q(x)$ adalah distribusi data asli. Menggunakan sampel data dengan distribusi $p(x)$, *generalization loss* dan *training loss* dapat dihitung dengan persamaan 2.13. Fungsi ini disebut sebagai ***cross entropy loss***. Perhatikan, masih banyak fungsi lain yang bisa digunakan untuk mengukur *loss*.

$$H = -\sum_{i=1}^{N} q(x) \log\left(p(x)\right) \tag{2.13}$$

Tentunya sekarang kamu bertanya-tanya. Kita tidak mengetahui bagaimana $q(x)$ aslinya, bagaimana cara menghitung *generalization loss*? Nah, untuk itulah ada teknik-teknik pendekatan distribusi populasi $q(x)$, misalnya **maximum likelihood method, maximum posterior method** dan *Bayesian method* (silahkan dieksplorasi). Ekspektasi terhadap biasanya $q(x)$ diaproksimasi dengan menggunakan data sampel yang kita miliki. Bentuk persamaan 2.13 memiliki kaitan dengan *confidence*. Konsep ini akan dijelaskan lebih detil pada bab 5.

Secara lebih filosofis, berkaitan dengan meminimalkan *loss*; tugas *machine learning* adalah untuk menemukan struktur tersembunyi (*discovering hidden structure*). Hal ini sangat erat kaitannya dengan *knowledge discovery* dan *data*

mining. Bila kamu membuka forum di internet, kebanyakan akan membahas perihal *learning machine* yang memaksimalkan akurasi (meminimalkan *error*). Selain harus memaksimalkan akurasi (meminimalkan salah *assignment*), kita juga harus mampu membuat model yang cukup generik. Artinya tidak hanya memiliki kinerja tinggi pada *training data*, tapi juga mampu memiliki kinerja yang baik untuk *unseen data*. Hal ini dapat tercapai apabila model yang dihasilkan melakukan inferensi yang mirip dengan inferensi sebenarnya (konsep asli). Kami tekankan kembali, meminimalkan *loss* adalah hal yang lebih penting, dimana meminimalkan *error* dapat digunakan sebagai sebuah *proxy* untuk mengestimasi *loss* (pada banyak kasus).

2.8 Hypothesis Testing

Diberikan dua model pembelajaran mesin dengan kinerja A dan B. Kita ingin memutuskan model pembelajaran mesin mana yang "lebih baik". Perhatikan, kualitas model tidak hanya tergantung pada satu metrik (misal akurasi), tetapi kita juga mempertimbangkan kompleksitas model (*running time*) dan sebagainya. Hal ini membuat keputusan model mana yang "lebih baik" menjadi cukup kompleks. Demi penyederhanaan pembahasan, anggap kita hanya melihat dari sisi performa yang diukur menggunakan skor tertentu.

Banyak makalah penelitian pembelajaran mesin yang hanya melihat nilai kinerja secara mentah. Artinya, apabila kinerja $A > B$ maka model A memiliki "lebih baik" dari model B. Hal ini terkadang tidak sesuai dengan prinsip statistik, dimana suatu model bisa saja mendapatkan kinerja A secara kebetulan. Konsep *Statistical hypothesis testing* menjadi penting untuk menarik konklusi apakah memang benar $A > B$ secara tidak kebetulan.

Konsep ini menjadi semakin penting saat menggunakan *artificial neural network* (ANN). Parameter model ANN pada umumnya diinisiasi secara *random*. Artinya, apabila kita melatih arsitektur yang sama sebanyak dua kali, kita belum tentu mendapatkan model dengan kinerja sama persis. Reimers dan Gurevych [13] menjelaskan betapa pentingnya melatih model ANN berkali-kali.

Kita hanya dapat menyimpulkan dengan benar model mana yang "lebih baik" hanya setelah melakukan *statistical hypothesis testing*. Ada banyak cara untuk melakukan *hypothesis testing*, dan tes yang digunakan berbeda tergantung metrik *performance measure*. Buku ini tidak akan membahas dengan detil, dan kami merekomendasikan untuk membaca paper oleh Dror et al. [14, 15]. Kami harap pembaca dapat menangkap bahwa memutuskan apakah suatu model A lebih baik dari B tidak cukup hanya dengan melihat ukuran kinerja secara numerik, tetapi hal ini harus dibuktikan menggunakan *statistical hypothesis testing*.

2.9 Teori Informasi

Kami tidak akan membahas bagian ini terlalu detail, jika kamu membaca buku, topik ini sendiri bisa mencapai satu buku [16]. Mudah-mudahan bab ini dapat memberikan gambaran (serius, ini sekedar gambaran!). *Information Theory*/Teori Informasi menjawab dua pertanyaan fundamental, pertama: bagaimana cara kompresi data terbaik (jawab: *entropy*); kedua: apakah cara transmisi komunikasi terbaik (jawab: channel capacity) [16]. Dalam *statistical learning theory*, fokus utama adalah menjawab pertanyaan pertama, yaitu bagaimana melakukan kompresi informasi. Contoh aplikasi *entropy* adalah *decision tree learning*.

Pada *machine learning*, kita ingin fitur pembelajaran yang digunakan mampu melambangkan *information source properties*. Artinya, kita ingin memilih fitur yang memuat informasi terbanyak (relatif terhadap *information source*). Karena hal tersebut, mengerti entropy menjadi penting. Ada sebuah strategi pemilihan fitur (*feature selection*) dengan membangun *decision tree*. Awalnya kita bentuk *training data* dengan semua kemungkinan fitur, kemudian mengambil beberapa fitur yang dekat dengan *root*. Hal tersebut dimaksudkan untuk mencari fitur yang memuat banyak informasi. Kemudian, fitur tersebut dapat dicoba pada algoritma learning lainnya. Detil akan dijelaskan pada bab 6.

2.9.1 Entropy

Diberikan sebuah random variabel x, kita ingin mengetahui seberapa banyak informasi yang kita dapatkan ketika kita mengobservasi sebuah nilai spesifik x_i. Kuantitas informasi yang kita dapatkan bisa dipandang sebagai "*degree of surprise*" [8]. Misalkan kita mengetahui seorang teman A sering makan es krim. Suatu ketika kita diberitahu bahwa dia sedang makan es krim, tentu kita tidak heran lagi karena hal tersebut sudah lumrah. Tetapi, apabila kita diberitahu bahwa teman A tidak memakan es krim yang diberikan teman B (padahal kita tahu dia suka), maka akan ada efek "kaget". Kasus kedua memuat lebih banyak informasi karena suatu kejadian yang seharusnya tidak mungkin, terjadi. Hal ini dikuantifikasi dengan persamaan ***Shannon Entropy*** 2.14.

$$S(x) = -\sum_{i=1}^{N} p(x_i)\log(p(x_i)) \tag{2.14}$$

Mari kita ambil contoh dari Bishop [8]. Misalkan sebuah *random variable* x memiliki 8 kemungkinan kejadian yang kemungkinannya sama (yaitu $\frac{1}{8}$). *Entropy* untuk kasus ini adalah (log dalam basis 2) diberikan oleh

$$S = -8\frac{1}{8}\log(\frac{1}{8}) = 3 \tag{2.15}$$

Sekarang mari kita ambil contoh dari [16]. Misalkan sebuah *random variable* x memiliki 8 kemungkinan kejadian $\{a, b, c, d, ..., h\}$ dengan peluang $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{64}, \frac{1}{64}, \frac{1}{64}, \frac{1}{64}$
Maka *entropy*-nya adalah 2. Dari contoh ini, kita tahu bahwa distribusi yang uniform memiliki *entropy* yang lebih besar dibanding distribusi yang tidak uniform. Banyaknya informasi sebanding dengan **turunnya** nilai *entropy*.

Seperti yang telah diceritakan sebelumnya, *event* yang memiliki "efek kaget" memiliki banyak informasi. Dari sisi *information transmission*, dapat diinterpretasikan kita dapat mengirimkan data sebuah distribusi dengan jumlah bit lebih sedikit untuk distribusi yang uniform. Distribusi yang memberikan nilai *entropy* maksimal adalah distribusi Gaussian [8]. Nilai *entropy* bertambah seiring *variance* distribusi bertambah. Dari sisi fisika, kamu dapat mempelajari *entropy* pada *statistical mechanics* (*microstate*, *macrostate*).

Perhatikan, *entropy* (persamaan 2.14) dan *cross entropy* (persamaan 2.13) memiliki persamaan agak berbeda (adanya distribusi asli q). Tetapi interpretasi kedua formula adalah sama. Distribusi yang memiliki nilai cukup uniform memiliki nilai *entropy*/*cross entropy* yang tinggi, sementara memiliki nilai rendah untuk distribusi yang condong ke nilai tertentu (*skewed*).

2.9.2 Relative Entropy dan Mutual Information

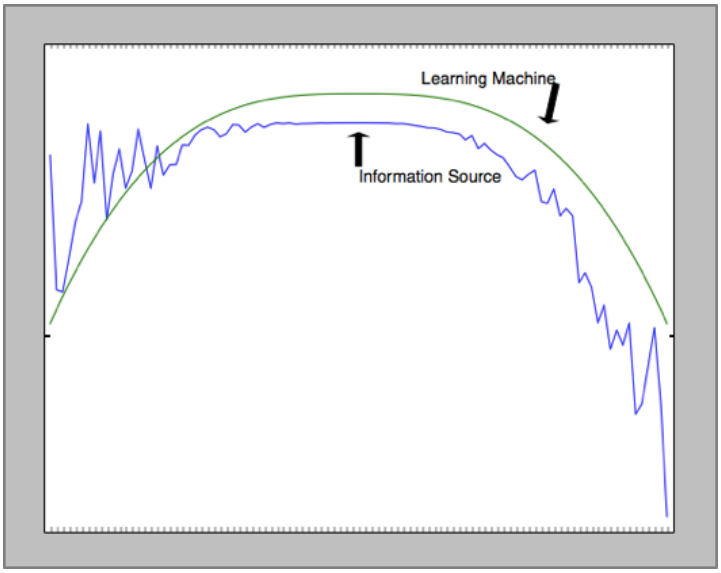
Kami harap kamu masih ingat materi bab 1, karena materi bagian ini juga menyinggung kembali materi tersebut. Misalkan kita mempunyai data dengan *probability density function* $q(x)$. Sebuah *learning machine* mengaproksimasi data tersebut dengan *probability density function* $p(x)$. Ingat! *Machine learning* adalah pendekatan (*approximation*). Ketika kita melakukan aproksimasi, seringkali aproksimasi yang dilakukan tidaklah tepat seperti pada Gambar 2.6.

Tentunya kita ingin tahu seberapa bagus aproksimasi kita, untuk mengukurnya terdapat sebuah perhitungan yang bernama **Kullback-Leibler Divergence** (KL-divergence). Secara konseptual, dirumuskan sebagai persamaan 2.16. Perlu diperhatikan $KL(q||p) \neq KL(p||q)$ (kecuali $p = q$).

$$KL(q||p) = -\int q(x) \log\left(\frac{q(x)}{p(x)}\right) dx \tag{2.16}$$

Persamaan 2.16 dapat diminimalkan jika dan hanya jika $q(x) = p(x)$. Kita dapat menganggap KL-divergence sebagai ukuran seberapa jauh aproksimasi dan distribusi populasi. Akan tetapi, kita tidak mengetahui $q(x)$. Karena itu, kita harus mengaproksimasi KL-divergence. Misalkan kita diberikan *training data* $\mathbf{x} = \{x_1, x_2, ...x_n\}$ yang kita asumsikan diambil (*drawn*) dari suatu distribusi $q(x)$. Lalu kita membuat *learning machine* $p(x \mid \mathbf{w})$. Ekspektasi terhadap $q(x)$ dapat diaproksimasi dengan menggunakan data sampel ini, dituangkan menjadi persamaan 2.17 [8].

$$KL(q||p) \approx \frac{1}{N} \sum_{i=1}^{N} \left(-\log(p(x_i \mid \mathbf{w})) + \log(q(x_i))\right) \tag{2.17}$$



Gambar 2.6: Information source vs learning machine.

KL-divergence disebut juga sebagai *relative entropy*.[3] Dari sisi pemrosesan informasi, KL-divergence dapat diinterpretasikan sebagai berapa informasi tambahan rata-rata untuk mengirimkan data distribusi dengan menggunakan fungsi aproksimasi dibanding menggunakan distribusi sebenarnya, seberapa pengurangan ketidakyakinan terhadap *posterior* seiring diberikannya data observasi yang baru. Dengan kata lain, **seiring diberikan observasi yang baru, kita semakin yakin terhadap nilai posterior** (semakin banyak jumlah sampel yang kita miliki maka model lebih dapat dipercaya). KL-Divergence = Cross Entropy - Entropy (buktikan!).[4]

2.10 Matriks

Subbab ini adalah pengingat untuk operasi perjumlahan, pengurangan, perkalian, dan transpose matriks karena banyak digunakan di buku ini. Diberikan dua buah matriks **U** dan **V**. **U** dan **V** dapat dijumlahkan jika dan hanya jika dimensi kedua matriks itu sama. Perjumlahan matriks dinotasikan dengan

[3] kamu dapat mencoba library entropy di scipy (python) untuk mendapat gambaran lebih detil
`https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.entropy.html`

[4] https://towardsdatascience.com/entropy-cross-entropy-and-kl-divergence-explained-b09cdae917a

$\mathbf{U} + \mathbf{V} = \mathbf{C}$. Matriks $\mathbf{C}$ memiliki dimensi yang sama dengan $\mathbf{U}$ dan $\mathbf{V}$. Nilai elemen baris ke-i dan kolom ke-j ($\mathbf{C}_{i,j}$) dihitung sebagai penjumlahan nilai elemen matriks $\mathbf{U}$ dan $\mathbf{V}$ pada baris dan kolom yang bersesuaian, seperti diilustrasikan pada persamaan 2.18. Pengurangan dua buah matriks dilakukan serupa.

$$\mathbf{C}_{i,j} = \mathbf{U}_{i,j} + \mathbf{V}_{i,j} \tag{2.18}$$

Dua buah matriks $\mathbf{U}$ dan $\mathbf{V}$ dapat dikalikan jika $\mathbf{U}$ memiliki kolom sebanyak baris pada $\mathbf{V}$. Misalkan matriks $\mathbf{U}$ berdimensi $N \times M$ dan $\mathbf{V}$ berdimensi $M \times O$, maka kedua matriks tersebut dapat dikalikan dan menghasilkan matriks $\mathbf{C}$ dengan dimensi $N \times O$ (dimensi baris $\mathbf{U}$ dan kolom $\mathbf{V}$), dimana tiap elemen pada matriks $\mathbf{C}$ dihitung dengan persamaan 2.19 (operasi antara vektor baris dan vektor kolom).

$$\mathbf{C}_{x,y} = \sum_{i=1}^{M} \mathbf{U}_{x,i} + \mathbf{V}_{i,y} \tag{2.19}$$

Selain perkalian antar dua buah matriks, sebuah matriks juga dapat dikalikan dengan skalar, dinotasikan dengan $a\mathbf{U}$. Hasil perkalian adalah sebuah matriks dengan dimensi yang sama dengan $\mathbf{U}$, dimana tiap elemen dikalikan dengan nilai skalar.

$$(a\mathbf{U})_{i,j} = a \times \mathbf{U}_{i,j} \tag{2.20}$$

Suatu matriks $\mathbf{U}$ berdimensi $N \times M$ apabila di transpose menghasilkan matriks $\mathbf{U}^T$ berdimensi $M \times N$, dimana elemen ke-i, j pada matriks $\mathbf{U}^T$ adalah elemen ke-j, i pada matriks $\mathbf{U}$, seperti diilustraiskan pada persamaan 2.19.

$$\mathbf{U}^T_{i,j} = \mathbf{U}_{j,i} \tag{2.21}$$

Ada satu istilah lagi yang perlu kamu ketahui yaitu **tensor**. Tensor adalah generalisasi untuk vektor (1 dimensi) dan matriks (2 dimensi) yang memiliki N dimensi. Tensor sering digunakan untuk notasi pada *artificial neural network*. Tetapi demi kemudahan pengertian, penulis menggunakan notasi matriks.

2.11 Bacaan Lanjutan

Untuk lebih mengerti, silahkan membaca buku *statistical mechanis* oleh Hitoshi Nishimori [17], buku probabilitas dan statistika oleh Walpole et al. [18] atau Brian Caffo [10], buku aljabar linear oleh Gilbert Strang [19] dan buku *statistical learning theory* oleh James et al. [20].

Soal Latihan

2.1. KL-divergence
Cari tahu lebih lanjut apa itu Kullback-Leibler (KL) Divergence. Apa hubungan KL-divergence dengan *utility function*? Pada kasus apa saja kita dapat menggunakan KL-divergence sebagai *utility function*?

2.2. Utility Function
Selain *utility function* yang telah disebutkan, sebutkan dan jelaskan *utility function* lainnya!

2.3. Gaussian Mixture Model

(a) Sebutkan algoritma-algoritma *machine learning* yang (*in a sense*) dapat mengaproksimasi *Gaussian Mixture Model*!
(b) Apa yang begitu spesial pada GMM sehingga algoritma *machine learing* mencoba mengaproksimasi GMM?

3

Data Analytics

> "Hiding within those mounds of data is knowledge that could change the life of a patient, or change the world"
>
> Atul Butte

Bab ini memuat penjelasan tahapan-tahapan umum untuk analisis data, serta beberapa karakteristik tipe data. Materi pada bab ini dapat dianggap sebagai kerangka berpikir (*framework*) atau langkah kerja. Karena buku ini hanya bersifat pengantar, materi yang disampaikan mungkin kurang lengkap. Penulis menyarankan pembaca untuk membaca buku oleh Witten et al. [21] dan Jeff Leek [22].

3.1 Pengenalan Data Analytics

Secara umum, subbab ini adalah ringkasan dari buku Jeff Leek [22]. Untuk detailnya, kamu dapat membaca buku tersebut secara langsung. Penulis merekomendasikan buku tersebut karena ringkas dan mudah dipahami bahkan oleh pemula.

Kita tahu di dunia ini ada banyak masalah. Masalah adalah ketika tujuan yang diinginkan tidak tercapai (*current state* bukanlah *desired state*). Agar *current state* menjadi *desired state*, kita melakukan kegiatan yang disebut penyelesaian masalah (*problem solving*). Tiap bidang (domain) mendefinisikan permasalahan secara berbeda. Oleh karena itu, mengetahui teknik *machine learning* tanpa mengetahui domain aplikasi adalah sesuatu yang kurang baik (semacam buta). Kamu memiliki ilmu, tetapi tidak tahu ilmunya mau digunakan untuk apa. Contohnya, bidang keilmuan pemrosesan bahasa alami (*natural language processing*) menggunakan *machine learning* untuk mengklasifikasikan teks; bidang keilmuan pemrosesan suara menggunakan *machine*

learning untuk mentranskripsikan suara manusia menjadi teks. Tiap bidang merepresentasikan permasalahan ke dalam formulasi yang berbeda. Bisa jadi bentuk komputasi (representasi) pada bidang satu berbeda dengan bidang lainnya. Hal ini perlu kamu ingat karena interpretasi representasi sangat bergantung pada konteks permasalahan (domain). Buku ini adalah pengenalan teknik yang bersifat umum.

Seperti yang sudah dijelaskan pada bab-bab sebelumnya. *Machine learning* adalah inferensi berdasarkan data. *Raw data* atau data mentah adalah sekumpulan fakta (*record*, *event*) yang kemungkinan besar tidak memberikan penjelasan apapun. Sama halnya dengan kebanyakan data di dunia nyata, *raw data* bersifat tidak rapih, misalnya mengandung *missing value*, i.e., ada data yang tidak memiliki label padahal data lainnya memiliki label (ingat kembali materi bab 1). Agar mampu menganalisis *raw data* menggunakan teknik *machine learning*, pertama-tama kita harus merapikan data sesuai dengan format yang kita inginkan (*dataset*). Kemudian, mencari fitur-fitur yang dapat merepresentasikan data. Kedua kegiatan ini secara umum dikenal dengan istilah *pre-processing*. Setelah *pre-processing*, kita menggunakan teknik-teknik yang ada untuk menemukan pola-pola yang ada di data (membangun model). Pada beberapa bidang, *pre-processing* adalah tahapan yang memakan waktu paling banyak pada eksperimen *machine learning*, sedangkan proses melatih model mungkin memakan waktu jauh lebih singkat.

Dalam komunitas peneliti basis data, dipercaya bahwa data memiliki sangat banyak relasi yang mungkin tidak bisa dihitung. Teknik *machine learning* hanya mampu mengeksplorasi sebagian relasi yang banyak itu. Lalu, kita analisis informasi yang kita dapatkan menjadi pengetahuan yang digunakan untuk memecahkan permasalahan atau membuat keputusan.

Setelah kita menganalisis data dan mendapatkan pengetahuan baru, pengetahuan yang kita temukan dari data pada umumnya dipresentasikan (konferensi, rapat, dsb). Hal umum yang dipaparkan saat presentasi, yaitu:

1. *Performance measure*. Seberapa "bagus" model atau metode yang kamu usulkan, dibandingkan menggunakan model yang sudah ada. *Performance measure* biasa disajikan dalam bentuk tabel. Perhatikan, mengatakan model/metode kamu "lebih bagus" adalah suatu hal subjektif, untuk itu gunakanlah metode kuantitatif, seperti *p-value* dalam statistik (*hypothesis testing*)[1] untuk mengatakan bahwa memang metode kamu lebih baik (berbeda) dari *baseline*.

2. Tren. Bagaimana pola-pola umum yang ada di data, sesuai dengan tujuan analisis (permasalahan). Biasa disajikan dalam bentuk teks, kurva, atau grafik.

3. *Outlier*. Sajikan data-data yang "jarang" atau tidak sesuai dengan tren yang ada. Apa beda sifat data *outlier* ini dibanding data pada tren? Kamu

[1] https://onlinecourses.science.psu.edu/statprogram/node/138

harus menganalisis hal ini untuk meningkatkan *performance measure* pada penelitian atau analisis mendatang. Apakah *outlier* ini penting untuk diurus atau bisa dipandang sebelah mata tanpa membahayakan keputusan/sistem? Tidak jarang kamu mendapat inspirasi untuk meningkatkan kinerja sistem setelah menganalisis *outlier*.

Langkah kerja yang dijelaskan ini adalah pekerjaan rutin *data scientist*. Penulis ingin menekankan sekali lagi, bahwa memahami *machine learning* saja tidak cukup, **kamu harus memahami domain permasalahan** agar mampu melakukan analisis dengan tepat. Terdapat banyak hal yang hanya mampu kamu pahami dari menganalisis data, apabila kamu mengerti domain aplikasi.

3.2 Nilai Atribut dan Transformasi

Perhatikan Tabel 3.1 yang merupakan contoh *dataset* pada *machine learning*. **Dataset** adalah kumpulan sampel. Seorang anak ingin bermain tenis, tetapi keputusannya untuk bermain tenis (*play*) tergantung pada empat variabel {*outlook, temperature, humidity, windy*}. Keempat variabel ini disebut **fitur**. Setiap fitur memiliki **atribut** nilai dengan **tipe data** dan **range** tertentu. Keputusan untuk bermain (*play*) disebut sebagai label atau kelas (*class*). Pada bab 1 kamu telah diperkenalkan *supervised learning* dan *unsupervised learning*. Pada *supervised learning*, kita ingin mengklasifikasikan apakah seorang anak akan bermain atau tidak, diberikan fitur-fitur yang memuat kondisi observasi. Pada *unsupervised learning*, informasi kolom *play* tidak diberikan, kita harus mengelompokkan data tersebut sesuai dengan fitur-fiturnya (contoh lebih nyata diberikan pada bab 4).

Dari segi data statistik, terdapat beberapa tipe atribut [23]:

1. **Nominal**. Nilai atribut bertipe nominal tersusun atas simbol-simbol yang berbeda, yaitu suatu himpunan terbatas. Sebagai contoh, fitur *outlook* pada Tabel 3.1 memiliki tipe data **nominal** yaitu nilainya tersusun oleh himpunan {*sunny, overcast, rainy*}. Pada tipe nominal, tidak ada urutan ataupun jarak antar atribut. Tipe ini sering juga disebut **kategorial** atau **enumerasi**. Secara umum, tipe *output* pada *supervised learning* adalah data nominal.

2. **Ordinal**. Nilai ordinal memiliki urutan, sebagai contoh $4 > 2 > 1$. Tetapi jarak antar suatu tipe dan nilai lainnya tidak harus selalu sama, seperti $4-2 \neq 2-1$. Atribut ordinal kadang disebut sebagai **numerik** atau **kontinu**.

3. **Interval**. Tipe interval memiliki urutan dan *range* nilai yang sama. Sebagai contoh $1-5, 6-10, dst$. Kita dapat mentransformasikan/ mengkonversi nilai numerik menjadi nominal dengan cara merubahnya menjadi

id	outlook	temperature	humidity	windy	play (class)
1	sunny	hot	high	false	no
2	sunny	hot	high	true	no
3	overcast	hot	high	false	yes
4	rainy	mild	high	false	yes
5	rainy	cool	normal	false	yes
6	rainy	cool	normal	true	no
7	overcast	cool	normal	true	yes
8	sunny	mild	high	false	no
9	sunny	cool	normal	false	yes
10	rainy	mild	normal	false	yes
11	sunny	mild	normal	true	yes
12	overcast	mild	high	true	yes
13	overcast	hot	normal	false	yes
14	rainy	mild	high	true	no

Tabel 3.1: Contoh dataset *play tennis* (UCI machine learning repository).

interval terlebih dahulu. Lalu, kita dapat memberikan nama (simbol) untuk masing-masing interval. Misalkan nilai numerik dengan *range* $1-100$ dibagi menjadi 5 kategori dengan masing-masing interval adalah $\{1-20, 21-40, \ldots, 81-100\}$. Setiap interval kita beri nama, misal interval $81-100$ diberi nama *nilai A*, interval $61-80$ diberi nama *nilai B*.

4. **Ratio**. Tipe *ratio* (rasio) didefinisikan sebagai perbandingan antara suatu nilai dengan nilai lainnya, misalkan massa jenis (fisika). Pada tipe *ratio* terdapat *absolute zero* (semacam *ground truth*) yang menjadi acuan, dan *absolute zero* ini memiliki makna tertentu.

Secara umum, data pada *machine learning* adalah nominal atau numerik (ordinal). Variabel yang kita prediksi yaitu *play* disebut **kelas/*class*/label**. Untuk setiap baris pada Tabel 3.1, baris kumpulan nilai variabel non-kelas disebut **vektor fitur/*feature vector***. Contohnya pada Tabel 3.1, *feature vector*-nya untuk data dengan $id = 4$ adalah $x_4 = \{$*outlook=rainy, temperature=mild, humidity=high, windy=false*$\}$. *Feature vector* adalah representasi dari suatu observasi/data. Pada *machine learning*, kita melakukan operasi terhadap data pada representasi *feature vector*-nya. Kami serahkan pada pembaca untuk mencari contoh dataset dengan tipe numerik sebagai pekerjaan rumah.[2]

[2] https://www.cs.waikato.ac.nz/ml/weka/datasets.html

3.3 Ruang Konsep

Dengan data yang diberikan, kita ingin melakukan generalisasi aturan/ konsep yang sesuai dengan data. Hal ini disebut sebagai *inductive learning*. Cara paling sederhana untuk *inductive learning* adalah mengenumerasi seluruh kemungkinan kombinasi nilai sebagai *rule*, kemudian mengeleminasi *rule* yang tidak cocok dengan contoh. Metode ini disebut *list-then-eleminate*. Silahkan baca buku Tom Mitchell [4] untuk penjelasakn lebih rinci. Kemungkinan kombinasi nilai ini disebut sebagai ruang konsep (***concept space***). Sebagai contoh pada Tabel 3.1 himpunan nilai masing-masing atribut yaitu:

- $outlook = \{sunny, overcast, rainy\}$
- $temperature = \{hot, mild, cold\}$
- $humidity = \{high, normal\}$
- $windy = \{true, false\}$
- $play = \{yes, no\}$

sehingga terdapat $3 \times 3 \times 2 \times 2 \times 2 = 72$ kemungkinan kombinasi. Tentunya kita tidak mungkin mengenumerasi seluruh kemungkinan kombinasi nilai karena secara praktikal, atribut yang digunakan banyak. Terlebih lagi, apabila mengenumerasi kombinasi atribut bertipe numerik.

Ada algoritma lain yang mendaftar "seluruh kemungkinan kombinasi" bernama *candidate-elemination algorithm* yang lebih efisien dibanding *list-then-eliminate*. Akan tetapi, algoritma ini *computationally expensive* secara praktikal, dalam artian memiliki kompleksitas yang besar dan tidak bisa menyelesaikan permasalahan nyata. Kamu dapat membaca algoritma ini pada buku Tom Mitchell [4] juga. Selain *inductive learning*, kita juga dapat melakukan *deductive learning* yaitu melakukan inferensi dari hal general menjadi lebih spesifik. Walau demikian, secara praktis kita sebenarnya melakukan *inductive learning*.

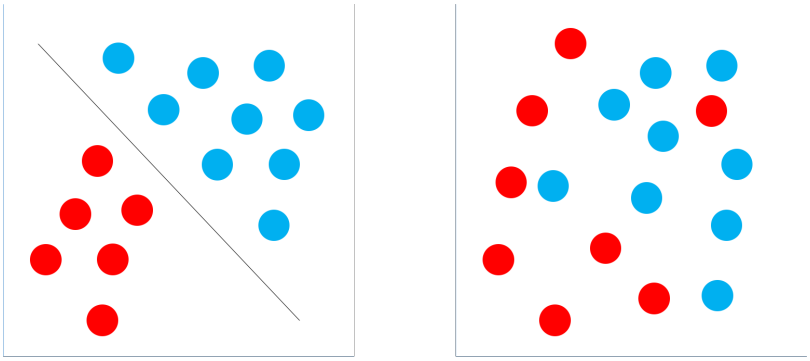
3.4 Linear Separability

id	humidity	windy	swim (class)
1	high	high	yes
2	normal	normal	no

Tabel 3.2: Contoh dataset *linearly separable*.

Perhatikan Tabel 3.2. Data pada tabel tersebut kita sebut ***linearly separable***. Sederhananya, untuk suatu nilai tertentu, fitur hanya berkorespondensi dengan kelas tertentu. Ambil contoh pada Tabel 3.2, saat *humidity=high* maka

swim=yes. Secara geometris, bila kita proyeksikan *feature vector* ke suatu ruang dimensi, memisahkan kelas satu dan kelas lainnya dapat diperoleh dengan cara menciptakan garis linier (*linear line*)–secara lebih umum, menggunakan *hyperplane*.[3] Ilustrasi dapat dilihat pada Gambar 3.1. Sementara pada Tabel 3.1, bila kita hanya melihat fitur *humidity* saja, ketika *humidity=high* bisa jadi *play=yes* atau *play=no*. Kasus ini disebut ***non-linearly separable***. Hidup kita tentu akan mudah apabila seluruh data bersifat *linearly separable*, sayangnya kebanyakan data yang ada bersifat *non-linearly separable*.



Gambar 3.1: *Linearly* vs *Non-Linearly Separable*.

Untuk memudahkan proses pada data *non-linearly separable*, kita pertamatama mentransformasikan data menjadi *linearly-separable*. Kita dapat menggunakan teknik transformasi data menggunakan *kernel function* seperti *radial basis function*.[4] Pada umumnya, *kernel function* mentransformasi data menjadi lebih tinggi (semacam menambah fitur). Misal dari data yang memiliki dua fitur, ditransformasi menjadi memiliki tiga fitur. Akan tetapi, hal ini tidak praktikal untuk banyak kasus (dijelaskan pada bab 11). Cara lainnya adalah memisahkan data menggunakan model non-linear, contoh: *artificial neural network*. Hal ini penting dipahami karena data yang bersifat *linearly separable* mudah dipisahkan satu sama lain sehingga mudah untuk melakukan *classification* atau *clustering*.

3.5 Seleksi Fitur

Pada subbab sebelumnya, telah dijelaskan bahwa kita dapat mentransformasi data *non-linearly separable* menjadi *linearly separable* dengan cara menam-

[3] https://en.wikipedia.org/wiki/Hyperplane
[4] Silakan baca teknik transformasi lebih lanjut pada literatur lain.

bah dimensi data. Pada bab ini, penulis akan menjelaskan justru kebalikannya! Pada permasalahan praktis, kita seringkali menggunakan banyak fitur (*computationally expensive*). Kita ingin menyederhanakan fitur-fitur yang digunakan, misalkan dengan memilih subset fitur awal, atas dasar beberapa alasan [22, 4]:[5]

1. Menyederhanakan data atau model agar lebih mudah dianalisis.
2. Mengurangi waktu *training* (mengurangi kompleksitas model dan inferensi).
3. Menghindari *curse of dimensionality*. Hal ini dijelaskan lebih lanjut pada bab 12.
4. Menghapus fitur yang tidak informatif.
5. Meningkatkan generalisasi dengan mengurangi *overfitting*.

Salah satu contoh cara seleksi fitur adalah menghapus atribut yang memiliki *variance* bernilai 0. Berdasarkan *information theory* atau *entropy*, fitur ini tidak memiliki nilai informasi yang tinggi. Dengan kata lain, atribut yang tidak dapat membedakan satu kelas dan lain bersifat tidak informatif. Kamu dapat membaca beberapa contoh algoritma seleksi fitur pada library sklearn.[6]

3.6 Classification, Association, Clustering

Pada *supervsied learning*, kita memprediksi kelas berdasarkan *feature vector* yang merepresentasikan suatu sampel (data/observasi). *Feature vector* bisa diibaratkan sebagai sifat-sifat atau keadaan yang diasosiasikan dengan kelas. Pada *supervised learning*, setiap *feature vector* berkorespondisi dengan kelas tertentu. Mencari kelas yang berkorespondensi terhadap suatu *input* disebut **klasifikasi** (***classification***). Contoh klasifikasi adalah mengkategorikan gambar buah (e.g. apel, jeruk, dsb). Sementara itu, apabila kita ingin mencari hubungan antara satu atribut dan atribut lainnya, disebut **association**. Sebagai contoh pada Tabel 3.1, apabila $outlook = sunny$, maka sebagian besar $humidity = high$. Di lain pihak, pada *unsupervised learning* tidak ada kelas yang berkorespondensi; kita mengelompokkan data dengan sifat-sifat yang mirip, disebut ***clustering***. Contoh *clustering* adalah pengelompokkan barang di supermarket. Perlu kamu catat bahwa *unsupervised learning* $\neq$ *clustering*. *Clustering* adalah salah satu *task* pada *unsupervised learning*.

Pada Tabel 3.1, hanya ada dua kelas, klasifikasi data ini disebut ***binary classification***. Apabila kelas klasifikasi lebih dari dua (*mutually exclusive*), disebut ***multi-class classification***. Apabila kelas-kelas tersebut tidak bersifat *mutually exclusive*, maka kita melakukan ***multi-label classification***. Mohon bedakan antara ***multi-label classification*** dan ***multi-level/hierarchical classification***. Pada *multi-level/hierarchical classification*, pertama-tama kita melakukan klasifikasi untuk suatu kelas generik, lalu

[5] https://en.wikipedia.org/wiki/Feature_selection
[6] http://scikit-learn.org/stable/modules/feature_selection.html

dilanjutkan mengklasifikan data ke kelas yang lebih spesifik. Contoh *multi-level classification* adalah *kingdom* (biologi), pertama diklasifikasikan ke *kingdom animalia*, lalu lebih spesifiknya ke *phylum Vertebrata*, dst. *Multi-label classification* hanya proses klasifikasi ke dalam banyak "kelas" tanpa tinjauan hirarkis.

Multi-class classification yang telah dijelaskan sebelumnya disebut juga sebagai *hard classification*, artinya apabila data diklasifikasikan ke kelas tertentu, maka tidak mungkin data berada di kelas lainnya (ya atau tidak). *Multi-label classification* bersifat lebih *soft*, karena dapat mengklasifikasikan ke beberapa kelas, misal data X memiliki masuk ke kategori kelas A, B dan C sekaligus (dengan nilai probabilitas masing-masing).

3.7 Mengukur Kinerja

Pada bab 1, sudah dijelaskan bahwa kita harus mengukur kinerja model dengan cara yang kuantitatif. Pada saat proses latihan, kita ingin model mengoptimalkan suatu nilai *utility function*, misal meminimalkan nilai *error* atau *entropy*. Pada saat latihan, model pembelajaran mesin dapat mengoptimalkan *utility function* yang berbeda-beda (tergantung algoritma).

Kita juga dapat mengevaluasi model secara eksternal dengan melewatkan data pada model (umumnya *validation* dan *testing data*), kemudian mengevaluasi prediksi final model tersebut dengan ukuran *performance measure*. *Performance measure* dapat mengukur kinerja model yang dikonstruksi oleh algoritma yang berbeda (akan lebih jelas sembari kamu membaca buku ini).

Sebagai contoh, kamu dapat membandingkan kinerja prediksi model dengan menggunakan metrik seperti akurasi, presisi, *recall*, F1-measure, BLEU [24], ROUGE [25], *intra-cluster similarity*, dsb. Masing-masing *performance measure* mengukur hal yang berbeda. Perlu kamu ketahui bahwa memilih ukuran kinerja tergantung pada domain permasalahan. Misalkan pada translasi otomatis, peneliti menggunakan ukuran BLEU; pada peringkasan dokumen, menggunakan ROUGE. Sementara itu, pada *information retrival*/sistem temu balik informasi menggunaan presisi, *recall*, F1-measure, atau *mean average precision* (MAP). Pada domain klasifikasi gambar, menggunakan akurasi. Masing-masing *performance measure* dapat memiliki karakteristik nilai optimal yang berbeda. Kamu harus mengerti domain permasalahan untuk mengerti cara mencapai titik optimal. Sebagai pengantar, diktat ini tidak dapat membahas seluruh domain. Dengan demikian, kamu harus membaca lebih lanjut literatur spesifik domain, misal buku pemrosesan bahasa alami atau sistem temu balik informasi, dsb. Sebagai contoh, untuk permasalahan klasifikasi, akurasi sering digunakan. Akurasi didefinisikan pada persamaan 3.1. Buku ini akan membahas *performance measure* dengan lebih lengkap pada bab 9.

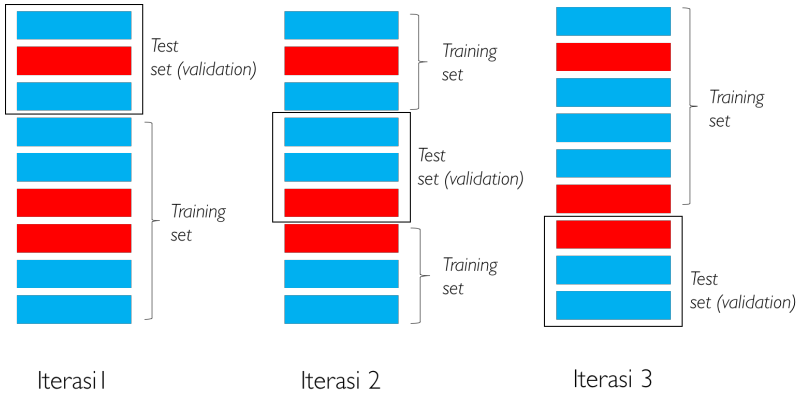
$$\text{akurasi} = \frac{\#input \text{ diklasifikasikan dengan benar}}{\text{banyaknya data}} \tag{3.1}$$

3.8 Evaluasi Model

Ada beberapa hal yang perlu kamu catat tentang proses evaluasi suatu model pembelajaran mesin:

1. ***Data splitting***. Seperti yang sudah dijelaskan pada subbab 1.5, pada umumnya kita memiliki *training*, *validation*/*validation*, dan *testing data*. Mesin dilatih menggunakan *training data*, saat proses *training*, *performance measure* diukur berdasarkan kemampuan mengenali/ mengeneralisasi *validation data*. Perlu diketahui, *performance measure* diukur menggunakan *validation data* untuk menghindari *overfitting* dan *underfitting*. Setelah selesai dilatih, maka model hasil pembelajaran dievaluasi dengan *testing data*. *Training*, *validation*, dan *testing* data tersusun oleh data yang independen satu sama lain (tidak beririsan) untuk memastikan model yang dihasilkan memiliki generalisasi cukup baik.
2. ***Overfitting dan Underfitting***. *Overfitting* adalah keadaan ketika model memiliki kinerja baik hanya untuk *training data*/*seen examples* tetapi tidak memiliki kinerja baik untuk *unseen examples*. *Underftting* adalah keadaan ketika model memiliki kinerja buruk baik untuk *training data* dan *unseen examples*. Hal ini akan dibahas lebih detil pada subbab 5.8.
3. ***Cross validation***. *Cross validation* adalah teknik untuk menganalisis apakah suatu model memiliki generalisasi yang baik (mampu memiliki kinerja yang baik pada *unseen examples*). Seperti yang kamu ketahui, kita dapat membagi data menjadi *training*, *validation*, dan *testing data*. Saat proses *training*, kita latih model dengan *training data* serta dievaluasi menggunakan *validation data*. Teknik *cross validation* bekerja dengan prinsip yang sama, yaitu membagi sampel asli menjadi beberapa subsampel dengan partisi sebanyak K (K-*fold*). Ilustrasi diberikan oleh Gambar 3.2. Persegi panjang melambangkan suatu sampel. Saat proses *training*, kita bagi data menjadi *training data* dan *test data* (i.e., *validation data*). Hal ini diulang sebanyak K kali. Kita evaluasi kemampuan generalisasi model dengan merata-ratakan kinerja pada tiap iterasi. Setelah itu, model dengan kinerja terbaik (pada iterasi teretentu) digunakan lebih lanjut untuk proses *testing* atau dipakai secara praktis. Perlu diperhatikan, setiap subsampel sebaiknya memiliki distribusi yang sama dengan sampel aslinya (keseluruhan sampel); i.e., pada contoh, proporsi warna biru dan merah adalah sama tiap partisi tiap iterasi. Konsep tersebut lebih dikenal dengan ***stratified sampling***.[7]

[7] https://en.wikipedia.org/wiki/Stratified_sampling



Gambar 3.2: Ilustrasi 3-*fold cross validation*. Warna merah dan biru melambangkan sampel (*instance*) yang tergolong ke dua kelas berbeda.

3.9 Kategori Jenis Algoritma

Algoritma pembelajaran mesin dapat dibagi menjadi beberapa kategori. Dari sudut pandang apakah algoritma memiliki parameter yang harus dioptimasi, dapat dibagi menjadi:[8]

1. Parametrik. Pada kelompok ini, kita mereduksi permasalahan sebagai optimisasi parameter. Kita mengasumsikan permasalahan dapat dilambangkan oleh fungsi dengan bentuk tertentu (e.g., linier, polinomial, dsb). Contoh kelompok ini adalah model linier.
2. Non parametrik. Pada kelompok ini, kita tidak mengasumsikan permasalahan dapat dilambangkan oleh fungsi dengan bentuk tertentu. Contoh kelompok ini adalah *Naive Bayes*, *decision tree* (ID3) dan *K-Nearest Neighbors*.

Dari sudut pandang lainnya, jenis algoritma dapat dibagi menjadi:

1. Model linear, contoh regresi linear, regresi logistik, *support vector machine*.
2. Model probabilistik, contoh *Naive Bayes*, *hidden markov model*.
3. Model non-linear, yaitu (*typically*) *artificial neural network*.

Selain kedua skema pengkategorian ini, terdapat skema pengkategorian lain (silahkan eksplorasi sendiri).

[8] *Artificial Neural Network* dapat dikategorikan ke dalam keduanya.

3.10 Tahapan Analisis

Bagian ini adalah ringkasan bab ini. Untuk menganalisis data, terdapat langkah yang perlu kamu perhatikan

1. Memutuskan tujuan analisis data (*defining goal*)
2. Mendapatkan data
3. Merapihkan data
4. Merepresentasikan data sebagai *feature vector*
5. Melakukan transformasi dan/atau *feature selection* (mengurasi dimensi *feature vector*)
6. Melatih model (*training*) dan menganlisis kinerjanya pada *validation* (*development*) *data*.
7. Melakukan *testing* dan analisis model baik secara kuantitatif dan kualitatif
8. Menyajikan data (presentasi)

Saat ini, mungkin kamu merasa bab ini kurang berguna. Hal ini disebabkan karena konsep yang dipaparkan oleh bab ini bersifat *high-level*. Kamu mungkin harus membaca bab ini kembali setelah selesai membaca seluruh buku agar mendapatkan pemahaman lebih mendalam.

Soal Latihan

3.1. Konversi atribut
Sebutkan dan jelaskan macam-macam cara untuk mengkonversi atribut! Sebagai contoh, numerik-nominal dan nominal-numerik.

3.2. Transformasi data
Sebutkan dan jelaskan macam-macam cara transformasi data (e.g. merubah *non-linearly separable* menjadi *linearly separable*)

3.3. Seleksi fitur
Bacalah algoritma seleksi fitur pada *library* sklearn. Jelaskan alasan (*rationale*) dibalik penggunaan tiap algoritma yang ada!

3.4. Inductive Learning
Jelaskanlah algoritma *list-then-eliminate* dan *candidate-elimination*!

3.5. Tahapan analisis
Agar mampu memahami tahapan analisis data dan pembelajaran mesin secara lebih praktikal, kerjakanlah tutorial berikut!
https://scikit-learn.org/stable/tutorial/basic/tutorial.html

Bagian II

Algoritma Pembelajaran Mesin

4

Algoritma Dasar

> "It is a capital mistake to
> theorize before one has data."
>
> Arthur Conan Doyle

Sebelum masuk ke algoritma *machine learning* yang cukup modern/ matematis, kami akan memberi contoh algoritma yang lebih mudah yaitu **Naive Bayes**, **K-means**, dan **K-nearest-neighbor**. Algoritma-algoritma ini tergolong *non-parametrik*. Bab ini akan memuat contoh sederhana *supervised* dan *unsupervised learning*. Mudah-mudahan bab ini memberikan kamu gambaran aplikasi *machine learning* sederhana.

4.1 Naive Bayes

Naive Bayes adalah algoritma *supervised learning* yang sangat sederhana [26]. Idenya mirip dengan probabilitas bayesian pada bab 2. Secara formal, persamaan *Naive Bayes* untuk klasifikasi diberikan pada persamaan 4.1 dimana c_i adalah suatu nilai kelas, C adalah pilihan kelas (himpunan), t adalah fitur (satu fitur, bukan *feature vector*) dan F adalah banyaknya fitur. Kita memprediksi kelas berdasarkan probabilitas kemunculan nilai fitur pada kelas tersebut.

Pertama, kita hitung *likelihood* suatu *feature vector* diklasifikasikan ke kelas tertentu berdasarkan bagaiman probabilitas korespondensi fitur-fiturnya terhadap kelas tersebut (persamaan 4.1). Kemudian, kita normalisasi *likelihood* semua kelas untuk mendapatkan probabilitas *class-assignment* (softmax– persamaan 4.2). Akhirnya, kita pilih kelas dengan probabilitas tertinggi (persamaan 4.3).

$$\text{likelihood}(c_i) = P(c_i) \prod_{f=1}^{F} P(t_f|c_i) \tag{4.1}$$

id	outlook	temperature	humidity	windy	play (class)
1	sunny	hot	high	false	no
2	sunny	hot	high	true	no
3	overcast	hot	high	false	yes
4	rainy	mild	high	false	yes
5	rainy	cool	normal	false	yes
6	rainy	cool	normal	true	no
7	overcast	cool	normal	true	yes
8	sunny	mild	high	false	no
9	sunny	cool	normal	false	yes
10	rainy	mild	normal	false	yes
11	sunny	mild	normal	true	yes
12	overcast	mild	high	true	yes
13	overcast	hot	normal	false	yes
14	rainy	mild	high	true	no

Tabel 4.1: Contoh dataset *play tennis* (UCI machine learning repository)

	outlook			temperature			humidity			windy		play (class)	
	yes	no		yes	no		yes	no		yes	no	yes	no
sunny	2	3	hot	2	3	high	3	4	false	6	2	9	5
overcast	4	0	mild	4	2	normal	6	1	true	3	3		
rainy	3	2	cool	3	1								

Tabel 4.2: Frekuensi setiap nilai atribut

	outlook			temperature			humidity			windy		play (class)	
	yes	no		yes	no		yes	no		yes	no	yes	no
sunny	2/9	3/5	hot	2/9	3/5	high	3/9	4/5	false	6/9	2/5	9/14	5/14
overcast	4/9	0/5	mild	4/9	2/5	normal	6/9	1/5	true	3/9	3/5		
rainy	3/9	2/5	cool	3/9	1/5								

Tabel 4.3: Probabilitas setiap nilai atribut

$$P_{assignment}(c_i) = \frac{\text{likelihood}(c_i)}{\sum_{c_j \epsilon C} \text{likelihood}(c_j)} \tag{4.2}$$

$$\hat{c_i} = \arg\max_{c_i \epsilon C} P_{assignment}(c_i) \tag{4.3}$$

Agar mendapatkan gambaran praktis, mari kita bangun model Naive Bayes untuk Tabel 4.1. Tabel ini disebut sebagai **dataset**, yaitu memuat *entry* data (tiap baris disebut sebagai ***instance***). Kita anggap Tabel 4.1 sebagai ***training data***. Untuk menghitung probabilitas, pertama-tama kita hitung terlebih dahulu frekuensi nilai atribut seperti pada Tabel 4.2, setelah itu kita bangun model probabilitasnya seperti pada Tabel 4.3.

Untuk menguji kebenaran model yang telah kita bangun, kita menggunakan ***testing data***, diberikan pada Tabel 4.4. *testing data* berisi *unseen example* yaitu contoh yang tidak ada pada *training data*.

id	outlook	temperature	humidity	windy	play (class)
1	sunny	cool	high	true	no

Tabel 4.4: Contoh testing data *play tennis* [7]

$$
\begin{aligned}
\text{likelihood}(play = yes) &= P(yes)P(sunny \mid yes)P(cool \mid yes)P(high \mid yes) \\
&\quad P(true \mid yes) \\
&= \frac{9}{14} * \frac{2}{9} * \frac{3}{9} * \frac{3}{9} * \frac{3}{9} \\
&= 0.0053 \\
\text{likelihood}(play = no) &= P(no)P(sunny \mid no)P(cool \mid no)P(high \mid no) \\
&\quad P(true \mid no) \\
&= \frac{5}{14} * \frac{3}{5} * \frac{1}{5} * \frac{4}{5} * \frac{3}{5} \\
&= 0.0206 \\
P_{assignment}(play = yes) &= \frac{\text{likelihood}(play = yes)}{\text{likelihood}(play = yes) + \text{likelihood}(play = no)} \\
&= \frac{0.0053}{0.0053 + 0.0206} \\
&= 0.205 \\
P_{assignment}(play = no) &= \frac{\text{likelihood}(play = no)}{\text{likelihood}(play = yes) + \text{likelihood}(play = no)} \\
&= \frac{0.0206}{0.0053 + 0.0206} \\
&= 0.795
\end{aligned}
$$

Karena $P_{assignment}(play = no) > P_{assignment}(play = yes)$ maka diputuskan bahwa kelas untuk *unseen example* adalah $play = no$. Proses klasifikasi untuk data baru sama seperti proses klasifikasi untuk *testing data*, yaitu kita ingin menebak kelas data. Karena model berhasil menebak kelas pada *training data* dengan tepat, akurasi model adalah 100% (kebetulan contohnya hanya ada satu).

Perhatikan! Kamu mungkin berpikir kita dapat langsung menggunakan *likelihood* untuk mengklasifikasi, karena probabilitas dengan *likelihood* terbesar akan dipilih. Hal ini cukup berbahaya apabila *likelihood* untuk masing-masing kelas memiliki jarak yang cukup dekat. Sebagai contoh, menghitung probabilitas apabila (kasus abstrak) *likelihood*=$\{0.70, 0.60\}$, sehingga probabilitas kelas menjadi=$\{0.54, 0.46\}$. Karena perbedaan probabilitas kelas relatif tidak terlalu besar (contoh ini adalah penyederhanaan), kita mungkin harus

berpikir kembali untuk mengklasifikasikan *instance* ke kelas pertama. Hal ini berkaitan dengan seberapa yakin kamu mengklasifikasikan suatu sampel ke kelas tertentu. Perbedaan *likelihood* yang besar menandakan keyakinan, sementara perbedaan yang tipis menandakan ketidakyakinan.

Perhatikan, jumlah *likelihood* mungkin lebih dari 1. Sementara, probabilitas atau jumlahnya berada pada range $[0, 1]$ $(0 \leq P \leq 1)$. Pada contoh sebelumnya, nilai *likelihood* diubah menjadi bentuk probabilitas dengan menggunakan teknik *softmax*. Fungsi *softmax* berbentuk seperti pada persamaan 4.2.

4.2 K-means

Pada *supervised learning* kita mengetahui kelas data untuk setiap *feature vector*, sedangkan untuk *unsupervised learning* kita tidak tahu. Tujuan *unsupervised learning* salah satunya adalah melakukan ***clustering***. Yaitu mengelompokkan data-data dengan karakter mirip. Untuk melakukan pembelajaran menggunakan **K-means** [27], kita harus mengikuti langkah-langkah sebagai berikut:

1. Tentukan jumlah kelompok yang kita inginkan.
2. Inisiasi **centroid** untuk setiap kelompok (pada bab ini, secara acak). Centroid adalah data yang merepresentasikan suatu kelompok (ibaratnya ketua kelompok).
3. Hitung kedekatan suatu data terhadap *centroid*, kemudian masukkan data tersebut ke kelompok yang **centroid**-nya memiliki sifat terdekat dengan dirinya.
4. Pilih kembali **centroid** untuk masing-masing kelompok, yaitu dari anggota kelompok tersebut (semacam memilih ketua yang baru).
5. Ulangi langkah-langkah sebelumnya sampai tidak ada perubahan anggota untuk semua kelompok.

id	rich	intelligent	good looking
1	yes	yes	yes
2	yes	no	no
3	yes	yes	no
4	no	no	no
5	no	yes	no
6	no	no	yes

Tabel 4.5: Contoh dataset orang kaya

Perhatikan Tabel 4.5, kita akan mengelompokkan data pada tabel tersebut menjadi dua *clusters* (dua kelompok) yaitu k_1, k_2 menggunakan algoritma

id	perbedaan dengan $centroid_1$	perbedaan dengan $centroid_2$	assignment
2	2	2	k_1
3	1	3	k_1
4	3	1	k_2
5	2	2	k_1

Tabel 4.6: *Assignment* K-means langkah 1

id	perbedaan dengan $centroid_1$	perbedaan dengan $centroid_2$	assignment
1	2	3	k_1
3	1	3	k_1
5	3	1	k_2
6	2	2	k_1

Tabel 4.7: *Assignment* K-Means langkah 2

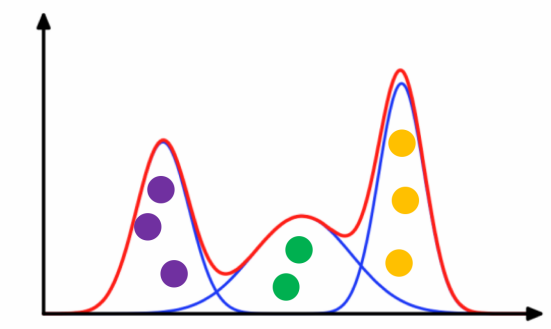
K-means. Pertama-tama kita inisiasi centroid secara acak, id_1 untuk k_1 dan id_6 untuk k_2. Kita hitung kedekatan data lainnya terhadap **centroid**. Untuk mempermudah contoh, kita hitung perbedaan data satu dan lainnya dengan menghitung perbedaan nilai atribut (nilai atributnya sama atau tidak). Apabila perbedaan suatu data terhadap kedua centroid bernilai sama, kita masukkan ke kelas dengan nomor urut lebih kecil.

Setelah langkah ini, kelompok satu beranggotakan $\{id_1, id_2, id_3, id_5\}$ sementara kelompok dua beranggotakan $\{id_4, id_6\}$. Kita pilih kembali centroid untuk masing-masing kelompok yang mana berasal dari anggota kelompok itu sendiri. Misal kita pilih secara acak,[1] centroid untuk kelompok pertama adalah id_2 sementara untuk kelompok kedua adalah id_4. Kita hitung kembali *assignment* anggota kelompok yang ilustrasinya dapat dilihat pada Tabel 4.7. Hasil langkah ke-2 adalah perubahan anggota kelompok, $k_1 = \{id_1, id_2, id_3, id_5\}$ dan $k_2 = \{id_4, id_6\}$. Anggap pada langkah ke-3 kita memilih kembali id_2 dan id_4 sebagai centroid masing-masing kelompok sehingga tidak ada perubahan keanggotaan.

Bila kamu membaca buku literatur lain, kemungkinan besar akan dijelaskan bahwa *clustering* itu memiliki **hubungan erat dengan *Gaussian Mixture Model***. Secara sederhana, **satu *cluster* (atau satu kelas)** sebenarnya seolah-olah dapat dipisahkan dengan kelas lainnya oleh distribusi gaussian. Perhatikan Gambar 4.1! Suatu *cluster* atau kelas, seolah olah "dibungkus" oleh suatu distribusi gaussian. Distribusi seluruh dataset dapat diaproksimasi dengan *Gaussian Mixture Model* (GMM).

Ingat kembali bahwa data memiliki suatu pola (dalam statistik disebut distribusi), kemudian pada bab 2 telah disebutkan bahwa GMM dipercaya dapat mengaproksimasi fungsi apapun (silahkan perdalam pengetahuan statistik

[1] Cara lain memilih akan dijelaskan pada bab 10.



Gambar 4.1: Ilustrasi hubungan Clustering, kelas, dan Gaussian

kamu untuk hal ini). Dengan demikian, *machine learning* yang mempunyai salah satu tujuan untuk menemukan pola dataset, memiliki hubungan yang sangat erat dengan distribusi gaussian karena pola tersebut dapat diaproksimasi dengan distribusi gaussian.

4.3 K-nearest-neighbor

Ide **K-nearest-neighbor** (KNN) adalah mengelompokkan data ke kelompok yang memiliki sifat termirip dengannya [28]. Hal ini sangat mirip dengan **K-means**. Bila K-means digunakan untuk *clustering*, KNN digunakan untuk klasifikasi. Algoritma klasifikasi ini disebut juga algoritma malas karena tidak mempelajari cara mengkategorikan data, melainkan hanya mengingat data yang sudah ada.[2] Pada subbab 4.2, kita telah mengelompokkan data orang kaya menjadi dua kelompok.

KNN mencari K *feature vector* dengan sifat termirip, kemudian mengelompokkan data baru ke kelompok *feature vector* tersebut. Sebagai ilustrasi mudah, kita lakukan klasifikasi algoritma KNN dengan $K = 3$ untuk data baru $\{rich = no, intelligent = yes, good\ looking = yes\}$. Kita tahu pada subbab sebelumnya bahwa kelompok satu $k_1 = \{id_1, id_2, id_3, id_5\}$ dan $k_2 = \{id_4, id_6\}$, pada Tabel 4.8. *feature vector* termirip dimiliki oleh data dengan id_1, id_5, id_6 seperti diilustrasikan pada Tabel 4.8. Kita dapat menggunakan strategi untuk mengurusi permasalahan ini, misalnya memberikan prioritas memasukkan ke kelompok yang rata-rata perbedaannya lebih kecil.[3] Dengan strategi tersebut, kita mengklasifikasikan data baru ke kelompok pertama.

[2] https://sebastianraschka.com/faq/docs/lazy-knn.html
[3] Silahkan eksplorasi cara lain juga!

id	perbedaan
1	1
2	3
3	3
4	2
5	1
6	1

Tabel 4.8: Perbedaan data baru vs data orang kaya

Soal Latihan

4.1. Data numerik

(a) Carilah suatu contoh dataset numerik.
(b) Pikirkanlah strategi untuk mengklasifikasi data numerik pada algoritma Naive Bayes dan *K-nearest-neighbor*!

4.2. K-means, KNN, GMM, EM

Buktikan bahwa K-means, K-nearest-neighbor, *Gaussian Mixture Model*, dan Expectation Maximization Algorithm memiliki hubungan! (apa kesamaan mereka).

4.3. K-means

(a) Cari tahu cara lain untuk memilih **centroid** pada algoritma K-means (selain cara acak) baik untuk data nominal dan numerik!
(b) Cari tahu cara lain untuk menghitung kedekekatan suatu data dengan **centroid** baik untuk data nominal dan numerik! Hint: *euclidian distance*, *manhattan distance*, *cosine similarity*.

5

Model Linear

> "Sometimes an entirely inaccurate formula is a handy way to move you in the right direction if it offers simplicity."
>
> Scott Adams

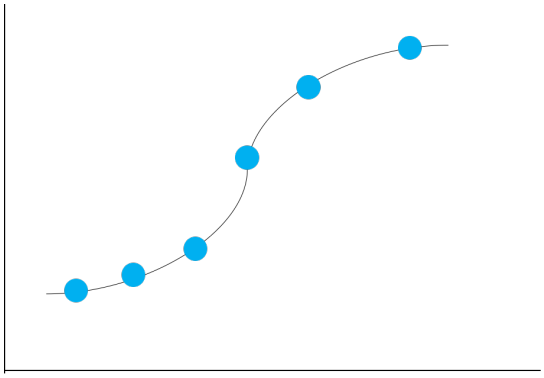
Bab ini membahas tentang model linear (algoritma parametrik), sebagai contoh pembelajaran yang sederhana karena cukup mudah dipahami idenya. Bab ini juga membahas *error function* dan *stochastic gradient descent*. Dimulai dari bab ini, kita akan memasuki materi lebih serius. Materi pada bab ini dijelaskan dengan sangat baik dan mendalam pada buku *An Introduction to Statistical Learning* [20].

5.1 Curve Fitting dan Error Function

Pertama, penulis ingin menceritakan salah satu bentuk *utility function* untuk model matematis bernama *error function*. Fungsi ini sudah banyak diceritakan pada bab-bab sebelumnya secara deskriptif. Mulai bab ini, kamu akan mendapatkan pengertian lebih jelas secara matematis.

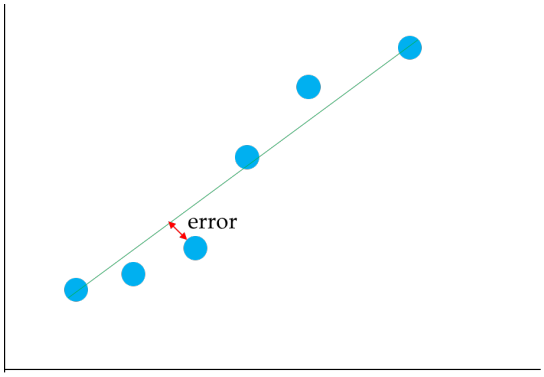
Error function paling mudah dijelaskan dalam permasalahan regresi. Diberikan $(x, y) \in \mathbb{R}$ sebagai *random variable*. Terdapat sebuah fungsi $f(x) \rightarrow y$, yang memetakan x ke y, berbentuk seperti pada Gambar 5.1. sekarang fungsi $f(x)$ tersebut disembunyikan (tidak diketahui), diberikan contoh-contoh pasangan (x_i, y_i), $i = 1, 2, ..., 6$; yang direpresentasikan dengan lingkaran biru pada ruang dua dimensi (titik sampel). Tugasmu adalah untuk mencari tahu $f(x)$! Dengan kata lain, kita harus mampu memprediksi sebuah bilangan riil y, diberikan suatu x.

Kamu berasumsi bahwa fungsi $f(x)$ dapat diaproksimasi dengan fungsi linear $g(x) = xw + b$. Artinya, kamu ingin mencari w dan b yang memberikan



Gambar 5.1: Contoh fungsi Sigmoid. Titik biru merepresentasikan sampel.

nilai sedemikian sehingga $g(x)$ mirip dengan $f(x)$. w adalah parameter sementara b adalah *bias*. Anggap kamu sudah berhasil melakukan pendekatan dan menghasilkan fungsi linear $g(x)$; seperti Gambar 5.2 (garis berwarna hijau). Akan tetapi, fungsi approksimasi ini tidak 100% tepat sesuai dengan fungsi aslinya (ini perlu ditekankan).[1] Jarak antara titik biru terhadap garis hijau disebut *error*.



Gambar 5.2: Pendekatan fungsi Sigmoid.

Salah satu cara menghitung *error* fungsi $g(x)$ adalah menggunakan ***squared error function*** dengan bentuk konseptual pada persamaan 5.1. Estimasi terhadap persamaan tersebut disajikan dalam bentuk diskrit pada persamaan 5.2, dimana (x_i, y_i) adalah pasangan *training data* (*input* dan *desired output*). Nilai *squared error* dapat menjadi tolak ukur untuk membandingkan kinerja

[1] Kamu patut curiga apabila model pembelajaran mesinmu memberikan kinerja 100%

suatu *learning machine* (model). Secara umum, bila nilainya tinggi, maka kinerja dianggap relatif buruk; sebaliknya bila rendah, kinerja dianggap relatif baik. Hal ini sesuai dengan konsep *intelligent agent* [5].

$$E(g) = \int \int \|y - g(x)\|^2 \, q(x, y) \, dx \, dy \tag{5.1}$$

$$E(g) = \frac{1}{N} \sum_{i=1}^{N} \|y_i - g(x_i)\|^2 \tag{5.2}$$

Secara konseptual, bentuk fungsi regresi dilambangkan sebagai persamaan 5.3 [9].

$$g(x) = \int y \; q(y \mid x) dy \tag{5.3}$$

Persamaan 5.3 dibaca sebagai "*expectation of* y, *with the distribution of* q." Secara statistik, regresi dapat disebut sebagai ekspektasi untuk y berdasarkan/dengan *input* x. Perlu diperhatikan kembali, regresi adalah pendekatan sehingga belum tentu 100% benar (hal ini juga berlaku pada model *machine learning* pada umumnya).

Kami telah memberikan contoh fungsi linear sederhana, yaitu $g(x) = xw + b$. Pada kenyataannya, permasalahan kita lebih dari persoalan skalar. Untuk $\mathbf{x}$ (input) yang merupakan vektor, biasanya kita mengestimasi dengan lebih banyak variable, seperti pada persamaan 5.4. Persamaan tersebut dapat ditulis kembali dalam bentuk aljabar linear sebagai persamaan 5.5.

$$g(\mathbf{x}) = x_1 w_1 + x_2 w_2 + \cdots + x_N w_N + b \tag{5.4}$$

$$g(\mathbf{x}) = \mathbf{x} \cdot \mathbf{w} + b \tag{5.5}$$

Bentuk persamaan 5.4 dan 5.5 relatif *interpretable* karena setiap fitur pada *input* (x_i) berkorespondensi hanya dengan satu parameter bobot w_i. Artinya, kita bisa menginterpretasikan seberapa besar/kecil pengaruh suatu fitur x_i terhadap keputusan (*output*) berdasarkan nilai w_i. Hal ini berbeda dengan algoritma non-linear (misal *artificial neural network*, bab 11) dimana satu fitur pada *input* bisa berkorespondensi dengan banyak parameter bobot. Perlu kamu ingat, model yang dihasilkan oleh fungsi linear lebih mudah dimengerti dibanding fungsi non-linear. Semakin suatu model pembelajaran mesin berbentuk non-linear, maka ia semakin susah dipahami.

Ingat kembali bab 1, *learning machine* yang direpresentasikan dengan fungsi g bisa diatur kinerjanya dengan parameter *training* $\mathbf{w}$. *Squared error* untuk *learning machine* dengan parameter *training* $\mathbf{w}$ diberikan oleh persamaan 5.2, dimana (x_i, y_i) adalah pasangan *input-desired output*. Selain untuk menghitung ***squared error*** pada *training data*, persamaan 5.2 juga dapat digunakan untuk menghitung *squared error* pada *test data*. Tujuan dari regresi/*machine learning* secara umum adalah untuk meminimalkan nilai *loss*

baik pada *training* maupun *unseen instances*. Misal, menggunakan *error function* sebagai *proxy* untuk *loss*. Selain *error function*, ada banyak fungsi lainnya seperti *Hinge*, *Log Loss*, *Cross-entropy loss*, *Ranking loss* [1].

5.2 Binary Classification

Binary classification adalah mengklasifikasikan data menjadi dua kelas (*binary*). Contoh model linear sederhana untuk *binary classification* diberikan pada persamaan 5.6. Perhatikan, pada persamaan 5.6, suatu data direpresentasikan sebagai *feature vector* $\mathbf{x}$, dan terdapat *bias* b.[2] Klasifikasi dilakukan dengan melewatkan data pada fungsi yang memiliki parameter. Fungsi tersebut menghitung bobot setiap fitur pada vektor dengan mengalikannya dengan parameter (*dot product*). Persamaan 5.6 dapat ditulis kembali sebagai persamaan 5.7, dimana x_i merupakan elemen ke-i dari vektor $\mathbf{x}$. Fungsi ini memiliki *range* $[-\infty, \infty]$. Pada saat ini, kamu mungkin bingung. Bagaimana mungkin fungsi regresi yang menghasilkan nilai kontinu digunakan untuk klasifikasi kelas kategorial. Kita dapat menggunakan *thresholding*, atau dengan memberikan batas nilai tertentu. Misal, bila $f(x) > threshold$ maka dimasukkan ke kelas pertama; dan sebaliknya $f(x) \leq threshold$ dimasukkan ke kelas kedua. *Threshold* menjadi bidang pembatas antara kelas satu dan kelas kedua (*decision boundary*, Gambar 5.3). Pada umumnya, teknik *threshold* diterapkan dengan menggunakan fungsi *sign* (sgn, Gambar 5.4) untuk merubah nilai fungsi menjadi $[-1, 1]$ sebagai *output* (persamaan 5.8); dimana -1 merepresentasikan *input* dikategorikan ke kelas pertama dan nilai 1 merepresentasikan *input* dikategorikan ke kelas kedua.

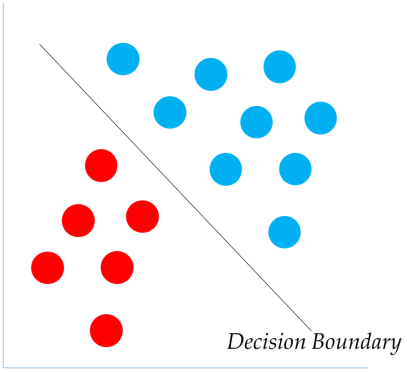
$$f(\mathbf{x}) = \mathbf{x} \cdot \mathbf{w} + b \tag{5.6}$$

$$f(\mathbf{x}) = x_1 w_1 + x_1 w_2 + \cdots + x_N w_N + b \tag{5.7}$$

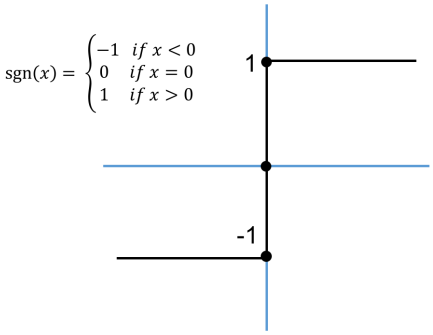
$$\text{output} = \text{sgn}(f(\mathbf{x}) \tag{5.8}$$

Seperti halnya fungsi regresi, kita juga dapat menghitung performa *binary classifier* sederhana ini menggunakan *squared error function* (umumnya menggunakan akurasi), dimana nilai target fungsi berada pada range $[-1, 1]$. Secara sederhana, model *binary classifier* mencari ***decision boundary***, yaitu garis (secara lebih umum, *hyperplane*) pemisah antara kelas satu dan lainnya. Sebagai contoh, garis hitam pada Gambar 5.3 adalah *decision boundary*.

[2] Perhatikan! *Bias* pada variabel fungsi memiliki arti yang berbeda dengan *statistical bias*



Gambar 5.3: Contoh *decision boundary*.



Gambar 5.4: Fungsi *sign*.

5.3 Log-linear Binary Classification

Pada subbab sebelumnya, telah dijelaskan fungsi binary classifier memetakan data menjadi nilai $[-1, 1]$, dengan -1 merepresentasikan kelas pertama dan 1 merepresentasikan kelas kedua. Tidak hanya kelas yang berkorespondensi, kita juga terkadang ingin tahu seberapa besar peluang data tergolong pada kelas tersebut. Salah satu alternatif adalah dengan menggunakan fungsi sigmoid dibanding fungsi *sign* untuk merubah nilai fungsi menjadi $[0, 1]$ yang merepresentasikan peluang p data diklasifikasikan sebagai kelas tertentu $(1-p$ untuk kelas lainnya). Konsep ini dituangkan menjadi persamaan 5.9, dimana y merepresentasikan probabilitas *input* $\mathbf{x}$ digolongkan ke kelas tertentu, $\mathbf{x}$ merepresentasikan data (*feature vector*), dan b merepresentasikan *bias*. Ingat kembali materi bab 4, algoritma *Naive Bayes* melakukan hal serupa. Hasil fungsi sigmoid, apabila di-*plot* maka akan berbentuk seperti Gambar 5.1 (berbentuk karakter "S").

$$y = \sigma(f(\mathbf{x})) = \frac{1}{1 + e^{-(\mathbf{x}\cdot\mathbf{w}+b)}} \tag{5.9}$$

Perhatikan, persamaan 5.9 juga dapat diganti dengan persamaan 5.10 yang dikenal juga sebagai fungsi logistik.

$$y = \text{logistik}\,(f(\mathbf{x})) = \frac{e^{(\mathbf{x}\cdot\mathbf{w}+b)}}{1 + e^{(\mathbf{x}\cdot\mathbf{w}+b)}} \tag{5.10}$$

Varibel y adalah nilai probabilitas data masuk ke suatu kelas. Sebagai contoh, kita ingin nilai $y = 1$ apabila data cocok masuk ke kelas pertama dan $y = 0$ apabila masuk ke kelas kedua.

Ketika fungsi *machine learning* menghasilkan nilai berbentuk probabilitas, kita dapat menggunakan *cross entropy* sebagai *utility function*. Persamaan 5.11 adalah cara menghitung *cross entropy*, dimana $T(c_i)$ melambangkan *desired probability* (*desired output*). $P(c_i)$ melambangkan probabilitas (prediksi) *input* diklasifikasikan ke kelas c_i dan N melambangkan banyaknya kelas. Untuk *binary classification*, $T(c_1) = 1 - T(c_2)$. Pada umumnya, T bernilai antara [0, 1]. Nilai 1 untuk kelas yang berkorespondensi dengan *input* (*desired output*) dan 0 untuk kelas lainnya.

$$H = -\sum_{i=1}^{N} T(c_i) \log\,(P(c_i)) \tag{5.11}$$

Kita ingin meminimalkan nilai *cross entropy* untuk model pembelajaran mesin yang baik. Ingat kembali materi teori informasi, nilai *entropy* yang rendah melambangkan distribusi tidak *uniform*. Sementara, nilai *entropy* yang tinggi melambangkan distribusi lebih *uniform*. Artinya, nilai *cross entropy* yang rendah melambangkan *high confidence* saat melakukan klasifikasi. Kita ingin model kita sebiasa mungkin menghasilkan *output* bernilai 1 untuk mendiskriminasi seluruh data yang masuk ke kelas pertama, dan 0 untuk kelas lainnya. Dengan kata lain, model dapat mendiskriminasi data dengan pasti. *Cross entropy* bernilai tinggi apabila perbedaan nilai probabilitas masuk ke kelas satu dan kelas lainnya tidak jauh, e.g., $P(c_1) = 0.6$ & $P(c_2) = 0.4$. Semakin rendah nilai *cross entropy*, kita bisa meningkatkan "keyakinan" kita terhadap kemampuan klasifikasi model pembelajaran mesin, yaitu perbedaan nilai probabilitas masuk ke kelas satu dan kelas lainnya tinggi, e.g., $P(c_1) = 0.8$ & $P(c_2) = 0.2$.

5.4 Multi-class Classification

Subbab ini akan membahas tentang *multi-class classification*, dimana terdapat lebih dari dua kemungkinan kelas. Terdapat himpunan kelas C beranggotakan $\{c_1, c_2, \ldots, c_K\}$. Untuk suatu data dengan representasi *feature vector*-nya, kita ingin mencari tahu kelas yang berkorespondesi untuk data tersebut. Contoh

permasalahan ini adalah mengklasifikasi gambar untuk tiga kelas: *apel*, *jeruk*, atau *mangga*. Cara sederhana adalah memiliki tiga buah vektor parameter dan *bias* berbeda, $\mathbf{w}_{apel}$, $\mathbf{w}_{jeruk}$, $\mathbf{w}_{mangga}$, dan *bias* $b_{\{apel,jeruk,mangga\}}$. Untuk menentukan suatu data masuk ke kelas mana, kita dapat memprediksi skor tertinggi yang diberikan oleh operasi *feature vector* terhadap masing-masing vektor parameter. Konsep matematisnya diberikan pada persamaan 5.12, dimana $\hat{c}$ adalah kelas terpilih (keputusan), yaitu kelas yang memiliki nilai tertinggi. C melambangkan himpunan kelas.

$$\begin{aligned} c_{apel} &= \mathbf{x} \cdot \mathbf{w}_{apel} + b_{apel} \\ c_{jeruk} &= \mathbf{x} \cdot \mathbf{w}_{jeruk} + b_{jeruk} \\ c_{mangga} &= \mathbf{x} \cdot \mathbf{w}_{mangga} + b_{mangga} \\ \hat{c} &= \arg\max_{c_i \in C}(c_i) \end{aligned} \tag{5.12}$$

Tiga set parameter $\mathbf{w}_{c_i}$ dapat disusun sedemikian rupa sebagai matriks $\mathbf{W} \in \mathbb{R}^{d \times 3}$, dimana d adalah dimensi *feature vector* ($\mathbf{x} \in \mathbb{R}^{1 \times d}$). Demikian pula kita dapat susun bias menjadi vektor $\mathbf{b} \in \mathbb{R}^{1 \times 3}$ berdimensi tiga. Dengan demikian, persamaan 5.12 dapat ditulis kembali sebagai persamaan 5.13, dimana $\mathbf{c}$ adalah vektor yang memuat nilai fungsi terhadap seluruh kelas. Kita memprediksi kelas berdasarkan indeks elemen $\mathbf{c}$ yang memiliki nilai terbesar (persamaan 5.14). Analogika, seperti memilih kelas dengan nilai *likelihood* tertinggi.

$$\mathbf{c} = f(\mathbf{x}) = \mathbf{x} \cdot \mathbf{W} + \mathbf{b} \tag{5.13}$$

$$\hat{c} = \arg\max_{c_i \in \mathbf{c}}(c_i) \tag{5.14}$$

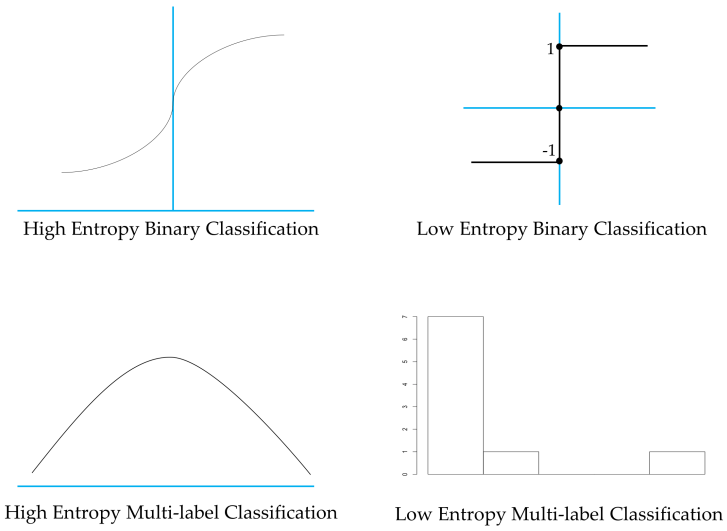
Seperti yang diceritakan pada subbab berikutnya, kita mungkin juga tertarik dengan probabilitas masing-masing kelas, bukan hanya *likelihood*-nya. Kita dapat menggunakan fungsi *softmax*[3] untuk hal ini. Fungsi *softmax* mentransformasi $\mathbf{c}$ agar jumlah semua nilainya berada pada range $[0, 1]$. Dengan itu, $\mathbf{c}$ dapat diinterpretasikan sebagai distribusi probabilitas. Konsep ini dituangkan pada persamaan 5.15, dimana c_i adalah elemen vektor ke-i, melambangkan probabilitas masuk ke kelas ke-i.

$$c_i = \frac{e^{(\mathbf{x} \cdot \mathbf{W} + \mathbf{b})_{[i]}}}{\sum_j e^{(\mathbf{x} \cdot \mathbf{W} + \mathbf{b})_{[j]}}} \tag{5.15}$$

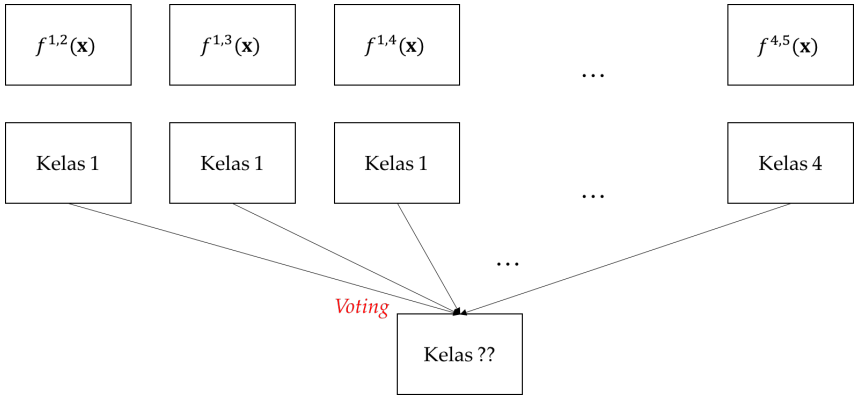
Karena *output* fungsi berupa distribusi probabilitas, kita juga dapat menghitung *loss* menggunakan *cross entropy*. Seperti yang sudah dijelaskan sebelumnya pada *binary classification*, kita ingin hasil perhitungan *cross entropy* seke-

[3] https://en.wikipedia.org/wiki/Softmax_function

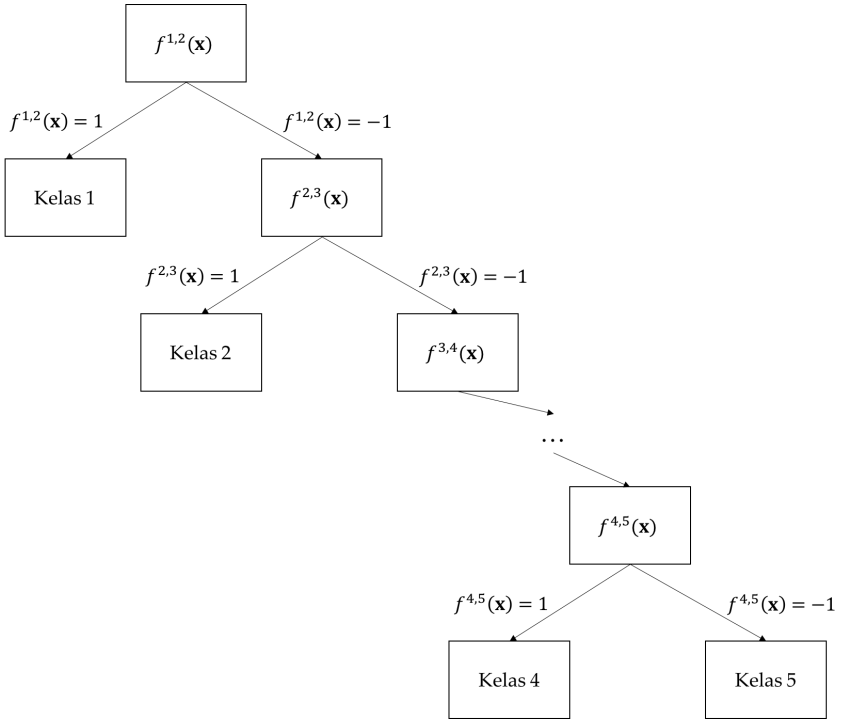
cil mungkin karena meminimalkan nilai *cross entropy* meningkatkan *confidence* saat proses klasifikasi. Hal ini dapat dianalogikan dengan jargon "*winner takes it all*". Sebagai ilustrasi, lihatlah Gambar 5.5. Kita ingin agar bentuk distribusi probabilitas *output* kelas ($\mathbf{c}$) condong ke salah satu sisi saja.



Gambar 5.5: *Classification Entropy.*



Gambar 5.6: *One versus one.*



Gambar 5.7: *One versus all.*

Selain mengekstensi suatu model untuk melakukan *multi-class classification* secara langsung, kita juga dapat menggabungkan beberapa model *binary classifier* untuk melakukan *multi-class classification*. Teknik ini memiliki dua varian yaitu *one versus one* dan *one versus all*.

Pada teknik *one versus one*, kita membuat sejumlah kombinasi pasangan kelas $\binom{N}{2}$, untuk N = banyaknya kelas. Kemudian, kita biarkan masing-masing model mengklasifikasikan *input* ke dalam satu dari dua kelas. Akhirnya, kita memilih kelas klasifikasi berdasarkan kelas yang paling sering muncul dari semua model. Hal ini diilustrasikan pada Gambar 5.6 (untuk lima kelas), dimana $f^{i,j}$ melambangkan *binary classifier* untuk kelas i dan kelas j.

Pada teknik *one versus all*, kita membuat sejumlah model juga, tetapi kombinasinya tidak sebanyak *one versus one*. Model pertama mengklasifikasikan *input* sebagai kelas pertama atau bukan kelas pertama. Setelah itu, dilanjutkan ke model kedua mengklasifikasikan *input* sebagai kelas kedua atau bukan kelas kedua, dan seterusnya. Hal ini diilustrasikan pada Gambar 5.7 (untuk lima kelas).

5.5 Multi-label Classification

Seperti halnya *multi-class classification*, kita dapat mendekomposisi *multi-label classification* menjadi beberapa *binary classifier* (analogi persamaan 5.12). Yang membedakan *multi-class* dan *multi-label* adalah output $\mathbf{c}$. Pada *multi-class classification*, $c_i \in \mathbf{c}$ melambangkan skor suatu *input* masuk ke kelas c_i. Keputusan akhir *class assignment* didapatkan dari elemen $\mathbf{c}$ dengan nilai terbesar. Untuk *multi-label classification* nilai $c_i \in \mathbf{c}$ melambangkan apakah suatu kelas masuk ke kelas c_i atau tidak. Bedanya, kita boleh meng-*assign* lebih dari satu kelas (atau tidak sama sekali). Misal $c_i \geq 0.5$, artinya kita anggap model tersebut layak masuk ke kelas c_i, tanpa harus membandingannya dengan nilai $c_j (i \neq j)$ lain. Inilah yang dimaksud dengan prinsip *mutual exclusivity*. Perhatikan Tabel 5.1 sebagai ilustrasi, dimana "1" melambangkan *class assignment*.

Instance	**Agama**	**Politik**	**Hiburan**
Berita-1	1	0	0
Berita-2	0	1	0
Berita-3	0	0	1

(a) Multi-class classification.

Instance	**Agama**	**Politik**	**Hiburan**
Berita-1	1	1	0
Berita-2	0	1	1
Berita-3	1	0	1

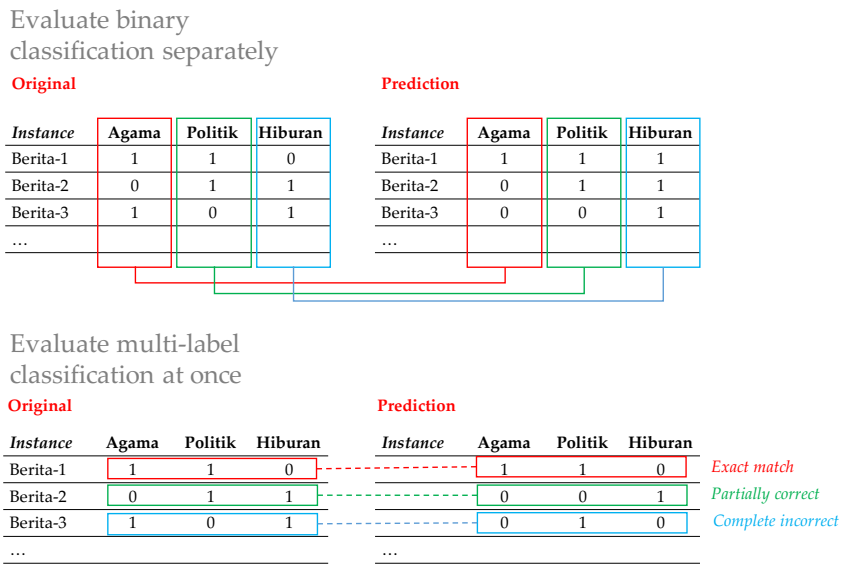
(b) Multi-label classification.

Tabel 5.1: Ilustrasi *multi-label* dan *multi-class classification*. Nilai "1" melambangkan TRUE dan "0" melambangkan FALSE (*class assignment*).

Sekali lagi, nilai $c_i \in \mathbf{c}$ bernilai $[0, 1]$ tetapi keputusan klasifikasi apakah suatu kelas masuk ke dalam c_i tidak bergantung pada nilai $c_j (i \neq j)$ lainnya. Berdasarkan prinsip *mutual exclusivity*, output $\mathbf{c}$ pada *classifier* 5.7 tidak ditransformasi menggunakan *softmax*. Pada umumnya, *multi-label classifier* melewatkan *output* $\mathbf{c}$ ke dalam fungsi sigmoid. Dengan demikian, persamaan 5.15 diganti menjadi persamaan 5.16 pada *multi-label classifier*.

$$c_i = \text{sigmoid}(c_i) \tag{5.16}$$

Berhubung cara menginterpretasikan *output* berbeda dengan *multi-class classification*, cara evaluasi kinerja juga berbeda. Ada dua pendekatan evaluasi pada *multi-label classification*. Pertama, kita evaluasi kinerja *binary classification* untuk setiap kelas. Pada pendekatan ini, seolah-olah kita memiliki banyak *binary classifier* untuk melakukan klasifikasi. Kedua, kita dapat mengevaluasi kinerja *multi-label classification* itu sendiri. Perhatikan Gambar 5.8! Pendekatan pertama ibarat membandingkan kolom *desired output* untuk masing-masing kelas dengan *prediction* yang berkorespondensi. Pada pendekatan kedua, kita membandingkan baris untuk tiap-tiap prediksi (seperti biasa). Saat kita membandingkan tiap-tiap baris prediksi, kita bisa jadi menda-



Gambar 5.8: Cara mengevaluasi *multi-label classifier*.

patkan prediksi tipe ***exact match*** (seluruh prediksi sama dengan *desired output*), ***partially correct*** (sebagian prediksi sama dengan *desired output*) atau ***complete incorrect*** (tidak ada prediksi yang sama dengan *desired output*). Dengan ini, evaluasi *multi-label classification* relatif lebih kompleks dibanding *multi-class classification* biasa. Untuk mendapatkan *multi-label classification accuracy*, kita dapat menghitung berapa seberapa banyak *exact match* dibandingkan banyaknya *instance* (walaupun hal ini terlalu ketat). Selain itu, kita dapat menghitung *loss* menggunakan *cross entropy* untuk mengevaluasi kinerja saat melatih *multi-label classifier*, layaknya *multi-class classifier*. Kamu dapat membaca [29] lebih lanjut untuk teknik evaluasi *multi-label classification*.

5.6 Pembelajaran sebagai Permasalahan Optimisasi

Salah satu tujuan dari pembelajaran (*training*) adalah untuk meminimalkan *error* sehingga kinerja *learning machine* (model) diukur oleh *squared error*. Dengan kata lain, *utility function* adalah meminimalkan *squared error*. Secara lebih umum, kita ingin meminimalkan/memaksimalkan suatu fungsi yang dijadikan tolak ukur kinerja (*utility function*), diilustrasikan pada per-

samaan 5.17, dimana θ adalah *learning parameter*,[4] $\hat{\theta}$ adalah nilai parameter paling optimal, dan $\mathcal{L}$ adalah *loss function*. Perhatikan, "arg min" dapat juga diganti dengan "arg max" tergantung **optimisasi** apa yang ingin dilakukan. Perubahan parameter dapat menyebabkan perubahan *loss*. Karena itu, *loss function* memiliki θ sebagai parameternya.

$$\hat{\theta} = \arg\min_{\theta} \mathcal{L}(\theta) \tag{5.17}$$

Sekarang, mari kita hubungkan dengan contoh yang sudah diberikan pada subbab sebelumnya. Kita coba melakukan estimasi *minimum squared error* E, dengan mencari nilai *learning parameters* $\mathbf{w}$ yang meminimalkan nilai *error* pada model linear (persamaan 5.18).[5] Parameter model pembelajaran mesin biasanya diinisialisasi secara acak atau menggunakan distribusi tertentu. Terdapat beberapa cara untuk memimalkan *squared error*. Yang penulis akan bahas adalah *stochastic gradient descent method*.[6] Selanjutnya apda buku ini, istilah *gradient descent*, *gradient-based method* dan *stochastic gradient descent* mengacu pada hal yang sama.

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} E(\mathbf{w}) \tag{5.18}$$

Bayangkan kamu sedang berada di puncak pegunungan. Kamu ingin mencari titik terendah pegunungan tersebut. Kamu tidak dapat melihat keseluruhan pegunungan, jadi yang kamu lakukan adalah mencari titik terendah (lokal) sejauh mata memandang, kemudian menuju titik tersebut dan menganggapnya sebagai titik terendah (global). Layaknya asumsi sebelumnya, kamu juga turun menuju titik terendah dengan cara melalui jalanan dengan kemiringan paling tajam, dengan anggapan bisa lebih cepat menuju ke titik terendah [9]. Sebagai ilustrasi, perhatikan Gambar 5.9! Seluruh area pegunungan adalah nilai *error* E yang mungkin (pada persoalanmu), dan titik terendah pada daerah tersebut adalah nilai *error* E terendah.

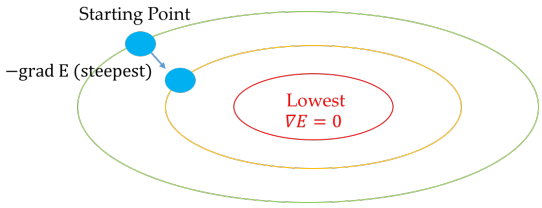
Jalanan dengan kemiringan paling tajam adalah $-\text{grad}\ E(\mathbf{w})$, dimana $E(\mathbf{w})$ adalah nilai *error* saat model memiliki parameter $\mathbf{w}$. Dengan definisi grad $E(\mathbf{w})$ diberikan pada persamaan 5.19 dan persamaan 5.20, dimana w_i adalah nilai elemen vektor ke-i.

$$\text{grad}\ E(\mathbf{w}) = \left(\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, ..., \frac{\partial E}{\partial w_F} \right) \tag{5.19}$$

[4] Dapat berupa skalar, vektor, matriks, atau tensor. Demi istilah yang lebih generik, kita gunakan θ.

[5] $\mathbf{w}$ boleh diganti dengan $\mathbf{W}$, saat ini penulis menggunakan vektor untuk menyederhanakan pembahasan.

[6] Konsep *Hill Climbing* dapat digunakan untuk memaksimalkan *utility function* Konsep tersebut sangat mirip dengan *gradient descent* https://en.wikipedia.org/wiki/Hill_climbing.



Gambar 5.9: *Stochastic Gradient Descent.*

$$\frac{\mathrm{d}\mathbf{w}}{\mathrm{d}t} = -\text{grad } E(\mathbf{w}); t = time \tag{5.20}$$

Ingat kembali materi diferensial. Gradien adalah turunan (diferensial) fungsi. Untuk mencari turunan paling terjal, sama halnya mencari nilai −gradient terbesar. Dengan demikian, menghitung −grad $E(\mathbf{w})$ terbesar sama dengan jalanan turun paling terjal. Tentunya seiring berjalannya waktu, kita mengubah-ubah parameter $\mathbf{w}$ agar kinerja model optimal. Nilai optimal diberikan oleh turunan $\mathbf{w}$ terhadap waktu, yang bernilai sama dengan −grad $E(\mathbf{w})$. Bentuk diskrit persamaan 5.20 diberikan pada persamaan 5.21,

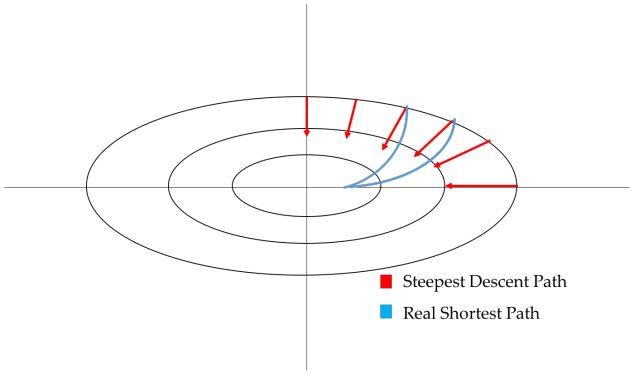
$$\mathbf{w}(t+1) = \mathbf{w}(t) - \eta \text{ grad } E(\mathbf{w}(t)) \tag{5.21}$$

dimana η disebut ***learning rate*** dan $\mathbf{w}(t)$ adalah nilai $\mathbf{w}$ saat waktu/iterasi t. *Learning rate* digunakan untuk mengatur seberapa pengaruh keterjalan terhadap pembelajaran. Silahkan mencari sumber tambahan lagi agar dapat mengerti *learning rate* secara lebih dalam/matematis. Pada implementasi, η juga sering diubah-ubah nilainya sepanjang waktu. Semakin kita sudah dekat dengan tujuan (titik *loss* terendah), kita mengurangi nilai η, ibaratnya seperti mengerem kalau sudah dekat dengan tujuan [30].

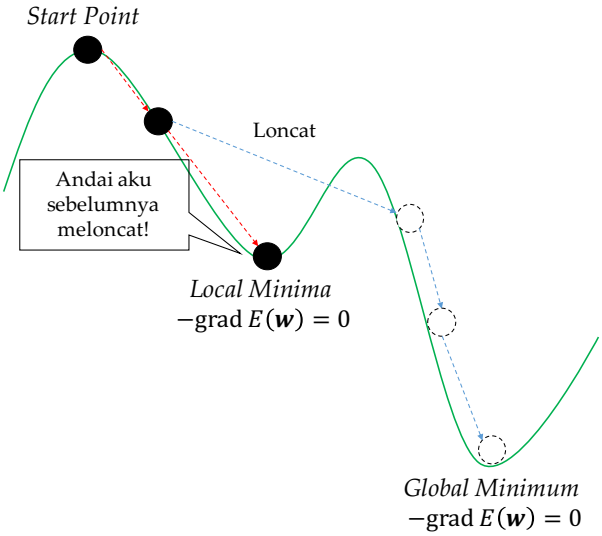
Walaupun kamu berharap bisa menuju titik terendah dengan menelusuri jalan terdekat dengan kemiringan paling tajam, tapi kenyataanya hal tersebut bisa jadi bukanlah jalan tercepat, seperti yang diilustrasikan pada Gambar 5.10. Warna merah melambangkan jalan yang dilalui *gradient descent*, sementara warna biru melambangkan jalanan terbaik (tercepat).

Pandangan kita yang terbatas layaknya kita tidak bisa melihat keseluruhan pengunungan secara keseluruhan (anggap ada kabut), kita juga tidak bisa melihat keseluruhan nilai *error* untuk semua parameter $\mathbf{w}$. Secara filosofis, hal tersebut juga berlaku saat membaca buku, oleh karena itu sebaiknya kamu membaca beberapa buku saat belajar.

Dalam *local point of view*, *steepest gradient descent* adalah cara tercepat menuju titik terendah, tetapi tidak pada *global point of view*. Kita dapat macet/berhenti saat sudah mencapai *local minima*, yaitu nilai minimum pada suatu daerah lokal saja. Sebagai ilustrasi, perhatikan Gambar 5.11. Kamu berada di puncak pegunungan kemudian turun bertahap. Kemudian, kamu sampai di suatu daerah landai (−grad $E(\mathbf{w}) = 0$). Kamu pikir daerah landai



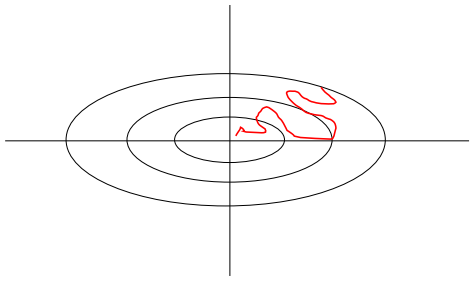
Gambar 5.10: *Stochastic Gradient Descent* 2.



Gambar 5.11: *Stuck at local minima.*

tersebut adalah titik terendah, tetapi, kamu ternyata salah. Untuk menghindari hal tersebut, kita menggunakan *learning rate* (η). Apabila nilai *learning rate* (η) pada persamaan 5.21 relatif kecil, maka dinamika perubahan parameter $\mathbf{w}$ juga kecil. Tetapi, bila nilainya besar, maka jalanan menuju titik terendah akan bergoyang-goyang (*swing*), seperti pada Gambar 5.12. Goyangan tersebut ibarat "meloncat-loncat" pada ilustrasi Gambar 5.11. Kemampuan untuk "meloncat" ini dapat menghindarkan model *stuck* di *local minima*.

Untuk mengontrol *learning parameter* $\mathbf{w}$ sehingga memberikan nilai $E(\mathbf{w})$ terendah, persamaan *steepest gradient descent* dapat ditambahkan dengan



Gambar 5.12: *Swing*.

momentum (α) pada persamaan 5.23. Alfa adalah momentum karena dikalikan dengan hasil perbedaan descent pada tahap sebelumnya. Alfa adalah parameter kontrol tambahan untuk mengendalikan *swing* yang sudah dibahas sebelumnya.

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \eta \text{ grad } E(\mathbf{w}(t)) + \alpha(\Delta\mathbf{w}) \tag{5.22}$$

$$\Delta\mathbf{w} = \mathbf{w}(t) - \mathbf{w}(t-1) \tag{5.23}$$

Apabila gradien bernilai 0, artinya model sudah berada pada titik *local/global optimum*. Kondisi ini disebut sebagai **konvergen** (*converging*). Sementara model yang tidak menuju titik optimal, malah menuju ke kondisi yang semakin tidak optimum, disebut **divergen** (*diverging*).

5.7 Batasan Model Linear

Model linear, walaupun mudah dimengerti, memiliki beberapa batasan. Ada dua batasan paling kentara [20]: (1) *additive assumption* dan (2) *linear assumption*.

Additive assumption berarti model linear menganggap hubungan antara *input* dan *output* adalah linear. Artinya, perubahan nilai pada suatu fitur x_i pada input $\mathbf{x}$ akan merubah nilai *output* secara independen terhadap fitur lainnya. Hal ini terkadang berakibat fatal karena fitur satu dan fitur lainnya dapat berinteraksi satu sama lain. Solusi sederhana untuk permasalahan ini adalah dengan memodelkan interaksi antar-fitur, seperti diilustrasikan pada persamaan 5.24 untuk *input* yang tersusun atas dua fitur.

$$f(\mathbf{x}) = x_1w_1 + x_1w_2 + x_1x_2w_3 + b \tag{5.24}$$

Dengan persamaan 5.24, apabila x_1 berubah, maka kontribusi x_2 terhadap *output* juga akan berubah (dan sebaliknya). Akan tetapi, seringkali interaksi

antar-fitur tidaklah sesederhana ini. Misal, semua fitur berinteraksi satu sama lain secara non-linear.

Linear assumption berarti perubahan pada suatu fitur x_i mengakibatkan perubahan yang konstan terhadap *output*, walaupun seberapa besar/kecil nilai x_i tersebut. Seringkali, perubahan pada *output* sesungguhnya bergantung juga pada nilai x_i itu sendiri, bukan hanya pada Δx_i. Solusi sederhana untuk permasalahan ini adalah memodelkan fungsi linear sebagai fungsi polinomial dengan orde (M) tertentu, diilustrasikan pada persamaan 5.25. Akan tetapi, pemodelan inipun tidaklah sempurna karena rawan *overfitting*.

$$f(\mathbf{x}) = x_1 w_1 + +x_2^2 w_2 + \cdots + x_M^M w_M + b \tag{5.25}$$

Asumsi yang sebelumnya dijelaskan pada pemodelan polinomial, dapat diekstensi menjadi *generalized additive model* (GAM) untuk mengatasi masalah *linear assumption*, seperti diilustrasikan pada persamaan 5.26 [20]. Artinya, kita melewatkan setiap fitur x_i pada suatu fungsi g_i, sehingga delta x_i tidak mengakibatkan perubahan yang konstan terhadap *output*. Ekstensi ini dapat memodelkan hubungan non-linear antara fitur dan *output*.

$$f(\mathbf{x}) = g_1(x_1) + +g_2(x_2) + \cdots + g_N(x_N) + b \tag{5.26}$$

Tetapi, GAM masih saja memiliki batasan *additive assumption*. Dengan demikian, interaksi antar-variabel tidak dapat dimodelkan dengan baik.

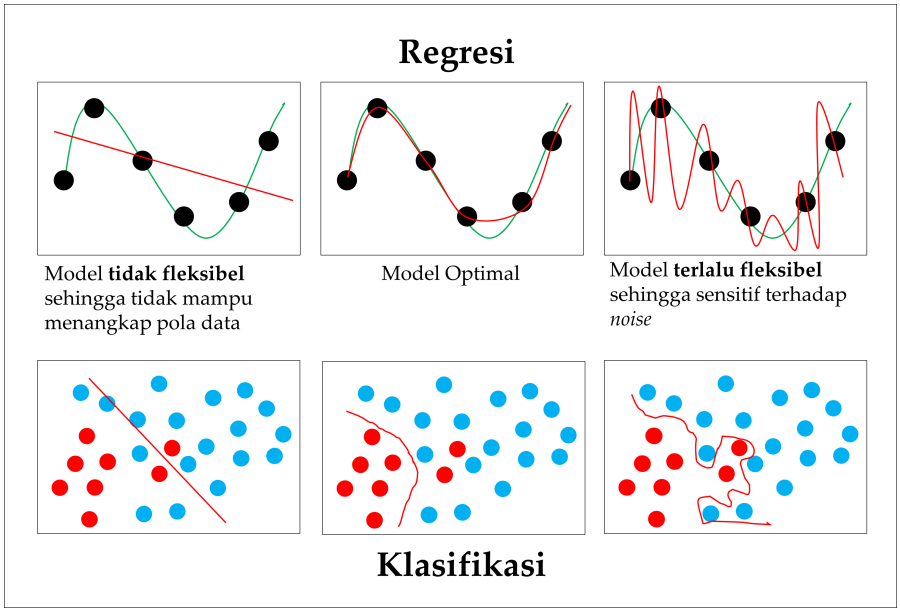
5.8 Overfitting dan Underfitting

Tujuan *machine learning* adalah membuat model yang mampu memprediksi data yang belum pernah dilihat (*unseen instances*) dengan tepat; disebut sebagai generalisasi (*generalization*). Seperti yang sudah dijelaskan pada bab pertama, kita dapat membagi dataset menjadi *training*, *validation*, dan *testing* dataset. Ketiga dataset ini berasal dari populasi yang sama dan dihasilkan oleh distribusi yang sama (*identically and independently distributed*). Dalam artian, ketiga jenis dataset mampu melambangkan (merepresentasikan) karakteristik yang sama.[7] Dengan demikian, kita ingin *loss* atau *error* pada *training*, *validation*, dan *testing* bernilai kurang lebih bernilai sama (i.e., kinerja yang sama untuk data dengan karakteristik yang sama). Akan tetapi, ***underfitting*** dan ***overfitting*** mungkin terjadi.

Underfitting adalah keadaan ketika kinerja model bernilai buruk baik pada *training* atau *validation* maupun *testing data*. *Overfitting* adalah keadaan ketika kinerja model bernilai baik untuk *training* tetapi buruk pada *unseen data*. Hal ini diilustrasikan pada Gambar 5.13. *Underfitting* terjadi akibat model yang terlalu tidak fleksibel, yaitu memiliki kemampuan yang rendah untuk mengestimasi variasi fungsi. Sedangkan, *overfitting* terjadi ketika

[7] Baca teknik *sampling* pada buku statistika.

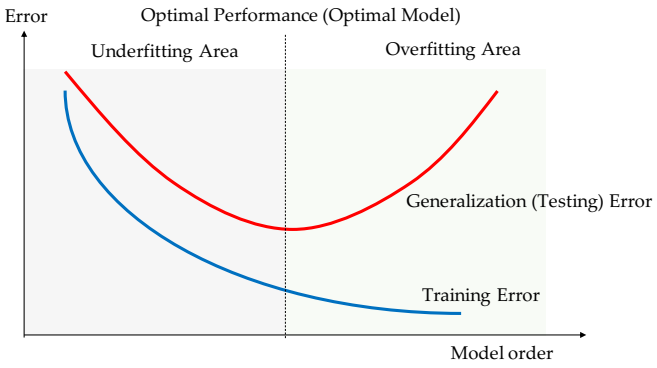
[8] Rekonstruksi https://www.inf.ed.ac.uk/teaching/courses/iaml/slides/eval-2x2.pdf



Gambar 5.13: *Underfitting* vs *Overfitting*.[8]

model terlalu fleksibel, yaitu memiliki kemampuan yang terlalu tinggi untuk mengestimasi banyak fungsi atau terlalu mencocokkan diri terhadap *training* data. Perhatikan kembali Gambar 5.13, dataset asli diambil (*sampled*) dari fungsi polinomial orde-3. Model *underfitting* hanya mampu mengestimasi dalam orde-1 (kemampuan terlalu rendah), sedangkan model *overfitting* mampu mengestimasi sampai orde-9 (kemampuan terlalu tinggi).

Apabila kita gambarkan grafik kinerja terhadap konfigurasi model (*model order*), fenomena *underfitting* dan *overfitting* dapat diilustrasikan seperti Gambar 5.14. Model yang ideal adalah model yang memiliki kinerja yang baik pada *training*, *validation*, dan *testing* data. Artinya, kita ingin perbedaan kinerja model pada berbagai dataset bernilai sekecil mungkin. Untuk menghindari *overfitting* atau *underfitting*, kita dapat menambahkan fungsi ***noise/bias*** (selanjutnya disebut *noise/bias* saja) dan regularisasi (subbab 5.9). Hal yang paling perlu pembaca pahami adalah untuk jangan merasa senang ketika model *machine learning* yang kamu buat memiliki kinerja baik pada *training data*. Kamu harus mengecek pada *validation* dan *testing* data, serta memastikan kesamaan karakteristik data, e.g., apakah *training* dan *testing* data benar diambil dari distribusi yang sama. Selain itu, kamu juga harus memastikan apakah data yang digunakan mampu merepresentasikan kompleksitas pada permasalahan asli/dunia nyata. Sering kali, dataset yang digunakan pada banyak eksperimen adalah *toy dataset*, semacam simplifikasi permasalahan dengan banyak sampel yang relatif sedikit. Kamu harus hati-hati terhadap



Gambar 5.14: *Selection Error*.

overclaiming, i.e., menjustifikasi model dengan performa baik pada *toy dataset* sebagai model yang baik secara umum.

5.9 Regularization

Gradient-based method mengubah-ubah parameter model $\mathbf{w}$ sehingga *loss* dapat diminimalkan. Perhatikan kembali Gambar 5.2, ibaratnya agar fungsi aproksimasi kita menjadi sama persis dengan fungsi asli pada Gambar 5.1. Perhatikan, karena nilai $\mathbf{w}$ berubah-ubah seiring waktu, bisa jadi urutan *training data* menjadi penting.[9] Pada umumnya, kita menggunakan *batch method* agar kinerja model tidak bias terhadap urutan data. Artinya, menghitung *loss* untuk beberapa data sekaligus. Hal ini akan kamu lebih mengerti setelah mambaca bab 11.

Selain permasalahan model yang sensitif terhadap urutan *training data*, model yang kita hasilkan bisa jadi *overfitting* juga. Yaitu memiliki kinerja baik pada *training data*, tetapi memiliki kinerja buruk untuk *unseen data*. Salah satu cara menghindari *overfitting* adalah dengan menggunakan *regularization*. Idenya adalah untuk mengontrol kompleksitas parameter (i.e. konfigurasi parameter yang lebih sederhana lebih baik). Dengan ini, objektif *training* pada persamaan 5.17 dapat kita ubah menjadi persamaan 5.27, dimana $\mathrm{R}(\mathbf{w})$ adalah fungsi *regularization* dan λ adalah parameter kontrol.

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \mathcal{L}(\mathbf{w}) + \lambda \mathrm{R}(\mathbf{w}) \tag{5.27}$$

[9] Baca buku Yoav Goldberg [1] untuk mendapat penjelasan lebih baik.

Pilihan umum untuk fungsi *regularization* pada umumnya adalah L_2 dan L_1 norm. L_2 *regularization* menggunakan jumlah kuadrat dari *Euclidean norm*[10] seluruh parameter, seperti pada persamaan 5.28. Sedangkan, L_1 *regularization* menggunakan jumlah dari nilai *absolute-value norm* seluruh parameter, diberikan pada persamaan 5.29.

$$\mathrm{R}(\mathbf{w}) = \|\mathbf{w}\|_2^2 = \sum_i (w_i)^2 \tag{5.28}$$

$$\mathrm{R}(\mathbf{w}) = \|\mathbf{w}\| = \sum_i |w_i| \tag{5.29}$$

Perlu kamu perhatikan, metode *gradient descent* memiliki syarat bahwa fungsi yang digunakan haruslah dapat diturunkan (*differentiable*). L_2 dapat diturunkan pada seluruh poin, sementara L_1 tidak dapat diturunkan pada semua poin.[11] Kita dapat menggunakan teknik *subgradient* untuk menyelesaikan permasalahan ini. Kami serahkan pembaca untuk mengeksplorasi teknik tersebut sendiri. Implikasi penggunakan regularisasi terhadap kompleksitas model dibahas lebih lanjut pada bab 9.

Selain itu, ada hal lainnya yang perlu diperhatikan saat menggunakan *gradient descent* yaitu apakah suatu fungsi *convex* atau *concave*. Izinkan saya mengutip pernyataan dari buku Yoav Goldberg [1] halaman 31 secara langsung menggunakan bahasa Inggris agar tidak ada makna yang hilang.

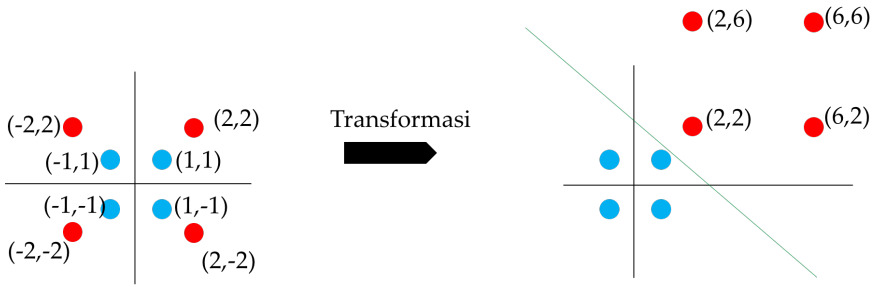
Convexity. In gradient-based optimization, it is common to distinguish between *convex* (or *concave*) functions and *non-concave* functions. A *convex function* is a function whose second-derivative is always non-negative. As a consequence, convex functions have a single minimum point. Similarly, *concave functions* are functions whose second-derivatives are always negative or zero, and as a consequence have a single maximum point. Convex (concave) functions have the property that they are easy to minimize (maximize) using gradient-based optimization–simply follow the gradient until an extremum point is reached, and once it is reached we know we obtained the global extremum point. In contrast, for functions that are neither convex or concave, a gradient-based optimization procedure may converge to a local extremum point, missing the global optimum.

5.10 Transformasi Data

Seperti yang sudah dijelaskan sebelumnya, alangkah baik apabila semua data memiliki hubungan secara linear atau bersifat *linearly separable*. Kenyataan-

[10] https://en.wikipedia.org/wiki/Norm_(mathematics)
[11] https://stats.stackexchange.com/questions/136895/why-is-the-l1-norm-in-lasso-not-differentiable

nya, kebanyakan data bersifat *non-linearly separable*. Kita dapat mentransformasi data yang bersifat *non-linearly separable* menjadi *linearly-separable* sebelum menggunakan model linear untuk mengklasifikasikan data. Sebagai contoh, perhatikan Gambar 5.15. Pada gambar bagian kiri, terdapat empat titik yang *non-linearly separable*. Titik-titik itu ditransformasi sehingga menjadi gambar bagian kanan. Fungsi transformasi yang digunakan diberikan pada persamaan 5.30.



Gambar 5.15: Contoh transformasi [7].

$$\phi(x,y) = \begin{cases} \sqrt{x^2+y^2} \geq 2 \rightarrow (4 - x + \|x-y\|, 4 - x + \|x-y\|) \\ \sqrt{x^2+y^2} \leq 2 \rightarrow (x,y) \end{cases} \tag{5.30}$$

Secara umum, fungsi transformasi tidaklah sesederhana contoh yang sudah diberikan. Fungsi transformasi pada umumnya menambah dimensi data (misal dari dua dimensi menjadi tiga dimensi). Beberapa fungsi transformasi (dikenal juga dengan istilah *kernel*) yang terkenal diantaranya:[12]

1. Fisher Kernel
2. Graph Kernel
3. Kernel Smoother
4. Polynomial Kernel
5. Radial Basis Function Kernel
6. String Kernel

Untuk penjelasan masing-masing *kernel*, silahkan membaca literatur lain lebih lanjut. Model linear yang memanfaatkan fungsi-fungsi *kernel* ini adalah *support vector machine* (SVM) [31, 32]. Perhatikan, algorithma SVM sebenarnya sangatlah penting. Akan tetapi, perlu kami informasikan bahwa buku ini tidak memuat materi SVM secara detil. Dengan demikian, kami harap pembaca dapat mencari referensi lain tentang SVM. Metode transformasi pun tidaklah

[12] https://en.wikipedia.org/wiki/Kernel_method

sempurna, seperti yang akan dijelaskan pada bab 11, karena memang data secara alamiah memiliki sifat yang non-linear. Dengan demikian, lebih baik apabila kita langsung saja memodelkan permasalahan dengan fungsi non-linear.

5.11 Bacaan Lanjutan

Kami harap pembaca mampu mengeksplorasi materi *kernel method* dan *support vector machine* (SVM). Kami mencantumkan materi SVM pada buku ini sedemikian pembaca mampu mendapatkan intuisi, tetapi tidaklah detil. Kami sarankan kamu membaca pranala https://www.svm-tutorial.com/ karena ditulis dengan cukup baik. Mengerti materi SVM dan *convex optimization* secara lebih dalam akan sangat membantu pada bab-bab berikutnya. Selain itu, kami juga menyarankan pembaca untuk melihat kedua pranala tambahan tentang *learning rate* dan momentum:

1. http://users.ics.aalto.fi/jhollmen/dippa/node22.html
2. http://www.willamette.edu/~gorr/classes/cs449/momrate.html

Soal Latihan

5.1. Hill Climbing
Baca dan jelaskanlah konsep Hill Climbing!

5.2. Variasi Optimisasi
Baca dan jelaskanlah variasi konsep optimisasi lain, selain *stochastic gradient descent*!

5.3. Convex Optimization
Bacalah literatur yang memuat materi tentang *convex optimization*! Jelaskan pada teman-temanmu apa itu fungsi *convex* dan *concave*, tidak lupa isi materi yang kamu baca! Bagaimana hubungan *convex optimization* dan pembelajaran mesin?

5.4. Sum of Squared Errors
Salah satu cara untuk mencari nilai parameter pada regresi adalah menggunakan teknik *sum of squared errors*. Jelaskanlah bagaimana cara kerja metode tersebut! (Hint: baca buku oleh James et al. [20])

5.5. Linear Discriminant Analysis
Jelaskan prinsip dan cara kerja *linear discriminant analysis*! (Hint: baca buku oleh James et al. [20])

6

Pohon Keputusan

> "Sometimes you make the right decision, sometimes you make the decision right."
>
> Phil McGraw

Bab ini akan menjelaskan salah satu varian pohon keputusan yaitu ID3 oleh Quinlan [33, 34] yang terinspirasi oleh teori informasi [35]. Algoritma ini sudah cukup tua, tetapi layak dimengerti. ID3 adalah salah satu varian dari *supervised learning*.

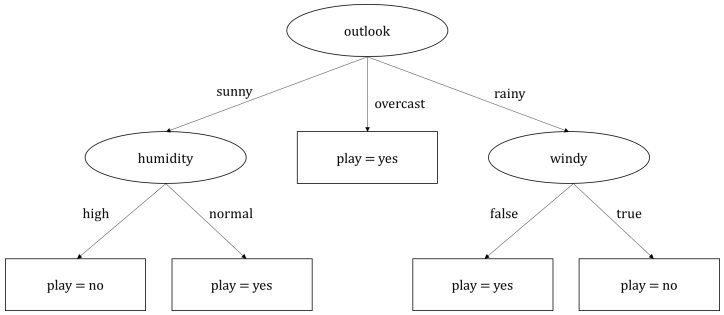
6.1 Inductive Learning

Salah satu bentuk "kecerdasan" sederhana kemungkinan adalah dalam bentuk aturan (*rule*) yang merepresentasikan pengetahuan. Misalkan, untuk menentukan apakah suatu pasien terserang penyakit tertentu, dokter mencari tahu gejala-gejala yang ada. Berdasarkan gejala-gejala yang ada, dokter memutuskan bahwa pasien memiliki suatu penyakit. Pada zaman dahulu, peneliti mentranskripsi aturan-aturan (*if then*) eksplisit (berdasarkan pengetahuan ahli) untuk membuat agen cerdas (*expert system*). Aturan sangat berguna, tetapi proses transkripsi pengetahuan sang ahli menjadi aturan formal (matematis) adalah hal yang sulit. Terlebih lagi, aturan-aturan yang sudah dibangun cenderung tidak dapat diubah dengan mudah.

Pada era *big data* seperti sekarang, kita dapat mengotomatisasi hal tersebut dengan membuat aturan-aturan secara otomatis berdasarkan contoh data yang ada (*machine learning*). Pendekatan ini disebut *inductive learning*, yaitu mengembangkan aturan klasifikasi yang dapat menentukan kelas suatu *instance* berdasarkan nilai atributnya (*feature vector*). Cara paling sederhana diberikan pada subbab 3.3 yaitu mendaftarkan seluruh kemungkinan aturan yang ada, kemudian menghapus yang kurang cocok. Algoritma lebih baik adalah dengan membangun pohon keputusan (*decision tree*).

6.2 ID3

Decision tree adalah varian dari *inductive learning*. ID3 adalah salah satu algoritma varian *decision tree* [34]. *Decision tree* dibangun berdasarkan asumsi bila atribut yang ada memberikan informasi yang cukup memadai maka kita mampu membangun *decision tree* yang mampu mengklasifikasikan seluruh *instance* di *training data* [34]. Akan tetapi, kita tentunya ingin melakukan generalisasi, yaitu *decision tree* yang juga mampu mengklasifikasikan objek dengan benar untuk *input* yang tidak ada di *training data* (*unseen instances*). Oleh karena itu, kita harus mampu mencari hubungan antara kelas dan nilai atribut.



Gambar 6.1: *Final decision tree*

id	outlook	temperature	humidity	windy	play (class)
1	sunny	hot	high	false	no
2	sunny	hot	high	true	no
3	overcast	hot	high	false	yes
4	rainy	mild	high	false	yes
5	rainy	cool	normal	false	yes
6	rainy	cool	normal	true	no
7	overcast	cool	normal	true	yes
8	sunny	mild	high	false	no
9	sunny	cool	normal	false	yes
10	rainy	mild	normal	false	yes
11	sunny	mild	normal	true	yes
12	overcast	mild	high	true	yes
13	overcast	hot	normal	false	yes
14	rainy	mild	high	true	no

Tabel 6.1: Contoh dataset *play tennis* (UCI machine learning repository).

Strategi pembangunan ID3 adalah berdasarkan *top-down* rekursif. Pertama, kita pilih atribut untuk *root* pohon, lalu membuat cabang untuk setiap nilai atribut yang mungkin. Untuk masing-masing cabang, kita buat *subtree*. Kita hentikan proses ini ketika kita sudah mencapai *leaf* (tidak bisa mencabang lebih jauh). *Leaf* ditandai apabila seluruh *instance* pada cabang tersebut memiliki kelas yang sama. Atribut yang sudah dipilih pada *ancestor* tidak akan dicoba pada percabangan di cabang tertentu.

Sebagai contoh, perhatikanlah Gambar 6.1 yang merupakan hasil ID3 untuk Tabel 6.1. Bentuk elips merepresentasikan nama atribut, sementara *edge* (panah) merepresentasikan nilai atribut. Bentuk segi empat merepresentasikan klasifikasi kelas (*leaf*). Pohon keputusan pada Gambar 6.1 dapat dikonversi menjadi kumpulan aturan klasifikasi berbentuk logika preposisi dengan menelusuri setiap cabang pada pohon tersebut, yaitu:

- if *outlook=sunny* and *humidity=high* then *play=no*
- if *outlook=sunny* and *humidity=normal* then *play=yes*
- if *outlook=overcast* then *play=yes*
- if *outlook=rainy* and *windy=false* then *play=yes*
- if *outlook=rainy* and *windy=true* then *play=no*

Karena *decision tree* dapat interpretasikan sebagai logika preposisi, model ini tergolong *interpretable*. Artinya, manusia dapat mengerti proses yang terjadi pada model dengan mudah.

Pada setiap langkah membangun ID3, kita menggunakan ***information gain*** untuk memilih kandidat atribut terbaik. *Information gain* mengukur kemampuan suatu atribut untuk memisahkan *training data* berdasarkan kelas [7].

Sebelum masuk ke perumusan *information gain*, penulis akan mengingatkan **entropy** terlebih dahulu. Entropy (derajat ketidakteraturan) adalah informasi yang dibutuhkan untuk memprediksi sebuah kejadian, diberikan distribusi probabilitas. Secara matematis, entropy didefinisikan pada persamaan 6.1 (**x** adalah kumpulan nilai probabilitas).

$$\text{entropy}(\mathbf{x}) = -\sum_{i=1}^{N} x_i \log x_i \tag{6.1}$$

Kita juga definisikan **Info** sebagai persamaan 6.2, dimana c_i adalah banyaknya *instance* diklasifikasikan sebagai kelas ke-i (atau secara lebih umum, ke *node-i*).

$$\text{Info}(c_1, c_2, \ldots, c_N) = \text{entropy}(\frac{c_1}{\sum_j^N c_j}, \frac{c_2}{\sum_j^N c_j}, \ldots, \frac{c_N}{\sum_j^N c_j}) \tag{6.2}$$

Information gain dihitung sebagai persamaan 6.3, dimana c_i adalah jumlah *instance* untuk kelas ke-i, v adalah nilai atribut, c^v adalah banyaknya *instance* ketika dicabangkan dengan nilai atribut v, c_x^v adalah banyaknya *instance* kelas saat percabangan. *Information gain* dapat dimaknai sebagai pengurangan entropy karena melakukan percabangan.

$$IG(c_1, c_2, \ldots, c_N) = \text{Info}(c_1, c_2, \ldots, c_N) - \sum_{v \epsilon V} \frac{c^v}{\sum_{i=1}^N c_i} \text{Info}({c_1}^v, {c_2}^v, \ldots, {c_N}^v) \tag{6.3}$$

Sebagai contoh, mari kita hitung *Information gain* untuk atribut *outlook* sebagai *root*. Dari keseluruhan data terdapat 9 *instance* untuk *play* = *yes* dan 5 *instance* untuk *play* = *no*. Kita hitung Info semesta sebagai (*log* basis 2)

$$\begin{aligned}
\text{Info}([9,5]) &= \text{entropy}\left(\left[\frac{9}{14}, \frac{5}{14}\right]\right) \\
&= -\frac{9}{14}\log\left(\frac{9}{14}\right) - \frac{5}{14}\log\left(\frac{5}{14}\right) \\
&= 0.940
\end{aligned}$$

Kita hitung *entropy* untuk masing-masing nilai atribut *outlook* sebagai berikut:

- *outlook* = *sunny*
  Ada dua *instance* dengan *play* = *yes* dan tiga *instance* dengan *play* = *no* saat *outlook* = *sunny*, dengan demikian kita hitung Info-nya.

$$\begin{aligned}
\text{Info}([2,3]) &= \text{entropy}\left(\frac{2}{5}, \frac{3}{5}\right) \\
&= -\frac{2}{5}\log\left(\frac{2}{5}\right) - \frac{3}{5}\log\left(\frac{3}{5}\right) \\
&= 0.971
\end{aligned}$$

- *outlook* = *overcast*
  Ada empat *instance* dengan *play* = *yes* dan tidak ada *instance* dengan *play* = *no* saat *outlook* = *overcast*, dengan demikian kita hitung Info-nya.

$$\begin{aligned}
\text{Info}([4,0]) &= \text{entropy}\left(\frac{4}{4}, \frac{0}{4}\right) \\
&= -\frac{4}{4}\log\left(\frac{4}{4}\right) - \frac{0}{4}\log\left(\frac{0}{4}\right) \\
&= 0
\end{aligned}$$

  Perhatikan log 0 pada matematika adalah tidak terdefinisi, tapi kita anggap 0 log 0 sebagai 0 dalam komputasi.

- $outlook = rainy$
  Ada tiga *instance* dengan $play = yes$ dan dua *instance* dengan $play = no$ saat $outlook = rainy$, dengan demikian kita hitung Info-nya.

$$\begin{aligned} Info([3,2]) &= entropy\left(\frac{3}{5}, \frac{2}{5}\right) \\ &= -\frac{3}{5}\log\left(\frac{3}{5}\right) - \frac{2}{5}\log\left(\frac{2}{5}\right) \\ &= 0.971 \end{aligned}$$

Kita hitung *information gain* untuk atribut *outlook* sebagai

$$\begin{aligned} IG(outlook) &= \text{Info}\left([9,5]\right) - \\ &\quad \left(\frac{5}{14} \times \text{Info}\left([3,2]\right) + \frac{4}{14} \times \text{Info}([4,0]) + \frac{5}{14} \times \text{Info}([3,2])\right) \\ &= 0.940 - \left(\frac{5}{14} \times 0.971 + \frac{4}{14} \times 0 + \frac{5}{14} \times 0.971\right) \\ &= 0.940 - 0.693 \\ &= 0.247 \end{aligned}$$

Dengan metode yang sama, kita hitung *information gain* untuk atribut lainnya.

- $IG(temperature) = 0.029$
- $IG(humidity) = 0.152$
- $IG(windy) = 0.048$

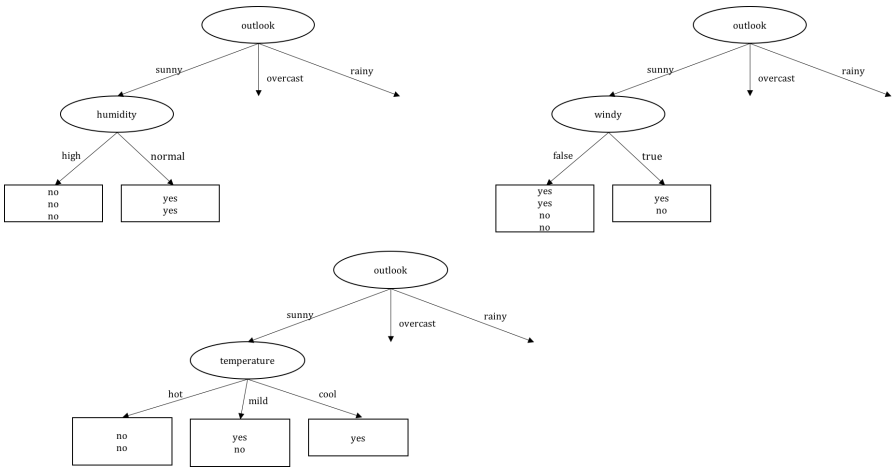
Dengan demikian, kita memilih atribut *outlook* sebagai *root*.

Kita lanjutkan lagi membuat *subtree* setelah memilih atribut *outlook* sebagai *root*. Kita hitung atribut yang tepat pada cabang $outook = sunny$, seperti diilustrasikan pada Gambar 6.2.

Pada $outlook = sunny$, terdapat dua *instance* dengan kelas $play = yes$ dan tiga *instance* dengan kelas $play = no$. Kita hitung *information gain* saat melanjutkan cabang dengan atribut *humidity*.

$$\begin{aligned} IG(humidity) &= \text{Info}([2,3]) - \left(\frac{3}{5} \times \text{Info}([0,3]) + \frac{2}{5} \times \text{Info}([2,0])\right) \\ &= 0.971 - 0 \\ &= 0.971 \end{aligned}$$

Untuk setiap kedalaman, kita coba menggunakan atribut yang belum pernah dicoba pada level-level lebih atas, seperti yang sudah diilutrasikan. Proses ini dilanjutkan sampai kita tidak bisa atau tidak perlu mencabang lagi.



Gambar 6.2: Percabangan

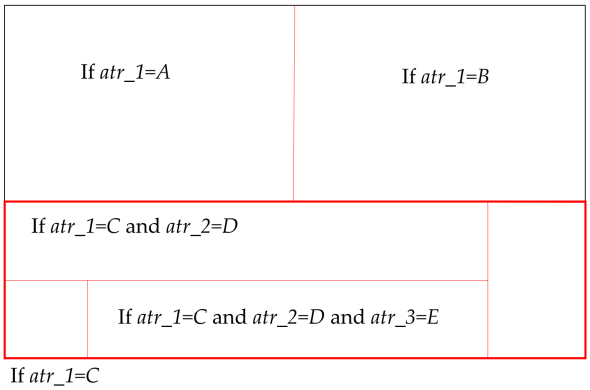
6.3 Isu pada ID3

Pada algoritma *decision tree* secara umum, terdapat beberapa isu diantara lain [7, 36]:

1. Mudah overfitting
2. Masalah menangani atribut kontinu
3. *Information gain* memiliki bias terhadap atribut yang memiliki banyak nilai (*highly-branching attributes*)
4. Data dengan *missing value*. Beberapa sel pada tabel dataset tidak terisi.
5. Data dengan *unseen value*. Misal nilai atribut yang tidak pernah dilihat pada *training data*, muncul saat *testing*.

6.4 Pembagian Ruang Konsep

Ada hubungan antara algorithma *decision tree* dan model linear. Pada model linear, kita semacam membagi-bagi ruang konsep (semesta data) menjadi ruang per kelas menggunakan garis pembatas linear. *Decision tree* melakukan hal yang hampir sama, karena percabangan *decision tree* dapat dianggap sebagai linear. Sebagai contoh perhatikan ilustrasi Gambar 6.3, dimana semesta adalah suatu ruang konsep. Tiap ruang merepresentasikan suatu cabang (dari *root* sampai *leaf*) pada *decision tree*. Garis-garis yang membentuk ruang-ruang pemisah disebut sebagai ***decision boundary***.



Gambar 6.3: Ilustrasi pembagian ruang konsep

Soal Latihan

6.1. Isu
Pada subbab 6.3, telah disebutkan isu-isu yang ada pada ID3, sebutkan dan jelaskan bagaimana cara menangani masing-masing isu tersebut!

6.2. Gain Ratio
Selain *information gain*, kita dapat menggunakan cara lain untuk memilih atribut bernama *gain ratio*. Jelaskan perbedaan keduanya! Yang mana lebih baik?

6.3. C4.5
ID3 disempurnakan kembali oleh pembuat aslinya menjadi C4.5. Jelaskanlah perbedaan ID3 dan C4.5, beserta alasan strategi penyempurnaan!

6.4. Pruning
Jelaskan apa itu *pruning* dan bagaimana cara melakukan *pruning* untuk *decision tree*!

6.5. Association Rule
Terdapat beberapa algoritma *association rule* yang membentuk aturan-aturan seperti *decision tree*.
(a) Jelaskanlah algoritma PRISM dan Apriori!
(b) Jelaskan perbedaan *association rule* dan *inductive learning*!

6.6. Final Decision Tree
Lanjutkanlah proses konstruksi ID3 untuk Tabel 6.1 hingga membentuk *decision tree* akhir seperti pada Gambar 6.1!

6.7. Variant
Jelaskanlah *Random Forest*, *Bagging* dan *Boosting* pada *Decision Tree*!

7

Support Vector Classifier

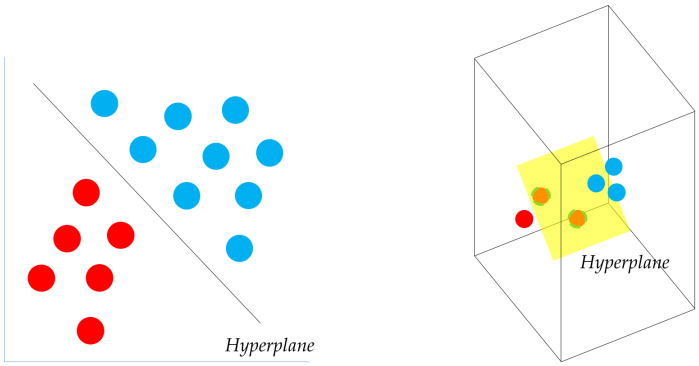
> "More data beats clever algorithms, but better data beats more data."
>
> Peter Norvig

Saat membaca judul bab ini, kamu mungkin berpikir bahwa *support vector machine* akan dibahas. *Support vector classifier* dan *support vector machine* adalah dua hal yang berbeda, walau kedua istilah tersebut sering digunakan pada konteks yang mirip [20]. *Support vector classifier* sesungguhnya adalah konsep yang lebih sederhana. Walaupun bab ini menyinggung kulit *support vector machine*, kami tidak membahas secara rinci. Akan tetapi, kami berharap pembaca mampu mendapatkan intuisi. Kamu dapat menganggap bab ini sebagai kelanjutan cerita bab 5. Referensi utama bab ini adalah buku karangan James et al. [20].

7.1 Maximal Margin Classifier

Ingat kembali kedua bab sebelumnya bahwa model klasifikasi mencari suatu *decision boundary* untuk memisahkan data pada kelas satu dan kelas lainnya. Apabila kamu memiliki data berdimensi dua, maka *decision boundary* yang kita dapat berupa garis. Pada data tiga dimensi, *decision boundary* berupa sebuah bidang (*plane*). Sebagai ilustrasi, lihatlah Gambar 7.1. Secara umum, konsep bidang pemisah disebut sebagai *hyperplane*. Untuk data berdimensi F, bidang pemisah kita memiliki dimensi $F - 1$.

Secara matematis, *hyperplane* didefinisikan pada persamaan 7.1 (dapat ditulis ulang seperti persamaan 7.2), dimana x melambangkan suatu fitur, $\mathbf{x}$ adalah *input* dalam bentuk *feature vector* dan F melambangkan banyaknya fitur. Ingat kembali, ruas kiri pada persamaan adalah bentuk dasar pada model linear. Dengan demikian, kita mengasumsikan bahwa data dapat dipisahkan



Gambar 7.1: Ilustrasi *hyperplane*.

secara linear.

$$x_1 w_1 + x_2 w_2 + \cdots + x_F w_F + b = 0 \tag{7.1}$$

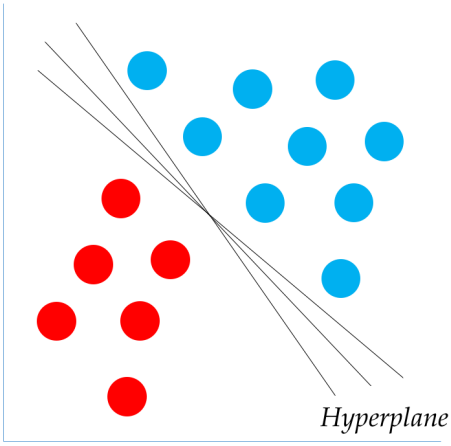
$$\mathbf{x} \cdot \mathbf{w} + b = 0 \tag{7.2}$$

Untuk permasalahan klasifikasi dua kelas, kita dapat memisahkan keputusan berdasarkan letak data pada *hyperplane*, misal di atas atau di bawah *hyperplane* pada Gambar 7.1. Secara lebih matematis, seperti pada persamaan 7.3 dan 7.4. Konsep ini mirip seperti yang sudah dijelaskan pada bab 5 tentang melakukan *binary classification* menggunakan fungsi *sign* dan *thresholding*.

$$\text{if } \mathbf{x} \cdot \mathbf{w} + b > 0, \text{then class } A \tag{7.3}$$

$$\text{if } \mathbf{x} \cdot \mathbf{w} + b < 0, \text{then class } B \tag{7.4}$$

Apabila kita memang mampu memisahkan data dua kelas secara sempurna dengan suatu *hyperplane* (*linearly separable*), pilihan *hyperplane* yang dapat kita buat tidaklah satu. Artinya, kita dapat menggeser-geser garis pembatas, disamping tetap memisahkan data secara sempurna, seperti diilustrasikan pada Gambar 7.2. *Hyperplane* terbaik dari beberapa pilihan yang ada adalah yang memiliki *maximal margin*. Artinya, suatu *hyperplane* yang memiliki jarak terjauh terhadap data pada suatu kelas. Dengan demikian, ada jarak yang besar dari *hyperplane* dengan data. Ilustrasi diberikan pada Gambar 7.3. Kita harap, suatu *hyperplane* yang memiliki *margin* besar dapat melakukan klasifikasi dengan baik pada data yang belum kita lihat, karena kita memberikan *margin* yang besar untuk suatu data baru masuk ke daerah kelas masing-masing. Bentuk lingkaran yang memiliki *border* berwarna hitam pada Gambar 7.3 menandakan data terluar pada masing-masing kelas, dikenal



Gambar 7.2: Ilustrasi banyak pilihan *hyperplane*. Garis hitam melambangkan opsi *hyperplane* untuk memisahkan data secara sempurna.

sebagai ***support vectors***. Garis putus-putus melambangkan garis yang dibentuk oleh masing-masing *support vectors* untuk masing-masing kelas (*margin*).

Apabila kita definisikan (simbolkan) kelas pertama dengan *output* bernilai 1 dan kelas kedua bernilai -1 (ingat kembali materi fungsi *sign*), maka *support vectors* adalah poin $\mathbf{x}$ yang memenuhi kriteria pada persamaan 7.5. Dengan demikian, semua data berlabel 1 dan -1 memenuhi persamaan 7.6, dimana y melambangkan kategori. Hal inilah yang disebut sebagai ***maximal margin classifier***. Kita mencari *support vectors* yang memberikan *hyperplane* yang memiliki *maximal margin*. Lihatlah ilustrasi pada Gambar 7.4!

$$|\mathbf{x} \cdot \mathbf{w} + b| = 1 \tag{7.5}$$

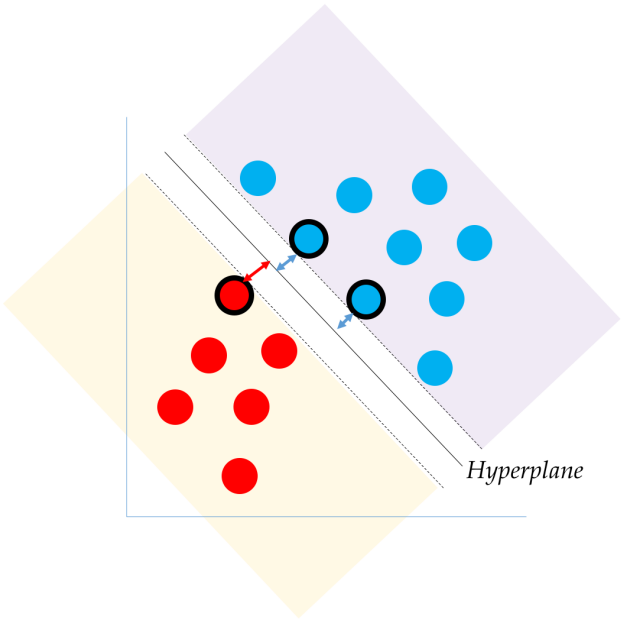
$$y_i(\mathbf{x} \cdot \mathbf{w} + b) \geq 1 \tag{7.6}$$

Misalkan kita ambil satu *support vector* dari masing-masing kelas. Pada kelas pertama, ia memenuhi $\mathbf{x}^{c_1} \cdot \mathbf{w} + b = 1$. Pada kelas kedua, ia memenuhi $\mathbf{x}^{c_2} \cdot \mathbf{w} + b = -1$. Apabila kita kurangi kedua persamaan tersebut, kita mendapatkan persamaan 7.7. Persamaan 7.8 memberikan perhitungan *margin* antara *support vectors* dan *hyperplane* yang memberikan nilai maksimal.

$$\mathbf{w} \cdot (\mathbf{x}^{c_1} - \mathbf{x}^{c_2}) = 2 \tag{7.7}$$

$$\frac{\mathbf{w}}{\|\mathbf{w}\|} \cdot (\mathbf{x}^{c_1} - \mathbf{x}^{c_2}) = \frac{2}{\|\mathbf{w}\|} \tag{7.8}$$

Sekarang, kita formalisasi *maximal margin classifier* secara matematis. Objektifnya adalah memaksimalkan *margin* (persamaan 7.9) dengan menjaga setiap *training data* diklasifikasikan dengan benar (persamaan 7.10).



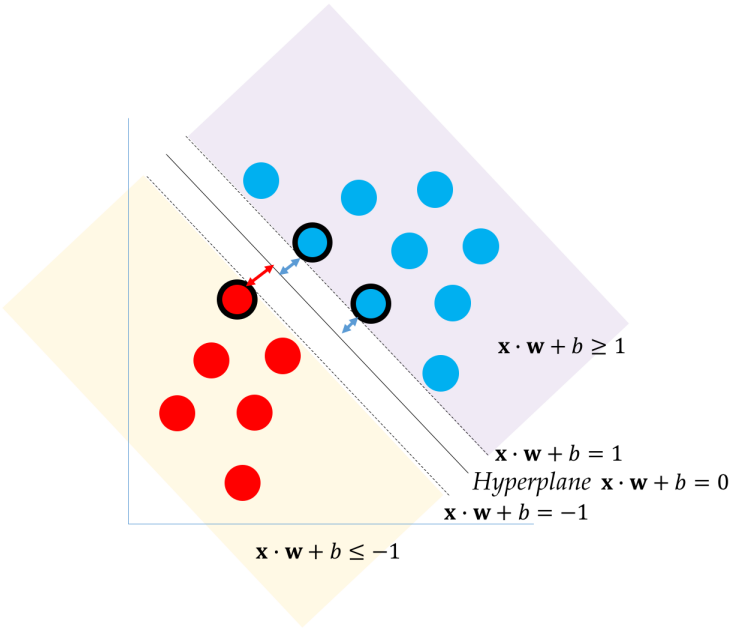
Gambar 7.3: *Maximal Margin Hyperplane*. Bentuk lingkaran yang memiliki *border* di-*bold* berwarna hitam menandakan data terluar pada masing-masing kelas, dikenal sebagai *support vectors*. Garis putus-putus melambangkan garis yang dibentuk oleh masing-masing *support vectors* untuk masing-masing kelas (*margin*).

$$\text{Objective} : \text{maximize Margin} = \frac{2}{\|\mathbf{w}\|} \tag{7.9}$$

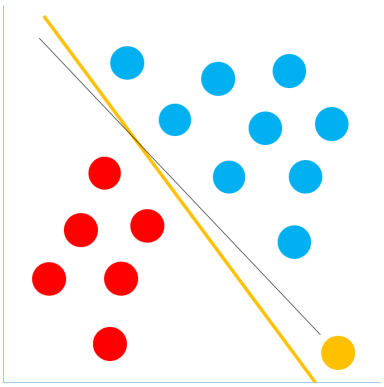
$$\text{Subject to} : y_i(\mathbf{x}_i \cdot \mathbf{w} + b) \geq 1 \tag{7.10}$$

Tidak sama dengan *model linear* yang sudah dijelaskan sebelumnya, kita ingin mengoptimasi sebuah fungsi sembari memenuhi kendala (*constraint*) dari fungsi lain. Ini adalah bentuk *integer linear programming*, dan solusi untuk *maximal margin classifier* dapat diselesaikan menggunakan *lagrange multiplier*. Untuk detail penyelesaiannya, silahkan membaca sumber lainnya.

Seperti yang kamu sadari, *maximal margin classifier* hanya bergantung pada subset *training data* yang dipilih sebagai *support vectors*. Dengan demikian, metode ini sangat sensitif terhadap tiap-tiap observasi. Satu observasi baru yang menjadi *support vector* dapat merubah *decision boundary*. Kita kenal ini sebagai *overfitting*. Ilustrasi permasalahan ini diberikan pada Gambar 7.5. Selain itu, *maximal margin classifier* mengasumsikan bahwa data bersifat *linearly separable*, walaupun kebanyakan data tidak bersifat demikian.



Gambar 7.4: *Maximal Margin Classifier*.



Gambar 7.5: *Maximal Margin Classifier* sangatlah sensitif terhadap perubahan *training data*. Lingkaran berwarna oranye melambangkan *training data* baru. Akibat kemuculan data baru ini, *decision boundary* awal (garis berwarna hitam) berubah secara cukup dramatis (garis berwarna oranye).

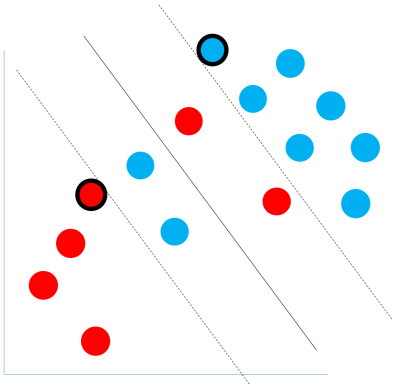
7.2 Support Vector Classifier

Support Vector Classifier adalah ekstensi dari *maximal margin classifier*. Ide utamanya adalah relaksasi kendala pada persamaan 7.10. Sebelumnya, *maximal margin classifier* mengasumsikan bahwa data bersifat *linearly separable* dan dipisahkan secara sempurna. Akan tetapi, kenyataan tidaklah demikian. Ide *support vector classifier* adalah memperbolehkan beberapa data diklasifikasikan dengan salah. Kita modifikasi *constraint* persamaan 7.10 menjadi persamaan 7.11 untuk memperbolehkan model salah mengklasifikasikan data dengan parameter kontrol ϵ, melambangkan apakah suatu observasi boleh berada pada ruang yang tidak tepat. Kita juga dapat membatasi seberapa banyak kesalahan yang dibuat dengan *constraint* baru yang diberikan pada persamaan 7.12.

$$\text{Maximize margin, subject to : } y_i(\mathbf{x}_i \cdot \mathbf{w} + b) \geq 1(1 - \epsilon_i) \tag{7.11}$$

$$\epsilon_i \geq 0; \sum \epsilon_i \leq C \tag{7.12}$$

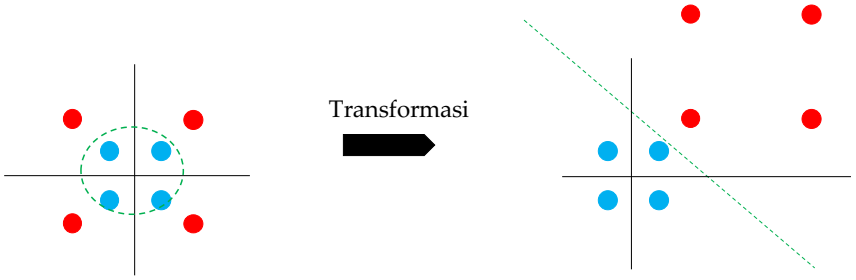
Ilustrasi *support vector classifier* diberikan pada Gambar 7.6. Disini, kita sedikit modifikasi definisi *support vectors* sebagai observasi yang tepat jauh pada *margin* atau pada daerah yang tidak sesuai dengan kelasnya [20].



Gambar 7.6: *Support Vector Classifier*. Lingkaran dengan *border* di-*bold* berwarna hitam melambangkan *support vectors*, garis putus-putus melambangkan *margin*.

Walaupun memperbolehkan beberapa observasi boleh tidak berada pada ruang yang tepat (berdasarkan *margin*), *support vector classifier* masih memiliki asumsi bahwa *decision boundary* berbentuk suatu fungsi linear. Ini adalah batasan utama model ini.

7.3 Support Vector Machine



Gambar 7.7: Ilustrasi transformasi data. Garis berwarna hijau (*dashed*) melambangkan *decision boundary*.

Berhubung banyak *decision boundary* tidak dapat dimodelkan dengan suatu bentuk atau persamaan linear, kita harus memodelkan *decision boundary* sebagai fungsi non-linear. Ekstensi *support vector classifier* adalah ***support vector machine*** yang menggunakan teknik ***kernel***. Suatu fungsi *kernel* mentransformasi data ke ruang (*space* atau dimensi) lainnya (biasanya ke dimensi lebih tinggi). Data yang ditransformasi ini (pada dimensi lebih tinggi), kita harapkan dapat dipisahkan dengan fungsi linear. Apabila kita lihat balik pada dimensi asli, *decision boundary* pada dimensi yang baru memodelkan suatu *decision boundary* non-linear pada dimensi aslinya. Hal ini diilustrasikan pada Gambar 7.7. Fungsi *kernel* ini ada banyak, beberapa yang terkenal diantaranya:[1]

1. Polynomial Kernel (persamaan 7.13)

$$k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j + 1)^d \tag{7.13}$$

2. Radial Basis Function Kernel (persamaan 7.14, σ^2 melambangkan varians)

$$k(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right) \tag{7.14}$$

Yang membedakan *support vector machine* dan *support vector classifier* adalah mengintegrasikan fungsi *kernel* pada model.

Sebelum membahas bagaimana fungsi *kernel* diintegrasikan pada SVM, kita sedikit bahas kembali *support vector classifier*. Ingat kembali *support vector classifier* mencari *maximal margin* (persamaan 7.9) dengan kendala persamaan 7.11 dan 7.12. Suatu *support vector classifier* yang memenuhi seluruh kendala tersebut dapat direpresentasikan sebagai persamaan 7.15 dimana

[1] https://data-flair.training/blogs/svm-kernel-functions/

$\mathbf{x}'$ melambangkan suatu data baru, $\mathbf{x}_i$ adalah suatu *instance* pada *training data* dan N adalah banyaknya *training data*. Operasi $\langle \mathbf{x}', \mathbf{x}_i \rangle$ melambangkan *inner product*. Cara menghitung *inner product* dari dua vektor diberikan pada persamaan 7.16, dimana F melambangkan panjangnya vektor. *Inner product* dapat diinterpretasikan sebagai perhitungan kemiripan dua vektor.

$$f(\mathbf{x}') = \beta_0 + \sum_{i=1}^{N} \alpha_i \langle \mathbf{x}', \mathbf{x}_i \rangle \tag{7.15}$$

$$\langle \mathbf{a}, \mathbf{b} \rangle = \sum_{j=1}^{F} a_j b_j \tag{7.16}$$

Untuk menghitung parameter β dan α pada persamaan 7.15, kita membutuhkan $\binom{N}{2}$ kombinasi pasangan *instance* yang ada pada *training data*. Akan tetapi, pada saat melewatkan suatu *input* baru pada persamaan tersebut, kita sebenarnya cukup menghitung seberapa kedekatan (kemiripan) antara *input* dan *support vectors*. Hal ini disebabkan α bernilai 0 untuk *instance* selain *support vectors*, i.e., diatur hanya bernilai *nonzero* untuk *support vectors*. Artinya, keputusan klasifikasi bergantung pada pilihan *support vectors*. Dengan demikian, kita dapat menulis kembali persamaan 7.15 sebagai persamaan 7.17, dimana S melambangkan *support vectors*.

$$f(\mathbf{x}') = \beta_0 + \sum_{\mathbf{x}_i \in S} \alpha_i \langle \mathbf{x}', \mathbf{x}_i \rangle \tag{7.17}$$

Ketika menggunakan fungsi *kernel*, kita menghitung kemiripan (*inner product*) antara dua vektor pada dimensi transformasi. Kita mengganti *inner product* menggunakan suatu fungsi *kernel*, ditulis sebagai persamaan 7.18. Persamaan inilah yang dikenal sebagai ***support vector machine*** (SVM). Untuk memodelkan *non-linear decision boundary*, kita menggunakan fungsi yang bersifat non-linear sebagai *kernel*.

$$f(\mathbf{x}') = \beta_0 + \sum_{\mathbf{x}_i \in S} \alpha_i k(\mathbf{x}', \mathbf{x}_i) \tag{7.18}$$

Untuk memahami SVM lebih dalam, kamu bisa membaca buku karangan Bishop [8].

7.4 Klasifikasi lebih dari dua kelas

Penjelasan pada bab ini berfokus pada *binary classification*. Memang, *maximal margin classifier* dan ekstensinya difokuskan pada permasalahan *binary classification*. Kita dapat mengekstensinya untuk *multi-class classification*. Ada dua teknik yang umumnya digunakan, yaitu *one versus one* dan *one versus all* seperti yang sudah dijelaskan pada bab 5. Kita dapat mendekomposisi *classifier* untuk *multi-label classification* menjadi beberapa *binary classifiers*, seperti yang sudah dijelaskan pada bab 5.

7.5 Tips

Untuk memahami materi yang disampaikan pada bab ini secara lebih dalam, kami menyarankan kamu untuk mempelajari *optimization theory* dan *operation research* (*i.e., integer linear programming*). Penulis harus mengakui tidak terlalu familiar dengan teori-teori tersebut. Sejarah dan perkembangan *support vector machine* dapat dibaca pada paper-paper yang diberikan di pranala http://www.svms.org/history.html. Walaupun penjelasan pada bab ini hanya bersifat "kulit"-nya saja, kami harap pembaca mampu mendapatkan intuisi.

Soal Latihan

7.1. Metode *Kernel*
Baca dan jelaskanlah konsep metode *kernel* yang dijelaskan pada buku Bishop [8]!

7.2. Fungsi *Kernel*
Jelaskanlah macam-macam fungsi *kernel* yang sering digunakan untuk *support vector machine*!

7.3. SVM-rank
Walau umumnya digunakan untuk permasalahan klasifikasi, SVM juga dapat diaplikasikan pada permasalahan *ranking* (*learning to rank*), dikenal sebagai SVM-*rank*.[2]. Permasalahan ini adalah inti dari *search engine* Jelaskanlah bagaimana cara kerja SVM-*rank*!

[2] https://www.cs.cornell.edu/people/tj/svm_light/svm_rank.html

8

Hidden Markov Model

> "Probability is expectation founded upon partial knowledge. A perfect acquaintance with all the circumstances affecting the occurrence of an event would change expectation into certainty, and leave neither room nor demand for a theory of probabilities."
>
> George Boole

Hidden Markov Model (HMM) adalah algoritma yang relatif cukup lama [37]. Tetapi algoritma ini penting untuk diketahui karena digunakan sebagai teknik dasar untuk *automatic speech recognition* (ASR) dan *part-of-speech* (POS) *tagging*. Bab ini akan membahas ide dasar HMM serta aplikasinya pada POS *tagging* (*natural language processing*). Aplikasi tersebut dipilih karena penulis lebih familiar dengan POS *tagging*. Selain itu, POS *tagging* relatif lebih mudah dipahami dibandingkan aplikasi HMM pada ASR.[1] HMM adalah kasus spesial ***Bayesian Inference*** [12, 38, 5]. Untuk mengerti *Bayesian Inference*, ada baiknya kamu membaca materi ***graphical model*** pada buku *pattern recognition and machine learning* [8]. Bab ini relatif lebih kompleks secara matematis dibanding bab-bab sebelumnya. Oleh karena itu, kami harap kamu membaca dengan sabar.

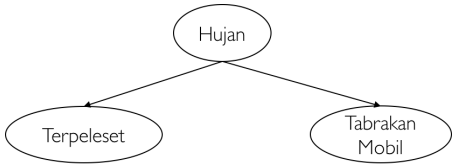
8.1 Probabilistic Reasoning

Pada logika matematika (*first order logic*), ketika kita memiliki premis "bila hujan, maka ayu terpeleset." Pada level *first order logic*, apabila "hujan" ter-

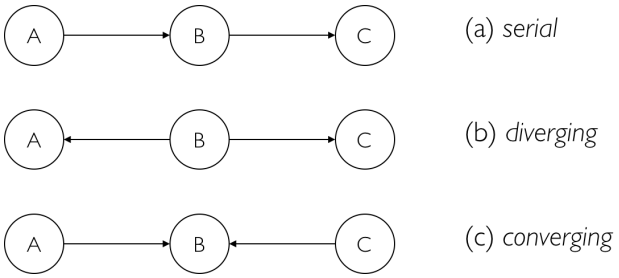
[1] Karena perlu dijelaskan juga cara transformasi sinyal suara menjadi data diskrit.

jadi, maka "terpeleset" juga pasti akan terjadi. Tetapi tidak sebaliknya, apabila kita "terpeleset", belum tentu "hujan" juga terjadi. Pada *probabilistic reasoning* kita mengkuantifikasi kepastian atau ketidakpastian itu. Apabila "hujan" terjadi, berapa besar kemungkinan "terpeleset" juga terjadi (dan sebaliknya).

Perhatikan Gambar 8.1! Gambar ini menerangkan hubungan pengkondisian *events*, disebut **Bayesian Network**. Panah dari "hujan" ke "terpeleset" merepresentasikan bahwa "hujan" adalah kondisi yang menyebabkan "terpeleset" terjadi (*causality*). Pada kerangka *probabilistic reasoning*, kita berpikir "karena ada yang terpeleset, mungkin ada hujan dan kecelakaan juga terjadi." Tetapi, apabila ada cerita lain bahwa jika ada seseorang yang "terpeleset" dan memang "hujan"; belum tentu "kecelakaan" terjadi.



Gambar 8.1: Contoh *Bayesian Network*.



Gambar 8.2: Tipe jaringan kausalitas.

Berdasarkan hubungan/jaringan kausalitas antara *events*, ada beberapa kerangka berpikir yang dapat digunakan: *serial*, *diverging*, dan *converging* [5]; diilustrasikan pada Gambar 8.2. Karakteristik dari masing-masing tipe kausalitas adalah sebagai berikut [5]:

(a) *Serial*. Bila kita mengetahui A maka kita bisa mengetahui sesuatu tentang B dan C. Tetapi apabila kita mengetahui B, mengetahui A tidak akan membantu inferensi kita terhadap C. Mengetahui C akan membuat kita mengetahui sesuatu tentang A ketika kita tidak mengetahui B. Artinya, hubungan antara A dan C di-*block* ketika B diketahui. Dengan bahasa

lebih matematis, A dan C bersifat **independen kondisional** (*conditionally independent*) ketika B diketahui. Perhatikan, disebut *kondisional* karena independen jika kondisi (mengetahui B) terpenuhi.

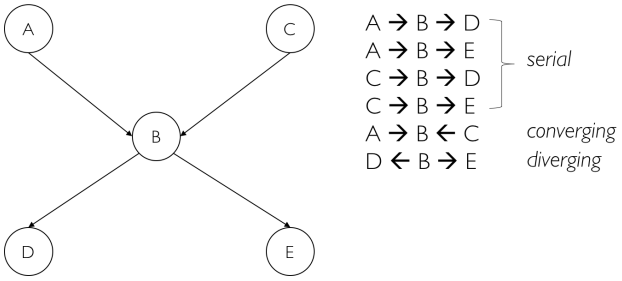
(b) *Diverging*. Bila kita mengetahui A maka kita bisa mengetahui sesuatu tentang C, dan sebaliknya. Tetapi apabila B diketahui, maka hubungan antara A dan C menjadi terputus. Dengan bahasa lebih matematis, A dan C independen kondisional ketika B diketahui.

(c) *Converging*. Tanpa mengetahui B, kita tidak bisa mencari tahu hubungan antara A dan C. Dengan bahasa lebih matematis, A dan C dependen kondisional ketika B diketahui.

Dua buah *events* X dan Y disebut ***d-separated*** apabila terdapat sebuah *intermediate variable* Z diantara mereka dan memenuhi kondisi:

1. Koneksi bersifat *serial* atau *diverging*, dan Z diketahui.
2. Koneksi bersifat *converging* dan Z tidak diketahui.
3. Bila tidak memenuhi kondisi yang disebutkan, artinya dua buah variabel tersebut tidak *d-separated*, disebut sebagai ***d-connected***.

Konsep ini penting dipahami untuk mengerti ide dasar *markov assumption* yang dibahas pada subbab berikutnya.



Gambar 8.3: Contoh inferensi.

Perhatikan Gambar 8.3! Dengan konsep yang sudah dipahami, mari kita coba lakukan inferensi pada jaringan tersebut. *Joint distribution* untuk seluruh *event* diberikan pada persamaan 8.1.

$$P(A, B, C, D, E) = P(D \mid B)\ P(E \mid B)\ P(B \mid A, C)\ P(A)\ P(C) \tag{8.1}$$

Sekarang, mari kita hanya lihat subgraf $\{A, B, E\}$ dengan koneksi tipe *serial*. Bila A diketahui, maka kita dapat melakukan inferensi terhadap C, seperti pada persamaan 8.2.

$$P(E \mid A) = P(E \mid B)\ P(B \mid A)\ P(A) + P(E \mid \neg B)\ P(\neg B \mid A)\ P(A) \tag{8.2}$$

Tetapi, apabila A dan B diketahui, maka inferensi terhadap E dilakukan seperti pada persamaan 8.3.

$$P(E \mid B, A) = P(E \mid B)\ P(B) \tag{8.3}$$

Operasi serupa dapat diaplikasikan pada koneksi tipe *converging* dan *diverging*. Perhatikan subgraf $\{A, B, C\}$ dengan tipe koneksi *converging* pada Gambar 8.3. Apabila B tidak diketahui, berarti A dan C terpisah (independen). Apabila B dan A diketahui, maka hubungan A dan C dapat dihitung sebagai persamaan 8.4.

$$P(C \mid B, A) = P(C \mid B)\ P(B \mid A)P(A) \tag{8.4}$$

Untuk koneksi tipe *diverging*, silahkan coba bagaimana mencari proses inferensinya! (pekerjaan rumah).

8.2 Generative Model

Pada *supervised learning* kamu sudah mengetahui bahwa kita memodelkan $p(y \mid x)$, memodelkan *target* y (label) ketika diberikan *input* x,[2] yaitu mencari tahu *decision boundary* antara keputusan. x dan y dapat berupa vektor, skalar, gambar, dan lain sebagainya. Sementara pada *unsupervised learning*, kita ingin mengaproksimasi distribusi asli dari sebuah *input* sebagai $p(x)$.

Berbeda dengan keduanya, *generative model* memodelkan $p(x, y)$. Persamaan itu dapat difaktorkan sebagai $p(x, y) = p(y \mid x)p(x)$. Pada umumnya, kita lebih tertarik dengan nilai y yang menyebabkan $p(x, y)$ bernilai maksimum, berhubung x akan selalu tetap–karena x adalah *fixed input*, y terbaiklah yang ingin kita temukan. Berbeda dengan *supervised learning*, *generative model* dapat difaktorkan menjadi $p(y \mid x)$ dan $p(x)$. Karena berbentuk *joint probability*, *generative model* memodelkan peluang kemunculan bersamaan. Kita ingin mengetahui seberapa mungkin suatu data x dihasilkan, diberikan y. Artinya seberapa mungkin *input* diobservasi untuk suatu *output*. Salah satu contoh *generative model* adalah *Hidden Markov Model* (HMM). HMM memodelkan observasi menggunakan proses *Markovian* dengan *state* yang tidak diketahui secara jelas (*hidden*). Kamu akan mengerti kalimat sebelumnya setelah membaca penjelasan buku ini seluruhnya.

Ide utama HMM adalah menyelesaikan persoalan ***sequence tagging***. Diberi-kan *input* $\mathbf{x}$ berupa sekuens (sekuens sinyal, sekuens kata, sekuens gambar, dsb). Kita ingin memodelkan sekuens *output* terbaik $\mathbf{y}$ untuk input tersebut.[3] *Output* ke-i bergantung pada *input* dari awal sampai ke-i dan *output* dari awal sampai sebelumnya $p(y_i \mid y_1, \ldots, y_{i-1}, x_1, \ldots, x_i)$. Berdasarkan pemaparan subbab 8.1, apabila suatu *event* dikondisikan variabel lain dengan

[2] Parameter $\mathbf{w}$ dihilangkan untuk menyederhanakan penjelasan.
[3] x_i dan y_i dapat berupa vektor

tipe koneksi *serial*, maka kita dapat mengabaikan banyak variabel historis. Hal ini dituangkan dalam persamaan 8.5,

$$p(y_i \mid y_1, \ldots, y_{i-1}, x_1, \ldots, x_i) = p(y_i \mid y_{i-1}, x_i) \tag{8.5}$$

Persamaan ini disebut ***first-order markov assumption***, yaitu suatu *event* yang seharusnya dikondisikan oleh suatu histori hanya bergantung pada *event* sebelumnya. Terdapat pula *second*, *third*, dst *markov assumption* yaitu bergantung pada dua, tiga, dst *events* sebelumnya. Walaupun hanya berupa penyederhanaan, asumsi ini memberi kita banyak keuntungan dari sisi komputasi.

8.3 Part-of-speech Tagging

Pada bidang pemrosesan bahasa alami (*natural language processing*), peneliti tertarik untuk mengetahui kelas kata untuk masing-masing kata di tiap kalimat. Misalkan kamu diberikan sebuah kalimat "*Budi menendang bola*". Setelah proses POS *tagging*, kamu akan mendapat "*Budi/Noun menendang/Verb bola/Noun*" (Gambar 8.4). Hal ini sangat berguna pada bidang pemrosesan bahasa alami, misalkan untuk memilih *noun* pada kalimat. Kelas kata disebut sebagai *syntactic categories*. Pada bahasa Inggris, kita mempunyai kelas kata yang dikenal dengan *Penn Treebank POS Tags*,[4] diberikan pada Tabel 8.1.

POS tag	Noun	Verb	Noun
Kata	Budi	Menendang	Bola

Gambar 8.4: Contoh POS *tagging*.

POS *tagging* adalah salah satu bentuk pekerjaan ***sequential classification***. Diberikan sebuah sekuens kata (membentuk satu kalimat), kita ingin menentukan kelas setiap kata/*token* pada kalimat tersebut. Kita ingin memilih sekuens kelas kata *syntactic categories* yang paling cocok untuk kata-kata/*tokens* pada kalimat yang diberikan. Secara formal, diberikan sekuens kata-kata $w_1, w_2, \ldots, w_T$, kita ingin mencari sekuens kelas kata $c_1, c_2, \ldots, c_T$ sedemikian sehingga kita memaksimalkan nilai probabilitas 8.6 [12, 38].

$$\hat{c_1}, \hat{c_2}, \ldots, \hat{c_T} = \underset{c_1, c_2, \ldots, c_T; c_i \epsilon C}{\arg\max} P(c_1, c_2, \ldots, c_N | w_1, w_2, \ldots, w_T) \tag{8.6}$$

Dimana C adalah daftar kelas kata. Akan tetapi, menghitung persamaan 8.6 sangatlah sulit karena dibutuhkan data yang sangat banyak (kombinasi sekuens

[4] https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

No.	Tag	Description
1.	CC	Coordinating conjunction
2.	CD	Cardinal number
3.	DT	Determiner
4.	EX	Existential there
5.	FW	Foreign word
6.	IN	Preposition or subordinating conjunction
7.	JJ	Adjective
8.	JJR	Adjective, comparative
9.	JJS	Adjective, superlative
10.	LS	List item marker
11.	MD	Modal
12.	NN	Noun, singular or mass
13.	NNS	Noun plural
14.	NNP	Proper noun singular
15.	NNPS	Proper noun plural
16.	PDT	Predeterminer
17.	POS	Possessive ending
18.	PRP	Personal pronoun
19.	PRP$	Possessive pronoun
20.	RB	Adverb
21.	RBR	Adverb, comparative
22.	RBS	Adverb, superlative
23.	RP	Particle
24.	SYM	Symbol
25.	TO	to
26.	UH	Interjection
27.	VB	Verb base form
28.	VBD	Verb past tense
29.	VBG	Verb gerund or present participle
30.	VBN	Verb past participle
31.	VBP	Verb non-3rd person singular present
32.	VBZ	Verb 3rd person singular present
33.	WDT	Wh-determiner
34.	WP	Wh-pronoun
35.	WP$	Possessive wh-pronoun
36.	WRB	Wh-adverb

Tabel 8.1: Penn Treebank POS Tag.

kata sangat sulit untuk didaftar/sangat banyak). Teori Bayes digunakan untuk melakukan aproksimasi permasalahan ini. Ingat kembali teori Bayes seperti pada persamaan 8.7.

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)} \tag{8.7}$$

Dengan menggunakan teori Bayes, kita dapat mentransformasi persamaan 8.6 menjadi persamaan 8.8.

$$\hat{c_1}, \hat{c_2}, \ldots, \hat{c_T} = \underset{c_1, c_2, \ldots, c_T; c_i \epsilon C}{\arg\max} \frac{P(w_1, w_2, \ldots, w_T | c_1, c_2, \ldots, c_T) P(c_1, c_2, \ldots, c_T)}{P(w_1, w_2, \ldots, w_T)} \tag{8.8}$$

Untuk suatu sekuens input, $P(w_1, w_2, \ldots, w_T)$ (*language model*) akan selalu sama sehingga dapat diabaikan (karena operasi yang dilakukan adalah mengubah-ubah atau mencari **c**). Oleh karena itu, persamaan 8.8 dapat disederhanakan menjadi 8.9.

$$\hat{c_1}, \hat{c_2}, \ldots, \hat{c_T} = \underset{c_1, c_2, \ldots, c_T; c_i \epsilon C}{\arg\max} P(w_1, w_2, \ldots, w_T | c_1, c_2, \ldots, c_T) P(c_1, c_2, \ldots, c_T) \tag{8.9}$$

Pada persamaan 8.9, kombinasi sekuens kelas kata jauh lebih sedikit dibanding kombinasi sekuens kata (karena kelas kata jumlahnya lebih terbatas). Ingat kembali $P(c_1, c_2, \ldots, c_T)$ disebut ***prior***, $P(w_1, w_2, \ldots, w_T | c_1, c_2, \ldots, c_T)$ disebut ***likelihood*** (bab 2).

Persamaan 8.9 masih dapat disederhanakan kembali menggunakan ***markov assumption***, yaitu dengan membuat asumsi saling lepas pada sekuens (disebut ***independence assumption***). Terdapat dua asumsi, pertama, kategori suatu kata hanya bergantung pada dirinya sendiri tanpa memperhitungkan kelas kata disekitarnya, seperti pada persamaan 8.10. Asumsi kedua adalah suatu kemunculan kategori kata hanya bergantung pada kelas kata sebelumnya, seperti pada persamaan 8.11.

$$P(w_1, w_2, \ldots, w_T | c_1, c_2, \ldots, c_T) = \prod_{i=1}^{T} P(w_i | c_i) \tag{8.10}$$

$$P(c_1, c_2, \ldots, c_T) = \prod_{i=1}^{T} P(c_i | c_{i-1}) \tag{8.11}$$

Dengan demikian, persamaan 8.9 disederhanakan kembali menjadi persamaan 8.12 yang disebut ***bigram assumption*** atau ***first-order markov chain***, dimana T melambangkan panjangnya sekuens.

$$\hat{c_1}, \hat{c_2}, \ldots, \hat{c_T} = \underset{c_1, c_2, \ldots, c_T; c_i \epsilon C}{\arg\max} \prod_{i=1}^{T} P(w_i | c_i) P(c_i | c_{i-1}) \tag{8.12}$$

Kita dapat membuat ekstensi persamaan 8.12 dengan ***trigram assumption***, ***quadgram assumption***, dan seterusnya. $P(c_i | c_{i-1}, c_{i-2})$ untuk *trigram*, $P(c_i | c_{i-1}, c_{i-2}, c_{i-3})$ untuk *quadgram*. Walau menghitung probabilitas seluruh sekuens adalah hal yang susah, hal tersebut dapat dimodelkan dengan

lebih baik menggunakan *recurrent neural network* (subbab 13.2). Anda dapat membaca subbab tersebut kemudian, ada baiknya kita mengerti pendekatan yang lebih sederhana terlebih dahulu.

Sebagai contoh, untuk kalimat "*budi menendang bola*", peluang kalimat tersebut memiliki sekuens kelas kata "*noun*, *verb*, *noun*" adalah

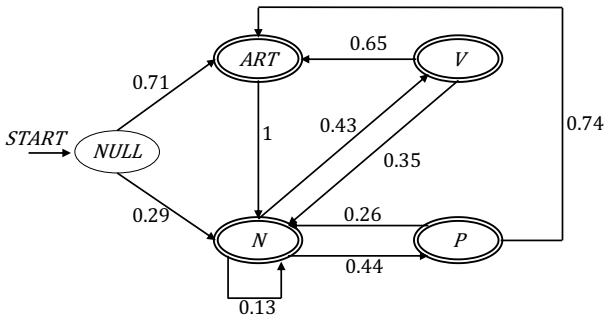
$$P(noun, verb, noun) = P(budi \mid noun)P(noun \mid null)P(menendang \mid verb) \\ P(verb \mid noun)P(bola \mid noun)P(noun \mid verb) \tag{8.13}$$

8.4 Hidden Markov Model Tagger

Pada subbab sebelumnya, POS *tagging* telah didefinisikan secara matematis. Kita sudah mengetahui permasalahan yang ingin kita selesaikan. Subbab ini adalah formalisasi *hidden markov model tagger*.

Ingat kembali persamaan 8.12 untuk POS tagging. $P(w_i|c_i)$ disebut *likelihood* dan $P(c_i|c_{i-1})$ disebut *prior*, *multiplication of probabilities* ($\prod$) melambangkan *markov chain*. ***Markov chain*** adalah kasus spesial *weighted automaton*[5] yang mana sekuens *input* menentukan *states* yang akan dilewati oleh automaton. Sederhananya, automaton mencapai *goal state* setelah mengunjungi berbagai *states*. Total bobot *outgoing edges* untuk masing-masing *state* pada automaton haruslah bernilai satu apabila dijumlahkan. Kasus spesial yang dimaksud adalah *emission* (dijelaskan kemudian).

Sebagai contoh, perhatikan Tabel 8.2. *ART* adalah *article*, *N* adalah *noun*, *V* adalah *verb* dan *P* adalah *preposition*. Mereka adalah contoh kelas kata yang disederhanakan demi membuat contoh yang mudah. Tabel 8.2 yang merepresentasikan probabilitas transisi kelas kata, ketika dikonversi menjadi *weighted automaton*, akan menjadi Gambar 8.5.



Gambar 8.5: *Weighted automaton* [38].

[5] Kami berasumsi kamu sudah mempelajari automata sebelum membaca buku ini.

Bigram	**Estimate**
$P(ART \mid null)$	0.71
$P(N \mid null)$	0.29
$P(N \mid ART)$	1
$P(V \mid N)$	0.43
$P(N \mid N)$	0.13
$P(P \mid N)$	0.44
$P(N \mid V)$	0.35
$P(ART \mid V)$	0.65
$P(ART \mid P)$	0.74
$P(N \mid P)$	0.26

Tabel 8.2: Probabilitas bigram [38].

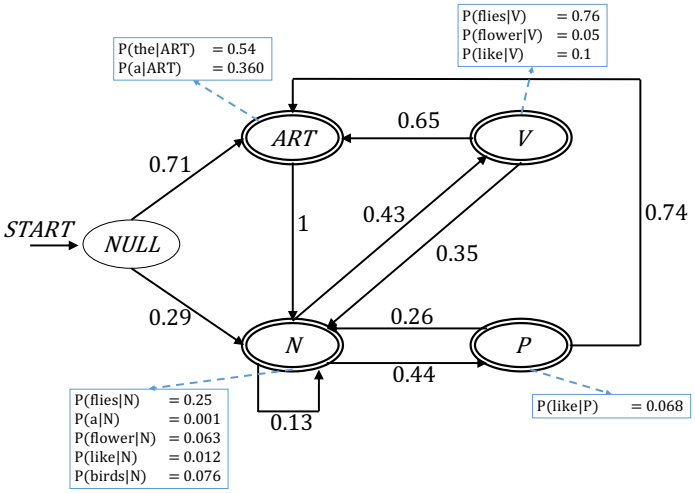
Tabel 8.2 dan Gambar 8.5 telah merepresentasikan probabilitas *prior*, sekarang kita ingin model yang kita punya juga mencakup ***lexical emission probabilities***, yaitu *likelihood* pada persamaan 8.12.

$P(the \mid ART)$	0.54	$P(a \mid ART)$	0.360
$P(flies \mid N)$	0.025	$P(a \mid N)$	0.001
$P(flies \mid V)$	0.076	$P(flower \mid N)$	0.063
$P(like \mid V)$	0.1	$P(flower \mid V)$	0.05
$P(like \mid P)$	0.068	$P(birds \mid N)$	0.076
$P(like \mid N)$	0.012		

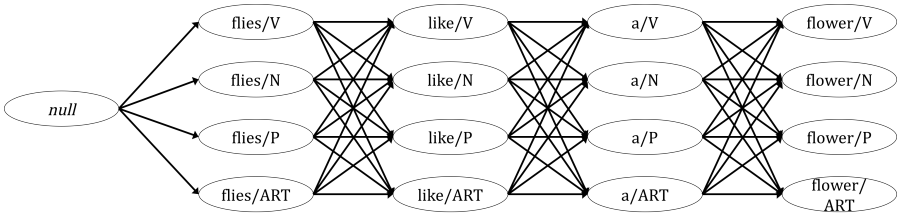
Tabel 8.3: *Lexical emission probabilities* [38]. Tabel ini hanya memuat informasi yang dibutuhkan untuk ilustrasi cerita (penyederhanaan).

Seumpama kita mempunyai *lexical emission probabilities* seperti pada Tabel 8.3. Setiap *state* pada automaton, dapat menghasilkan/meng-*output*-kan suatu kata (*word*) dengan probabilitas pada Tabel 8.3. Kita kembangkan lagi Gambar 8.5 dengan tambahan informasi *lexical emission probabilities* menjadi Gambar 8.6. Automaton ini disebut **hidden markov model** (HMM). Kata *hidden* berarti, untuk setiap kata pada sekuens, kita tidak mengetahui kata tersebut dihasilkan oleh *state* mana secara model (baru diketahui saat *running*). Misalkan, kata *flies* dapat dihasilkan oleh *state* N (*noun*) atau V (*verb*) [38].

Diberikan kalimat "*flies like a flower*", untuk menghitung sekuens kelas kata untuk kalimat tersebut, kita menyelusuri automaton Gambar 8.6. Hasil penelusuran memberikan kita kombinasi sekuens yang mungkin seperti pada Gambar 8.7. Pekerjaan berikutnya adalah, dari seluruh kombinasi sekuens yang mungkin, yaitu $\prod_{i=1}^{T} P(w_i|c_i)P(c_i|c_{i-1})$. Bagaimana cara kita menen-



Gambar 8.6: *Hidden Markov Model*.



Gambar 8.7: Sekuens yang mungkin (*brute force*).

tukan sekuens terbaik (paling optimal), yaitu sekuens dengan probabilitas tertinggi, diberikan pada subbab berikutnya.

Secara formal, ***hidden markov model tagger*** didefinisikan oleh beberapa komponen [12]:

1. $Q = \{q_1, q_2, \ldots, q_S\}$ yaitu himpunan ***states***; S menunjukkan banyaknya *states*.
2. $\mathbf{A} = a_{0,0}, a_{1,1}, a_{2,2}, \ldots, a_{S,S}$ yaitu **transition probability matrix** dari suatu *state* i menuju *state* j; dimana $\sum_{j=0}^{S} a_{i,j} = 1$. Indeks 0 merepresentasikan *start state* (*null state*). S melambangkan banyaknya *states*.
3. $\mathbf{o} = o_1, o_2, \ldots, o_T$ yaitu sekuens **observasi** (kata/*input*); T adalah panjang *input*.
4. $\mathbf{b} = b_i(o_w)$ yaitu sekuens dari *observation likelihood*, atau disebut dengan *emission probabilities*, merepresentasikan sebuah observasi kata o_w dihasilkan oleh suatu *state-i*.

5. q_0, q_F yaitu kumpulan *state* spesial yang terdiri dari **start state** dan **final state(s)**.

8.5 Algoritma Viterbi

Pada subbab sebelumnya, telah didefinisikan permasalahan POS *tagging* dan *Hidden Markov Model* untuk menyelesaikan permasalahan tersebut. Pada bab ini, kamu akan mempelajari cara mencari sekuens *syntactical categories* terbaik diberikan suatu observasi kalimat menggunakan **algoritma Viterbi**. Hal ini disebut proses ***decoding*** pada HMM. Algoritma Viterbi adalah salah satu algoritma *dynamic programming* yang prinsip kerjanya mirip dengan *minimum edit distance*.[6] Ide utama algoritma Viterbi adalah mengingat sekuens untuk setiap posisi tertentu (setiap iterasi, setiap panjang kalimat). Apabila kita telah sampai pada kata terakhir, kita lakukan *backtrace* untuk mendapatkan sekuens terbaik.

---

function VITERBI(*observations* of len T, *state-graphs* of len S)

Initialization Step
create a path of probability matrix viterbi[S,T]
for each state s **from** 1 **to** S **do**
  viterbi[s,1] $\leftarrow a_{0,s} \times b_s(o_1)$
  backpointer[s,1] $\leftarrow 0$

Iteration Step
for each time step t **from** 2 **to** T **do**
  for each state s **from** 1 **to** S **do**
    viterbi[s,t] $\leftarrow \arg\max_{j=1,S}$ *viterbi*$[j, t-1] \times a_{s,j} \times b_s(o_t)$
    backpointer[s,t] $\leftarrow$ index of j that gave the max above

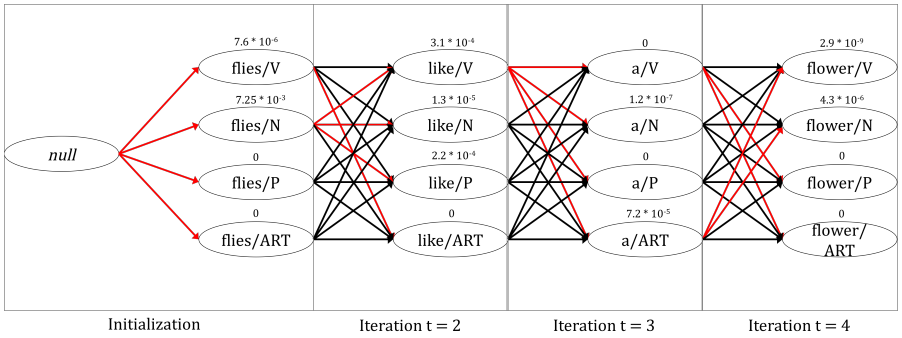
Sequence Identification Step
$c_T \leftarrow i$ that maximizes viterbi[i, T]
for $i = T - 1$ **to** 1 **do**
  $c_i \leftarrow$ backpointer[c_{i+1},$i+1$]

Gambar 8.8: Algoritma Viterbi [12, 38].

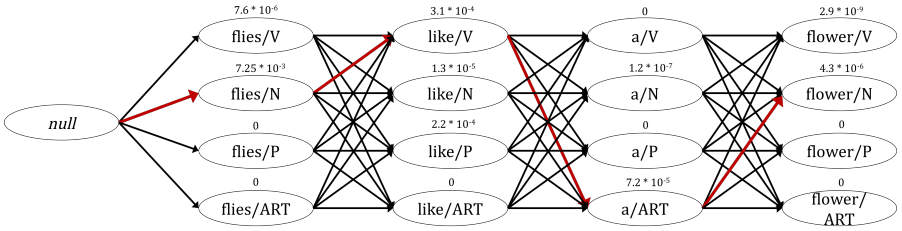
Perhatikan Gambar 8.8 yang menunjukkan *pseudo-code* untuk algoritma Viterbi. Variabel c berarti kelas kata, a adalah *transition probability*, dan b adalah *lexical-generation probability*. Pertama-tama, algoritma tersebut membuat suatu matriks berukuran SxT dengan S adalah banyaknya *states* (tidak

[6] https://en.wikipedia.org/wiki/Edit_distance

termasuk *start state*) dan T (*time*) adalah panjang sekuens. Pada setiap iterasi, kita pindah ke observasi kata lainnya. Gambar 8.9 adalah ilustrasi algoritma Viterbi untuk kalimat *input* (*observed sequence*) "*flies like a flower*" dengan *lexical generation probability* pada Tabel 8.3 dan *transition probabilities* pada Tabel 8.2 (bigram) [38]. Panah berwarna merah melambangkan *backpointer* yaitu *state* mana yang memberikan nilai tertinggi untuk ekspansi ke *state* berikutnya. Setelah *iteration step* selesai, kita lakukan *backtrace* terhadap *state* terakhir yang memiliki nilai probabilitas tertinggi dan mendapat hasil seperti pada Gambar 8.10. Dengan itu, kita mendapatkan sekuens "flies/N like/V a/ART flower/N".



Gambar 8.9: Ilustrasi algoritma Viterbi per iterasi. Panah berwarna merah melambangkan *backpointer–state* mana yang memberikan nilai tertinggi untuk ekspansi.



Gambar 8.10: Viterbi *backtrace*.

Apabila kamu hanya ingin mengetahui HMM tanpa variabel yang perlu dilatih/diestimasi, kamu dapat berhenti membaca sampai subbab ini. Apabila kamu ingin mengetahui bagaimana HMM dapat mengestimasi parameter, kamu dapat melanjutkan membaca subbab berikutnya.

8.6 Proses Training Hidden Markov Model

Hidden Markov Model (HMM) adalah salah satu varian *supervised learning*,[7] diberikan sekuens *input* dan *output* yang bersesuaian sebagai *training data*. Pada kasus POS *tagging*, yaitu *input*-nya adalah sekuens kata dan *output*-nya adalah sekuens kelas kata (masing-masing kata/*token* berkorespondensi dengan kelas kata). Saat melatih HMM, kita ingin mengestimasi parameter **A** dan **b** yaitu *transition probabilities* dan *emission probabilities/lexical-generation probabilities* (ingat kembali definisi HMM secara formal pada subbab 8.4). Kita melatih HMM dengan menggunakan **Algoritma Forward-Backward** (**Baum-Welch Algorithm**).

Cara paling sederhana untuk menghitung *emission probabilities* atau *transition probabilities* adalah dengan menghitung kemunculan pada sampel (*training data*). Sebagai contoh, *emission probability* suatu kata untuk setiap kelas kata diberikan pada persamaan 8.14, dimana N melambangkan banyaknya kemunculan kata w_i terlepas dari kelasnya.

$$P(w_i|c_i) = \frac{count(w_i, c_i)}{\sum_{j=1}^{N} count(w_i, c_j)} \tag{8.14}$$

Akan tetapi, perhitungan tersebut mengasumsikan *context-independent*, artinya tidak mempedulikan keseluruhan sekuens. Estimasi lebih baik adalah dengan menghitung seberapa mungkin suatu kategori c_i pada posisi tertentu (indeks kata/*token* pada kalimat) pada semua kemungkinan sekuens, diberikan input $w_1, w_2, \ldots, w_T$. Kami ambil contoh, kata *flies* sebagai *noun* pada kalimat "*The flies like flowers*", dihitung sebagai penjumlahan seluruh sekuens yang berakhir dengan *flies* sebagai *noun*. Probabilitas $P(flies/N \mid The\ flies) = \frac{P(flies/N\ \&\ The\ flies)}{P(The\ flies)}$.

Agar lebih *precise*, secara formal, kita definisikan terlebih dahulu ***forward probability*** sebagai 8.15, dimana $\alpha_i(t)$ adalah probabilitas untuk menghasilkan kata $w_1, w_2, \ldots, w_t$ dengan w_t dihasilkan (*emitted*) oleh c_i.

$$\alpha_i(t) = P(w_t/c_i \mid w_1, w_2, \ldots, w_t) \tag{8.15}$$

Pseudo-code perhitungan kemunculan kelas c_i sebagai kategori pada posisi tertentu diberikan oleh Gambar 8.11, dengan c_i adalah kelas kata ke-i dan w_i adalah kata ke-i, a adalah *transition probability*, b adalah *emission probability*, dan S melambangkan banyaknya *states*.

Sekarang, kita definisikan juga ***backward probability*** $\beta_i(t)$, yaitu probabilitas untuk menghasilkan sekuens $w_t, \ldots, w_T$ dimulai dari *state* w_t/c_i (c_i menghasilkan w_t). Hal ini serupa dengan *forward probability*, tetapi *backward probability* dihitung dari posisi ke-t sampai ujung akhir (t ke T). Anda dapat melihat *pseudo-code* pada Gambar 8.12, dengan a adalah *transition probability* dan b adalah *emission probability*.

[7] Walaupun ia termasuk *generative model*. tetapi komponen utama yang dimodelkan adalah $p(y \mid x)$

Initialization Step
for $i = 1$ **to** S **do**
  $\alpha_i(t) \leftarrow b_i(o_1) \times a_{0,i}$

Comparing the Forward Probabilities
for $t = 2$ **to** T **do**
  for $i = 1$ **to** S **do**
    $\alpha_i(t) \leftarrow \sum_{j=1}^{S}(a_{ji} \times \alpha_j(t-1)) \times b_i(o_t)$

Gambar 8.11: Algoritma *forward* [38].

Initialization Step
for $i = 1$ **to** S **do**
  $\beta_i(T) \leftarrow P(c_i)$ # assigned using a particular class c_i

Comparing the Backward Probabilities
for $t = T - 1$ **to** t **do**
  for $i = 1$ **to** S **do**
    $\beta_i(t) \leftarrow \sum_{j=1}^{S}(a_{ji} \times \beta_i(t+1)) \times b_j(o_{j+1})$

Gambar 8.12: Algoritma *backward* [38].

Gabungan *forward* dan *backward probability* dapat digunakan untuk mengestimasi $\gamma_j(t)$ yaitu probabilitas berada pada *state* c_j pada waktu ke-t dengan persamaan 8.16.

$$\gamma_j(t) = \frac{\alpha_i(t) \times \beta_i(t)}{\sum_{j=1}^{S} \alpha_j(t) \times \beta_j(t)} \tag{8.16}$$

Kita mengestimasi probabilitas keberadaan pada *state* tertentu, berdasarkan pengaruh probabilitas keseluruhan sekuens.

Dengan menggunakan *forward probability* dan *backward probability* sekaligus, kita definisikan $\xi_t(i,j)$ yaitu probabilitas berada di *state-i* pada waktu ke t dan *state-j* pada waktu ke-$(t+1)$ dengan persamaan 8.17.

$$\xi_t(i,j) = \frac{\alpha_i(t) \times a_{ij} \times b_j(o_{t+1}) \times \beta_j(t+1))}{\alpha_S(T)} \tag{8.17}$$

Dengan a_{ij} adalah *transition probability* dan $b_j(o_{t+1})$ adalah *emission probability* (ingat kembali definisi formal HMM pada subbab 8.4). Pada setiap iterasi, kita ingin memperbaharui kembali parameter HMM yaitu **A** dan **b**. Kita hitung kembali *transition probability* (nilai yang lama di-*update*), diberikan oleh persamaan 8.18.

$$a_{ij}{}' = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \sum_{j=1}^{S} \xi_t(i,j)} \tag{8.18}$$

Kita juga menghitung kembali *emission probability* (nilai yang lama di-*update*), diberikan oleh persamaan 8.19.

$$b_j(o_k)' = \frac{\sum_{t=1,o_k=w_k}^{T} \gamma_j(t)}{\sum_{t=1}^{T} \gamma_j(t)} \tag{8.19}$$

$\sum_{t=1,o_k=w_k}^{T}$ berarti jumlah observasi w_k pada waktu t.

Initialize A and b
Iterate until convergence
E-step

$\gamma_j(t) = \frac{\alpha_i(t) \times \beta_i(t)}{\sum_{j=1}^{S} \alpha_j(t) \times \beta_j(t)}$

$\xi_t(i,j) = \frac{\alpha_i(t) \times a_{ij} \times b_j(o_{t+1}) \times \beta_j(t+1))}{\alpha_S(T)}$

M-step

update $a_{ij}{}' = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \sum_{j=1}^{S} \xi_t(i,j)}$

update $b_j(o_k)' = \frac{\sum_{t=1,o_k=w_k}^{T} \gamma_j(t)}{\sum_{t=1}^{T} \gamma_j(t)}$

Gambar 8.13: Algoritma *forward-backward* (EM) [12].

Keseluruhan proses ini adalah cara melatih HMM dengan menggunakan kerangka berpikir ***Expectation Maximization***: terdiri dari **E-step** dan **M-step** [12]. Pada E-step, kita mengestimasi probabilitas berada di suatu *state* c_j menggunakan $\gamma_j(t)$ dan mengestimasi transisi $\xi_t(i,j)$ berdasarkan parameter **A** dan **b** yang sudah diberikan pada tahap iterasi *training* (*epoch*) sebelumnya. Pada M-step, kita menggunakan γ dan ξ untuk mengestimasi kembali parameter **A** dan **b**. Hal ini dijelaskan secara formal pada Gambar 8.13. Walaupun HMM menggunakan *independence assumption*, tetapi kita dapat mengikutsertakan **pengaruh probabilitas keseluruhan sekuens untuk perhitungan probabilitas keberadaan kita pada suatu *state* i pada saat (*time* t) tertentu**. Metode ini dapat dianggap sebagai suatu cara optimalisasi menggunakan *smoothing*[8] terhadap nilai parameter (**A** dan **b**). Kita mencapai titik *local optimal* apabila tidak ada perubahan parameter.

[8] https://en.wikipedia.org/wiki/Smoothing

Kami ingin mengakui bahwa penjelasan pada subbab 8.6 mungkin kurang baik (kurang memuaskan). Kami harap kamu mencari referensi lain yang lebih baik untuk subbab ini.

Soal Latihan

8.1. Data Numerik
Pada bab ini, diberikan contoh aplikasi Hidden Markov Model (HMM) untuk POS *tagging*, dimana data kata adalah data nominal. Berikan strategi penggunaan HMM untuk data numerik! Misal, pada *automatic speech recognizer*.

8.2. Ekstensi Algoritma Viterbi
Buatlah ekstensi algoritma Viterbi untuk asumsi trigram!

8.3. Maximum Entropy Markov Model

(a) Jelaskan konsep *maximum entropy*!
(b) Jelaskan *maximum entropy markov model*!

8.4. Gibbs Sampling

(a) Jelaskan bagaimana Gibbs sampling digunakan pada HMM!
(b) Jelaskan penggunaan variasi/ekstensi Gibbs sampling pada HMM!

8.5. Latent Dirichlet Allocation
Salah satu materi yang berkaitan erat dengan HMM adalah *Latent Dirichlet Allocation* (LDA) yang merupakan anggota keluarga *graphical model*. Jelaskan apa itu LDA serta bagaimana cara kerja LDA!

9

Seleksi Fitur dan Metode Evaluasi

> "The key to artificial intelligence has always been the representation."
>
> Jeff Hawkins

Bab ini membahas beberapa tips dan trik yang sebenarnya sudah disinggung pada bab 3 dan bab 5. Hanya saja, beberapa hal lebih baik dijelaskan secara lebih mendalam ketika sudah mempelajari beberapa algoritma pembelajaran mesin, seperti pentingnya *feature engineering*, *feature selection* dan penjelasan lebih lanjut *cross validation*. Kamu dapat menganggap bab ini sebagai kelanjutan tips dan trik pembelajaran mesin lanjutan bab 3 dan bab 5.

9.1 Feature Engineering

Record (data) pada pembelajaran mesin pada umumnya dikonversi menjadi suatu vektor (*feature vector*) yang merepresentasikannya dalam bentuk matematis. Fitur-fitur biasanya tersusun atas atribut yang kita anggap memiliki pengaruh terhadap *output*. Sebagai contoh, tekanan darah diprediksi berdasarkan usia, jenis kelamin dan BMI. Seringkali, seseorang membutuhkan keahlian suatu bidang agar dapat memilih fitur yang tepat. Proses untuk mencari (atau me-*list*) kandidat fitur, disebut sebagai aktivitas ***feature engineering***.

Seperti kata mutiara yang kami berikan pada awal bab ini, kunci pembelajaran mesin adalah representasi permasalahan. Fitur yang dipilih untuk merepresentasikan data adalah bagaimana cara kamu merepresentasikan masalah juga. Karena membutuhkan tingkat keahlian tertentu, proses memilih fitur tidaklah mudah. Bisa jadi (sering), orang memilih fitur yang tidak representatif! Dengan demikian, tidak heran apabila seseorang mempublikasikan makalah ilmiah atas kontribusinya menentukan fitur baru pada domain tertentu.

Konon tetapi, proses pemilihan fitur yang bersifat manual ini cukup bahaya karena rentan dengan *bias*, yaitu kemampuan seseorang pada domain tertentu. Alangkah baiknya, apabila kita dapat memilih fitur yang memang benar-benar diperlukan (murni) secara otomatis. Hal tersebut akan dibahas lebih lanjut pada materi *artificial neural network*. Hal ini salah satu alasan yang membuatnya populer.

9.2 High Dimensional Data

Pada kenyataan, fitur yang kita gunakan untuk membangun model pembelajaran mesin tidaklah sesederhana yang dicontohkan pada subbab sebelumnya. Seringkali, kita berhadapan dengan data yang memiliki sangat banyak fitur (*high dimensional data*). Sebagai contoh, seorang *marketing analyst* mungkin saja ingin mengerti pola seseorang berbelanja pada *online shop*. Untuk mengerti pola tersebut, ia menganalisis seluruh kata kunci pencarian (*search term*) *item*. Misalnya, seseorang yang ingin membeli meja makan juga mungkin akan membeli kursi (sebagai satu paket). Data pada analisis semacam ini berdimensi besar. Seberapa besar dimensi yang dapat disebut *high dimension* adalah hal yang relatif.

Sayangnya, data dengan dimensi yang sangat besar membawa beberapa masalah pada pembelajaran mesin. Pertama, model pembelajaran susah untuk memiliki kinerja yang optimal pada data berdimensi tinggi. Semakin banyak fitur yang dipakai, semakin kompleks suatu model pembelajaran mesin harus memodelkan permasalahan. Berhubung kita memiliki banyak fitur, *search space* untuk mencari konfigurasi parameter optimal sangatlah luas. Hal ini dapat dianalogikan seperti mencari seseorang pada gedung berlantai satu vs. gedung berlantai 20. Kedua, hal ini menyebabkan mudah terjadi *overfitting* karena ada sangat banyak konfigurasi fitur walaupun kita hanya memiliki data yang terbatas.[1] Ketiga, data dengan dimensi yang besar susah untuk diproses secara komputasi (*computationally expensive*), baik dari segi memori dan waktu. Karenanya hal ini, kita ingin agar fitur-fitur yang kita gunakan sesedikit mungkin. Dengan kata lain, kita ingin representasi permasalahan sesederhana mungkin dari sesi memori dan *computational processing*.

9.3 Feature Selection

Pada pembelajaran mesin, pada umumnya kita menggunakan banyak (lebih dari satu) fitur. Artinya kita merepresentasikan setiap *record* (*instance*) atau *input* sebagai suatu vektor $\mathbf{x} \in \mathbb{R}^{1 \times F}$; dimana F melambangkan dimensi vektor atau banyaknya fitur. Seringkali, F bernilai besar sehingga model yang kita miliki kompleks. Kita tentunya ingin mengurangi kompleksitas dengan

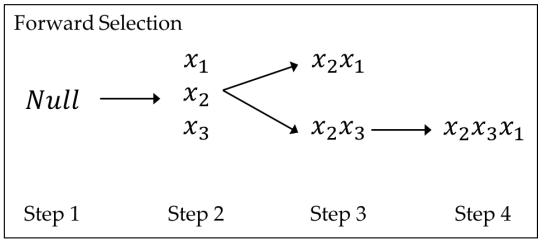
[1] Curse of Dimensionality

alasan-alasan yang sudah disebutkan pada subbab 9.2. Alasan lainnya karena belum tentu semua fitur berguna. Cara termudah adalah dengan menghapus fitur yang memiliki nilai varians = 0. Sayang sekali, hal ini tidak selalu terjadi. Subbab ini membahas teknik-teknik yang dapat digunakan untuk menyederhanakan fitur (mengurangi dimensi input).

9.3.1 Subset Selection (Feature Ablation)

Cara paling intuitif untuk mencari tahu kombinasi fitur terbaik adalah dengan mencoba seluruh kombinasi fitur. Misal kita mempunyai fitur sebanyak F, kita bisa pilih untuk menggunakan atau tidak menggunakan masing-masing fitur, menghasilkan kombinasi sebanyak 2^F. Dari keseluruhan kombinasi tersebut, kita pilih suatu kombinasi fitur yang memerikan kinerja terbaik. Akan tetapi, metode *brute force* ini terlalu memakan waktu. Kita dapat juga menggunakan teknik *greedy* yaitu *forward selection* dan *backward selection*. ***Forward*** dan ***backward selection*** sering juga disebut sebagai ***feature ablation***.

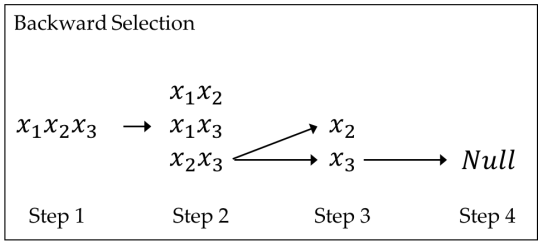
Pada *forward selection*, kita mulai dengan suatu model yang tidak menggunakan fitur apapun sama sekali, yaitu meng-*assign* kelas yang paling sering muncul di dataset, pada *input*. Setelah itu, kita tambahkan satu per satu fitur pada setiap langkah. Langkah berikutnya, kita gunakan satu fitur. Diantara F pilihan fitur, kita cari fitur yang memberi nilai terbaik. Kemudian, pada tahap berikutnya, kita kombinasikan fitur yang kita pilih pada langkah sebelumnya dengan fitur yang tersisa. Hal ini terus diulang sampai kita sudah menggunakan seluruh fitur pada model kita. Untuk mencari model terbaik, kita hanya perlu mencari kombinasi fitur yang memberikan nilai kinerja terbaik. *Forward selection* diilustrasikan pada Gambar 9.1. Dibanding *brute force* yang bersifat eksponensial, *forward selection* membentuk suatu deret aritmatika kombinasi fitur yang dicoba, yaitu $F + (F-1) + \cdots + 1 = F(F+1)/2$. Apabila kita memiliki fitur sebanyak $F = 10$, kombinasi *brute force* menghasilkan $1,024$ kombinasi, sementara *forward selection* hanya sebanyak 45.



Gambar 9.1: Ilustrasi *forward selection* untuk tiga fitur.

Backward selection adalah kebalikan dari *forward selection*. Apabila pada *forward selection*, kita menambahkan satu fitur tiap langkah, *backward selec-*

tion mengurangi satu fitur pada tiap langkah. Ilustrasi diberikan pada Gambar 9.2. Seperti *forward selection*, *backward selection* juga hanya mencoba sebanyak $F(F+1)/2$ kombinasi. Kita juga dapat menggabungkan *forward* dan *backward selection* menjadi metode *hybrid*, yaitu menambah satu fitur pada tiap langkah, serta memperbolehkan untuk menghilangkan fitur juga [20].



Gambar 9.2: Ilustrasi *backward selection* untuk tiga fitur.

9.3.2 Shrinkage

Ingat kembali materi bab 5 tentang *regularization*. Seperti yang sudah dijelaskan, kita ingin agar model kita sesederhana mungkin. Mengurangi dimensi fitur adalah cara mengurangi kompleksitas. Ingat kembali, dengan menggunakan suatu fungsi regularisasi, objektif pembelajaran adalah meminimalkan *loss* dan kompleksitas, seperti pada persamaan 9.1. Kompleksitas model dapat dihitung menggunakan L_2 (Ridge, persamaan 9.2) atau L_1 (Lasso, persamaan 9.3) norm. Karena kita ingin meminimalkan norm, artinya kita juga membuat parameter model pembelajaran mesin bernilai dekat dengan nol. Pada metode Ridge, kita ingin meminimalkan fungsi eksponensial, sementara fungsi skalar pada Lasso. Artinya, Lasso lebih cenderung untuk menghasilkan suatu model yang bersifat *sparse*. Dengan kata lain, Lasso melakukan *subset feature selection* seperti yang sudah dijelaskan pada subbab sebelumnya. Sementara itu, Ridge cenderung tidak meng-nol-kan parameter, melainkan hanya **dekat** dengan nol [20]. Kamu mungkin berpikir, kenapa kita tidak menggunakan Lasso saja, berhubung ia mengeleminasi fitur. Pada metode Ridge, semua fitur tetap digunakan walaupun nilainya diturunkan (*shrink*) agar dekat dengan nol. Hal ini, walaupun tidak mengeleminasi fitur, dapat mengurangi variasi kinerja model.[2] (dijelaskan pada subbab 9.5)

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \mathcal{L}(\mathbf{w}) + \lambda \mathrm{R}(\mathbf{w}) \tag{9.1}$$

$$\mathrm{R}(\mathbf{w}) = \|\mathbf{w}\|_2^2 = \sum_i (w_i)^2 \tag{9.2}$$

[2] baca buku [20] untuk pembuktiannya

$$\mathrm{R}(\mathbf{w}) = \|\mathbf{w}\| = \sum_i |w_i| \tag{9.3}$$

Secara singkat, Ridge digunakan untuk menghasilkan model yang varian kinerjanya kecil. Sementara itu, Lasso digunakan untuk menghasilkan model yang mudah dimengerti (*interpretability*) dengan mengeliminasi fitur.

9.3.3 Principal Components Analysis (Dimensionality Reduction)

Teknik-teknik sebelumnya berfokus pada cara mengeleminasi fitur. Akan tetapi, penghapusan suatu fitur berpotensi pada model yang tidak mampu mengerti kompleksitas permasalahan. Dengan kata lain, *oversimplification*. Dibanding menghapus fitur, cara lain untuk mengurangi kompleksitas komputasi adalah mentransformasi data ke dalam dimensi lebih kecil. Untuk *input* yang memiliki F fitur, kita kurangi dimensi *input* menjadi $M < F$ (*dimensionality reduction*). Apabila kita mampu mengurangi dimensi *input*, maka kita juga dapat mengurangi jumlah parameter pada model pembelajaran mesin, yang artinya mengurangi kompleksitas komputasi dan meningkatkan *interpretability*.

Ide utama *dimensionality reduction* adalah mentransformasi data dari suatu *space* ke *space* lainnya, dimana data direpresentasikan dengan dimensi lebih kecil. Dengan catatan, data dengan dimensi lebih kecil harus mampu merepresentasikan karakteristik data pada dimensi aslinya! Dengan demikian, satu fitur pada dimensi yang baru mungkin memuat informasi beberapa fitur pada dimensi aslinya. Walaupun model yang kita hasilkan lebih sederhana secara jumlah parameter, tetapi kita membutuhkan usaha ekstra untuk mengerti representasi fitur-fitur pada dimensi yang baru.

Teknik yang digunakan untuk mengurangi dimensi adalah *principal component analysis*. Ide dasarnya adalah mencari bentuk data (pada dimensi lebih kecil) yang memiliki nilai varians tinggi untuk tiap fiturnya, karena varians yang tinggi berarti kemampuan fitur yang tinggi dalam melakukan diskriminasi (klasifikasi). Kita ingin mencari suatu arah (vektor, matriks, tensor) dimana data kita memiliki varians tertinggi. Pada dimensi yang baru, kita berharap data kita tersebar menjadi beberapa kelompok yang mudah dibedakan (*easily separable*). Arah ini dikenal sebagai **eigenvector**, dan nilai varians pada arah tersebut dikenal sebagai **eigenvalue**. Kami harap kamu ingat materi kuliah aljabar linear. Eigenvector yang memiliki nilai eigenvalue tertinggi disebut sebagai ***principal components***, yaitu semacam bentuk data yang ringkas. Selain mengurangi dimensi, teknik ini juga mengurangi (atau meniadakan) interaksi antar-fitur. Dengan demikian, kita dapat menggunakan *additive assumption* saat melakukan pembelajaran (ingat kembali materi bab 5).

Saat melakukan analisis ini, kita membuat suatu asumsi bahwa sejumlah *principal components* cukup untuk merepresentasikan variasi yang ada pada data [20]. Dengan kata lain, kita mengasumsikan arah (eigenvector) ketika

data memiliki varians tertinggi, adalah arah yang berasosiasi dengan *output*. *Singular Value Decomposition* adalah salah satu teknik untuk melakukan *principal component analysis*. Hal tersebut akan dibahas pada bab 12. Buku ini juga akan memberi contoh konkret *dimensionality reduction* dengan *artificial neural network* pada bab 12.

9.4 Evaluasi Kinerja Model

Buku ini telah menjelaskan bahwa model pembelajaran mesin mengoptimisasi *utility function* pada saat proses latihan (*training*). Pada umumnya, *utility function* berbentuk *cross entropy* atau *error function* untuk arsitektur *linear model* dan *neural network*. Untuk arsitektur lain, seperti *support vector machine*, kita mencari *support vectors* yang memberikan *margin* (*decision boundary*) terbaik untuk memisahkan dua kelas. Model *decision tree* dibangun berdasarkan *information gain* tiap atribut data. Sementara, model *Hidden Markov Model* mengoptimalkan *forward* dan *backward probability*. Kegiatan evaluasi model tidak berhenti saat *training* telah mencapai nilai *utility function* yang optimal, tetapi kita juga mengevaluasi kinerja model pada *validation data* (baik mengukur *utility function* dan *performance measure*). Terakhir, kita juga mengevaluasi prediksi model dengan mengukur *performance measure* pada *test data*. Subbab ini akan menjabarkan beberapa contoh *performance measure* untuk mengevaluasi hasil prediksi model pada kasus *supervised learning*.

Kelas A	Kelas B	Kelas C	Kelas D
0.2	0.6	0.1	0.1

Tabel 9.1: Contoh *class-assignment probability* pada *multi-class classification*.

Ingat kembali persoalan *multi-class classification*. Tujuan suatu model pembelajaran mesin adalah mengoptimalkan *class-assignment probability*, yaitu probabilitas masing-masing kelas untuk suatu (*given*) *input* (Tabel 9.1). Dengan meminimalkan *cross-entropy* saat proses latihan, kita harap model bisa memberikan nilai probabilitas yang tinggi untuk kelas yang tepat, dan nilai yang rendah untuk kelas lainnya (*high confidence*). Namun, urusan evaluasi model tidak hanya sampai di titik ini saja. *Class-assignment probability* berikutnya dikonversi menjadi keputusan final, yaitu memilih **satu** kelas mana yang tepat untuk *input* yang diberikan. Dengan kata lain, kita harus mengevaluasi penerjemahan "*raw*" *class-assignment probability* menjadi sebuah prediksi final. Pada kasus Tabel 9.1, *output* atau prediksi final dari model bersangkutan adalah "Kelas B".

Sebagai contoh lain, *support vector classifier* memberikan skor +1 atau −1 untuk *input*. Skor tersebut berkorespondensi dengan kelas A atau kelas B. *Output* final model pun adalah kelas yang tepat untuk *input* yang diberikan. Hal yang serupa terjadi pada *Naive Bayes*, *Linear Model*, *Decision Tree* dan *supervised learning model* lain yang dijelaskan pada buku ini (termasuk *neural network*). Walaupun model-model tersebut mengoptimasi *utility function* atau dikonstruksi dengan cara yang berbeda, prediksi final *multi-class* classification adalah kelas yang sesuai untuk *input*. Subbab ini membahas cara **mengevaluasi keputusan prediksi final model**.

9.4.1 Akurasi

Akurasi adalah *performance measure* paling sederhana dan sering digunakan saat mengevaluasi kinerja model. Akurasi didefinisikan sebagai proporsi prediksi yang benar dibagi dengan banyaknya sampel. Diberikan *desired output* (juga dikenal dengan "*gold standard*") $\mathbf{y}$ dan hasil prediksi $\hat{\mathbf{y}}$, kita menghitung proporsi banyaknya elemen yang sama antara $\mathbf{y}$ dan $\hat{\mathbf{y}}$. Secara matematis, akurasi diformulasikan pada persamaan 9.4, dimana N merepresentasikan banyaknya sampel.

$$\begin{aligned} \text{Akurasi} &= \frac{1}{N}\sum_{i=1}^{N} \text{verdict}_i \\ \text{verdict}_i &= \begin{cases} 1, & y_i = \hat{y}_i \\ 0, & y_i \neq \hat{y}_i \end{cases} \end{aligned} \tag{9.4}$$

Sebagai contoh, suatu model pembelajaran mesin mengklasifikan gambar buah menjadi "apel" dan "jeruk" (*binary classification*). *Desired output* dan prediksi model diberikan pada Tabel 9.2. Pada tabel ini, ada lima prediksi yang sama dengan *desired output*, yaitu pada sampel dengan ID=$[1, 2, 4, 8, 9]$. Dengan demikian, akurasi model adalah $\frac{5}{10} = 0.5$.

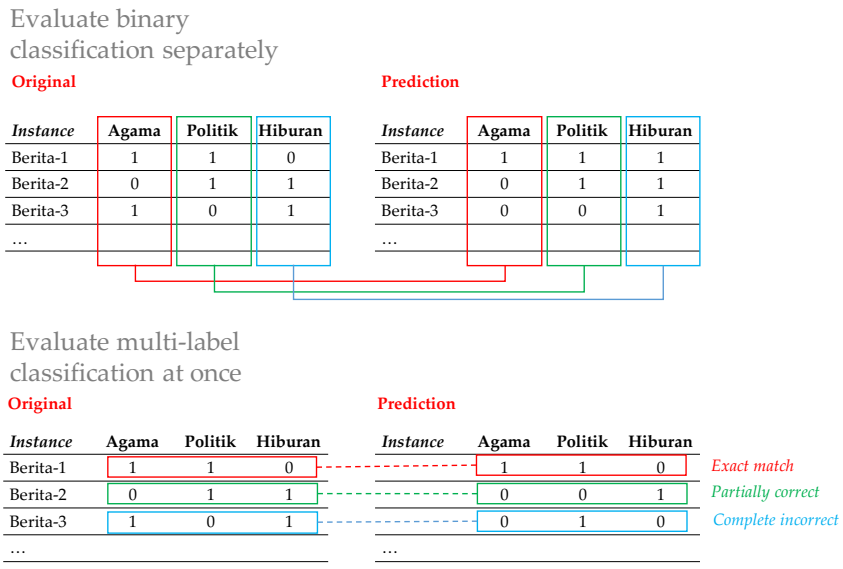
Perhitungan akurasi sama halnya untuk *multi-class classification*, tetapi menjadi semakin rumit untuk *multi-label classification*. Ingat, pada *multi-label classification*, suatu *input* dapat diklasifikasikan menjadi beberapa kelas. *Multi-label classification* dapat dievaluasi dengan dua cara. Pertama, mengevaluasi kinerja pada masing-masing kelas (seolah-olah kita memiliki beberapa *binary classifier*). Cara kedua adalah mengevaluasi prediksi *multi-label* secara sekaligus. Perbedaan kedua metode diilustrasikan pada Gambar 9.3. Saat mengevaluasi secara sekaligus, akurasi dapat dihitung sebagai proporsi banyaknya *exact-match* dibagi banyaknya sampel.

9.4.2 F1 Score

Metrik yang tidak kalah pentingnya adalah F1 score. F1 score untuk suatu kelas F1^k didefinisikan sebagai *harmonic mean* (persamaan 9.5) antara pre-

Instance	*Desired Output*	Hasil Prediksi
1	Jeruk	Jeruk
2	Jeruk	Jeruk
3	Jeruk	Apel
4	Apel	Apel
5	Jeruk	Apel
6	Apel	Jeruk
7	Jeruk	Apel
8	Apel	Apel
9	Apel	Apel
10	Jeruk	Apel

Tabel 9.2: Contoh hasil prediksi model.



Gambar 9.3: Cara mengevaluasi *multi-label classifier* (Gambar 5.8).

sisi (persamaan 9.6) dan recall (persamaan 9.7) kelas tersebut. Perhatikan, $\mathbf{y}^k$ (persamaan 9.7) melambangkan *desired output* untuk kelas k dan $\hat{\mathbf{y}}^k$ (persamaan 9.6) adalah prediksi untuk kelas k. Perhatikan, presisi dan recall berbeda dari sisi variabel pembagi. Presisi dibagi oleh banyaknya ($\|...\|$) prediksi pada kelas k, sementara recall dibagi oleh banyaknya prediksi **seharusnya** (*desired output*) pada kelas k.

$$\text{F1}^k = 2\frac{\text{Presisi}^k \times \text{Recall}^k}{\text{Presisi}^k + \text{Recall}^k} \tag{9.5}$$

$$\text{Presisi}^k = \frac{\text{Jumlah prediksi benar pada kelas } k}{\|\hat{\mathbf{y}}^k\|} \tag{9.6}$$

$$\text{Recall}^k = \frac{\text{Jumlah prediksi benar pada kelas } k}{\|\mathbf{y}^k\|} \tag{9.7}$$

Penjelasan sebelumnya cukup *high-level*, sekarang mari kita masuk ke ilustrasi nyata pada Tabel 9.2. Untuk memudahkan perhitungan, kita susun *confusion matrix* prediksi terlebih dahulu, seperti diilustrasikan pada Tabel 9.3. *Confusion matrix* amatlah berguna untuk analisis data, khususnya saat menghitung F1 score.

		Prediksi apel	Prediksi jeruk	$\|\mathbf{y}^k\|$
Gold	apel	3	1	4
	jeruk	4	2	6
	$\|\hat{\mathbf{y}}^k\|$	7	3	

Tabel 9.3: *Confusion matrix* prediksi apel dan jeruk berdasarkan Tabel 9.2.

Berikut adalah cara membaca *confusion matrix* pada Tabel 9.3:

- Ada 3 sampel yang diprediksi sebagai "apel" dengan benar.
- Ada 1 sampel yang seharusnya diprediksi sebagai "apel", tetapi diprediksi sebagai "jeruk".
- Ada 4 sampel yang seharusnya diprediksi sebagai "jeruk", tetapi diprediksi sebagai "apel".
- Ada 2 sampel yang diprediksi sebagai "jeruk" dengan benar.

Nilai $\|\hat{\mathbf{y}}^k\|$ untuk k="apel" adalah 7, sementara 3 untuk k="jeruk" (jumlah nilai pada tiap kolom). Nilai $\|\mathbf{y}^k\|$ adalah 4 dan 6 untuk k="apel" dan k="jeruk" (jumlah nilai pada tiap baris). Nilai presisi, recall, dan F1 score untuk masing-masing kelas diberikan pada Tabel 9.4.

Kategori	Presisi	Recall	F1 score
Apel	$\frac{3}{7}$=0.43	$\frac{3}{4}$=0.75	$2\frac{0.43\times0.75}{0.43+0.75}$=0.55
Jeruk	$\frac{2}{3}$=0.67	$\frac{2}{6}$=0.33	$2\frac{0.67\times0.33}{0.67+0.33}$=0.44

Tabel 9.4: Perhitungan nilai presisi, recall dan F1 score untuk masing-masing kelas untuk contoh Tabel 9.2.

Untuk mendapatkan nilai F1 secara "umum", kita merata-ratakan F1^{apel} dan F1^{jeruk}. Prosedur ini disebut ***macro-averaging***, yaitu merata-ratakan

nilai F1 pada masing-masing kelas. Hasilnya disebut sebagai "macro-averaged F1" atau disingkat "F1-macro". F1-macro untuk contoh pada Tabel 9.3 adalah 0.49.

Terdapat dua variasi lain, yaitu ***weighted-average*** (F1-weighted) dan ***micro-average*** (F1-micro). Pada F1-weighted, kita memberikan bobot untuk masing-masing kelas, berdasarkan banyaknya *instance* yang seharusnya diklasifikasikan ke kelas tersebut, i.e., banyaknya *desired output* untuk masing-masing kelas. F1-weighted untuk contoh Tabel 9.4 diberikan pada persamaan 9.8. Variasi ini berbeda dengan F1-macro yang menganggap bobot masing-masing kelas adalah sama.

$$\text{F1-weighted} = \frac{4 \times 0.55 + 6 \times 0.44}{10} = 0.48 \tag{9.8}$$

F1-macro dan F1-weighted memisahkan kinerja untuk masing-masing kelas. Variasi terakhir, F1-micro, melihat hasil prediksi secara keseluruhan tanpa pemisahan kinerja untuk masing-masing kelas. Sebagai contoh, nilai presisi dan recall secara keseluruhan diberikan pada persamaan 9.9 dan 9.10. Pada kasus ini, banyaknya prediksi sama dengan banyaknya prediksi seharusnya. Dengan demikian, nilai presisi = recall. F1-micro adalah nilai rata-rata presisi dan recall secara keseluruhan. Karenanya F1-micro = presisi = recall. Pada kasus *multi-class classification*, nilai F1-micro sama dengan akurasi.

$$\text{Precision} = \frac{\text{Jumlah prediksi benar}}{\text{Banyaknya prediksi}} = \frac{5}{10} \tag{9.9}$$

$$\text{Recall} = \frac{\text{Jumlah prediksi benar}}{\text{Banyaknya prediksi seharusnya}} = \frac{5}{10} \tag{9.10}$$

Ringkasan variasi F1 score diberikan pada Tabel 9.5. F1-macro memiliki keunggulan dibanding akurasi (F1-micro) dan F1-weighted pada kasus *imbalanced dataset*. Sebagai contoh, kita memiliki sebuah dataset untuk permasalahan *binary classification* dimana 90% *input* ditetapkan sebagai kelas A dan 10% sebagai kelas B. Apabila suatu model pembelajaran mesin memprediksi seluruh data ke kelas A, maka ia akan mendapatkan akurasi 90%. Pada kasus ini, sesungguhnya model pembelajaran mesin tidak mempelajari distribusi data dengan benar. Dalam artian, kamu dapat "tertipu" karena nilai akurasi yang besar, padahal model tersebut sebenarnya tidak memiliki performa yang baik. F1-weighted juga akan memberikan nilai yang cukup baik pada kasus ini. Apabila kita mengukur kinerja menggunakan F1-macro, maka kinerja model akan terlihat jauh lebih kecil. Cerita ini juga berkaitan dengan *overclaiming* (subbab 9.6). Akibat penggunaan atau interpretasi metrik yang kurang tepat, kita mengasumsikan model yang kita buat sudah memiliki kinerja yang cukup bagus.

Metrik	Deskripsi
F1-macro	Memisahkan kinerja untuk masing-masing kelas, kemudian merata-ratakan nilai kinerja.
F1-weighted	Memisahkan kinerja untuk masing-masing kelas, kemudian merata-ratakan nilai kinerja dimana setiap kelas memiliki bobot berdasarkan banyaknya *desired output* untuk tiap kelas.
F1-micro	Melihat hasil prediksi secara keseluruhan tanpa pemisahan kinerja untuk masing-masing kelas. Nilai F1-micro sama dengan akurasi pada *multi-class classification*.

Tabel 9.5: Penjelasan variasi F1 score.

9.4.3 Evaluasi Output Berstruktur

Kasus-kasus yang sudah diceritakan pada subbab ini adalah contoh kasus sederhana baik pada *binary*, *multi-class* dan *multi-label classification*. Diberikan sebuah *input* (e.g., teks, gambar, vektor, suara), kita ingin memprediksi satu atau beberapa kelas (dari beberapa opsi) yang sesuai untuk merepresentasikan *input*. Pada kasus yang disebutkan, hasil prediksi suatu *instance* adalah independen dari hasil prediksi untuk *instance* lainnya.

Sekarang, mari kita melanjutkan cerita ke kasus yang lebih kompleks, yaitu *output* berstuktur. Pada kasus ini, diberikan sebuah *input*, *output* model adalah sebuah data terstruktur, e.g., sekuens kategori, graf, teks.

Sekuens Kategori

Pada *output* dengan struktur berupa sekuens kategori $\mathbf{y}$, suatu elemen y_i pada sekuens bisa saja bergantung pada elemen lainnya, misal pada elemen sebelumnya. Sebagai contoh, pada persoalan Part-of-Speech (POS) *tagging* (yang sudah dijelaskan pada buku ini), diberikan *input* sekuens kata $\mathbf{x}$, kita ingin memprediksi kelas kata $\mathbf{y}$ yang bersesuaian (Gambar 9.4). Pada setiap iterasi i, model memprediksi kelas kata y_i yang cocok bagi *input* x_i. Tujuan model adalah memprediksi sekuens $\mathbf{y}$ paling optimal. Perhatikan, kelas kata pada posisi i (y_i) bisa jadi bergantung pada kelas kata di posisi $i-1$ (y_{i-1}), seperti yang sudah diceritakan pada buku ini. Artinya, suatu elemen *output* bergantung pada elemen *output* lainnya. Walaupun demikian, kita dapat mengevaluasi tiap elemen *output* secara independen. Misal, dengan menghitung akurasi apakah tiap kelas kata pada prediksi $\hat{y}_i$ memang sesuai dengan *desired output* y_i.

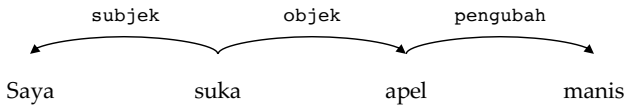
Graf

Evaluasi menjadi cukup kompleks apabila kita memprediksi *output* dengan struktur lebih rumit, misalnya sebuah graf $G \in (E, V)$; dimana E adalah kumpulan *edge* dan V adalah kumpulan *vertex*. Hal ini cukup lumrah pada

POS tag	Noun	Verb	Noun
Kata	Budi	Menendang	Bola

Gambar 9.4: Contoh POS *tagging* (Gambar 8.4).

bidang pemrosesan bahasa alami, sebagai contoh pada permasalahan *constituent parsing*, *dependency parsing*, dan *discourse parsing* (silakan di eksplorasi lebih lanjut). Gambar 9.5 memberikan contoh *dependency parsing*.



Gambar 9.5: Ilustrasi *dependency parsing*, dimana kita mencari hubungan antar-kata (direpresentasikan oleh *edge* pada graf). Sebagai contoh, "saya" adalah subjek dari "suka"; "apel" adalah objek dari "suka"; "manis" adalah kata pengubah "apel".

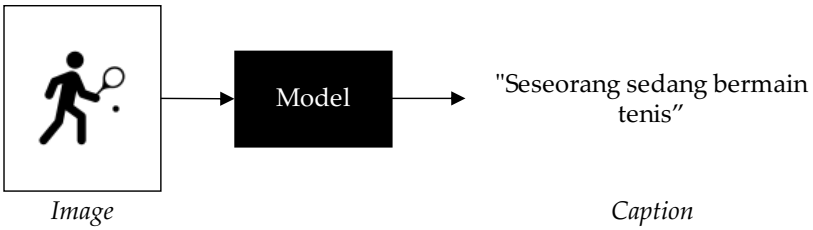
Kita dapat mengevaluasi suatu prediksi struktur dengan cara memfaktorkan struktur tersebut menjadi unit lebih kecil, kemudian mengevaluasi performa model pada satuan unit yang lebih kecil. Kinerja model pada satuan unit diagregat atau digabungkan menjadi satu skor untuk merepresentasikan keseluruhan performa dalam memprediksi suatu struktur [39]. Pemilihan unit atau teknik faktorisasi struktur bergantung pada domain permasalahan. Sebagai contoh, suatu model X yang mampu memprediksi *edges* pada graph dengan cukup akurat ternyata tidak dapat memprediksi suatu jalan (*path*) dari *node* A ke B yang seharusnya ada pada *desired output*. Dengan demikian, kamu mungkin ingin memfaktorkan graf tersebut menjadi sekumpulan *paths*.

Hal terpenting saat mengevaluasi *structured output prediction* adalah mengevaluasi berbagai aspek dan mempertimbangkan berbagai perspektif. Artinya, memfaktorkan struktur menjadi beberapa (≥ 1) macam unit, dari unit yang "kecil" (e.g., *edge*) sampai unit yang "besar" (e.g., *subtree* pada struktur pohon; *path* atau subgraph pada graf). Pemilihan unit pada proses faktorisasi bergantung pada domain permasalahan.

Teks

Evaluasi menjadi semakin kompleks apabila kita tidak dapat mendefinisikan metrik secara matematis. Misal pada *image captioning task*: diberikan sebuah gambar dan suatu model harus menjelaskan apa isi gambar tersebut (ilustrasi pada Gambar 9.6). Dari sisi "bentuk" *output*, teks adalah sebuah sekuens kata. Akan tetapi, setiap kata memiliki maknanya tersendiri. Makna kata

dapat berubah pada konteksnya, i.e., ketika beberapa kata membentuk sebuah klausa atau kalimat. Selain itu, teks memiliki struktur "implisit". Sebagai contoh, kata-kata pada teks harus membentuk suatu kesatuan makna sebagai kalimat. Tiap kalimat juga harus sesuai dengan aturan tata bahasa. Hal ini membuat teks menjadi lebih rumit (dan unik) dibanding sekuens kategori.



Gambar 9.6: Ilustrasi *image captioning task*.

Pada kasus *image captioning task* seperti diatas, bagaimana cara kamu mengevaluasi apakah model mampu memberikan penjelasan yang merefleksikan situasi pada gambar? Sebagai contoh, apabila diberikan gambar "seseorang sedang bermain tenis", dan model memprediksi "seseorang sedang bermain basket," maka model memberikan penjelasan yang salah. Tetapi apabila kamu melihat dari sisi proporsi banyaknya kata yang sama, terdapat banyak kata pada *desired output* yang beririsan dengan prediksi, yaitu "seseorang sedang bermain."

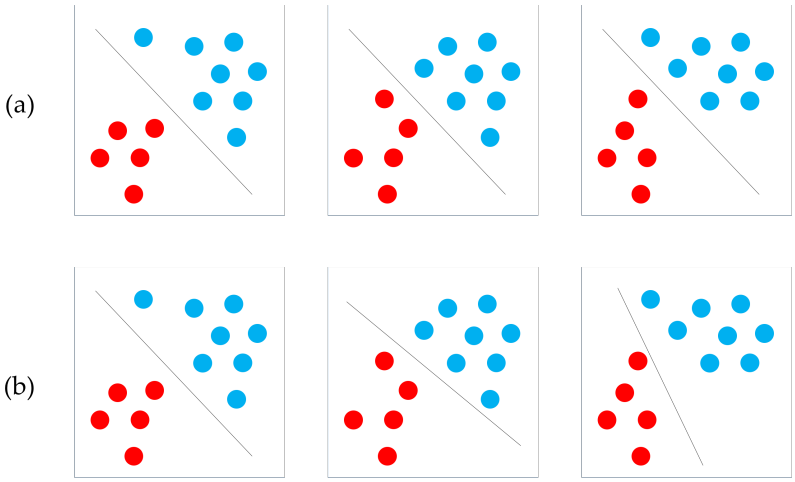
Hal serupa juga harus diperhatikan pada persoalan mesin translasi bahasa. Apakah konten translasi memang benar sama dengan konten aslinya? Bagaimana dengan gaya penulisan? Pada sistem peringkasan teks, apakah hasil ringkasan memang benar sesuai dengan teks aslinya, bukan menambah informasi atau mengada-ada?

Cerita pada subbab 9.4 mengilustrasikan bahwa evaluasi model adalah perkara yang kompleks. Pemilihan metrik yang salah dapat mengakibatkan kamu menganggap bahwa model sudah "cukup" baik, padahal model tersebut memberikan banyak prediksi yang salah. Subbab ini juga sekilas menjelaskan bahwa bentuk *output* pada *supervised learning* bisa saja memiliki struktur yang rumit.

9.5 Cross Validation

Ingat kembali materi bab-bab sebelumnya bahwa pada umumnya kita membagi dataset menjadi tiga kelompok: *training*, *validation/development* dan

testing. Melatih dan mengukur kinerja model menggunakan *training data* adalah hal yang tidak bijaksana karena kita tidak dapat mengetahui kemampuan generalisasi model. Kita melatih model pembelajaran mesin menggunakan *training data* yang dievaluasi kinerjanya (saat *traning*) menggunakan *validation data*. Setelah itu, kita uji model kembali menggunakan *testing data*. Pada umumnya, ketiga kelompok tersebut memiliki data dengan karakteristik yang sama. Artinya, dataset disampel dari distribusi yang sama. Misal pada *training data*, ada pembagian 30:30:40 untuk data kelas pertama, kedua dan ketiga. Distribusi persebaran tersebut juga harus sama pada *validation* dan *testing data*. Hal penting yang harus diperhatikan adalah tidak boleh ada data yang sama (*overlap*) pada ketiga kelompok tersebut.



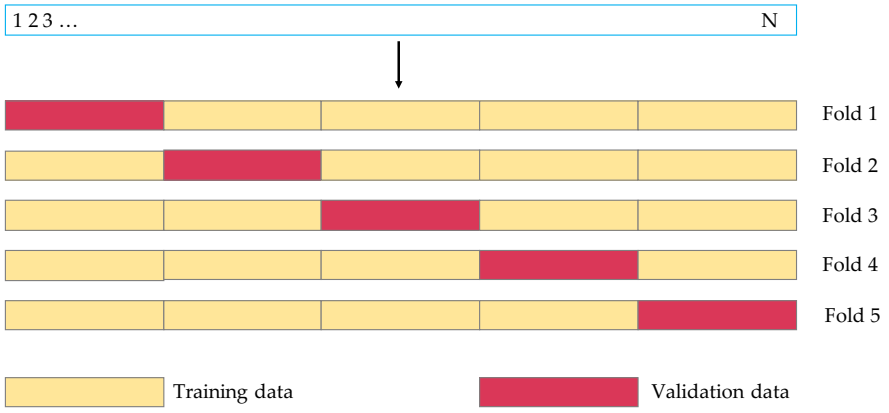
Gambar 9.7: Model yang stabil (a) memberikan *decision boundary* (garis hitam) yang serupa walaupun *input* sedikit diubah-ubah (variasi kinerja yang kecil, untuk set *input* yang berbeda). Model yang kurang stabil (b) memberikan *decision boundary* yang berbeda untuk *input* yang sedikit diubah-ubah (variasi kinerja yang besar, untuk set *input* yang berbeda).

Pada dunia nyata, kita belum tentu memiliki ketiga kelompok tersebut. Penyebabnya ada banyak, misal dataset yang kecil (hanya memiliki sedikit *records*) atau pengadaan *testing data* mahal. Apabila kita tidak memiliki *testing data*, *validation data* dapat menjadi pengganti (*proxy*).

Akan tetapi, permasalahan *statistical bias* dapat terjadi apabila kita hanya menguji model pada satu set data karena kita tidak mengetahui variasi kinerja model [20]. Sebisanya, kita ingin mengevaluasi kinerja model pada beberapa dataset, dan mengevaluasi variasi kinerja model. Kegiatan semacam ini membuat kita mengetahui apakah suatu model cukup stabil[3] dan fleksi-

[3] https://en.wikipedia.org/wiki/Stability_(learning_theory)

bel. Stabil berarti kinerja model tidak begitu berubah, diberikan *input* yang sedikit berubah (ilustrasi diberikan pada Gambar 9.7). Fleksibel berarti model yang mampu menangkap karakteristik data dengan baik. Misal kita menggunakan model linear dengan persamaan polinomial orde-1, orde-2 dan orde-3. Saat kita menaikkan orde persamaan sampai dengan orde-3, kinerja model terus meningkat pada *validation* data, walaupun kurang lebih sama pada *training data*. Lalu, kita mencoba menggunakan persamaan polinomial orde-4, dan kinerja model menurun pada *validation* data. Artinya, persamaan polinomial orde-3 adalah yang paling optimal untuk memodelkan permasalahan yang kita miliki. Apabila kita tidak memiliki *validation* data, kita tidak akan mengetahui hal ini. Stabil berkaitan dengan nilai varians, sedangkan fleksibel berkaitan dengan perubahan terhadap ukuran kinerja. Fleksibilitas dapat dianalisis dengan mem-*plot* grafik kinerja model pada *training data* dan *validation/testing data* untuk mengetahui *underfitting* dan *overfitting* seperti yang sudah dijelaskan pada bab 5.



Gambar 9.8: Ilustrasi 5-*fold-cross-validation*. Kotak berwarna merah melambangkan subset yang dipilih sebagai *validation data*.

Kita dapat menganalisis stabilitas dengan menggunakan teknik *cross validation* seperti yang sudah dijelaskan pada bab 3. Intinya adalah, kita membagi-bagi *dataset* yang kita miliki menjadi K bagian. Kita latih model menggunakan $K-1$ bagian, kemudian menguji prediksi model pada satu bagian lainnya (sebagai *validation data*) yang mengaproksimasi kinerja pada *testing data* (Ilustrasi pada Gambar 9.8). Perhatikan, walau disebut *validation data*, group ini digunakan sebagai *testing data* pada saat proses latihan. Perlu diperhatikan, distribusi tiap-tiap kelas pada subset data haruslah sama (*stratified sampling*).

Prosedur ini disebut sebagai K-***fold-cross-validation***. Untuk K=N; N=jumlah data, disebut sebagai ***leave-one-out-cross-validation***. Den-

gan teknik *cross validation*, kita memiliki sejumlah K model pembelajaran mesin dan K buah nilai kinerja. Kita dapat mengukur stabilitas model dengan menghitung varians kinerja dari K model. Nilai rata-rata kinerja yang diberikan adalah sebuah estimasi terhadap kinerja model pada *testing data* (yang sesungguhnya tidak ada), diberikan pada persamaan 9.11. Kita ingin model yang stabil, karena nilai varians yang kecil berarti eksperimen dapat diulangi, dan kita mendapatkan kinerja yang sama saat eksperimen diulang. Varians yang terlalu besar dapat berarti kita mendapatkan suatu nilai kinerja secara *random chance* (untung-untungan belaka).

$$\text{Performa} = \frac{1}{K}\sum_{i=1}^{K}\text{KinerjaModel}_i \tag{9.11}$$

9.6 Replicability, Overclaiming dan Domain Dependence

Pada dunia pembelajaran mesin, *replicability* adalah hal yang sangat penting. Artinya, eksperimen yang kamu lakukan dapat diulangi kembali oleh orang lain, serta mendapat kinerja yang kurang lebih sama. Untuk ini, biasanya dataset dipublikasi pada domain publik agar dapat digunakan oleh banyak orang, atau mempublikasi kode program. Selain *replicability*, kamu juga harus memperhatikan *overclaiming*. Banyak orang yang menggunakan *toy dataset* (berukuran sangat kecil) yang tidak merepresentasikan permasalahan aslinya. Akan tetapi, mereka mengklaim bahwa model mereka memiliki generalisasi yang baik. Kamu harus menginterpretasikan kinerja model secara bijaksana. Kinerja yang baik pada *toy dataset* belum tentu berarti kinerja yang baik pada dunia nyata. Sebagai contoh, artikel yang dibahas pada *post*[4] ini adalah contoh *overclaiming*.

Perlu kamu perhatikan juga, mendapatkan kinerja yang bagus pada suatu dataset belum tentu berarti model yang sama dapat mencapai kinerja yang baik pada dataset lainnya. Misalnya, model yang dilatih untuk mengklasifikasikan kategori berita belum tentu dapat mengklasifikasikan laporan medis. Hal ini banyak terjadi pada *supervised learning* [40]. Selain itu, kamu harus memperhatikan karakteristik *performance metric* yang digunakan agar tidak salah menginterpretasikan kualitas model.

Soal Latihan

9.1. Seleksi Fitur
Jelaskanlah algoritma seleksi fitur selain yang sudah dijelaskan pada bab ini! (Saran: baca *survey paper*)

[4] Pranala Post Yoav Goldberg

9.2. AIC dan BIC
Jelaskan *Akaike Information Criterion* (AIC) dan *Bayesian Information Criterion* (BIC)!

9.3. AUC
Jelaskan *Area Under Curve* (AUC)!

9.4. Dependency Parsing
Salah satu permasalahan penting pada bidang pemrosesan bahasa alami adalah *dependency parsing*, dimana kita ingin memprediksi hubungan antar kata pada suatu kalimat. Permasalahan ini sangatlah baik untuk mengilustrasikan model yang memprediksi *output* berupa graph. Bacalah paper oleh Kipperwasser dan Goldberg [39] tentang cara mereka memodelkan persoalan tersebut dan jelaskan pada temanmu. Apakah evaluasi yang mereka lakukan sudah cukup?

9.5. Bootstrapping
Jelaskan apa itu teknik *bootstrapping* (evaluasi model)! Bagaimana perbedaanya *bootstrapping* dan *cross validation*?

9.6. Varians
Jelaskan mengapa fitur yang baik memiliki varians yang tinggi, sementara kinerja model yang baik memiliki varians yang rendah!

10

Clustering

> "Most of us cluster somewhere in the middle of most statistical distributions. But there are lots of bell curves, and pretty much everyone is on a tail of at least one of them. We may collect strange memorabillia or read esoteric books, hold unsual religious beliefs or wear odd-sized shoes, suffer rare diseases or enjoy obscure moveis."
>
> Virginia Postrel

Pada bab 4, kamu sudah mempelajari salah satu teknik *clustering* yang cukup umum yaitu K-means. Bab ini akan mengupas *clustering* secara lebih dalam. Kami sarankan kamu untuk membaca paper [41], walaupun relatif lama, tetapi paper tersebut memberikan penjelasan yang mudah dimengerti tentang *clustering*. Selain itu, kamu juga dapat membaca *paper* oleh Saad et al. [42].

Clustering adalah pengelompokkan data dengan sifat yang mirip. Data untuk *clustering* tidak memiliki label (kelas). Secara umum, algoritma *clustering* dapat dikategorikan menjadi dua macam berdasarkan hasil yang dinginkan [43]: (1) *partitional*, yaitu menentukan partisi sebanyak K dan (2) *hierarchical*, yaitu mengelompokan data berdasarkan struktur taksonomi. Contoh algoritma *partitional* adalah **K-means** pada subbab 10.1, sementara contoh algoritma *hierarchical* adalah **agglomerative clustering** pada subbab 10.2.

10.1 K-means, Pemilihan Centroid, Kemiripan Data

Algoritma K-means mengelompokkan data menjadi sebanyak K kelompok sesuai yang kita definisikan. Algoritma ini disebut juga sebagai *flat clustering*, artinya kelompok satu memiliki kedudukan sejajar dengan kelompok lainnya. Kita tinjau kembali tahapan-tahapan algoritma K-means sebagai berikut:

1. Tentukan sebanyak K kelompok yang kita inginkan.
2. Inisiasi **centroid** untuk setiap kelompok. Centroid ibarat seperti "ketua kelompok", yang merepresentasikan kelompok.
3. Hitung kedekatan suatu data terhadap *centroid*, kemudian masukkan data tersebut ke kelompok yang **centroid**-nya memiliki sifat terdekat dengan dirinya.
4. Pilih kembali centroid untuk masing-masing kelompok, yaitu dari anggota kelompok tersebut.
5. Ulangi langkah-langkah sebelumnya sampai tidak ada perubahan anggota untuk semua kelompok.

Perhatikan, ada dua hal penting pada algoritma K-means yaitu: (1) memilih centroid dan (2) Perhitungan kemiripan data. Pada bab 4, dijelaskan salah satu metode pemilihan centroid paling sederhana yaitu secara acak. Pada kenyataannya, inisiasi centroid untuk setiap kelompok/*cluster* dapat dilakukan secara acak; tetapi pada tahap berikutnya, secara umum centroid dipilih menggunakan nilai rata-rata/*mean*. Dengan demikian, centroid bisa saja merupakan suatu vektor yang tidak ada *entry*-nya di dataset.

Diberikan sekumpulan data $\mathbf{D} = \{\mathbf{d}_1, \mathbf{d}_2, \cdots, \mathbf{d}_N\}$; maka centroid c untuk *cluster* itu dihitung dengan persamaan 10.1,

$$\mathbf{c} = \frac{1}{N} \sum_{i=1}^{N} \sum_{e=1}^{F} \mathbf{d}_{i[e]} \tag{10.1}$$

yaitu nilai rata-rata setiap elemen *feature vector* untuk seluruh anggota *cluster* tersebut, dimana N adalah banyaknya anggota *cluster*, F adalah dimensi vektor, $\mathbf{d}_i$ adalah anggota ke-i dalam representasi *feature vector* dan $\mathbf{d}_{i[e]}$ melambangkan elemen ke-e pada vektor $\mathbf{d}_i$. Dengan ini, centroid secara umum bisa jadi tidak merupakan elemen anggota *cluster* (centroid bukan sebuah *instance* data).

Pada bab 4 dijelaskan salah satu metode perhitungan kemiripan data sederhana yaitu dengan menghitung banyaknya nilai atribut yang sama diantara dua *feature vector*. Selain metode tersebut, terdapat banyak perhitungan kemiripan data lainnya tergantung pada tipe, contohnya:

1. **Numerik**. Euclidean Distance, Manhattan Distance, Cosine Distance, dsb.
2. **Boolean**. Jaccard Dissimilarity, Rogers Tanimoto Dissimilarity, dsb.
3. **String**. Levenshtein Distance, Hamming Distance, dsb.

Perhitungan yang paling populer digunakan adalah ***cosine similarity*** [1] (kebetulan pada kebanyakan kasus kita bekerja dengan data numerik), didefinisikan pada persamaan 10.2, yaitu *dot product* antara dua vektor dibagi dengan perkalian *norm* kedua vektor.

$$\text{cosSim}(\mathbf{d}_i, \mathbf{d}_j) = \frac{\mathbf{d}_i \cdot \mathbf{d}_j}{\|\mathbf{d}_i\| \, \|\mathbf{d}_j\|} \tag{10.2}$$

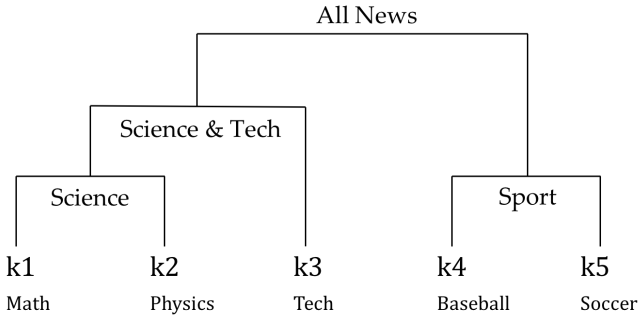
Clusters yang terbentuk, nantinya dapat digunakan sebagai pengelompokkan untuk label klasifikasi. Seumpama $cluster_1$ dapat dianggap sebagai data untuk kelas ke-1, dst.

10.2 Hierarchical Clustering

Hierarchical clustering adalah teknik untuk membentuk pembagian bersarang (*nested partition*). Berbeda dengan K-means yang hasil *clustering*-nya berbentuk *flat* atau rata, *hierarchical clustering* memiliki satu *cluster* paling atas yang mencakup konsep seluruh *cluster* dibawahnya. Ada dua cara untuk membentuk *hierarchical clustering* [41]:

1. **Agglomerative**. Dimulai dari beberapa *flat clusters*; pada setiap langkah iterasi, kita menggabungkan dua *clusters* termirip. Artinya, kita harus mendefinisikan arti "kedekatan" dua *clusters*.

2. **Divisive**. Dimulai dari satu *cluster* (seluruh data), kemudian kita memecah belah *cluster*. Artinya, kita harus mendefinisikan *cluster* mana yang harus dipecah dan bagaimana cara memecahnya.



Gambar 10.1: Ilustrasi *hierarchical clustering*.

[1] https://en.wikipedia.org/wiki/Cosine_similarity

Sebagai contoh, algoritma *hierarchical clustering* menghasilkan struktur hirarkis seperti pada gambar 10.1 yang disebut **dendogram**. Dendogram melambangkan taksonomi, sebagai contoh taksonomi dokumen berita *sport*, dipecah menjadi *baseball* dan *soccer*.

Sejauh ini, teknik **agglomerative clustering** lebih populer, karena pendekatan ini bersifat *bottom-up*. Secara umum, pendekatan *bottom-up* memang relatif lebih populer dibanding pendekatan *top-down*. Langkah-langkah *agglomerative clustering* sebagai berikut:

1. Sediakan sebanyak K *cluster*. Kamu dapat menganggap satu *instance* data sebagai satu *cluster*.
2. Gabung dua *clusters* paling mirip.
3. Ulangi langkah ke-2 sampai hanya satu *cluster* tersisa.

Perhatikan, untuk menggabungkan dua *clusters* termirip, kita membutuhkan definisi "mirip". Definisi tersebut dikuantifikasi dengan formula matematis (seperti definisi kemiripan data pada subbab 10.1). Perhitungan kemiripan *clusters* dapat dihitung dengan tiga metode [43] (untuk data numerik):

1. ***Single Link***. Nilai kemiripan dua *clusters* $\mathbf{U}$ dan $\mathbf{V}$ dihitung berdasarkan nilai kemiripan **maksimum** diantara anggota kedua *clusters* tersebut[2].

$$\text{Sim}_{singleLink}(\mathbf{U}, \mathbf{V}) = \max_{\mathbf{u}_i \in \mathbf{U}, \mathbf{v}_j \in \mathbf{V}} \text{cosSim}(\mathbf{u}_i, \mathbf{v}_j) \tag{10.3}$$

2. ***Complete Link***. Nilai kemiripan dua *clusters* dihitung berdasarkan nilai kemiripan **minimum** diantara anggota kedua *clusters* tersebut.

$$\text{Sim}_{completeLink}(\mathbf{U}, \mathbf{V}) = \min_{\mathbf{u}_i \in \mathbf{U}, \mathbf{v}_j \in \mathbf{V}} \text{cosSim}(\mathbf{u}_i, \mathbf{v}_j) \tag{10.4}$$

3. **UPGMA (*Average Link*)**. Nilai kemiripan dua *clusters* dihitung berdasarkan nilai kemiripan **rata-rata** diantara anggota kedua *clusters* tersebut.

$$\text{Sim}_{\text{UPGMA}}(\mathbf{U}, \mathbf{V}) = \frac{1}{|U|\,|V|} \sum_{\mathbf{u}_i \in \mathbf{U}, \mathbf{v}_j \in \mathbf{V}} \text{cosSim}(\mathbf{u}_i, \mathbf{v}_j) = \frac{\mathbf{c}^U \mathbf{c}^V}{|U|\,|V|} \tag{10.5}$$

dimana $|U|$ adalah banyaknya data pada *cluster* $\mathbf{U}$ dan $\mathbf{c}^U$ adalah centroid untuk *cluster* $\mathbf{U}$.

[2] *Cluster* dapat dianggap sebagai matriks karena merupakan kumpulan *feature vector*.

10.3 Evaluasi

Diberikan sekumpulan data $\mathbf{D} = \{\mathbf{d}_1, \mathbf{d}_2, \cdots, \mathbf{d}_N\}$ untuk suatu *cluster*. Saat tidak tersedianya informasi label/kelas untuk setiap data, kualitas hasil *clustering* dapat dihitung dengan tiga kriteria yaitu:

1. ***Intra-cluster similarity***, yaitu menghitung rata-rata kedekatan antara suatu anggota dan anggota *cluster* lainnya.

$$I = \frac{1}{N^2} \sum_{\mathbf{d}_i \mathbf{d}_j, i \neq j} \text{cosSim}(\mathbf{d}_i, \mathbf{d}_j) \tag{10.6}$$

   Perhitungan kedekatan antar tiap pasang anggota *cluster* sama halnya dengan menghitung *norm* dari centroid *cluster* tersebut, ketika centroid dihitung menggunakan *mean* (buktikan!).

$$I = \frac{1}{N^2} \sum_{\mathbf{d}_i \mathbf{d}_j, i \neq j} \text{cosSim}(\mathbf{d}_i, \mathbf{d}_j) = \|\mathbf{c}\|^2 \tag{10.7}$$

   Perhitungan ini dapat dinormalisasi sesuai dengan banyaknya anggota cluster

$$I' = \frac{\|\mathbf{c}\|^2}{N} \tag{10.8}$$

   Semakin tinggi kemiripan anggota pada suatu *cluster*, semakin baik kualitas *cluster* tersebut.

2. ***Inter-cluster similarity***, yaitu menghitung bagaimana perbedaan antara suatu *cluster* dan *cluster* lainnya. Hal tersebut dihitung dengan *cosine similarity* antara centroid suatu *cluster* dan centroid dari seluruh data [42], dimana $\mathbf{c}_k$ adalah centroid *cluster* ke-k, $\mathbf{c}^{\mathbf{D}}$ adalah centroid (*mean*) dari seluruh data, N_k adalah banyaknya anggota *cluster* ke-k, dan K adalah banyaknya *clusters*. Semakin kecil nilai *inter-cluster similarity*, maka semakin baik kualitas *clustering*.

$$E = \sum_{k=1}^{K} N_k \frac{\mathbf{c}_k \mathbf{c}^{\mathbf{D}}}{\|\mathbf{c}_k\|} \tag{10.9}$$

3. ***Hybrid***. Perhitungan *intra-cluster* dan *inter-cluster* mengoptimalkan satu hal sementara tidak memperdulikan hal lainnya. *Intra-cluster* menghitung keeratan anggota *cluster*, sementara *Inter-cluster* menghitung separasi antar *clusters*. Kita dapat menggabungkan keduanya sebagai *hybrid* (gabungan), dihitung dengan:

$$H = \frac{\sum_{k=1}^{K} I'_k}{E} = \frac{\sum_{k=1}^{K} \frac{\|\mathbf{c}_k\|^2}{N_k}}{\sum_{k=1}^{K} N_k \frac{\mathbf{c}_k \mathbf{c}^{\mathbf{D}}}{\|\mathbf{c}_k\|}} = \sum_{k=1}^{K} \frac{\|\mathbf{c}_k\|}{N_k^2 \mathbf{c}_k \mathbf{c}^{\mathbf{D}}} \tag{10.10}$$

   Semakin besar nilai perhitungan *hybrid*, semakin bagus kualitas *clusters*.

Apabila terdapat informasi label/ kelas untuk setiap data, kita juga dapat menghitung kualitas algoritma *clustering* (perhatikan! tujuan pengukuran adalah kualitas algoritma) dengan **Entropy** dan **Purity**.

Soal Latihan

10.1. Intra-cluster Evaluation
Buktikan kebenaran persamaan 10.7.

10.2. Entropy
Bagaimana cara menghitung kualitas algoritma *clustering*, jika diberikan informasi label/ kelas setiap data menggunakan: (hint, baca [41])
(a) Entropy
(b) Purity

10.3. Kompleksitas
Hitunglah kompleksitas algoritma untuk:
(a) K-means
(b) Agglomerative Clustering
(c) Divisive Clustering

10.4. Kemiripan Data
Sebutkanlah contoh perhitungan kemiripan untuk data *string*. Bagaimana adaptasi perhitungan tersebut pada formula-formula yang sudah diberikan pada algoritma K-means dan agglomerative clustering.

10.5. Agglomerative vs Divisive Clustering
Menurut kamu, mengapa pendekatan *bottom-up* (agglomerative) lebih populer dibanding *top-down* (divisive)? Apa perbedaan kedua pendekatan tersebut (keuntungan dan kerugian masing-masing)?

10.6. Agglomerative Link
Jelaskan apa kelebihan dan kekurangan masing-masing metode perhitungan kemiripan cluster pada agglomerative clustering!

Bagian III

Artificial Neural Network

11

Feedforward Neural Network

> "If you want to make information stick, it's best to learn it, go away from it for a while, come back to it later, leave it behind again, and once again return to it–to engage with it deeply across time. Our memories naturally degrade, but each time you return to a memory, you reactivate its neural network and help to lock it in."
>
> Joshua Foer

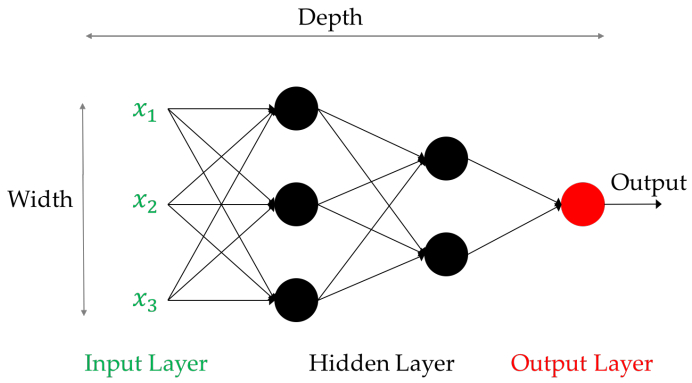
Penulis cukup yakin pembaca sudah menanti-nanti bagian ini. Bagian ketiga membahas algoritma *machine learning* yang sedang populer belakangan ini, yaitu *artificial neural network*. Buku ini lebih berfokus pada penggunaan *artifical neural network* untuk *supervised learning*. Pembahasan dimulai dari hal-hal sederhana (*single perceptron*, *multilayer perceptron*) sampai yang lebih kompleks. Sebelum membaca bab ini, ada baiknya kamu mengulang membaca bab 5 karena memberikan fondasi matematis untuk mengerti bab ini.

11.1 Definisi Artificial Neural Network

Masih ingatkah Anda materi pada bab-bab sebelumnya? *Machine learning* sebenarnya meniru bagaimana proses manusia belajar. Pada bagian ini, peneliti ingin meniru proses belajar tersebut dengan mensimulasikan jaringan saraf biologis (*neural network*) [44, 45, 46, 47]. Kami yakin banyak yang sudah tidak asing dengan istilah ini, berhubung *deep learning* sedang populer dan banyak yang membicarakannya (dan digunakan sebagai trik pemasaran). *Artificial neural network* adalah salah satu algoritma *supervised learning* yang

populer dan bisa juga digunakan untuk *semi-supervised* atau *unsupervised learning* [45, 47, 48, 49, 50]. Walaupun tujuan awalnya adalah untuk mensimulasikan jaringan saraf biologis, jaringan tiruan ini sebenenarnya simulasi yang terlalu disederhanakan, artinya simulasi yang dilakukan tidak mampu menggambarkan kompleksitas jaringan biologis manusia (menurut penulis).[1]

Artificial Neural Network (selanjutnya disingkat ANN), menghasilkan model yang sulit dibaca dan dimengerti oleh manusia karena memiliki banyak layer (kecuali *single perceptron*) dan sifat **non-linear** (merujuk pada fungsi aktivasi). Pada bidang riset ini, ANN disebut agnostik–kita percaya, tetapi sulit membuktikan kenapa konfigurasi parameter yang dihasilkan *training* bisa benar. Konsep matematis ANN itu sendiri cukup *solid*, tetapi *interpretability* model rendah menyebabkan kita tidak dapat menganalisa proses inferensi yang terjadi pada model ANN. Secara matematis, ANN ibarat sebuah graf. ANN memiliki neuron/*node* (*vertex*), dan sinapsis (*edge*). Topologi ANN akan dibahas lebih detil subbab berikutnya. Karena memiliki struktur seperti graf, operasi pada ANN mudah dijelaskan dalam notasi aljabar linear. Sebagai gambaran, ANN berbentuk seperti Gambar 11.1 (*deep neural network*, salah satu varian arsitektur). *Depth* (kedalaman) ANN mengacu pada banyaknya *layer*. Sementara *width* (lebar) ANN mengacu pada banyaknya unit pada *layer*.



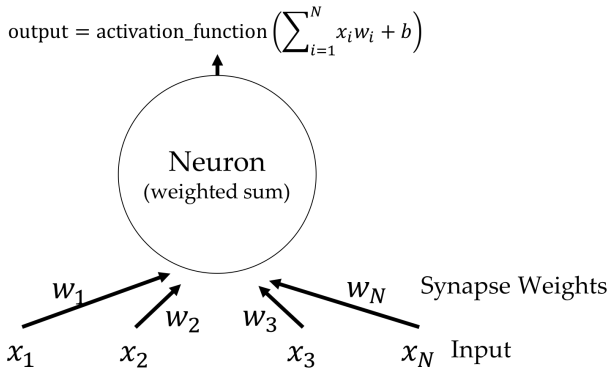
Gambar 11.1: *Deep Neural Network*.

11.2 Single Perceptron

Bentuk terkecil (minimal) sebuah ANN adalah *single perceptron* yang hanya terdiri dari sebuah neuron. Sebuah neuron diilustrasikan pada Gambar 11.2.

[1] `Quora: why is Geoffrey Hinton suspicious of backpropagation and wants AI to start over`

Secara matematis, terdapat *feature vector* **x** yang menjadi *input* bagi neuron tersebut. Ingat kembali, *feature vector* merepresentasikan suatu *data point*, *event* atau *instance*. Neuron akan memproses *input* **x** melalui perhitungan jumlah perkalian antara nilai *input* dan ***synapse weight***, yang dilewatkan pada **fungsi non-linear** [51, 52, 4]. Pada *training*, yang dioptimasi adalah nilai *synapse weight* (*learning parameter*). Selain itu, terdapat juga bias b sebagai kontrol tambahan (ingat materi *steepest gradient descent*). *Output* dari neuron adalah hasil fungsi aktivasi dari perhitungan jumlah perkalian antara nilai *input* dan *synapse weight*. Ada beberapa macam fungsi aktivasi, misal ***step function***, ***sign function***, ***rectifier*** dan ***sigmoid function***. Untuk selanjutnya, pada bab ini, fungsi aktivasi yang dimaksud adalah jenis ***sigmoid function***. Silahkan eksplorasi sendiri untuk fungsi aktivasi lainnya. Salah satu bentuk tipe ***sigmoid function*** diberikan pada persamaan 11.1. Bila di-*plot* menjadi grafik, fungsi ini memberikan bentuk seperti huruf "S".



Gambar 11.2: *Single Perceptron.*

$$\sigma(u) = \frac{1}{1 + e^{-u}} \tag{11.1}$$

Perhatikan kembali, Gambar 11.2 sesungguhnya adalah operasi aljabar linear. Single perceptron dapat dituliskan kembali sebagai 11.2.

$$o = f(\mathbf{x} \cdot \mathbf{w} + b) \tag{11.2}$$

dimana o adalah *output* dan f adalah fungsi **non-linear yang dapat diturunkan secara matematis** (*differentiable non-linear function*–selanjutnya disebut "fungsi non-linear" saja.). Bentuk ini tidak lain dan tidak bukan adalah persamaan model linear yang ditransformasi dengan fungsi non-linear.

Secara filosofis, ANN bekerja mirip dengan model linear, yaitu mencari *decision boudary*. Apabila beberapa model non-linear ini digabungkan, maka kemampuannya akan menjadi lebih hebat (subbab berikutnya). Yang menjadikan ANN "spesial" adalah penggunaan fungsi non-linear.

Untuk melakukan pembelajaran *single perceptron*, *training* dilakukan menggunakan ***perceptron training rule***. Prosesnya sebagai berikut [4, 51, 52]:

1. Inisiasi nilai *synapse weights*, bisa *random* ataupun dengan aturan tertentu. Untuk heuristik aturan inisiasi, ada baiknya membaca buku referensi [1, 11].
2. Lewatkan input pada neuron, kemudian kita akan mendapatkan nilai *output*. Kegiatan ini disebut ***feedforward***.
3. Nilai *output* (*actual output*) tersebut dibandingkan dengan *desired output*.
4. Apabila nilai *output* sesuai dengan *desired output*, tidak perlu mengubah apa-apa.
5. Apabila nilai *output* tidak sesuai dengan *desired output*, hitung nilai *error* (*loss*) kemudian lakukan perubahan terhadap *learning parameter* (*synapse weight*).
6. Ulangi langkah-langkah ini sampai tidak ada perubahan nilai *error*, nilai *error* kurang dari sama dengan suatu *threshold* (biasanya mendekati 0), atau sudah mengulangi proses latihan sebanyak T kali (*threshold*).

Salah satu cara melatih *neural network* adalah dengan mengoptimalkan *error function*, diberikan pada persamaan 11.3[2] (dapat diganti dengan *absolute value*). Perubahan nilai *synapse weight* saat proses latihan (apabila $E \neq 0$ diberikan pada persamaan 11.4, dimana y melambangkan *desired output*,[3] $o = f(\mathbf{x} \cdot \mathbf{w} + \mathbf{b})$ melambangkan *actual output* untuk $\mathbf{x}$ sebagai input, η adalah *learning rate*.

$$E(\mathbf{w}) = (y - o)^2 \tag{11.3}$$

$$\Delta w_i = \eta(y - o)x_i \tag{11.4}$$

Hasil akhir pembelajaran adalah konfigurasi *synapse weight* yang mengoptimalkan nilai *error*. Saat klasifikasi, kita melewatkan *input* baru pada jaringan yang telah dibangun, kemudian tinggal mengambil hasilnya. Pada contoh kali ini, seolah-olah *single perceptron* hanya dapat digunakan untuk melakukan *binary classification* (hanya ada dua kelas, nilai 0 dan 1). Untuk *multi-class classification*, kita dapat menerapkan berbagai strategi, misal *thresholding*, i.e., nilai *output* $0 - 0.2$ mengacu pada kelas pertama, $0.2 - 0.4$ untuk kelas kedua, dst.

[2] Pada umumnya, kita tidak menggunakan satu data, tetapi *batch-sized*.
[3] Pada contoh ini, kebetulan banyaknya *output* neuron hanyalah satu.

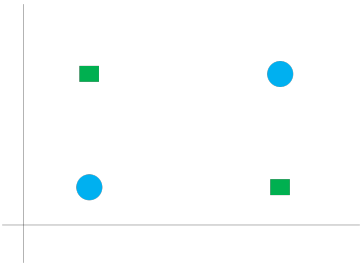
11.3 Permasalahan XOR

Sedikit sejarah, *perceptron* sebenarnya cukup populer sekitar tahun 1950-1960. Entah karena suatu sebab, *perceptron* menjadi tidak populer dan digantikan oleh model linear. Saat itu, belum ada algoritma yang bekerja dengan relatif bagus untuk melatih *perceptron* yang digabungkan (*multilayer perceptron*). Model linear mendapat popularitas hingga kira-kira sekitar tahun 1990'an atau awal 2000'an. Berkat penemuan *backpropagation* sekitar awal 1980,[4] *multilayer perceptron* menjadi semakin populer. Perlu dicatat, komunitas riset bisa jadi seperti cerita ini. Suatu teknik yang baru belum tentu bisa segera diimplementasikan karena beberapa kendala (misal kendala kemampuan komputasi).

Pada bab-bab sebelumnya, kamu telah mempelajari model linear dan model probabilistik. Kita ulangi kembali contoh data yang bersifat *non-linearly separable*, yaitu XOR yang operasinya didefinisikan sebagai:

- $\text{XOR}(0, 0) = 0$
- $\text{XOR}(1, 0) = 1$
- $\text{XOR}(0, 1) = 1$
- $\text{XOR}(1, 1) = 0$

Ilustrasinya dapat dilihat pada Gambar 11.3. Jelas sekali, XOR ini adalah fungsi yang tidak dapat diselesaikan secara langsung oleh model linear.

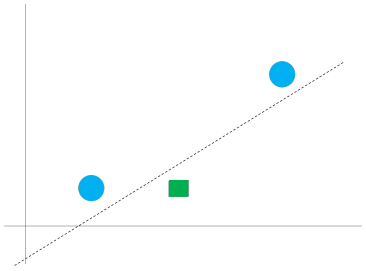


Gambar 11.3: Permasalahan XOR.

Seperti yang diceritakan pada bab model linear, solusi permasalahan ini adalah dengan melakukan transformasi data menjadi *linearly-separable*, misalnya menggunakan fungsi non-linear pada persamaan 11.5 dimana (x, y) adalah absis dan ordinat. Hasil transformasi menggunakan fungsi ini dapat dilihat pada Gambar 11.4. Jelas sekali, data menjadi *linearly separable*.

$$\phi(x, y) = (x \times y, x + y) \tag{11.5}$$

[4] http://people.idsia.ch/~juergen/who-invented-backpropagation.html



Gambar 11.4: XOR ditransformasi. Segiempat berwarna hijau sebenarnya melambangkan dua *instance* (yang setelah ditransformasi kebetulan berlokasi pada tempat yang sama).

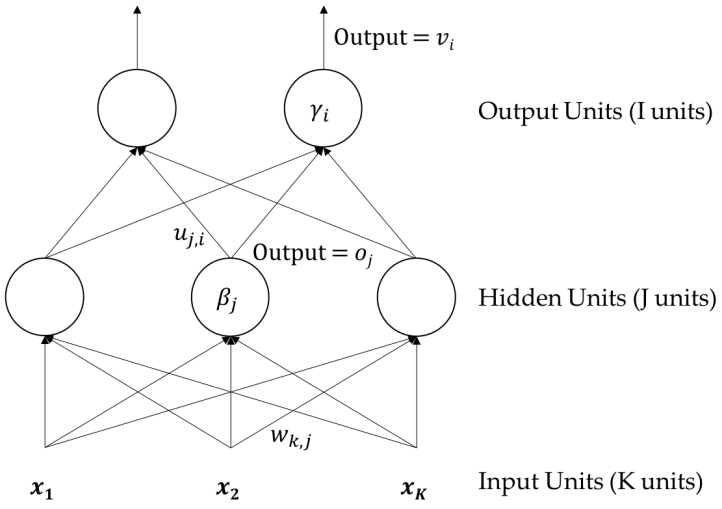
Sudah dijelaskan pada bab model linear, permasalahan dunia nyata tidak sesederhana ini (kebetulan ditransformasikan menjadi data dengan dimensi yang sama). Pada permasalahan praktis, kita harus mentransformasi data menjadi dimensi lebih tinggi (dari 2D menjadi 3D). Berbeda dengan ide utama linear model/*kernel method* tersebut, **prinsip ANN adalah untuk melewatkan data pada fungsi non-linear** (*non-linearities*). Sekali lagi penulis ingin tekankan, ANN secara filosofis adalah ***trainable non-linear mapping functions***. ANN mampu mentransformasi data ke *space*/ruang konsep yang berbeda (bisa pada dimensi lebih tinggi atau lebih rendah), lalu mencari *non-linear decision boundary* dengan *non-linear functions*. Interaksi antar-fitur juga dapat dimodelkan secara non-linear.

Perlu dicatat, pemodelan non-linear inilah yang membuat ANN menjadi hebat. ANN mungkin secara luas didefinisikan mencakup *single perceptron* tetapi secara praktis, **ANN sebenarnya mengacu pada *multilayer perceptron* dan arsitektur lebih kompleks** (dijelaskan pada subbab berikutnya). Pada masa ini, (hampir) tidak ada lagi yang menggunakan *single perceptron*. Untuk bab-bab kedepan, ketika kami menyebut ANN maka yang diacu adalah *multilayer perceptron* dan arsitektur lebih kompleks (*single perceptron* di-*exclude*). Hal ini disebabkan oleh *single perceptron* tidak dapat mempelajari XOR *function* secara independen tanpa *feature engineering*, sementara *multilayer perceptron* bisa [53].

11.4 Multilayer Perceptron

Kamu sudah belajar bagaimana proses *training* untuk *single perceptron*. Selanjutnya kita akan mempelajari *multilayer perceptron* (MLP) yang juga dikenal sebagai ***feedforward neural network***. Kami tekankan sekali lagi, **istilah "ANN" selanjutnya mengacu pada MLP dan arsitektur lebih kompleks**.

Perhatikan ilustrasi pada Gambar 11.5, *multilayer perceptron* secara literal memiliki beberapa *layers*. Pada buku ini, secara umum ada tiga *layers*: *input*, *hidden*, dan *output layer*. *Input layer* menerima *input* (tanpa melakukan operasi apapun), kemudian nilai *input* (tanpa dilewatkan ke fungsi aktivasi) diberikan ke *hidden units* (persamaan 11.6). Pada *hidden units*, *input* diproses dan dilakukan perhitungan hasil fungsi aktivasi untuk tiap-tiap neuron, lalu hasilnya diberikan ke *layer* berikutnya (persamaan 11.7, σ adalah fungsi aktivasi). *Output* dari *input layer* akan diterima sebagai input bagi *hidden layer*. Begitupula seterusnya *hidden layer* akan mengirimkan hasilnya untuk *output layer*. Kegiatan ini dinamakan ***feed forward*** [45, 4]. Hal serupa berlaku untuk *artificial neural network* dengan lebih dari tiga *layers*. Parameter neuron dapat dioptimisasi menggunakan metode *gradient-based optimization* (dibahas pada subabb berikutnya, ingat kembali bab 5). Perlu diperhatikan, MLP adalah gabungan dari banyak fungsi non-linear. Seperti yang disampaikan pada subbab sebelumnya, gabungan banyak fungsi non-linear ini lebih hebat dibanding *single perceptron*. Seperti yang kamu lihat pada Gambar 11.5, masing-masing neuron terkoneksi dengan semua neuron pada *layer* berikutnya. Konfigurasi ini disebut sebagai ***fully connected***. MLP pada umumnya menggunakan konfigurasi *fully connected*.



Gambar 11.5: *Multilayer Perceptron* 2.

$$o_j = \sigma\left(\sum_{k=1}^{K} x_k w_{k,j} + \beta_j\right) \tag{11.6}$$

$$v_i = \sigma\left(\sum_{j=1}^{J} o_j u_{j,i} + \gamma_i\right) = \sigma\left(\sum_{j=1}^{J} \sigma\left(\sum_{k=1}^{K} x_k w_{k,j} + \beta_j\right) u_{j,i} + \gamma_i\right) \tag{11.7}$$

Perhatikan persamaan 11.6 dan 11.7 untuk menghitung *output* pada *layer* yang berbeda. u, w adalah *learning parameters*. β, γ melambangkan *noise* atau *bias*. K adalah banyaknya *input units* dan J adalah banyaknya *hidden units*. Persamaan 11.7 dapat disederhanakan penulisannya sebagai persamaan 11.8. Persamaan 11.8 terlihat relatif lebih "elegan", dimana σ melambangkan fungsi aktivasi. Seperti yang disebutkan pada subbab sebelumnya, ANN dapat direpresentasikan dengan notasi aljabar linear.

$$\mathbf{v} = \sigma(\mathbf{o}\mathbf{U} + \gamma) = \sigma((\sigma(\mathbf{x}\mathbf{W} + \beta))\mathbf{U} + \gamma) \tag{11.8}$$

Untuk melatih MLP, algoritma yang umumnya digunakan adalah ***backpropagation*** [54]. Arti kata *backpropagation* sulit untuk diterjemahkan ke dalam bahasa Indonesia. Kita memperbaharui parameter (*synapse weights*) secara bertahap (dari *output* ke *input* layer, karena itu disebut *backpropagation*) berdasarkan *error*/*loss* (*output* dibandingkan dengan *desired output*). Intinya adalah mengkoreksi *synapse weight* dari *output layer* ke *hidden layer*, kemudian *error* tersebut dipropagasi ke layer sebelum-sebelumnya. Artinya, perubahan *synapse weight* pada suatu layer dipengaruhi oleh perubahan *synapse weight* pada layer setelahnya.[5] *Backpropagation* tidak lain dan tidak bukan adalah metode *gradient-based optimization* yang diterapkan pada ANN.

Pertama-tama diberikan pasangan *input* ($\mathbf{x}$) dan *desired output* ($\mathbf{y}$) sebagai *training data*. Untuk meminimalkan *loss*, algoritma *backpropagation* menggunakan prinsip *gradient descent* (ingat kembali materi bab model linear). Kamu akan memperlajari bagaimana cara menurunkan *backpropagation* menggunakan teknik *gradient descent*, yaitu menghitung *loss* ANN pada Gambar 11.5 **yang menggunakan fungsi aktivasi sigmoid**. Untuk fungsi aktivasi lainnya, pembaca dapat mencoba menurunkan persamaan sendiri!

Ingat kembali *chain rule* pada perkuliahan diferensial

$$f(g(x))' = f'(g(x))g'(x). \tag{11.9}$$

Error, untuk MLP diberikan oleh persamaan 11.10 (untuk satu data *point*), dimana I adalah banyaknya *output unit* dan θ adalah kumpulan *weight matrices* (semua parameter pada MLP). Kami inggatkan kembali perhitungan *error* bisa juga menggunakan nilai absolut.[6]

[5] Kata "setelah" mengacu *layer* yang menuju *output layer* "sebelum" mengacu layer yang lebih dekat dengan *input layer*.

[6] Kami menggunakan tanda kurung agar lebih mudah dibaca penurunan rumusnya.

$$E(\theta) = \frac{1}{2}\sum_{i=1}^{I}(y_i - v_i)^2 \tag{11.10}$$

Mari kita lakukan proses penurunan untuk melatih MLP. *Error/loss* diturunkan terhadap tiap *learning parameter*.

Diferensial $u_{j,i}$ diberikan oleh turunan *sigmoid function*

$$\begin{aligned}\frac{\delta E(\theta)}{\delta u_{j,i}} &= (y_i - v_i)\frac{\delta v_i}{\delta u_{j,i}} \\ &= (y_i - v_i)v_i(1 - v_i)o_j\end{aligned}$$

Diferensial $w_{k,j}$ diberikan oleh turunan *sigmoid function*

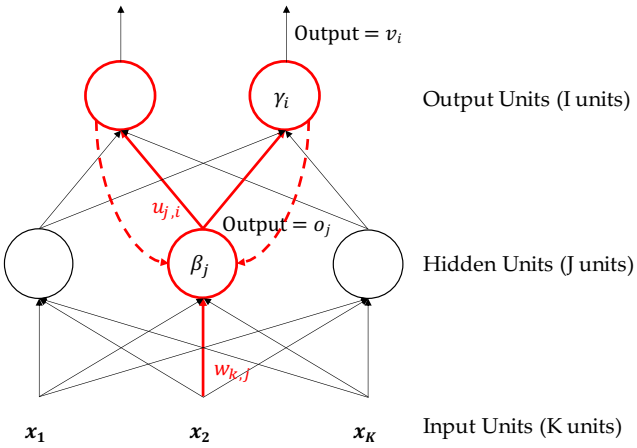
$$\begin{aligned}\frac{\delta E(\theta)}{\delta w_{k,j}} &= \sum_{i=1}^{I}(y_i - v_i)\frac{\delta v_i}{\delta w_{k,j}} \\ &= \sum_{i=1}^{I}(y_i - v_i)\frac{\delta v_i}{\delta o_j}\frac{\delta o_j}{\delta w_{k,j}} \\ &= \sum_{i=1}^{I}(y_i - v_i)(v_i(1 - v_i)u_{j,i})(o_j(1 - o_j)x_k)\end{aligned}$$

Perhatikan, diferensial $w_{k,j}$ memiliki $\sum$ sementara $u_{j,i}$ tidak ada. Hal ini disebabkan karena $u_{j,i}$ hanya berkorespondensi dengan satu *output* neuron. Sementara $w_{k,j}$ berkorespondensi dengan banyak *output* neuron. Dengan kata lain, nilai $w_{k,j}$ mempengaruhi hasil operasi yang terjadi pada banyak *output* neuron, sehingga banyak neuron mempropagasi *error* kembali ke $w_{k,j}$. Ilustrasi diberikan pada Gambar 11.6.

Metode penurunan serupa dapat juga digunakan untuk menentukan perubahan β dan γ. Jadi proses *backpropagation* untuk kasus Gambar 11.5 dapat diberikan seperti pada Gambar 11.7 dimana η adalah *learning rate*. Untuk *artificial neural network dengan* lebih dari 3 *layers*, kita pun bisa menurunkan persamaannya. Secara umum, proses melatih ANN (apapun variasi arsitekturnya) mengikuti *framework perceptron training rule* (subbab 11.2). Cerita pada buku ini menggunakan *error* sebagai *utility function* karena mudah diilustrasikan, serta contoh ini sering digunakan pada referensi lainnya. Pada permasalahan praktis, *cross entropy* adalah *utility function* yang sering digunakan untuk melatih ANN (untuk permasalahan klasifikasi).

11.5 Interpretability

Interpretability ada dua macam yaitu *model interpretability* (i.e., apakah struktur model pembelajaran mesin dapat dipahami) dan *prediction interpretability* (i.e., bagaimana memahami dan memverifikasi cara *input* dipetakan



Gambar 11.6: Justifikasi penggunaan $\sum$ pada penurunan dari *hidden* ke *input layer*.

(2) Hidden to Output

$$v_i = \sigma\left(\sum_{j=1}^{J} o_j u_{j,i} + \gamma_i\right)$$

(3) Output to Hidden

$$\delta_i = (y_i - v_i)v_i(1 - v_i)$$
$$\Delta u_{j,i} = -\eta(t)\delta_i o_j$$
$$\Delta\gamma_i = -\eta(t)\delta_i$$

(1) Input to Hidden Layer

$$o_j = \sigma\left(\sum_{k=1}^{K} x_k w_{k,j} + \beta_j\right)$$

(4) Hidden to Input

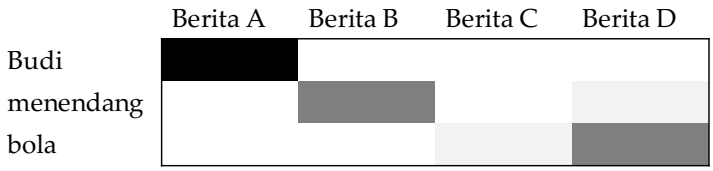
$$\varphi_j = \sum_{i=1}^{I} \delta_i u_{j,i} o_j(1 - o_j)$$
$$\Delta w_{k,j} = -\eta(t)\varphi_j x_k$$
$$\Delta\beta_j = -\eta(t)\varphi_j$$

Gambar 11.7: Proses latihan MLP menggunakan *backpropagation*.

menjadi *output*) [55]. Contoh teknik pembelajaran mesin yang mudah diinterpretasikan baik secara struktur dan prediksinya adalah *decision tree* (bab 6). Struktur *decision tree* berupa pohon keputusan mudah dimengerti oleh manusia dan prediksi (keputusan) dapat dilacak (*trace*). Seperti yang sudah dijelaskan pada bagian pengantar, ANN (MLP) biasanya dianggap sebagai metode *black box* atau susah untuk dinterpretasikan (terutama *model*

interpretability-nya). Hal ini disebabkan oleh kedalaman (*depth*) yaitu memiliki beberapa *layer* dan *non-linearities*. Suatu unit pada *output layer* dipengaruhi oleh kombinasi (*arbitrary combination*) banyak parameter pada *layers* sebelumnya yang dilewatkan pada fungsi non-linear. Sulit untuk mengetahui bagaimana pengaruh bobot suatu unit pada suatu *layer* berpengaruh pada *output layer*, beserta bagaimana pengaruh kombinasi bobot. Intinya, fitur dan *output* tidak memiliki korespondensi satu-satu. Berbeda dengan model linear, kita tahu parameter (dan bobotnya) untuk setiap *input*. Salah satu arah riset adalah mencari cara agar keputusan yang dihasilkan oleh ANN dapat dijelaskan [56],[7] salah satu contoh nyata adalah *attention mechanism* [57, 58] (subbab 13.4.4) untuk *prediction interpretability*. Survey tentang *interpretability* dapat dibaca pada *paper* oleh Doshi-Velez dan Kim [59].

Cara paling umum untuk menjelaskan keputusan pada ANN adalah menggunakan *heat map*. Sederhananya, kita lewatkan suatu data $\mathbf{x}$ pada ANN, kemudian kita lakukan *feed-forward* sekali (misal dari *input* ke *hidden layer* dengan parameter $\mathbf{W}$). Kemudian, kita visualisasikan $\mathbf{x} \cdot \mathbf{W}$ (ilustrasi pada Gambar 11.8). Dengan ini, kita kurang lebih dapat mengetahui bagian *input* mana yang berpengaruh terhadap keputusan di *layer* berikutnya.



Gambar 11.8: Contoh *heat map* pada persoalan klasifikasi teks. *Input* berupa kata-kata yang dimuat pada suatu berita. Output adalah kategori berita untuk *input*. Warna lebih gelap menandakan bobot lebih tinggi. Sebagai contoh, kata "menendang" berkorespondensi paling erat dengan kelas berita B.

11.6 Binary Classification

Salah satu strategi untuk *binary classification* adalah dengan menyediakan hanya satu *output unit* di jaringan. Kelas pertama direpresentasikan dengan -1, kelas kedua direpresentasikan dengan nilai 1 (setelah diaktivasi). Hal ini dapat dicapai dengan fungsi non-linear seperti sign[8] atau tanh.[9] Apabila kita tertarik dengan probabilitas masuk ke dalam suatu kelas, kita dapat meng-

[7] Karena struktur lebih susah, setidaknya beranjak dari keputusan terlebih dahulu
[8] https://en.wikipedia.org/wiki/Sign_function
[9] https://en.wikipedia.org/wiki/Hyperbolic_function

gunakan fungsi seperti sigmoid,[10] dimana *output* pada masing-masing neuron berada pada *range* nilai $[0, 1]$.

11.7 Multi-class Classification

Multilayer perceptron dapat memiliki lebih dari satu *output unit*. Seumpama kita mempunyai empat kelas, dengan demikian kita dapat merepresentasikan keempat kelas tersebut sebagai empat *output units*. Kelas pertama direpresentasikan dengan unit pertama, kelas kedua dengan unit kedua, dst. Untuk C kelas, kita dapat merepresentasikannya dengan C *output units*. Kita dapat merepresentasikan data harus dimasukkan ke kelas mana menggunakan *sparse vector*, yaitu bernilai 0 atau 1. Elemen ke-i bernilai 1 apabila data masuk ke kelas c_i, sementara nilai elemen lainnya adalah 0 (ilustrasi pada Gambar 11.9).*Output* ANN dilewatkan pada suatu fungsi softmax yang melambangkan probabilitas *class-assignment*; i.e., kita ingin *output* agar semirip mungkin dengan *sparse vector* (*desired output*). Pada kasus ini, *output* ANN adalah sebuah distribusi yang melambangkan *input* di-*assign* ke kelas tertentu. Ingat kembali materi bab 5, *cross entropy* cocok digunakan sebagai *utility function* ketika *output* berbentuk distribusi.

$$
\begin{array}{cccc}
c_1 & c_2 & c_3 & c_4 \\
\end{array}
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{array}{l}
\text{Label} = c_1 \\
\text{Label} = c_2 \\
\text{Label} = c_3 \\
\text{Label} = c_4
\end{array}
$$

Gambar 11.9: Ilustrasi representasi *desired output* pada *multi-class classification* menggunakan *sparse vector* (sering disebut sebagai "*one-hot vector*" di komunitas ANN).

11.8 Multi-label Classification

Seperti halnya *multi-class classification*, kita dapat menggunakan sebanyak C neuron untuk merepresentasikan C kelas pada *multi-label classification*. Seperti yang sudah dijelaskan pada bab 5, perbedaan *multi-class* dan *multi-label* terletak pada cara interpretasi *output* dan evaluasi *output*. Pada umumnya, *layer* terakhir diaktivasi dengan fungsi sigmoid, dimana tiap neuron n_i merepresentasikan probabilitas suatu dapat diklasifikasikan sebagai kelas c_i

[10] https://en.wikipedia.org/wiki/Sigmoid_function

$$\begin{array}{cccc} c_1 & c_2 & c_3 & c_4 \\ \left[\begin{array}{cccc} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{array}\right] & \begin{array}{l} \text{Label} = c_1, c_3 \\ \text{Label} = c_2 \\ \text{Label} = c_1, c_4 \\ \text{Label} = c_2, c_3, c_4 \end{array} \end{array}$$

Gambar 11.10: Ilustrasi representasi *desired output* pada *multi-label classification*.

atau tidak (Gambar 11.10). Pada umumnya, *binary cross entropy* digunakan sebagai *loss* (*utility function*) pada *multi-label classification*, yaitu perhitungan *cross entropy* untuk tiap-tiap *output* unit (bukan sekaligus semua *output unit* seperti pada *multi-class classification*).

11.9 Deep Neural Network

Deep Neural Network (DNN) adalah *artificial neural network* yang memiliki banyak *layer*. Pada umumnya, *deep neural network* memiliki lebih dari 3 *layers* (*input layer*, $N \geq 2$ *hidden layers*, *output layer*), dengan kata lain adalah MLP dengan lebih banyak *layer*. Karena ada relatif banyak layer, disebutlah *deep*. Proses pembelajaran pada DNN disebut sebagai *deep learning* (tidak ada bedanya dengan proses latihan pada jaringan yang lebih dangkal, ini hanyalah istilah keren saja) [9]. Jaringan *neural network* pada DNN disebut *deep neural network*.[11]

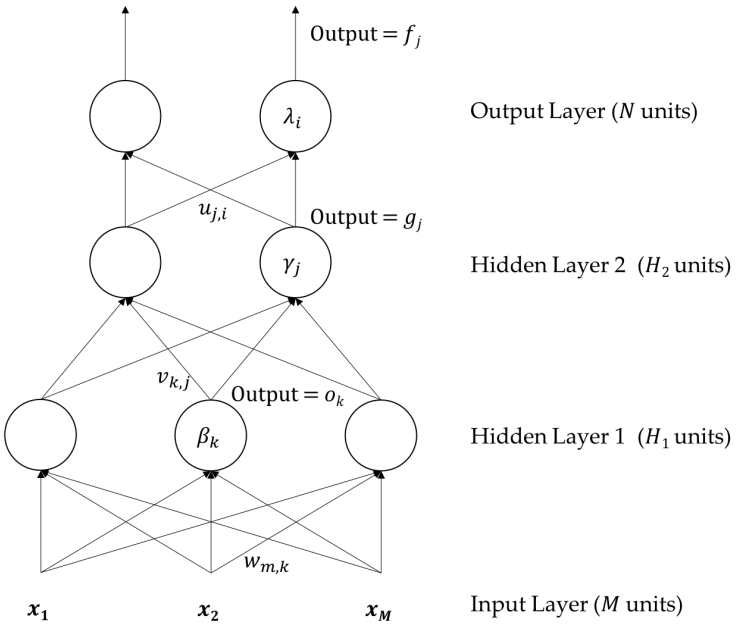
Perhatikan Gambar 11.11 yang memiliki 4 *layers*. Cara menghitung *final output* sama seperti MLP, diberikan pada persamaan 11.11 dimana β, γ, λ adalah *noise* atau *bias*, σ adalah fungsi aktivasi.

$$f_i = \sigma\left(\sum_{j=1}^{H_2} u_{j,i}\sigma\left(\sum_{k=1}^{H_1} v_{k,j}\sigma\left(\sum_{m=1}^{M} x_m w_{m,k} + \beta_k\right) + \gamma_j\right) + \lambda_i\right) \tag{11.11}$$

Cara melatih *deep neural network*, salah satunya dapat menggunakan *back-propagation*. Seperti pada bagian sebelumnya, kita hanya perlu menurunkan rumusnya saja. **Penurunan diserahkan pada pembaca sebagai latihan**. Hasil proses penurunan dapat dilihat pada Gambar 11.12.

Deep network terdiri dari banyak *layer* dan *synapse weight*, karenanya estimasi parameter susah dilakukan. Arti filosofisnya adalah susah/lama untuk menentukan relasi antara *input* dan *output*. Walaupun *deep learning* sepertinya kompleks, tetapi entah kenapa dapat bekerja dengan baik untuk permasalahan praktis [9]. *Deep learning* dapat menemukan relasi "tersembunyi"

[11] Terkadang disingkat menjadi *deep network* saja.



Gambar 11.11: *Deep Neural Network*.

(3) Hidden 2 to Output

$$f_i = \sigma\left(\sum_{j=1}^{H_2} g_j u_{j,i} + \lambda_i\right)$$

(4) Output to Hidden 2

$$\delta_i = (y_i - f_i) f_i (1 - f_i)$$
$$\Delta u_{j,i} = -\eta(t)\delta_i g_j$$
$$\Delta\lambda_i = -\eta(t)\delta_i$$

(2) Hidden 1 to Hidden 2

$$g_j = \sigma\left(\sum_{k=1}^{H_1} o_k v_{k,j} + \gamma_j\right)$$

(5) Hidden 2 to Hidden 1

$$\varphi_j = \sum_{i=1}^{N} \delta_i u_{j,i} g_j (1 - g_j)$$
$$\Delta v_{k,j} = -\eta(t)\varphi_j o_k$$
$$\Delta\gamma_j = -\eta(t)\varphi_j$$

(1) Input to Hidden Layer

$$o_k = \sigma\left(\sum_{m=1}^{M} x_m w_{m,k} + \beta_k\right)$$

(6) Hidden 1 to Input

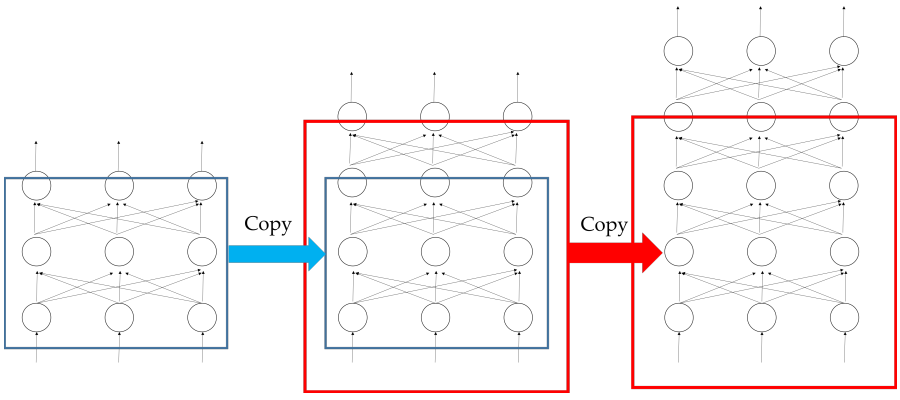
$$\mu_k = \sum_{j=1}^{H_2} \varphi_j v_{k,j} o_k (1 - o_k)$$
$$\Delta w_{m,k} = -\eta(t)\mu_k x_m$$
$$\Delta\beta_k = -\eta(t)\beta_k$$

Gambar 11.12: Proses latihan DNN menggunakan *backpropagation*.

antara *input* dan *output*, yang tidak dapat diselesaikan menggunakan *multi-layer perceptron* (3 layers). Perhatikan, kamu harus ingat bahwa satu langkah *feedforward* memiliki analogi dengan transformasi. Jadi, *input* ditransformasikan secara non-linear sampai akhirnya pada *output*, berbentuk distribusi *class-assignment*.

Banyak orang percaya *deep neural network* lebih baik dibanding *neural network* yang lebar tapi sedikit *layer*, karena terjadi lebih banyak transformasi. Maksud lebih banyak transformasi adalah kemampuan untuk merubah *input* menjadi suatu representasi (tiap *hidden layer* dapat dianggap sebagai salah satu bentuk representasi *input*) dengan langkah *hierarchical*. Seperti contoh permasalahan XOR, permasalahan *non-linearly separable* pun dapat diselesaikan apabila kita dapat mentransformasi data (representasi data) ke dalam bentuk *linearly separable* pada ruang yang berbeda. Keuntungan utama *deep learning* adalah mampu merubah data dari *non-linearly separable* menjadi *linearly separable* melalui serangkaian transformasi (*hidden layers*). Selain itu, *deep learning* juga mampu mencari *decision boundary* yang berbentuk non-linear, serta mengsimulasikan interaksi non-linear antar fitur.

Karena memiliki banyak parameter, proses latihan ANN pada umumnya lambat. Ada beberapa strategi untuk mempercepat pembelajaran menggunakan deep learning, misalnya: regularisasi, ***successive learning***, dan penggunaan ***autoencoder*** [9]. Sebagai contoh, arti *successive learning* adalah jaringan yang dibangun secara bertahap. Misal kita latih ANN dengan 3 *layers*, kemudian kita lanjutkan 3 *layers* tersebut menjadi 4 *layers*, lalu kita latih lagi menjadi 5 *layers*, dst. Hal ini sesuai dengan [60], yaitu mulai dari hal kecil. Ilustrasinya dapat dilihat pada Gambar 11.13. Menggunakan *deep learning* harus hati-hati karena pembelajaran cenderung *divergen* (artinya, *minimum squared error* belum tentu semakin rendah seiring berjalannya waktu–*swing* relatif sering).



Gambar 11.13: Contoh *successive learning*.

11.10 Tips

Pada contoh yang diberikan, *error* atau *loss* dihitung per tiap data point. Artinya begitu ada melewatkan suatu *input*, parameter langsung dioptimisasi sesuai dengan *loss*. Pada umumnya, hal ini tidak baik untuk dilakukan karena ANN menjadi tidak stabil. Metode yang lebih baik digunakan adalah teknik *minibatches*. Yaitu mengoptimisasi parameter untuk beberapa buah *inputs*. Jadi, update parameter dilakukan per *batch*. Perhitungan *error* juga berubah, diberikan pada persamaan 11.12 dimana B melambangkan ***batch size*** (banyaknya *instance* per *batch*), $\mathbf{y}$ adalah *desired output* dan $\mathbf{o}$ adalah *actual output*. Perhitungan *error* saat menggunakan *minibatches* secara sederhana adalah rata-rata (bisa diganti dengan penjumlahan saja) individual *error* untuk semua *instance* yang ada pada *batch* bersangkutan. Setelah menghitung *error* per *batch*, barulah *backpropagation* dilakukan.

$$E(\text{minibatch}) = \frac{1}{B}\sum_{i=1}^{B} \|\mathbf{y} - \mathbf{o}\|^2 \tag{11.12}$$

Data mana saja yang dimasukkan ke suatu *batch* dalam dipilih secara acak. Seperti yang mungkin kamu sadari secara intuitif, urutan data yang disajikan saat *training* mempengaruhi kinerja ANN. Pengacakan ini menjadi penting agar ANN mampu mengeneralisasi dengan baik. Kita dapat mengatur laju pembelajaran dengan menggunakan *learning rate*. Selain menggunakan *learning rate*, kita juga dapat menggunakan *momentum* (subbab 5.6).

Pada library/API *deep learning*, *learning rate* pada umumnya berubah-ubah sesuai dengan waktu. Selain itu, tidak ada nilai khusus (*rule-of-thumb*) untuk *learning rate* terbaik. Pada umumnya, kita inisiasi *learning rate* dengan nilai $\{0.001, 0.01, 0.1\}$ [1]. Biasanya, kita menginisiasi proses latihan dengan nilai *learning rate* cukup besar, kemudian mengecil seiring berjalannya waktu.[12] Kemudian, kita mencari konfigurasi parameter terbaik dengan metode ***grid-search***,[13] yaitu dengan mencoba-coba parameter secara *exhaustive* (*brute-force*) kemudian memilih parameter yang memberikan kinerja terbaik.

ANN sensitif terhadap inisialisasi parameter, dengan demikian banyak metode inisialisasi parameter misalkan, nilai *synapse weights* diambil dari distribusi binomial (silahkan eksplorasi lebih lanjut). Dengan hal ini, kinerja ANN dengan arsitektur yang sama dapat berbeda ketika dilatih ulang dari awal. Karena sensitif terhadap inisialisasi parameter, kamu bisa saja mendapatkan performa yang berbeda saat melatih suatu arsitektur ANN dengan data yang sama (untuk permasalahan yang sama). Untuk menghindari *bias* inisialisasi parameter, biasanya ANN dilatih beberapa kali (umumnya 5, 10, atau 20 kali). Kinerja ANN yang dilaporkan adalah nilai kinerja rata-rata

[12] Analogi: ngebut saat baru berangkat, kemudian memelan saat sudah dekat dengan tujuan agar tidak terlewat.

[13] https://en.wikipedia.org/wiki/Hyperparameter_optimization

dan varians (*variance*). Hal ini untuk menghindari *overclaiming*. Pada konteks pembahasan saat ini, artinya seseorang bisa saja mendapatkan model dengan performa sangat baik, padahal performa arsitektur yang sama bisa saja berkurang saat dicoba oleh orang lain (isu *replicability*).

Kamu mungkin sudah menyadari bahwa melatih ANN harus telaten, terutama dibanding model linear. Untuk model linear, ia akan memberikan konfigurasi parameter yang sama untuk *training data* yang sama (kinerja pun sama). Tetapi, ANN dapat konfigurasi parameter yang berbeda untuk *training data* yang sama (kinerja pun berbeda). Pada model linear, kemungkinan besar variasi terjadi saat mengganti data. Pada ANN, variasi kinerja ada pada seluruh proses! Untuk membandingkan dua arsitektur ANN pada suatu dataset, kita dapat menggunakan *statistical hypothesis testing* (arsitektur X lebih baik dari arsitektur Y secara signifikan dengan nilai $p < threshold$). Penulis merekomendasikan untuk membaca [14, 15] perihal *hypothesis testing*.

Apabila kamu pikir dengan seksama, ANN sebenarnya melakukan transformasi non-linear terhadap *input* hingga menjadi *output*. Parameter diperbarahui agar transformasi non-linear *input* bisa menjadi semirip mungkin dengan *output* yang diharapkan. Dengan hal ini, istilah "ANN" memiliki asosiasi yang dekat dengan "transformasi non-linear". Kami ingin kamu mengingat, ANN (apapun variasi arsitekturnya) adalah **gabungan fungsi non-linear**, dengan demikian ia mampu mengaproksimasi fungsi non-linear (*decision boundary* dapat berupa fungsi non-linear).

Deep learning menjadi penting karena banyaknya transformasi (banyaknya *hidden layers*) lebih penting dibanding lebar jaringan. Seringkali (pada permasalahan praktis), kita membutuhkan banyak transformasi agar *input* bisa menjadi *output*. Setiap transformasi (*hidden layer*) merepresentasikan *input* menjadi suatu representasi. Dengan kata lain, *hidden layer* satu dan *hidden layer* lainnya mempelajari bentuk representasi atau karakteristik *input* yang berbeda.

Curriculum learning juga adalah tips yang layak disebutkan (*mention*) [61]. Penulis kurang mengerti detilnya, sehingga pembaca diharapkan membaca sendiri. Intinya adalah memutuskan apa yang harus ANN pelajari terlebih dahulu (mulai dari mempelajari hal mudah sebelum mempelajari hal yang susah).

11.11 Regularization and Dropout

Seperti yang sudah dijelaskan pada model linear. Kita ingin model mengeneralisasi dengan baik (kinerja baik pada *training data* dan *unseen examples*). Kita dapat menambahkan fungsi regularisasi untuk mengontrol kompleksitas ANN. Regularisasi pada ANN cukup *straightforward* seperti regularisasi pada model linear (subbab 5.9). Kami yakin pembaca bisa mengeksplorasi sendiri.

Selain itu, agar ANN tidak "bergantung" pada satu atau beberapa *synapse weights* saja, kita dapat menggunakan *dropout*. *Dropout* berarti me-*nol*-kan

nilai *synapse weights* dengan nilai *rate* tertentu. Misalkan kita *nol*-kan nilai 30% *synapse weights* (*dropout rate*= 0.3) secara random. Hal ini dapat dicapai dengan teknik *masking*, yaitu mengalikan *synapse weights* dengan suatu *mask*.

Ingat kembali ANN secara umum, persamaan 11.13 dimana $\mathbf{W}$ adalah *synapse weights*, $\mathbf{x}$ adalah *input* (dalam pembahasan saat ini, dapat merepresentasikan *hidden state* pada suatu layer), b adalah *bias* dan f adalah fungsi aktivasi (non-linear). Kita buat suatu *mask* untuk *synapse weights* seperti pada persamaan 11.14, dimana $\mathbf{p}$ adalah vektor dan $p_i = [0, 1]$ merepresentasikan *synapse weight* diikutsertakan atau tidak. $r\%$ (*dropout rate*) elemen vektor $\mathbf{p}$ bernilai 0. Biasanya $\mathbf{p}$ diambil dari *bernoulli distribution* [1]. Kemudian, saat *feed forward*, kita ganti *synapse weights* menggunakan *mask* seperti pada persamaan 11.15. Saat menghitung *backpropagation*, turunan fungsi juga mengikutsertakan *mask* (gradient di-*mask*). Kami sarankan untuk membaca *paper* oleh Srivastava et al. [62] tentang *dropout* pada ANN. Contoh implementasi *dropout* dapat dilihat pada pranala berikut.[14] Teknik *regularization* dan *dropout* sudah menjadi metode yang cukup "standar" dan diaplikasikan pada berbagai macam arsitektur.

$$o = f(\mathbf{x} \cdot \mathbf{W} + b) \tag{11.13}$$

$$\mathbf{W}' = \mathbf{p} \cdot \mathbf{W} \tag{11.14}$$

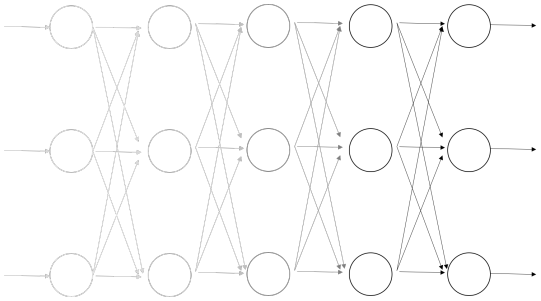
$$o = f(\mathbf{x} \cdot \mathbf{W}' + b) \tag{11.15}$$

11.12 Vanishing and Exploding Gradients

Pada beberapa kasus, nilai gradien ($\Delta\mathbf{W}$ - perubahan parameter) sangat kecil (mendekati nol - *vanishing*) atau sangat besar (*explode*). *Vanishing gradient problem* umum terjadi untuk ANN yang sangat dalam (*deep*), yaitu memiliki banyak *layer*. Hal ini juga terjadi pada arsitektur khusus, seperti *recurrent neural network* saat diberikan input yang panjang [63]. Turunan suatu fungsi bernilai lebih kecil dari fungsi tersebut. Artinya nilai gradient pada *input layer* bernilai lebih kecil dari *output layer*. Apabila kita memiliki banyak *layer*, nilai gradient saat *backpropagation* mendekati nol ketika diturunkan kembali dalam banyak proses. Ilustrasi *vanishing gradient* diberikan pada Gambar 11.14 (analogikan dengan *heat map*). Saat melakukan *backpropagation*, nilai gradien menjadi mendekati nol (warna semakin putih, delta nilai semakin menghilang). Penanganan permasalahan ini masih merupakan topik riset tersendiri. Sebagai contoh, pada arsitektur *recurrent neural network*, biasanya digunakan fungsi aktivasi *long short term memory* (LSTM) atau *gated recurrent unit* (GRU) untuk menangani *vanishing gradient problem*. Selain

[14] https://gist.github.com/yusugomori/cf7bce19b8e16d57488a

nilai gradien, nilai *synapse weights* juga bisa sangat kecil atau sangat besar. Hal ini juga tidak baik!



Gambar 11.14: Ilustrasi *vanishing gradient problem*.

11.13 Rangkuman

Ada beberapa hal yang perlu kamu ingat, pertama-tama jaringan *neural network* terdiri atas:

1. *Input layer*
2. *Hidden layer(s)*
3. *Output layer*

Setiap *edge* yang menghubungkan suatu *node* dengan *node* lainnya disebut *synapse weight*. Pada saat melatih *neural network* kita mengestimasi nilai yang "bagus" untuk *synapse weights*.

Kedua, hal tersulit saat menggunakan *neural network* adalah menentukan topologi. Kamu bisa menggunakan berbagai macam variasi topologi *neural network* serta cara melatih untuk masing-masing topologi. Tetapi, suatu topologi tertentu lebih tepat untuk merepresentasikan permasalahan dibanding topologi lainnya. Menentukan tipe topologi yang tepat membutuhkan pengalaman.

Ketiga, proses *training* untuk *neural network* berlangsung lama. Secara umum, perubahan nilai *synapse weights* mengikuti tahapan (*stage*) berikut [9]:

1. *Earlier state*. Pada tahap ini, struktur global (kasar) diestimasi.
2. *Medium state*. Pada tahap ini, *learning* berubah dari tahapan global menjadi lokal (ingat *steepest gradient descent*).
3. *Last state*. Pada tahap ini, struktur detail sudah selesai diestimasi. Harapannya, model menjadi konvergen.

Neural network adalah salah satu *learning machine* yang dapat menemukan *hidden structure* atau pola data "implisit". Secara umum, *learning machine*

tipe ini sering menjadi *overfitting/overtraining*, yaitu model memiliki kinerja sangat baik pada *training data*, tapi buruk pada *testing data/unseen example*. Oleh sebab itu, menggunakan *neural network* harus hati-hati.

Keempat, *neural network* dapat digunakan untuk *supervised*, *semi-supervised*, maupun *unsupervised learning*. Hal ini membuat *neural network* cukup populer belakangan ini karena fleksibilitas ini. Contoh penggunaan *neural network* untuk *unsupervised* learning akan dibahas pada bab 12. Semakin canggih komputer, maka semakin cepat melakukan perhitungan, dan semakin cepat melatih *neural network*. Hal ini adalah kemewahan yang tidak bisa dirasakan 20-30 tahun lalu.

Soal Latihan

11.1. Turunan

(a) Turunkanlah perubahan *noise/bias* untuk *training* pada MLP.
(b) Turunkanlah proses *training deep neural network* pada Gambar 11.12 termasuk perubahan *noise/bias*.

11.2. Neural Network Training

(a) Sebutkan dan jelaskan cara lain untuk melatih *artificial neural network* (selain *backpropagation*) (bila ada)!
(b) Apa kelebihan dan kekurangan *backpropagation*?
(c) Tuliskan persamaan MLP dengan menggunakan momentum! (kemudian berikan juga penurunan *backpropagation*-nya)

11.3. Regularization Technique

(a) Sebutkan dan jelaskan teknik *regularization* untuk *neural network*! (dalam bentuk formula)
(b) Mengapa kita perlu menggunakan teknik tersebut?

11.4. Softmax Function

(a) Apa itu *softmax function*?
(b) Bagaimana cara menggunakan *softmax function* pada *neural network*?
(c) Pada saat kapan kita menggunakan fungsi tersebut?
(d) Apa kelebihan fungsi tersebut dibanding fungsi lainnya?

11.5. Transformasi atribut

Secara alamiah *neural network* membutuhkan data dengan atribut numerik untuk klasifikasi. Jelaskan konversi/strategi penanganan atribut nominal pada *neural network*!

12

Autoencoder

> "The goal is to turn data into information, and information into insight."
>
> Carly Fiorina

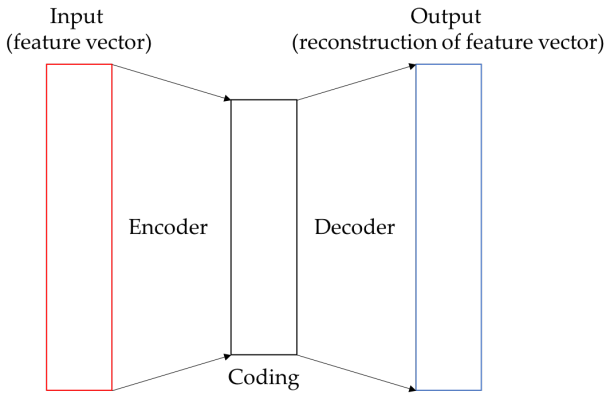
Bab ini memuat materi yang relatif sulit (karena agak *high level*). Bab ini memuat materi *autoencoder* serta penerapannya pada pemrosesan bahasa alami (***natural language processing***–NLP). Berhubung aplikasi yang diceritakan adalah aplikasi pada NLP, kami akan memberi sedikit materi (*background knowledge*) agar bisa mendapat gambaran tentang persoalan pada domain tersebut. Bagi yang tertarik belajar NLP, kami sarankan untuk membaca buku [64]. Teknik yang dibahas pada bab ini adalah ***representation learning*** untuk melakukan pengurangan dimensi pada *feature vector* (*dimensionality reduction*), teknik ini biasanya digolongkan sebagai *unsupervised learning*. Artinya, *representation learning* adalah mengubah suatu representasi menjadi bentuk representasi lain yang ekuvalen, tetapi berdimensi lebih rendah; sedemikian sehingga informasi yang terdapat pada representasi asli tidak hilang/terjaga. Ide dasar teknik ini bermula dari dekomposisi matriks pada aljabar linear.

12.1 Representation Learning

Pada bab model linear, kamu telah mempelajari ide untuk mentransformasi data menjadi dimensi lebih tinggi agar data tersebut menjadi *linearly separable*. Pada bab ini, kamu mempelajari hal sebaliknya, yaitu mengurangi dimensi. *Curse of dimensionality* dapat dipahami secara mendalam apabila kamu membaca buku [65]. Untuk melakukan klasifikasi maupun *clustering*, kita membutuhkan fitur. Fitur tersebut haruslah dapat membedakan satu *instance* dan *instance* lainnya. Seringkali, untuk membedakan *instance* satu dan

instance lainnya, kita membutuhkan *feature vector* yang berdimensi relatif "besar". Karena dimensi *feature vector* besar, kita butuh sumber daya komputasi yang besar juga (bab 9). Untuk itu, terdapat metode-metode ***feature selection***[1] untuk memilih fitur-fitur yang dianggap "representatif" dibanding fitur lainnya. Sayangnya, bila kita menggunakan metode-metode *feature selection* ini, tidak jarang kita kelihangan informasi yang memuat karakteristik data. Dengan kata lain, ada karakteristik yang hilang saat menggunakan *feature selection*.

Pertanyaan yang kita ingin jawab adalah apakah ada cara untuk merepresentasikan data ke dalam bentuk yang membutuhkan memori lebih sedikit tanpa adanya kehilangan informasi? Kita dapat memanfaatkan prinsip *principal component analysis* yang sudah kamu pelajari pada bab 9 untuk mereduksi dimensi data (mengurangi dimensi *input*), pada saat yang bersamaan, menjaga karakteristik data. ***Representation learning*** adalah metode untuk melakukan **kompresi** *feature vector* menggunakan *neural network*.[2] Proses melakukan kompresi disebut ***encoding***, hasil *feature vector* dalam bentuk terkompres disebut ***coding***, proses mengembalikan hasil kompresi ke bentuk awal disebut ***decoding***.[3] *Neural network* yang mampu melakukan proses *encoding* disebut ***encoder***, sedangkan ***decoder*** untuk proses *decoding* [66, 67, 68, 69, 70].



Gambar 12.1: Contoh autoencoder sederhana.

Contoh *representation learning* paling sederhana kemungkinan besar adalah ***autoencoder*** yaitu *neural network* yang dapat merepresentasikan data kemudian merekonstruksinya kembali. Ilustrasi *autoencoder* dapat dilihat pada

[1] http://scikit-learn.org/stable/modules/feature_selection.html
[2] Istilah *representation learning* pada umumnya mengacu dengan teknik menggunakan *neural network*.
[3] Bisa dianggap sebagai proses menginterpretasikan *coding*.

Gambar 12.1. Karena tujuan *encoder* untuk kompresi, bentuk terkompresi haruslah memiliki dimensi lebih kecil dari dimensi *input*. *Neural network* mampu melakukan "kompresi" dengan baik karena ia mampu menemukan *hidden structure* dari data. *Autoencoder* dilatih untuk memilimalkan meminimalkan *loss*. Kamu mungkin berpikir bahwa idealnya, *output* harus sama dengan *input*, yaitu *autoencoder* dengan tingkat *loss* 0%. Akan tetapi, kita sebenarnya tidak ingin *autoencoder* memiliki performa 100% (subbab 12.4).

Contoh klasik lainnya adalah ***N-gram language modelling***, yaitu memprediksi kata y_t diberikan suatu konteks (*surrounding words*) misal kata sebelumnya y_{t-1} (bigram), i.e., $P(y_t \mid y_{t-1})$ Apabila kita mempunyai *vocabulary* sebesar $40,000$ berarti suatu bigram model membutuhkan *memory* sebesar $40,000^2$ (kombinatorial). Apabila kita ingin memprediksi kata diberikan *history* yang lebih panjang (misal dua kata sebelumnya–trigram) maka kita membutuhkan *memory* sebesar $40,000^3$. Artinya, *memory* yang dibutuhkan berlipat secara eksponensial. Tetapi, terdapat strategi menggunakan *neural network* dimana parameter yang dibutuhkan tidak berlipat secara eksponensial walau kita ingin memodelkan konteks yang lebih besar [71].

12.2 Singular Value Decomposition

Sebelum masuk ke *autoencoder* secara matematis, penulis akan memberikan sedikit *overview* tentang dekomposisi matriks. Seperti yang sudah dijelaskan pada bab-bab sebelumnya, dataset dimana setiap *input* direpresentasikan oleh *feature vector* dapat disusun menjadi matriks $\mathbf{X}$ berukuran $N \times F$, dimana N adalah banyaknya sampel[4] dan F adalah dimensi fitur. Pada *machine learning*, dekomposisi atau reduksi dimensi sangat penting dilakukan terutama ketika dataset berupa *sparse matrix*. Dekomposisi berkaitan erat dengan *principal component analysis* (PCA) yang sudah kamu pelajari. Teknik PCA (melalui *eigendecomposition*) mendekomposisi sebuah matriks $\mathbf{X}$ menjadi tiga buah matriks, seperti diilustrasikan pada persamaan 12.1. Matriks $\mathbf{A}$ adalah kumpulan eigenvector dan λ adalah sebuah diagonal matriks yang berisi nilai eigenvalue, dimana $\mathbf{U}$ berukuran $N \times N$, $\mathbf{V}$ berukuran $N \times F$, dan $\mathbf{W}^T$ berukuran $F \times F$.

$$\mathbf{X} = \mathbf{A}\lambda\mathbf{A}^{-1} \tag{12.1}$$

PCA membutuhkan matriks yang kamu ingin dekomposisi berbentu simetris. Sedangkan, teknik *singular value decomposition* (SVD) tidak. Dengan konsep yang mirip dengna PCA, matriks $\mathbf{X}$ dapat difaktorisasi menjadi tiga buah matriks menggunakan teknik SVD, dimana operasi ini berkaitan dengan mencari eigenvectors, diilustrasikan pada persamaan 12.2.

$$\mathbf{X} = \mathbf{U}\ \mathbf{V}\ \mathbf{W}^T \tag{12.2}$$

[4] Banyaknya training data.

Perlu diperhatikan, matriks $\mathbf{V}$ adalah sebuah diagonal matriks (elemennya adalah nilai *singular value* dari $\mathbf{X}$). $\mathbf{U}$ disebut *left-singular vectors* yang tersusun atas eigenvector dari $\mathbf{X}\mathbf{X}^T$. Sementara, $\mathbf{W}$ disebut *right-singular vectors* yang tersusun atas eigenvector dari $\mathbf{X}^T\mathbf{X}$.

Misalkan kita mempunyai sebuah matriks lain $\hat{\mathbf{V}}$ berukuran $K \times K$, yaitu modifikasi matriks $\mathbf{V}$ dengan mengganti sejumlah elemen diagonalnya menjadi 0 (analogi seperti menghapus beberapa baris dan kolom yang dianggap kurang penting). Sebagai contoh, perhatikan ilustrasi berikut!

$$\mathbf{V} = \begin{bmatrix} \alpha_1 & 0 & 0 & 0 & 0 \\ 0 & \alpha_2 & 0 & 0 & 0 \\ 0 & 0 & \alpha_3 & 0 & 0 \\ 0 & 0 & 0 & \alpha_4 & 0 \end{bmatrix} \quad \hat{\mathbf{V}} = \begin{bmatrix} \alpha_1 & 0 & 0 \\ 0 & \alpha_2 & 0 \\ 0 & 0 & \alpha_3 \end{bmatrix}$$

Kita juga dapat me-nol-kan sejumlah baris dan kolom pada matriks $\mathbf{U}$ dan $\mathbf{W}$ menjadi $\hat{\mathbf{U}}$ $(N \times K)$ dan $\hat{\mathbf{W}}^T$ $(K \times F)$. Apabila kita mengalikan semuanya, kita akan mendapat matriks $\hat{\mathbf{X}}$ yang merupakan *approximation* untuk matriks asli $\mathbf{X}$, seperti diilustrasikan pada persamaan 12.3.

$$\hat{\mathbf{X}} = \hat{\mathbf{U}}\ \hat{\mathbf{V}}\ \hat{\mathbf{W}}^T \tag{12.3}$$

Suatu baris dari matriks $\mathbf{E} = \hat{\mathbf{U}}\ \hat{\mathbf{V}}$ dianggap sebagai aproksimasi baris matriks $\mathbf{X}$ berdimensi tinggi [1]. Artinya, menghitung *dot-product* $\mathbf{E}_i \cdot \mathbf{E}_j = \hat{\mathbf{X}}_i \cdot \hat{\mathbf{X}}_j$. Artinya, operasi pada matriks $\mathbf{E}$ kurang lebih melambangkan operasi pada matriks asli. Konsep ini menjadi fundamental *autoencoder* yang akan dibahas pada subbab berikutnya. Operasi data pada level *coding* dianggap merepresentasikan operasi pada bentuk aslinya. Matriks aproksimasi ini memanfaatkan sejumlah K arah paling berpengaruh pada data. Dengan analogi tersebut, sama seperti mentransformasi data ke bentuk lain dimana data hasil transformasi memiliki varians yang tinggi.

12.3 Ide Dasar Autoencoder

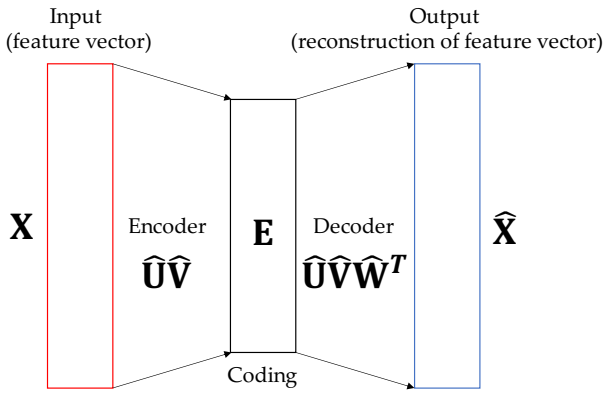
Seperti yang sudah dijelaskan *autoencoder* adalah *neural network* yang mampu merekonstruksi *input*. Ide dasar *autoencoder* tidak jauh dari konsep dekomposisi/*dimentionality reduction* menggunakan *singular value decomposition*. Diberikan dataset $\mathbf{X}$, kita ingin mensimulasikan pencarian matriks $\hat{\mathbf{X}}$ yang merupakan sebuah aproksimasi dari matriks asli. Arsitektur dasar *autoencoder* diberikan pada Gambar 12.1. Kita memberi input matriks $\mathbf{X}$ pada *autoencoder*, kemudian ingin *autoencoder* tersebut menghasilkan matriks yang sama. Dengan kata lain, *desired output* sama dengan *input*. Apabila dihubungkan dengan pembahasan ANN pada bab sebelumnya, *error function* untuk melatih *autoencoder* diberikan pada persamaan 12.4, dimana $\mathbf{y}$ adalah output dari jaringan dan Z adalah dimensi *output*, N adalah banyaknya sampel dan $\mathbf{x}_i$ adalah data ke-i (*feature vector* ke-i).

$$E(\theta) = \frac{1}{N} \sum_{i=j}^{Z} \left(\mathbf{x}_{i[j]} - \mathbf{y}_{i[j]} \right)^2 \tag{12.4}$$

Persamaan 12.4 dapat kita tulis kembali sebagai persamaan 12.5, dimana f melambangkan fungsi aktivasi dan θ adalah ANN (kumpulan *weight matrices*).[5]

$$E(\theta) = \frac{1}{N} \sum_{j=1}^{Z} \left(\mathbf{x}_{i[j]} - f(\mathbf{x}_i, \theta)_{[j]} \right)^2 \tag{12.5}$$

Seperti yang sudah dijelaskan sebelumnya, *desired output* sama dengan *input*. Tetapi seperti yang kamu ketahui, mencapai *loss* sebesar 0% adalah hal yang susah. Dengan demikian, kamu dapat memahami secara intuitif bahwa *autoencoder* melakukan aproksimasi terhadap data asli, tetapi tidak sama persis. Apabila *loss* sebesar 0%, ditakutkan bahwa *autoencoder* semata-mata hanya melakukan translasi (penggesaran) data saja. Gambar 12.2 mengilustrasikan hubungan antara *autoencoder* dan *singular value decomposition*.[6] Perhatikan, ada dua proses *encoding* yaitu merepresentasikan data ke dimensi lebih rendah dan *decoding*–rekonstruksi kembali.



Gambar 12.2: Hubungan autoencoder dan *singular value decomposition* (analogi).

Perhatikan, *hidden layer*/*coding* dapat dianalogikan sebagai $\mathbf{E} = \hat{\mathbf{U}}\ \hat{\mathbf{V}}$. Dengan kata lain, kita dapat melakukan operasi *dot-product* pada *coding* untuk merepresentasikan *dot-product* pada data asli $\mathbf{X}$. Ini adalah ide utama

[5] Pada banyak literatur, kumpulan *weight matrices* ANN sering dilambangkan dengan θ

[6] Hanya sebuah analogi.

autoencoder, yaitu meng-aproksimasi/mengkompresi data asli menjadi bentuk lebih kecil *coding*. Kemudian, operasi pada bentuk *coding* merepresentasikan operasi pada data sebenarnya.

Autoencoder terdiri dari *encoder* (sebuah *neural network*) dan *decoder* (sebuah *neural network*). *Encoder* merubah *input* ke dalam bentuk dimensi lebih kecil (dapat dianggap sebagai kompresi). *Decoder* berusaha merekonstruksi *coding* menjadi bentuk aslinya. Secara matematis, kita dapat menulis *autoencoder* sebagai persamaan 12.6, dimana dec melambangkan *decoder*, enc melambangkan *encoder* dan $\mathbf{x}$ adalah *input*. *Encoder* diberikan pada persamaan 12.7 yang berarti melewatkan *input* pada suatu *layer* di *neural network* untuk menghasilkan representasi $\mathbf{x}$ berdimensi rendah, disebut *coding* $\mathbf{c}$. $\mathbf{U}$ dan α melambangkan *weight matrix* dan *bias*.

$$f(\mathbf{d}, \theta) = \text{dec}(\text{enc}(\mathbf{x})) \tag{12.6}$$

$$\mathbf{c} = \text{enc}(\mathbf{x}) = g(\mathbf{x}, \mathbf{U}, \alpha) \tag{12.7}$$

Representasi $\mathbf{c}$ ini kemudian dilewatkan lagi pada suatu *layer* untuk merekonstruksi kembali *input*, kita sebut sebagai *decoder*. *Decoder* diberikan pada persamaan 12.8 dimana $\mathbf{W}$ dan β melambangkan *weight matrix* dan *bias*. Baik pada fungsi *encoder* dan *decoder*, σ melambangkan fungsi aktivasi.

$$f(\mathbf{d}, \theta) = \text{dec}(\mathbf{c}) = h(\mathbf{c}, \mathbf{W}, \beta) \tag{12.8}$$

Pada contoh sederhana ini, *encoder* dan *decoder* diilustrasikan sebagai sebuah *layer*. Kenyataannya, *encoder* dan *decoder* dapat diganti menggunakan sebuah *neural network* dengan arsitektur kompleks.

Sekarang kamu mungkin bertanya-tanya, bila *autoencoder* melakukan hal serupa seperti *singular value decomposition*, untuk apa kita menggunakan *autoencoder*? (mengapa tidak menggunakan aljabar saja?) Berbeda dengan teknik SVD, teknik *autoencoder* dapat juga mempelajari fitur nonlinear.[7] Pada penggunaan praktis, *autoencoder* adalah *neural network* yang cukup kompleks (memiliki banyak *hidden layer*). Dengan demikian, kita dapat "mengetahui" **berbagai macam representasi** atau transformasi data. *Framework autoencoder* yang disampaikan sebelumnya adalah *framework* dasar. Pada kenyataannya, masih banyak ide lainnya yang bekerja dengan prinsip yang sama untuk mencari *coding* pada permasalahan khusus. *Output* dari *neural network* juga bisa tidak sama *input*-nya, tetapi tergantung permasalahan (kami akan memberikan contoh persoalan *word embedding*). Selain itu, *autoencoder* juga relatif fleksibel; dalam artian saat menambahkan data baru, kita hanya perlu memperbaharui parameter *autoencoder* saja. Kami sarankan untuk membaca *paper* [72, 73] perihal penjelasan lebih lengkap tentang perbedaan dan persamaan SVD dan *autoencoder* secara lebih matematis.

Secara sederhana, *representation learning* adalah teknik untuk mengkompresi *input* ke dalam dimensi lebih rendah tanpa (diharapkan) ada kehilangan

[7] Hal ini abstrak untuk dijelaskan karena membutuhkan pengalaman.

informasi. Operasi vektor (dan lainnya) pada level *coding* merepresentasikan operasi pada bentuk aslinya. Untuk pembahasan *autoencoder* secara lebih matematis, kamu dapat membaca pranala ini.[8] Setelah *autoencoder* dilatih, pada umumnya *encoder* dapat digunakan untuk hal lainnya juga, e.g., klasifikasi kelas gambar.

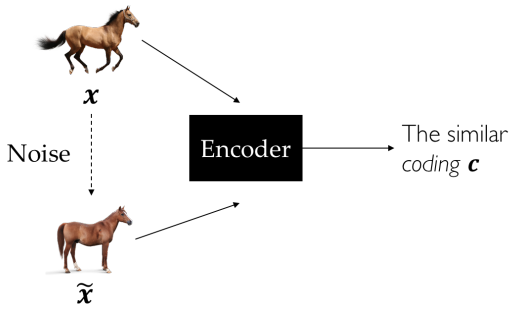
12.4 Resisting Perturbation

Pada subbab sebelumnya, telah dijelaskan bahwa mencapai performa 100% (100% rekonstruksi) pada *autoencoder* adalah hal yang tidak diinginkan. Hal ini disebabkan karena kita ingin menghindari *autoencoder* semata-mata hanya mempelajari *trivial identity function* [11], memiliki analogi dengan *one-to-one mapping*. Misalnya, suatu gambar kuda dipetakan ke *coding* $\mathbf{c}$, kemudian gambar kuda lainnya dipetakan ke *coding* $\hat{\mathbf{c}}$, dan $\hat{\mathbf{c}}$ tidak mirip dengan $\mathbf{c}$, i.e., *cosine similarity*-nya jauh. Artinya kita ingin *autoencoder* merepresentasikan dua hal yang mirip ke dalam bentuk representasi *coding* yang mirip juga! Walaupun kita ingin performa *autoencoder* tidak mencapai 100%, tapi kita masih ingin performanya dekat dengan 100%.

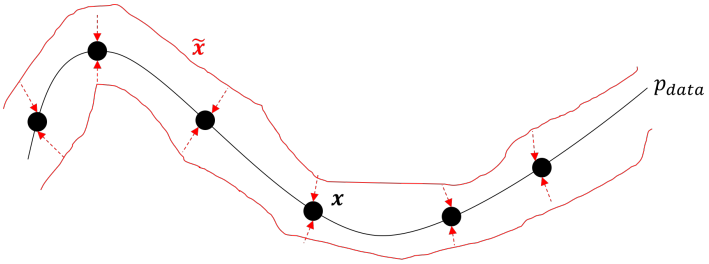
Tujuan utama *autoencoder* adalah mengekstraksi informasi penting tentang data yang ada (seperti *principal components*), bukan replikasi semata. Dengan alasan ini, *coding* pada umumnya memiliki dimensi lebih rendah dibanding *input*. Kita sebut arsitektur ini sebagai ***undercomplete autoencoder***. Apabila *coding* memiliki dimensi lebih besar dari *input*, disebut sebagai ***overcomplete autoencoder***–yang kemungkinan besar hanya mempelajari *trivial identity function* [11]. Kita dapat menggunakan teknik regularisasi pada *autoencoder* untuk memastikan tujuan kita tercapai, misal *sparse autoencoder*, *denoising autoencoder* dan *penalizing derivaties* [11]. Untuk mengilustrasikan permasalahan, buku ini membahas ***denoising autoencoder*** (silahkan baca buku [11] untuk teknik regularisasi lainnya).

Diberikan suatu *input* $\mathbf{x}$, kemudian kita lakukan *noise-injection* terhadap *input* tersebut, menghasilkan $\tilde{\mathbf{x}}$. Perhatikan Gambar 12.3, kita ingin *encoder* memberikan bentuk *coding* yang mirip bagi $\mathbf{x}$ dan $\tilde{\mathbf{x}}$. Kita ingin memaksa *autoencoder* untuk mempelajari sebuah fungsi yang tidak berubah terlalu jauh ketika *input* sedikit diubah. Hal ini disebut sebagai sifat ***resistance to perturbation***. Performa *autoencoder* yang bernilai 100% berbahaya karena *autoencoder* tersebut belum tentu mampu mempelajari sifat data, melainkan mampu "mengingat" *training data* saja (*mapping table*). Objektif *denoising autoencoder* diberikan pada persamaan 12.9, yaitu kemampuan merekonstruksi kembali data tanpa *noise*.

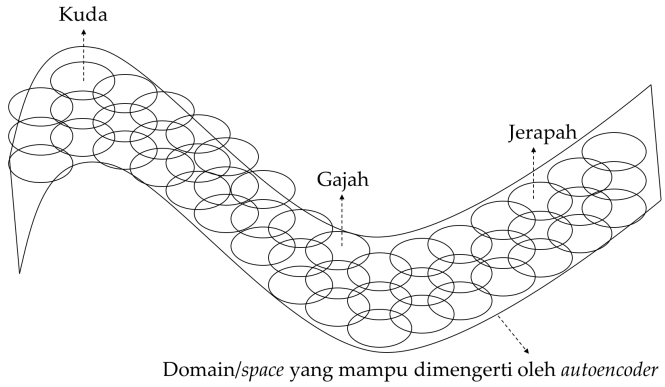
$$E(\theta) = \frac{1}{N}\sum_{j=1}^{Z}\left(\mathbf{x}_{i[j]} - f(\tilde{\mathbf{x}}_i, \theta)_{[j]}\right)^2 \tag{12.9}$$



Gambar 12.3: *Resisting Perturbation.*



Gambar 12.4: *Autoencoder* yang memiliki sifat *resistance to perturbation*, yaitu invarian terhadap sedikit perubahan.



Gambar 12.5: *Manifolds.*

Implikasi atau tujuan dari persamaan 12.9 diberikan pada Gambar 12.4 yang mengilustrasikan *invariant to slight changes*. Diberikan data dengan dis-

[8] https://jaan.io/what-is-variational-autoencoder-vae-tutorial/

tribusi asli p_{data}, dan data yang sudah terkena *noise* $\tilde{\mathbf{x}}$, *autoencoder* mampu mengembalikan $\tilde{\mathbf{x}}$ ke bentuk asli $\mathbf{x}$. Sebagai ilustrasi yang lebih "global", perhatikan Gambar 12.5 dimana suatu elips melambangkan ***manifolds***, dimana data yang mirip (e.g., kuda dengan berbagai macam pose) berada pada ruang yang cukup dekat satu sama lain. Kamu dapat memahami *resistance to perturbation* membuat *autoencoder* membentuk semacam "ruang lokal" yang merepresentasikan suatu data dan variannya.

12.5 Representing Context: Word Embedding

Subbab ini menceritakan salah satu aplikasi *autoencoder*. Pada domain NLP, kita ingin komputer mampu mengerti bahasa selayaknya manusia mengerti bahasa. Misalkan komputer mampu mengetahui bahwa "meja" dan "kursi" memiliki hubungan yang erat. Hubungan seperti ini tidak dapat terlihat berdasarkan teks tertulis, tetapi kita dapat menyusun kamus hubungan kata seperti WordNet.[9] WordNet memuat ontologi kata seperti hipernim, antonim, sinonim. Akan tetapi, hal seperti ini tentu sangat melelahkan, seumpama ada kata baru, kita harus memikirkan bagaimana hubungan kata tersebut terhadap seluruh kamus yang sudah dibuat. Pembuatan kamus ini memerlukan kemampuan para ahli linguistik.

Oleh sebab itu, kita harus mencari cara lain untuk menemukan hubungan kata ini. Ide utama untuk menemukan hubungan antarkata adalah ***statistical semantics hypothesis*** yang menyebutkan pola penggunaan kata dapat digunakan untuk menemukan arti kata [74]. Contoh sederhana, kata yang muncul pada "**konteks**" yang sama cenderung memiliki makna yang sama. Perhatikan "konteks" dalam artian NLP adalah kata-kata sekitar (*surrounding words*);[10] contohnya kalimat "budi menendang bola", "konteks" dari "bola" adalah "budi menendang". Kata "cabai" dan "permen" pada kedua kalimat "budi suka cabai" dan "budi suka permen" memiliki kaitan makna, dalam artian keduanya muncul pada konteks yang sama. Sebagai manusia, kita tahu ada keterkaitan antara "cabai" dan "permen" karena keduanya bisa dimakan.

Berdasarkan hipotesis tersebut, kita dapat mentransformasi kata menjadi sebuah bentuk matematis dimana kata direpresentasikan oleh pola penggunaannya [64]. Arti kata ***embedding*** adalah transformasi kata (beserta konteksnya) menjadi bentuk matematis (vektor), i.e., mirip/sama dengan *coding*. "Kedekatan hubungan makna" (***semantic relationship***) antarkata kita harapkan dapat tercermin pada operasi vektor. Salah satu metode sederhana untuk merepresentasikan kata sebagai vektor adalah ***Vector Space Model***. Konsep *embedding* dan *autoencoder* sangatlah dekat, tapi kami ingin menakankan bahwa *embedding* adalah bentuk representasi konteks.

[9] https://wordnet.princeton.edu/

[10] Selain *surrounding words*, konteks dalam artian NLP dapat juga berupa kalimat, paragraph, atau dokumen.

	Dokumen 1	Dokumen 2	Dokumen 3	Dokumen 4	⋯
King	1	0	0	0	⋯
Queen	0	1	0	1	⋯
Prince	1	0	1	0	⋯
Princess	0	1	0	1	⋯
⋯					

Tabel 12.1: Contoh 1-of-V encoding.

Semantic relationship dapat diartikan sebagai ***attributional*** atau ***relational similarity***. *Attributional similarity* berarti dua kata memiliki atribut/sifat yang sama, misalnya anjing dan serigala sama-sama berkaki empat, menggongong, serta mirip secara fisiologis. *Relational similarity* berarti derajat korespondensi, misalnya *anjing* : *menggongong* memiliki hubungan yang erat dengan *kucing* : *mengeong*.

12.5.1 Vector Space Model

Vector space model (VSM)[11] adalah bentuk *embedding* yang relatif sudah cukup lama tapi masih digunakan sampai saat ini. Pada pemodelan ini, kita membuat sebuah matriks dimana baris melambangkan kata, kolom melambangkan dokumen. Metode VSM ini selain mampu menangkap hubungan antarkata juga mampu menangkap hubungan antardokumen (*to some degree*). Asal muasalnya adalah *statistical semantics hypothesis*. Tiap sel pada matriks berisi nilai 1 atau 0. 1 apabila $kata_i$ muncul di $dokumen_i$ dan 0 apabila tidak. Model ini disebut ***1-of-V/1-hot encoding*** dimana V adalah ukuran kosa kata. Ilustrasi dapat dilihat pada Tabel 12.1.

Akan tetapi, *1-of-V encoding* tidak menyediakan banyak informasi untuk kita. Dibanding sangat ekstrim saat mengisi sel dengan nilai 1 atau 0 saja, kita dapat mengisi sel dengan frekuensi kemunculan kata pada dokumen, disebut ***term frequency*** (TF). Apabila suatu kata muncul pada banyak dokumen, kata tersebut relatif tidak terlalu "penting" karena muncul dalam berbagai konteks dan tidak mampu membedakan hubungan dokumen satu dan dokumen lainnya (***inverse document frequency***/IDF). Formula IDF diberikan pada persamaan 12.10. Tingkat kepentingan kata berbanding terbalik dengan jumlah dokumen dimana kata tersebut dimuat. N adalah banyaknya dokumen, $|d \epsilon D; t \epsilon d|$ adalah banyaknya dokumen dimana kata t muncul.

$$IDF(t, D) = \log\left(\frac{N}{|d \epsilon D; t \epsilon d|}\right) \tag{12.10}$$

Dengan menggunakan perhitungan TF-IDF yaitu $TF \times IDF$ untuk mengisi sel pada matriks Tabel 12.1, kita memiliki lebih banyak informasi. TF-IDF

[11] Mohon bedakan dengan VSM (*vector space model*) dan SVM (*support vector machine*)

sampai sekarang menjadi *baseline* pada ***information retrieval***. Misalkan kita ingin menghitung kedekatan hubungan antar dua dokumen, kita hitung ***cosine distance*** antara kedua dokumen tersebut (vektor suatu dokumen disusun oleh kolom pada matriks). Apabila kita ingin menghitung kedekatan hubungan antar dua kata, kita hitung *cosine distance* antara kedua kata tersebut dimana vektor suatu kata merupakan baris pada matriks. Tetapi seperti intuisi yang mungkin kamu miliki, mengisi *entry* dengan nilai TF-IDF pun akan menghasilkan *sparse matrix*.

Statistical semantics hypothesis diturunkan lagi menjadi empat macam hipotesis [74]:

1. *Bag of words*
2. *Distributional hypothesis*
3. *Extended distributional hypothesis*
4. *Latent relation hypothesis*

Silakan pembaca mencari sumber tersendiri untuk mengerti keempat hipotesis tersebut atau membaca *paper* Turney dan Pantel [74].

12.5.2 Sequential, Time Series dan Compositionality

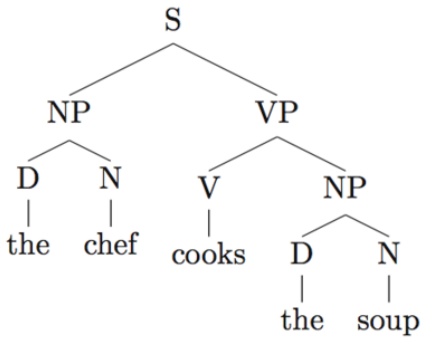
Bahasa manusia memiliki dua macam karakteristik yaitu adalah data berbentuk ***sequential data*** dan memenuhi sifat ***compositionality***. *Sequential data* adalah sifat data dimana suatu kemunculan $data_i$ dipengaruhi oleh data sebelumnya ($data_{i-1}, data_{i-2}, ...$). Perhatikan kedua kalimat berikut:

1. Budi melempar bola.
2. Budi melempar gedung bertingkat.

Pada kedua kalimat tersebut, kalimat pertama lebih masuk akal karena bagaimana mungkin seseorang bisa melempar gedung bertingkat. Keputusan kita dalam memilih kata berikutnya dipengaruhi oleh kata-kata sebelumnya, dalam hal ini "Budi melempar" setelah itu yang lebih masuk akal adalah "bola". Contoh lain adalah data yang memiliki sifat ***time series*** yaitu gelombang laut, angin, dan cuaca. Kita ingin memprediksi data dengan rekaman masa lalu, tapi kita tidak mengetahui masa depan. Kita mampu memprediksi cuaca berdasarkan rekaman parameter cuaca pada hari-hari sebelumnya. Ada yang berpendapat beda *time series* dan *sequential* (sekuensial) adalah diketahuinya sekuens kedepan secara penuh atau tidak. Penulis tidak dapat menyebutkan *time series* dan sekuensial sama atau beda, silahkan pembaca menginterpretasikan secara bijaksana.

Data yang memenuhi sifat ***compositionality*** berarti memiliki struktur hirarkis. Struktur hirarkis ini menggambarkan bagaimana unit-unit lebih kecil berinteraksi sebagai satu kesatuan. Artinya, interpretasi/pemaknaan unit yang lebih besar dipengaruhi oleh interpretasi/pemaknaan unit lebih kecil

(subunit). Sebagai contoh, kalimat "saya tidak suka makan cabai hijau". Unit "cabai" dan "hijau" membentuk suatu frasa "cabai hijau". Mereka tidak bisa dihilangkan sebagai satu kesatuan makna. Kemudian interaksi ini naik lagi menjadi kegiatan "makan cabai hijau" dengan keterangan "tidak suka", bahwa ada seseorang yang "tidak suka makan cabai hijau" yaitu "saya". Pemecahan kalimat menjadi struktur hirarkis berdasarkan *syntactical role* disebut ***constituent parsing***, contoh lebih jelas pada Gambar 12.6. *N* adalah *noun*, *D* adalah *determiner*, *NP* adalah *noun phrase*, *VP* adalah *verb phrase*, dan *S* adalah *sentence*. Selain bahasa manusia, gambar juga memiliki struktur hirarkis. Sebagai contoh, gambar rumah tersusun atas tembok, atap, jendela, dan pintu. Tembok, pintu, dan jendela membentuk bagian bawah rumah; lalu digabung dengan atap sehingga membentuk satu kesatuan rumah.



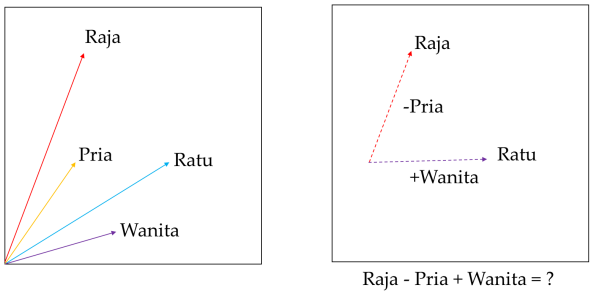
Gambar 12.6: Contoh constituent tree.[12]

12.5.3 Distributed Word Representation

Seperti yang disebutkan pada bagian sebelumnya, kita ingin hubungan kata (yang diinferensi dari konteksnya) dapat direpresentasikan sebagai operasi vektor seperti pada ilustrasi Gambar 12.7. Kata "raja" memiliki sifat-sifat yang dilambangkan oleh suatu vektor (misal 90% aspek loyalitas, 80% kebijaksanaan, 90% aspek kebangsaan, dst), begitu pula dengan kata "pria", "wanita", dan "ratu". Jika sifat-sifat yang dimiliki "raja" dihilangkan bagian sifat-sifat "pria"-nya, kemudian ditambahkan sifat-sifat "wanita" maka idealnya operasi ini menghasilkan vektor yang dekat kaitannya dengan "ratu". Dengan kata lain, raja yang tidak maskulin tetapi fenimin disebut ratu. Seperti yang disebutkan sebelumnya, ini adalah tujuan utama *embedding* yaitu merepresentasikan "makna" kata sebagai vektor sehingga kita dapat memanipulasi banyak hal berdasarkan operasi vektor. Hal ini mirip (tetapi tidak sama)

[12] source: Pinterest

dengan prinsip *singular value decomposition* dan *autoencoder* yang telah dijelaskan sebelumnya.

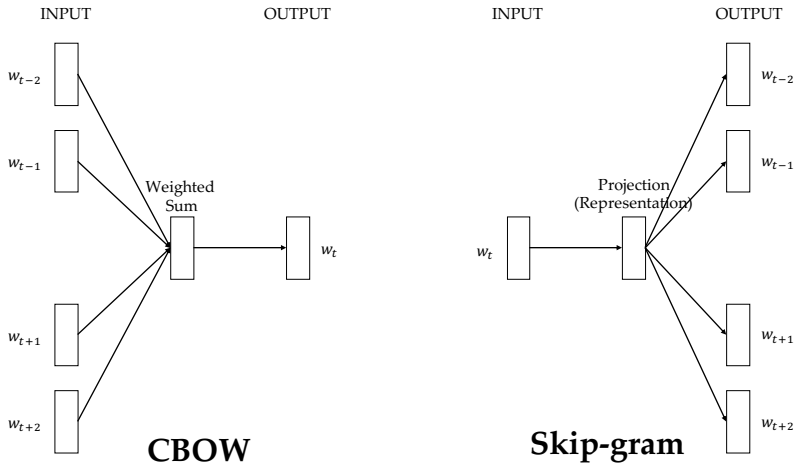


Gambar 12.7: Contoh operasi vektor kata.

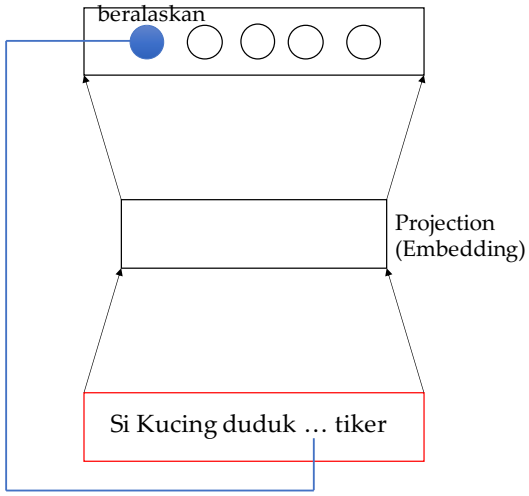
Selain *vector space model*, apakah ada cara lain yang mampu merepresentasikan kata dengan lebih baik? Salah satu kekurangan VSM adalah tidak memadukan sifat sekuensial pada konstruksi vektornya. Cara lebih baik ditemukan oleh [48, 49] dengan ekstensi pada [70]. Idenya adalah menggunakan teknik *representation learning* dan prinsip *statistical semantics hypothesis*. Metode ini lebih dikenal dengan sebutan *word2vec*. Tujuan *word2vec* masih sama, yaitu merepresentasikan kata sebagai vektor, sehingga kita dapat melakukan operasi matematis terhadap kata. *Encoder*-nya berbentuk ***Continous bag of words*** **(CBOW)** atau **Skip-gram**. Pada CBOW, kita memprediksi kata diberikan suatu "konteks". Pada arsitektur "Skip-gram" kita memprediksi konteks, diberikan suatu kata. Ilustrasi dapat dilihat pada Gambar 12.8. Bagian *projection layer* pada Gambar 12.8 adalah *coding* layer. Kami akan memberikan contoh CBOW secara lebih detil. Kedua arsitektur ini dapat dilatih menggunakan *one-hot encoding*, i.e., w_i merepresentasikan *one-hot encoding* untuk kata ke-i.

Perhatikan Gambar 12.9. Diberikan sebuah konteks "si kucing duduk ... tiker". Kita harus menebak apa kata pada "..." tersebut. Dengan menggunakan teknik *autoencoder*, *output layer* adalah distribusi probabilitas $kata_i$ pada konteks tersebut. Kata yang menjadi jawaban adalah kata dengan probabilitas terbesar, misalkan pada kasus ini adalah "beralaskan". Dengan arsitektur ini, prinsip sekuensial atau *time series* dan *statistical semantics hypothesis* terpenuhi (*to a certain extent*). Teknik ini adalah salah satu contoh penggunaan *neural network* untuk *unsupervised learning*. Kita tidak perlu mengkorespondensikan kata dan *output* yang sesuai karena *input vektor* didapat dari statistik penggunaan kata. Agar lebih tahu kegunaan vektor kata, kamu dapat mencoba kode dengan bahasa pemrograman Python 2.7 yang disediakan penulis.[13] Buku ini telah menjelaskan ide konseptual *word embedding* pada

[13] https://github.com/wiragotama/GloVe_Playground



Gambar 12.8: CBOW (*Continous bag of words*) vs Skip-gram, rekonstruksi [49].



Gambar 12.9: CBOW.

level abstrak, yaitu merepresentasikan kata dan konteksnya menjadi bentuk vektor. Apabila kamu tertarik untuk memahami detilnya secara matematis, kamu dapat membaca berbagai penelitian terkait.[14] Silahkan baca *paper* oleh Mikolov [48, 49] untuk detil implementasi *word embedding*.

12.5.4 Distributed Sentence Representation

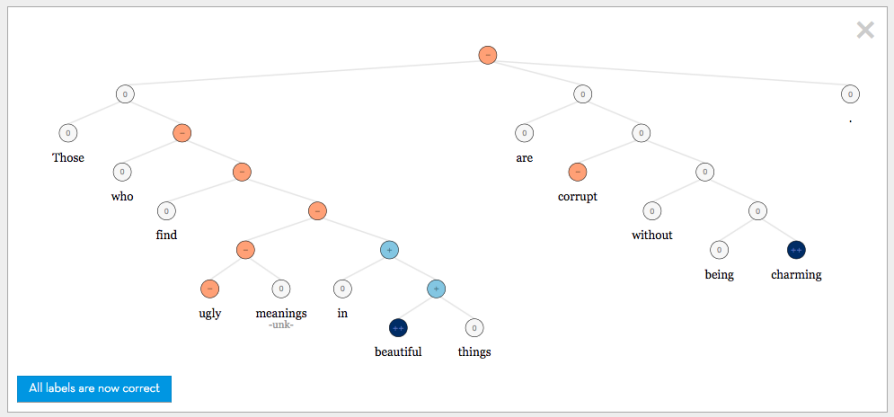
Kita sudah dapat merepresentasikan kata menjadi vektor, selanjutnya kita ingin mengonversi unit lebih besar (kalimat) menjadi vektor. Salah satu cara paling mudah adalah menggunakan nilai rata-rata representasi *word embedding* untuk semua kata yang ada pada kalimat tersebut (*average of its individual word embeddings*). Cara ini sering digunakan pada bidang NLP dan cukup *powerful*, sebagai contoh pada *paper* oleh Putra dan Tokunaga [75]. Pada NLP, sering kali kalimat diubah terlebih dahulu menjadi vektor sebelum dilewatkan pada algoritma *machine learning*, misalnya untuk analisis sentimen (kalimat bersentimen positif atau negatif). Vektor ini yang nantinya menjadi *feature vector* bagi algoritma *machine learning*.

Kamu sudah tahu bagaimana cara mengonversi kata menjadi vektor, untuk mengonversi kalimat menjadi vektor cara sederhananya adalah merata-ratakan nilai vektor kata-kata pada kalimat tersebut. Tetapi dengan cara sederhana ini, sifat sekuensial dan *compositional* pada kalimat tidak terpenuhi. Sebagai contoh, kalimat "anjing menggigit Budi" dan "Budi menggigit anjing" akan direpresentasikan sebagai vektor yang sama karena terdiri dari kata-kata yang sama. Dengan demikian, representasi kalimat sederhana dengan merata-ratakan vektor kata-katanya juga tidaklah sensitif terhadap urutan.[15] Selain itu, rata-rata tidak sensitif terhadap *compositionality*. Misal frase "bukan sebuah pengalaman baik" tersusun atas frase "bukan" yang diikuti oleh "sebuah pengalaman baik". Rata-rata tidak mengetahui bahwa "bukan" adalah sebuah *modifier* untuk sebuah frase dibelakangnya. Sentimen dapat berubah bergantung pada komposisi kata-katanya (contoh pada Gambar 12.10).

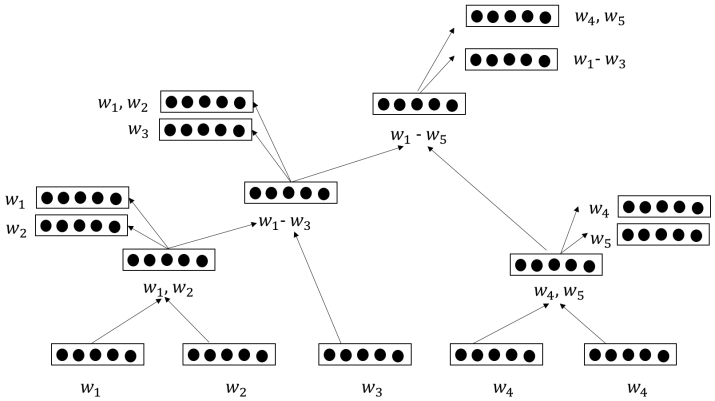
Cara lainnya adalah meng-*encode* kalimat sebagai *vektor* menggunakan *recursive autoencoder*. *Recursive* berarti suatu bagian adalah komposisi dari bagian lainnya. Penggunaan *recursive autoencoder* sangat rasional berhubung data memenuhi sifat *compositionality* yang direpresentasikan dengan baik oleh topologi *recursive neural network*. Selain itu, urutan susunan kata-kata juga tidak hilang. Untuk melatih *recursive autoencoder*, *output* dari suatu layer adalah rekonstruksi *input*, ilustrasi dapat dilihat pada Gambar 12.11. Pada setiap langkah *recursive*, *hidden layer/coding layer* berusaha men-*decode* atau merekonstruksi kembali vektor *input*.

Lebih jauh, untuk sentimen analisis pada kata, kita dapat menambahkan output pada setiap *hidden layer*, yaitu sentimen unit gabungan, seperti pada

[14] Beberapa orang berpendapat bahwa *evil is in the detail*.
[15] Karena ini *recurrent neural network* bagus untuk *language modelling*.
[16] http://nlp.stanford.edu:8080/sentiment/rntnDemo.html



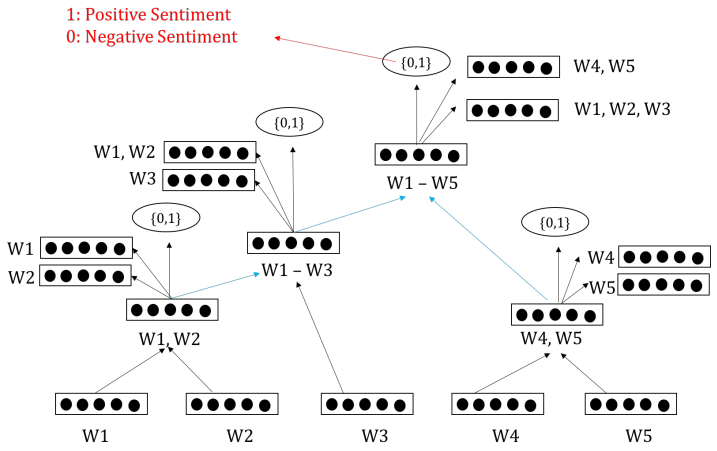
Gambar 12.10: Contoh analisis sentimen (Stanford).[16]



Gambar 12.11: Contoh recursive autoencoder.

Gambar 12.12. Selain menggunakan *recursive autoencoder*, kamu juga dapat menggunakan *recurrent autoencoder*. Kami silahkan pada pembaca untuk memahami *recurrent autoencoder*. Prinsipnya mirip dengan *recursive autoencoder*.

Teknik yang disampaikan mampu mengonversi kalimat menjadi vektor, lalu bagaimana dengan paragraf, satu dokumen, atau satu frasa saja? Teknik umum untuk mengonversi teks menjadi vektor dapat dibaca pada [69] yang lebih dikenal dengan nama ***paragraph vector*** atau ***doc2vec***.



Gambar 12.12: Contoh recursive autoencoder dengan sentiment[67].

12.6 Tips

Bab ini menyampaikan penggunaan *neural network* untuk melakukan *kompresi data* (*representation learning*) dengan teknik *unsupervised learning*. Hal yang lebih penting untuk dipahami bahwa ilmu *machine learning* tidak berdiri sendiri. Walaupun kamu menguasai teknik *machine learning* tetapi tidak mengerti domain dimana teknik tersebut diaplikasikan, kamu tidak akan bisa membuat *learning machine* yang memuaskan. Contohnya, pemilihan fitur *machine learning* pada teks (NLP) berbeda dengan gambar (*computer vision*). Mengerti *machine learning* tidak semata-mata membuat kita bisa menyelesaikan semua macam permasalahan. Tanpa pengetahuan tentang domain aplikasi, kita bagaikan orang buta yang ingin menyetir sendiri!

Soal Latihan

12.1. Penggunaan Autoencoder untuk Arsitektur Kompleks

(a) Pada bab ini, telah dijelaskan bahwa kita dapat menginisialisasi arsitektur *neural network* yang kompleks menggunakan *autoencoder*. Jelaskan pada kasus apa kita dapat melakukan hal tersebut!
(b) Jelaskan mengapa menginisiasi (sebagian) arsitektur kompleks menggunakan *autoencoder* adalah sesuatu yang masuk akal!

12.2. LSI dan LDA

(a) Jelaskanlah Latent Semantic Indexing (LSI) dan Latent Dirichlet Allocation (LDA)!

(b) Apa persamaan dan perbedaan antara LSI, LDA, dan *autoencoder*?

12.3. Variational Autoencoder

Jelaskan apa itu *variational autoencoder*! Deskripsikan perbedaannya dengan *autoencoder* yang sudah dijelaskan pada bab ini?

13

Arsitektur Neural Network

> "As students cross the threshold from outside to insider, they also cross the threshold from superficial learning motivated by grades to deep learning motivated by engagement with questions. Their transformation entails an awakening–even, perhaps, a falling in love."
>
> John C. Bean

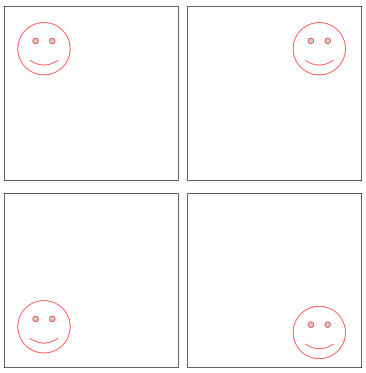
Seperti yang sudah dijelaskan pada bab 9 dan bab 12, data memiliki karakteristik (dari segi struktur) misal *sequential data*, *compositional data*, dsb. Terdapat arsitektur khusus *artificial neural network* (ANN) untuk menyelesaikan persoalan pada tipe data tertentu. Pada bab ini, kami akan memberikan beberapa contoh variasi arsitektur ANN yang cocok untuk tipe data tertentu. Penulis akan berusaha menjelaskan semaksimal mungkin ide-ide penting pada masing-masing arsitektur. Tujuan bab ini adalah memberikan pengetahuan konseptual (intuisi). Pembaca harus mengeksplorasi tutorial pemrograman untuk mampu mengimplementasikan arsitektur-arsitektur ini. Penjelasan pada bab ini bersifat abstrak dan kamu harus mengerti penjelasan bab-bab sebelumnya untuk mengerti konsep pada bab ini.

13.1 Convolutional Neural Network

Subbab ini akan memaparkan **ide utama** dari *convolutional neural network* (CNN) berdasarkan *paper* asli dari LeCun dan Bengio [76] (saat buku ini ditulis sudah ada banyak variasi). CNN memiliki banyak istilah dari bidang pemrosesan gambar (karena dicetuskan dari bidang tersebut), tetapi demi

mempermudah pemahaman intuisi CNN, diktat ini akan menggunakan istilah yang lebih umum juga.

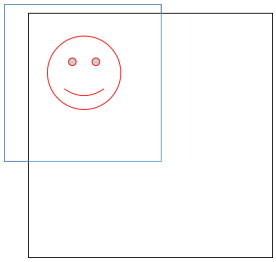
Sekarang, mari kita memasuki cerita CNN dari segi pemrosesan gambar. Objek bisa saja terlatak pada berbagai macam posisi seperti diilustrasikan oleh Gambar. 13.1. Selain tantangan variasi posisi objek, masih ada juga tantangan lain seperti rotasi objek dan perbedaan ukuran objek (*scaling*). Kita ingin mengenali (memproses) objek pada gambar pada berbagai macam posisi yang mungkin (***translation invariance***). Salah satu cara yang mungkin adalah dengan membuat suatu mesin pembelajaran (ANN) untuk regional tertentu seperti pada Gambar. 13.2 (warna biru) kemudian meng-*copy* mesin pembelajaran untuk mampu mengenali objek pada regional-regional lainnya. Akan tetapi, kemungkinan besar ANN *copy* memiliki konfigurasi parameter yang sama dengan ANN awal. Hal tersebut disebabkan objek memiliki informasi prediktif (*predictive information–feature vector*) yang sama yang berguna untuk menganalisisnya. Dengan kata lain, objek yang sama (*smiley*) memiliki pola *feature vector* yang mirip walaupun posisinya digeser-geser. ANN (MLP) bisa juga mempelajari prinsip *translation invariance*, tetapi memerlukan jauh lebih banyak parameter dibanding CNN (subbab berikutnya secara lebih matematis) yang memang didesain dengan prinsip *translation invariance* ("*built-in*").



Gambar 13.1: Motivasi *convolutional neural network*.

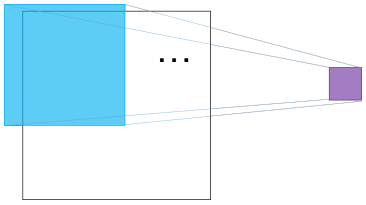
13.1.1 Convolution

Seperti yang sudah dijelaskan, motivasi CNN adalah untuk mampu mengenali aspek yang informatif pada regional tertentu (lokal). Dibanding meng-*copy* mesin pembelajaran beberapa kali untuk mengenali objek pada banyak regional, ide lebih baik adalah untuk menggunakan ***sliding window***. Setiap



Gambar 13.2: Motivasi *convolutional neural network*, solusi regional.

operasi pada *window*[1] bertujuan untuk mencari aspek lokal yang paling informatif. Ilustrasi diberikan oleh Gambar. 13.3. Warna biru merepresentasikan satu *window*, kemudian kotak ungu merepresentasikan aspek lokal paling informatif (disebut ***filter***) yang dikenali oleh *window*. Dengan kata lain, kita mentransformasi suatu *window* menjadi suatu nilai numerik (*filter*). Kita juga dapat mentransformasi suatu *window* (regional) menjadi d nilai numerik (d-*channels*, setiap elemen berkorespondensi pada suatu *filter*). *Window* ini kemudian digeser-geser sebanyak T kali, sehingga akhirnya kita mendapatkan vektor dengan panjang $d \times T$. Keseluruhan operasi ini disebut sebagai ***convolution***.[2]



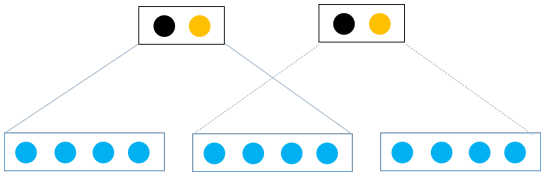
Gambar 13.3: *Sliding window*.

Agar kamu lebih mudah memahami prinsip ini, kami berikan contoh dalam bentuk 1-D pada Gambar. 13.4. Warna biru merepresentasikan *feature vector* (regional) untuk suatu *input* (e.g., regional pada suatu gambar, kata pada kalimat, dsb). Pada contoh ini, setiap 2 *input* ditransformasi menjadi vektor berdimensi 2 (2-*channels*); menghasilkan vektor berdimensi 4 (2 *window* $\times$ 2).

Pada contoh sebelumnya, kita menggunakan *window* selebar 2, satu *window* mencakup 2 data lokal; i.e., $window_1 = (x_1, x_2)$, $window_2 = (x_2, x_3)$, ... ; untuk suatu *input* $\mathbf{x}$. Kita juga dapat mempergunakan ***stride*** sebesar s,

[1] Dikenal juga sebagai *receptive field*.

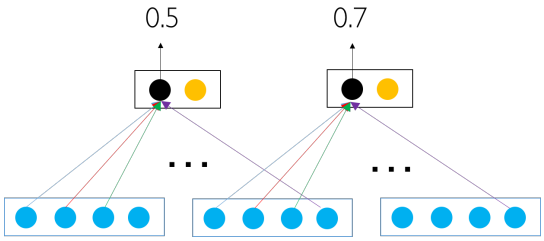
[2] Istilah *convolution* yang diterangkan pada konteks *machine learning* memiliki arti yang berbeda pada bidang *signal processing*.



Gambar 13.4: *1D Convolution.*

yaitu seberapa banyak data yang digeser untuk *window* baru. Contoh yang diberikan memiliki *stride* sebesar satu. Apabila kita memiliki *stride*=2, maka kita menggeser sebanyak 2 data setiap langkah; i.e., $window_1 = (x_1, x_2)$, $window_2 = (x_3, x_4)$, $\cdots$.

Selain *sliding window* dan *filter*, *convolutional layer* juga mengadopsi prinsip *weight sharing*. Artinya, *synapse weights* untuk suatu filter adalah sama walau *filter* tersebut dipergunakan untuk berbagai *window*. Sebagai ilustrasi, perhatikan Gambar. 13.5, warna yang sama pada *synapse weights* menunjukkan *synapse weights* bersangkutan memiliki nilai (*weight*) yang sama. Tidak hanya pada *filter* hitam, hal serupa juga terjadi pada *filter* berwarna oranye (i.e., *filter* berwarnya oranye juga memenuhi prinsip *weight sharing*). Walaupun memiliki konfigurasi bobot *synapse weights* yang sama, unit dapat menghasilkan *output* yang berbeda untuk *input* yang berbeda. Konsep *weight sharing* ini sesuai dengan cerita sebelumnya bahwa konfigurasi parameter untuk mengenali karakteristik informatif untuk satu objek bernilai sama walau pada lokasi yang berbeda. Dengan *weight sharing*, parameter *neural network* juga menjadi lebih sedikit dibanding menggunakan *multilayer perceptron* (*feed-forward neural network*).

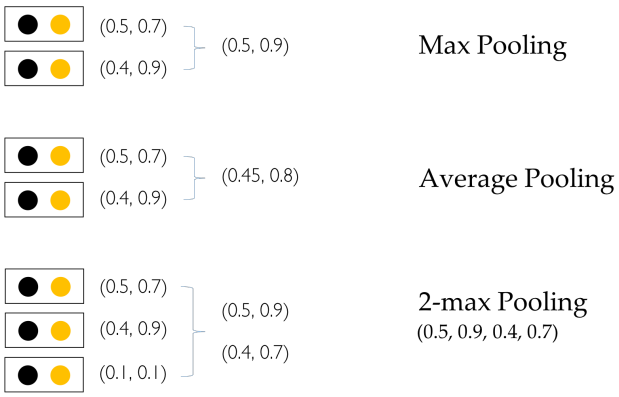


Gambar 13.5: Konsep *weight sharing.*

13.1.2 Pooling

Pada tahap *convolution*, kita merubah setiap *k-sized window* menjadi satu vektor berdimensi d (yang dapat disusun menjadi matriks $\mathbf{D}$). Semua vektor yang dihasilkan pada tahap sebelumnya dikombinasikan (*pooled*) menjadi

satu vektor **c**. Ide utamanya adalah mengekstrak informasi paling informatif (semacam meringkas). Ada beberapa teknik *pooling*, diantaranya: *max pooling*, *average pooling*, dan *K-max pooling*;[3] diilustrasikan pada Gambar. 13.6. *Max pooling* mencari nilai maksimum untuk setiap dimensi vektor. *Average pooling* mencari nilai rata-rata tiap dimensi. *K-max pooling* mencari K nilai terbesar untuk setiap dimensinya (kemudian hasilnya digabungkan). Gabungan operasi *convolution* dan *pooling* secara konseptual diilustrasikan pada Gambar. 13.7.



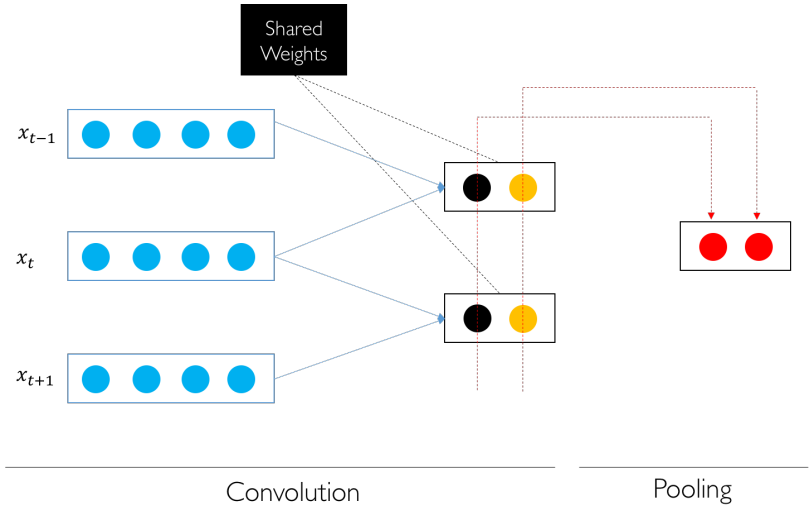
Gambar 13.6: Contoh *pooling*.

Setelah melewati berbagai operasi *convolution* dan *pooling*, kita akan memiliki satu vektor yang kemudian dilewatkan pada *multilayer perceptron* (*fully connected*) untuk melakukan sesuatu (tergantung permasalahan), misal klasifikasi gambar, klasifikasi sentimen, dsb (Ilustrasi pada Gambar. 13.8).

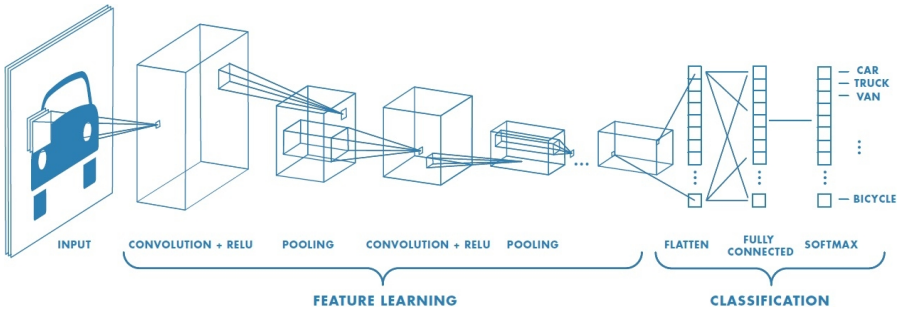
13.1.3 Rangkuman

Kemampuan utama *convolutional neural network* (CNN) adalah arsitektur yang mampu mengenali informasi prediktif suatu objek (gambar, teks, potongan suara, dsb) walaupun objek tersebut dapat diposisikan dimana saja pada *input*. Kontribusi CNN adalah pada *convolution* dan *pooling* layer. *Convolution* bekerja dengan prinsip *sliding window* dan *weight sharing* (mengurangi kompleksitas perhitungan). *Pooling layer* berguna untuk merangkum informasi informatif yang dihasilkan oleh suatu *convolution* (mengurangi dimensi). Pada ujung akhir CNN, kita lewatkan satu vektor hasil beberapa operasi *convolution* dan *pooling* pada *multilayer perceptron* (*feed-forward neural network*), dikenal juga sebagai ***fully connected layer***, untuk melakukan suatu pekerjaan, e.g., klasifikasi. Perhatikan, pada umumnya CNN tidak berdiri

[3] Kami ingin pembaca mengeksplorasi sendiri *dynamic pooling*.



Gambar 13.7: *Convolution* dan *pooling*.



Gambar 13.8: *Convolutional Neural Network*.[4]

sendiri, dalam artian CNN biasanya digunakan (dikombinasikan) pada arsitektur yang lebih besar.

13.2 Recurrent Neural Network

Ide dasar *recurrent neural network* (RNN) adalah membuat topologi jaringan yang mampu merepresentasikan data *sequential* (sekuensial) atau *time series* [77], misalkan data ramalan cuaca. Cuaca hari ini bergantung kurang lebih pada cuaca hari sebelumnya. Sebagai contoh apabila hari sebelumnya

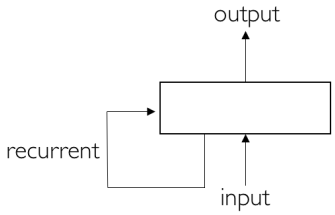
[4] mathworks.com

mendung, ada kemungkinan hari ini hujan.[5] Walau ada yang menganggap sifat data sekuensial dan *time series* berbeda, RNN berfokus sifat data dimana *instance* waktu sebelumnya $(t-1)$ mempengaruhi *instance* pada waktu berikutnya (t). Intinya, mampu mengingat *history*.

Secara lebih umum, diberikan sebuah sekuens *input* $\mathbf{x} = (x_1, \cdots, x_T)$. Data x_t (e.g., vektor, gambar, teks, suara) dipengaruhi oleh data sebelum-sebelumnya (*history*), ditulis sebagai $P(x_t \mid \{x_1, \cdots, x_{t-1}\})$. Kami harap kamu ingat kembali materi *markov assumption* yang diberikan pada bab 8. Pada *markov assumption*, diasumsikan bahwa data x_t (data point) hanya dipengaruhi oleh **beberapa data sebelumnya saja** (analogi: *windowing*). Setidaknya, asumsi ini memiliki dua masalah:

1. Menentukan *window* terbaik. Bagaimana cara menentukan banyaknya data sebelumnya (secara optimal) yang mempengaruhi data sekarang.
2. Apabila kita menggunakan *markov assumption*, artinya kita mengganggap informasi yang dimuat oleh data lama dapat direpresentasikan oleh data lebih baru, i.e., x_t juga memuat informasi $x_{t-J}, ..., x_{t-1}$; J adalah ukuran *window*. Penyederhanaan ini tidak jarang mengakibatkan informasi yang hilang.

RNN adalah salah satu bentuk arsitektur ANN untuk mengatasi masalah yang ada pada *markov assumption*. Ide utamanya adalah memorisasi,[6] kita ingin mengingat **keseluruhan** sekuens (dibanding *markov assumption* yang mengingat sekuens secara terbatas), implikasinya adalah RNN yang mampu mengenali dependensi yang panjang (misal x_t ternyata dependen terhadap x_1). RNN paling sederhana diilustrasikan pada Gambar. 13.9. Ide utamanya adalah terdapat *pointer* ke dirinya sendiri.



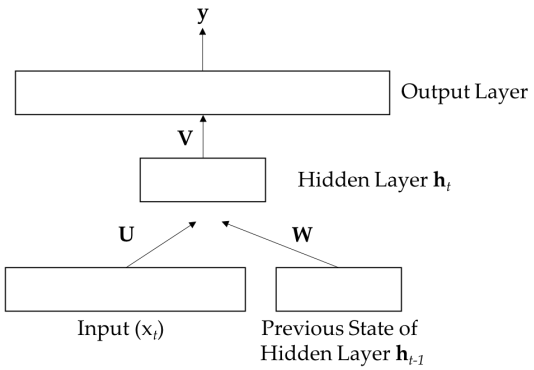
Gambar 13.9: Bentuk konseptual paling sederhana Recurrent Neural Network.

Ilustrasi Gambar. 13.9 mungkin sedikit susah dipahami karena berbentuk sangat konseptual. Bentuk lebih matematis diilustrasikan pada Gam-

[5] Mohon bertanya pada ahli meteorologi untuk kebenaran contoh ini. Contoh ini semata-mata pengalaman pribadi penulis.

[6] Tidak merujuk hal yang sama dengan *dynamic programming*.

bar. 13.10 [77]. Perhitungan *hidden state* pada waktu ke-t bergantung pada *input* pada waktu ke-t (x_t) dan *hidden state* pada waktu sebelumnya (h_{t-1}).



Gambar 13.10: Konsep Recurrent Neural Network.

Konsep ini sesuai dengan prinsip *recurrent* yaitu **mengingat** (memorisasi) kejadian sebelumnya. Kita dapat tulis kembali RNN sebagai persamaan 13.1.

$$\mathbf{h}_t = f(x_t, \mathbf{h}_{t-1}, b) \tag{13.1}$$

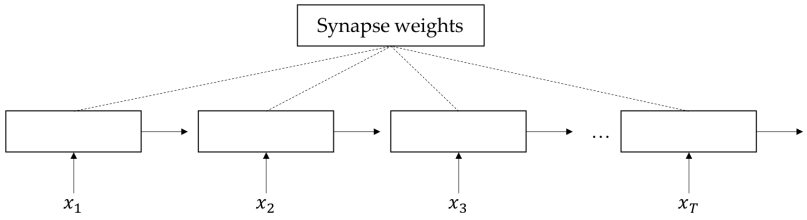
dimana f adalah fungsi aktivasi (non-linear, dapat diturunkan). Demi menyederhanakan penjelasan, penulis tidak mengikutsertakan *bias* (b) pada fungsi-fungsi berikutnya. Kami berharap pembaca selalu mengingat bahwa *bias* adalah parameter yang diikutsertakan pada fungsi *artificial neural network*. Fungsi f dapat diganti dengan variasi *neural network*,[7] misal menggunakan *long short-term memory network* (LSTM) [78]. Buku ini hanya akan menjelaskan konsep paling penting, silahkan eksplorasi sendiri variasi RNN.

Secara konseptual, persamaan 13.1 memiliki analogi dengan *full markov chain*. Artinya, *hidden state* pada saat ke-t bergantung pada semua *hidden state* dan *input* sebelumnya.

$$\begin{aligned}\mathbf{h}_t &= f(x_t, \mathbf{h}_{t-1}) \\ &= f(x_t, f(x_{t-1}, \mathbf{h}_{t-2})) \\ &= f(x_t, f(x_{t-1}, f(\{x_1, \cdots, x_{t-2}\}, \{\mathbf{h}_1, \cdots, \mathbf{h}_{t-3}\})))\end{aligned} \tag{13.2}$$

Training pada *recurrent neural network* dapat menggunakan metode *backpropagation*. Akan tetapi, metode tersebut kurang intuitif karena tidak mampu mengakomodasi *training* yang bersifat sekuensial *time series*. Untuk itu, terdapat metode lain bernama *backpropagation through time* [79].

[7] https://en.wikipedia.org/wiki/Recurrent_neural_network



Gambar 13.11: Konsep *feed forward* pada Recurrent Neural Network (RNN). Karena RNN menerima *input* berupa sekuens, kita memvisualisasikan proses *feed foward* dengan *unfolding* (atau *unrolling*) RNN pada keseluruhan sekuens input.

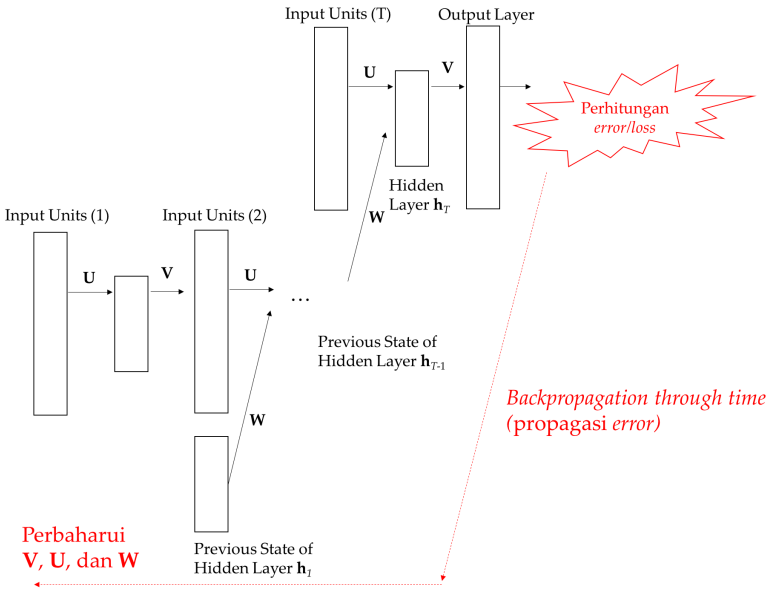
Sebagai contoh kita diberikan sebuah sekuens $\mathbf{x}$ dengan panjang T sebagai input, dimana x_t melambangkan input ke-i (**data point**), dapat berupa vektor, gambar, teks, atau apapun. Kita melakukan *feed forward* data tersebut ke RNN, diilustrasikan pada Gambar. 13.11. Perlu diingat, RNN mengadopsi prinsip *parameter sharing* (serupa dengan *weight sharing* pada CNN) dimana neuron yang sama diulang-ulang saat process *feed forward*. Setelah selesai proses *feed forward*, kita memperbaharui parameter (*synapse weights*) berdasarkan propagasi *error* (*backpropagation*). Pada *backpropagation* biasa, kita perbaharui parameter sambil mempropagasi *error* dari *hidden state* ke *hidden state* sebelumnya. Teknik melatih RNN adalah ***backpropagation through time*** yang melakukan *unfolding* pada *neural network*. Kita mengupdate parameter saat kita sudah mencapai *hidden state* paling awal. Hal ini diilustrasikan pada Gambar. 13.12.[8] Gambar. 13.12 dapat disederhanakan menjadi bentuk lebih abstrak (konseptual) pada Gambar. 13.13.

Kita mempropagasi *error* dengan adanya efek dari *next states of hidden layer*. *Synapse weights* diperbaharui secara *large update*. *Synapse weight* tidak diperbaharui per *layer*. Hal ini untuk merepresentasikan *neural network* yang mampu mengingat beberapa kejadian masa lampau dan keputusan saat ini dipengaruhi oleh keputusan pada masa lampau juga (ingatan). Untuk mengerti proses ini secara praktikal (dapat menuliskannya sebagai program), penulis sarankan pembaca untuk melihat materi tentang ***computation graph***[9] dan disertasi PhD oleh Mikolov [47].

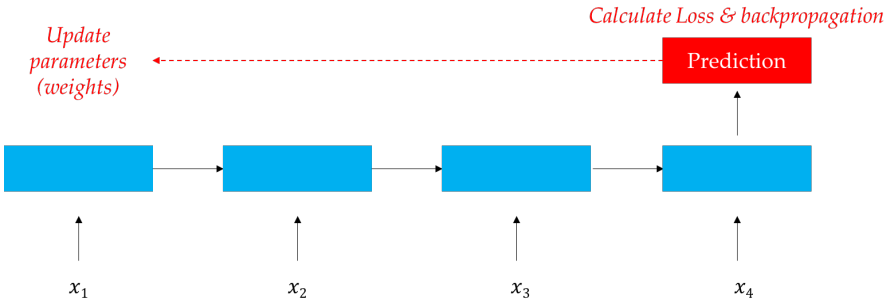
Walaupun secara konseptual RNN dapat mengingat seluruh kejadian sebelumnya, hal tersebut sulit untuk dilakukan secara praktikal untuk sekuens yang panjang. Hal ini lebih dikenal dengan *vanishing* atau *exploding gradient problem* [63, 80, 81]. Seperti yang sudah dijelaskan, ANN dan variasi arsitekturnya dilatih menggunakan teknik *stochastic gradient descent* (*gradient-based optimization*). Artinya, kita mengandalkan propagasi *error* berdasarkan tu-

[8] Prinsip ini mirip dengan *weight sharing*.

[9] https://www.coursera.org/learn/neural-networks-deep-learning/lecture/4WdOY/computation-graph



Gambar 13.12: Konsep *backpropagation through time* [47].



Gambar 13.13: Konsep *backpropagation through time* [1]. Persegi berwarna merah umumnya melambangkan *multi-layer perceptron*.

runan. Untuk sekuens *input* yang panjang, tidak jarang nilai *gradient* menjadi sangat kecil dekat dengan 0 (*vanishing*) atau sangat besar (*exploding*). Ketika pada satu *hidden state* tertentu, gradient pada saat itu mendekati 0, maka nilai tersebut yang dipropagasikan pada langkah berikutnya menjadi semakin kecil. Hal serupa terjadi untuk nilai gradient yang besar.

Berdasarkan pemaparan ini, RNN adalah teknik untuk merubah suatu sekuens *input*, dimana x_t merepresentasikan data ke-t (e.g., vektor, gambar, teks) menjadi sebuah *output* vektor $\mathbf{y}$. Vektor $\mathbf{y}$ dapat digunakan un-

tuk permasalahan lebih lanjut (buku ini memberikan contoh *sequence to sequence* pada subbab 13.4). Bentuk konseptual ini dapat dituangkan pada persamaan 13.3. Biasanya, nilai $\mathbf{y}$ dilewatkan kembali ke sebuah *multi-layer perceptron* (MLP) dan fungsi softmax untuk melakukan klasifikasi akhir (*final output*) dalam bentuk probabilitas, seperti pada persamaan 13.4.

$$\mathbf{y} = \text{RNN}(x_1, \cdots, x_N) \tag{13.3}$$

$$\text{final output} = \text{softmax}(\text{MLP}(\mathbf{y})) \tag{13.4}$$

Perhatikan, arsitektur yang penulis deskripsikan pada subbab ini adalah arsitektur paling dasar. Untuk arsitektur *state-of-the-art*, kamu dapat membaca *paper* yang berkaitan.

13.3 Part-of-speech Tagging Revisited

Pada bab sebelumnya, kamu telah mempelajari konsep dasar *recurrent neural network*. Selain digunakan untuk klasifikasi (i.e., *hidden state* terakhir digunakan sebagai *input* klasifikasi), RNN juga dapat digunakan untuk memprediksi sekuens seperti persoalan *part-of-speech tagging* (POS *tagging*) [82, 83, 84]. Kami harap kamu masih ingat materi bab 8 yang membahas apa itu persoalan POS *tagging*.

Diberikan sebuah sekuens kata $\mathbf{x} = \{x_1, \cdots, x_T\}$, kita ingin mencari sekuens *output* $\mathbf{y} = \{y_1, \cdots, y_T\}$ (*sequence prediction*); dimana y_i adalah kelas kata untuk x_i. Perhatikan, panjang *input* dan *output* adalah sama. Ingat kembali bahwa pada persoalan POS *tagging*, kita ingin memprediksi suatu kelas kata yang cocok y_i dari kumpulan kemungkinan kelas kata C ketika diberikan sebuah *history* seperti diilustrasikan oleh persamaan 13.5, dimana t_i melambangkan kandidat POS *tag* ke-i. Pada kasus ini, biasanya yang dicari tahu setiap langkah (*unfolding*) adalah probabilitas untuk memilih suatu kelas kata $t \in C$ sebagai kelas kata yang cocok untuk di-*assign* sebagai y_i.

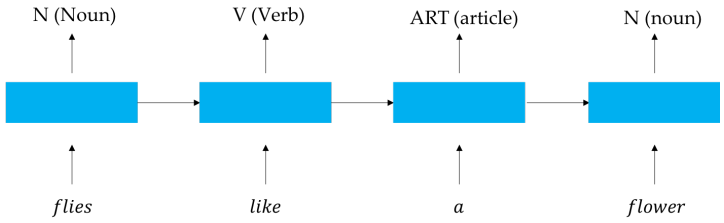
Ilustrasi diberikan oleh Gambar. 13.14.

$$y_1, \cdots, y_T = \underset{t_1, \cdots, t_T; t_i \in C}{\arg\max} P(t_1, \cdots, t_T \mid x_1, \cdots, x_T) \tag{13.5}$$

Apabila kita melihat secara sederhana (*markov assumption*), hal ini tidak lain dan tidak bukan adalah melakukan klasifikasi untuk setiap *instance* pada sekuens *input* (persamaan 13.6). Pada setiap *time step*, kita ingin menghasilkan *output* yang bersesuaian.

$$y_i = \underset{t_i \in C}{\arg\max} P(t_i | x_i) \tag{13.6}$$

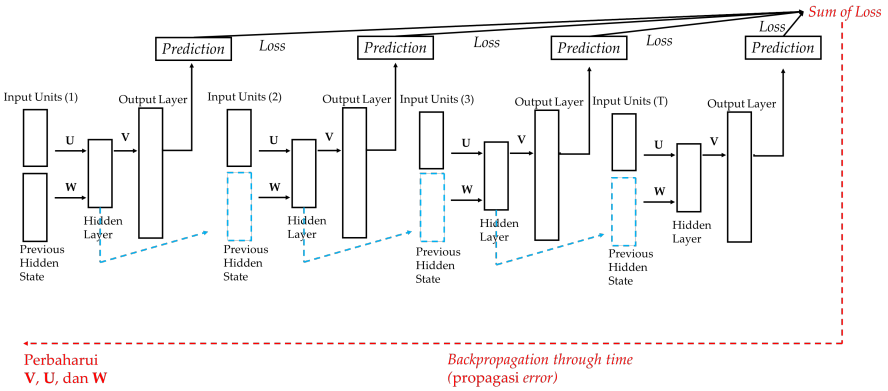
Akan tetapi, seperti yang sudah dibahas sebelum sebelumnya, *markov assumption* memiliki kelemahan. Kelemahan utama adalah tidak menggunakan



Gambar 13.14: POS *tagging* menggunakan Recurrent Neural Network.

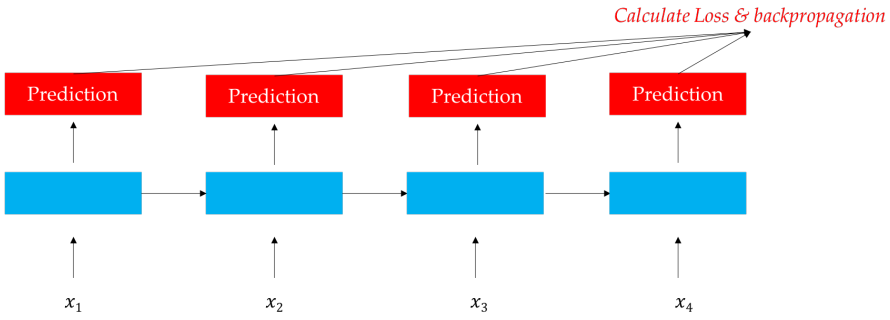
keseluruhan *history*. Persoalan ini cocok untuk diselesaikan oleh RNN karena kemampuannya untuk mengingat seluruh sekuens (berbeda dengan *hidden markov model* (HMM) yang menggunakan *markov assumption*). Secara teoritis (dan juga praktis–sejauh yang penulis ketahui.) Pada banyak persoalan, RNN menghasilkan performa yang lebih baik dibanding HMM. Tetapi hal ini bergantung juga pada variasi arsitektur. Dengan ini, persoalan POS *tagging* (*full history*) diilustrasikan oleh persamaan 13.7.

$$y_i = \underset{t_i \in C}{\arg\max}\, P(t_i | x_1, \cdots, x_T) \tag{13.7}$$



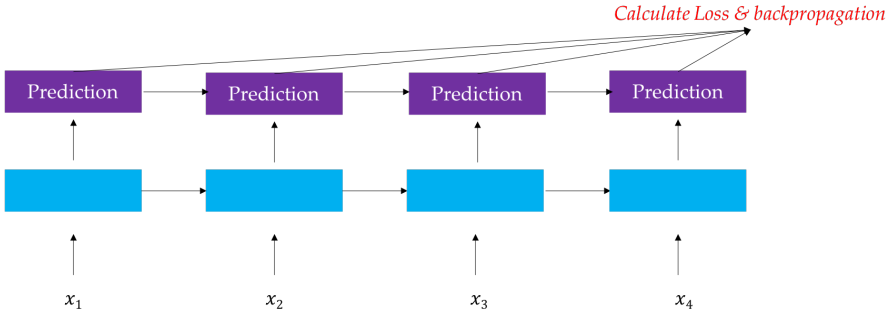
Gambar 13.15: *Sequence prediction* menggunakan Recurrent Neural Network.

Pada bab sebelumnya, kamu diberikan contoh persoalan RNN untuk satu *output*; i.e., diberikan sekuens *input*, *output*-nya hanyalah satu kelas yang mengkategorikan seluruh sekuens *input*. Untuk persoalan POS *tagging*, kita harus sedikit memodifikasi RNN untuk menghasilkan *output* bagi setiap elemen sekuens *input*. Hal ini dilakukan dengan cara melewatkan setiap *hidden layer* pada RNN pada suatu jaringan (anggap sebuah MLP+softmax).



Gambar 13.16: *Sequence prediction* menggunakan RNN (disederhakan) [1]. Persegi berwarna merah umumnya melambangkan *multi-layer perceptron*.

Kita lakukan prediksi kelas kata untuk setiap elemen sekuens *input*, kemudian menghitung *loss* untuk masing-masing elemen. Seluruh *loss* dijumlahkan untuk menghitung *backpropagation* pada RNN. Ilustrasi dapat dilihat pada Gambar. 13.15. Tidak hanya untuk persoalan POS *tagging*, arsitektur ini dapat juga digunakan pada persoalan *sequence prediction* lainnya seperti *named entity recognition*.[10] Gambar. 13.15 mungkin agak sulit untuk dilihat, kami beri bentuk lebih sederhananya (konseptual) pada Gambar. 13.16. Pada setiap langkah, kita menentukan POS *tag* yang sesuai dan menghitung *loss* yang kemudian digabungkan. *Backpropagation* dilakukan dengan mempertimbangkan keseluruhan (jumlah) *loss* masing-masing prediksi.



Gambar 13.17: *Sequence prediction* menggunakan Recurrent Neural Network (disederhakan), dimana prediksi saat waktu ke-t dipengaruhi oleh hasil prediksi pada waktu $t-1$.

Berdasarkan arsitektur yang sudah dijelaskan sebelumnya, prediksi POS *tag* ke-i bersifat independen dari POS *tag* lainnya. Padahal, POS *tag* lain-

[10] https://en.wikipedia.org/wiki/Named-entity_recognition

nya memiliki pengaruh saat memutuskan POS *tag* ke-i (ingat kembali materi bab 8); sebagai persamaan 13.8.

$$y_i = \arg\max_{t_i \in C} P(t_i \mid y_1, \cdots, y_{i-1}, x_1, \cdots, x_i) \tag{13.8}$$

Salah satu strategi untuk menangani hal tersebut adalah dengan melewatkan POS *tag* pada sebuah RNN juga, seperti para persamaan 13.9 [1] (ilustrasi pada Gambar. 13.17). Untuk mencari keseluruhan sekuens terbaik, kita dapat menggunakan teknik *beam search* (detil penggunaan dijelaskan pada subbab berikutnya). RNN^{x} pada persamaan 13.9 juga lebih intuitif apabila diganti menggunakan *bidirectional RNN* (dijelaskan pada subbab berikutnya).

$$\begin{aligned} &P(t_i \mid y_1, \cdots, y_{i-1}, x_1, \cdots, x_i) = \\ &\text{softmax}(\text{MLP}([\text{RNN}^{\text{x}}(x_1, \cdots, x_i); \text{RNN}^{\text{tag}}(t_1, \cdots, t_{i-1})])) \end{aligned} \tag{13.9}$$

13.4 Sequence to Sequence

Pertama-tama, kami ingin mendeskripsikan kerangka ***conditioned generation***. Pada kerangka ini, kita ingin memprediksi sebuah kelas y_i berdasarkan kelas yang sudah di-hasilkan sebelumnya (*history* yaitu $y_1, \cdots, y_{i-1}$) dan sebuah *conditioning context* $\mathbf{c}$ (berupa vektor).

Arsitektur yang dibahas pada subbab ini adalah variasi RNN untuk permasalahan *sequence generation*.[11] Diberikan sekuens *input* $\mathbf{x} = (x_1, \cdots, x_T)$. Kita ingin mencari sekuens *output* $\mathbf{y} = (y_1, \cdots, y_M)$. Pada subbab sebelumnya, x_i berkorespondensi langsung dengan y_i, e.g., y_i adalah kelas kata (kategori) untuk x_i. Tetapi, pada permasalahan saat ini, x_i tidak langsung berkorespondensi dengan y_i. Setiap y_i dikondisikan oleh **seluruh** sekuens *input* $\mathbf{x}$; i.e., *conditioning context* dan *history* $\{y_1, \cdots, y_{i-1}\}$. Panjang sekuens *output* M tidak mesti sama dengan panjang sekuens *input* T. Permasalahan ini masuk ke dalam kerangka *conditioned generation* dimana keseluruhan *input* $\mathbf{x}$ dapat direpresentasikan menjadi sebuah vektor $\mathbf{c}$ (*coding*). Vektor $\mathbf{c}$ ini menjadi variabel pengkondisi untuk menghasilkan *output* $\mathbf{y}$.

Pasangan *input-output* dapat melambangkan teks bahasa X–teks bahasa Y (translasi), teks-ringkasan, kalimat-*paraphrase*, dsb. Artinya ada sebuah *input* dan kita ingin menghasilkan (*generate/produce*) sebuah *output* yang cocok untuk *input* tersebut. Hal ini dapat dicapai dengan momodelkan pasangan *input-output* $p(\mathbf{y} \mid \mathbf{x})$. Umumnya, kita mengasumsikan ada kumpulan parameter θ yang mengontrol *conditional probability*, sehingga kita transformasi *conditional probability* menjadi $p(\mathbf{y} \mid \mathbf{x}, \theta)$. *Conditional probability* $p(\mathbf{y} \mid \mathbf{x}, \theta)$ dapat difaktorkan sebagai persamaan 13.10. Kami harap kamu mampu membedakan persamaan 13.10 dan persamaan 13.5 (dan 13.8) dengan jeli. Sedikit

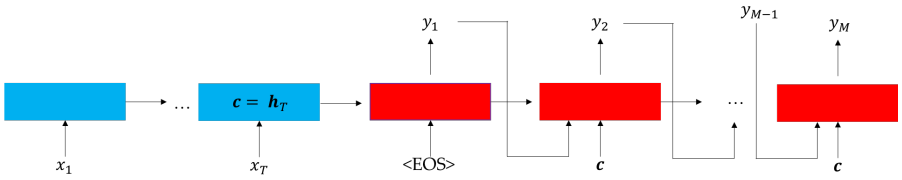
[11] Umumnya untuk bidang pemrosesan bahasa alami.

perbedaan pada formula menyebabkan makna yang berbeda. Objektif *training* adalah untuk meminimalkan *loss function*, sebagai contoh berbentuk *log likelihood* function diberikan pada persamaan 13.11, dimana $\mathbf{D}$ melambangkan *training data*.[12]

$$p(\mathbf{y} \mid \mathbf{x}, \theta) = \prod_{t=1}^{M} p(y_t \mid \{y_1, \ldots, y_{t-1}\}, \mathbf{x}, \theta), \tag{13.10}$$

$$L(\theta) = -\sum_{\{\mathbf{x},\mathbf{y}\} \in \mathbf{D}} \log p(\mathbf{y} \mid \mathbf{x}, \theta) \tag{13.11}$$

Persamaan 13.10 dapat dimodelkan dengan ***encoder-decoder*** model yang terdiri dari dua buah RNN dimana satu RNN sebagai *encoder*, satu lagi sebagai *decoder*. *Neural Network*, pada kasus ini, bertindak sebagai *controlling parameter* θ. Ilustrasi encoder-decoder dapat dilihat pada Gambar. 13.18. Gabungan RNN *encoder* dan RNN *decoder* ini disebut sebagai bentuk ***sequence to sequence***. Warna biru merepresentasikan *encoder* dan warna merah merepresentasikan *decoder*. "<EOS>" adalah suatu simbol spesial (*untuk praktikalitas*) yang menandakan bahwa sekuens *input* telah selesai dan saatnya berpindah ke *decoder*.



Gambar 13.18: Konsep *encoder-decoder* [81]. "<EOS>" adalah suatu simbol spesial (*untuk praktikalitas*) yang menandakan bahwa sekuens *input* telah selesai dan saatnya berpindah ke *decoder*.

Sebuah *encoder* merepresentasikan sekuens *input* $\mathbf{x}$ menjadi satu vektor $\mathbf{c}$.[13] Kemudian, *decoder* men-*decode* representasi $\mathbf{c}$ untuk menghasilkan (*generate*) sebuah sekuens *output* $\mathbf{y}$. Perhatikan, arsitektur kali ini berbeda dengan arsitektur pada subbab 13.3. *Encoder-decoder* (*neural network*) bertindak sebagai kumpulan parameter θ yang mengatur *conditional probability*. *Encoder-decoder* juga dilatih menggunakan prinsip *gradient-based optimization* untuk *tuning* parameter yang mengkondisikan *conditional probability* [81]. Dengan ini, persamaan 13.10 sudah didefinisikan sebagai *neural network* sebagai persamaan 13.12. "enc" dan "dec" adalah fungsi *encoder* dan *decoder*, yaitu sekumpulan transformasi non-linear.

[12] Ingat kembali materi *cross entropy*!
[13] Ingat kembali bab 12 untuk mengerti kenapa hal ini sangat diperlukan.

$$y_t = \text{dec}(\{y_1, \cdots, y_{t-1}\}, \text{enc}(\mathbf{x}), \theta) \tag{13.12}$$

Begitu model dilatih, *encoder-decoder* akan mencari *output* $\hat{\mathbf{y}}$ terbaik untuk suatu input $\mathbf{x}$, dillustrasikan pada persamaan 13.13. Masing-masing komponen *encoder-decoder* dibahas pada subbab-subbab berikutnya. Untuk abstraksi yang baik, penulis akan menggunakan notasi aljabar linear. Kami harap pembaca sudah familiar dengan representasi *neural network* menggunakan notasi aljabar linear seperti yang dibahas pada bab 11.

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y}} p(\mathbf{y} \mid \mathbf{x}, \theta) \tag{13.13}$$

13.4.1 Encoder

Seperti yang sudah dijelaskan, *encoder* mengubah sekuens *input* $\mathbf{x}$ menjadi satu vektor $\mathbf{c}$. Suatu data point pada sekuens *input* x_t (e.g., kata, gambar, suara, dsb) umumnya direpresentasikan sebagai *feature vector* $\mathbf{e}_t$. Dengan demikian, *encoder* dapat direpresentasikan dengan persamaan 13.14, dimana f adalah fungsi aktivasi non-linear; $\mathbf{U}$ dan $\mathbf{W}$ adalah matriks bobot (*weight matrices*–merepresentasikan *synapse weights*).

$$\begin{aligned} \mathbf{h}_t &= f(\mathbf{h}_{t-1}, \mathbf{e}_t) \\ &= f(\mathbf{h}_{t-1}\mathbf{U} + \mathbf{e}_t\mathbf{W}) \end{aligned} \tag{13.14}$$

Representasi *input* $\mathbf{c}$ dihitung dengan persamaan 13.15, yaitu sebagai *weighted sum* dari *hidden states* [57], dimana q adalah fungsi aktivasi non-linear. Secara lebih sederhana, kita boleh langsung menggunakan $\mathbf{h}_T$ sebagai konteks $\mathbf{c}$ [81] karena kita mengasumsikan $\mathbf{h}_T$ mengandung seluruh informasi yang ada di *input*.

$$\mathbf{c} = q(\{\mathbf{h}_1, \cdots, \mathbf{h}_T\}) \tag{13.15}$$

Walaupun disebut sebagai representasi keseluruhan sekuens *input*, informasi awal pada *input* yang panjang dapat hilang. Artinya $\mathbf{c}$ bisa saja memuat lebih banyak informasi *input* ujung-ujung akhir. Salah satu strategi yang dapat digunakan adalah dengan membalik (*reversing*) sekuens *input*. Sebagai contoh, *input* $\mathbf{x} = (x_1, \cdots, x_T)$ dibalik menjadi $(x_T, \cdots, x_1)$ agar bagian awal $(\cdots, x_2, x_1)$ lebih dekat dengan *decoder* [81]. Informasi yang berada dekat dengan *decoder* cenderung lebih diingat. Kami ingin pembaca mengingat bahwa teknik ini pun tidaklah sempurna.

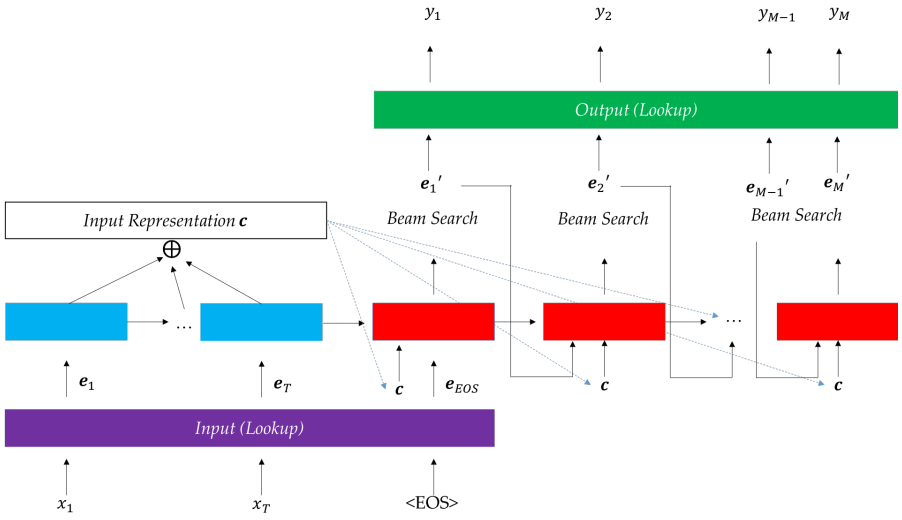
13.4.2 Decoder

Seperti yang sudah dijelaskan sebelumnya, *encoder* memproduksi sebuah vektor $\mathbf{c}$ yang merepresentasikan sekuens *input*. *Decoder* menggunakan representasi ini untuk memproduksi (*generate*) sebuah sekuens *output* $\mathbf{y}$ =

$(y_1, \cdots, y_M)$, disebut sebagai proses ***decoding***. Mirip dengan *encoder*, kita menggunakan RNN untuk menghasilkan *output* seperti diilustrasikan pada persamaan 13.16, dimana f merepresentasikan fungsi aktivasi non-linear; $\mathbf{H}$, $\mathbf{E}$, dan $\mathbf{C}$ merepresentasikan *weight matrices*. *Hidden state* $\mathbf{h}'_t$ melambangkan distribusi probabilitas suatu objek (e.g., POS tag, kelas kata yang **berasal dari suatu himpunan**) untuk menjadi *output* y_t. Umumnya, y_t adalah dalam bentuk *feature-vector* $\mathbf{e}'_t$.

$$\begin{aligned} \mathbf{h}'_t &= f(\mathbf{h}'_{t-1}, \mathbf{e}'_{t-1}, \mathbf{c}) \\ &= f(\mathbf{h}'_{t-1}\mathbf{H} + \mathbf{e}'_{t-1}\mathbf{E} + \mathbf{c}\mathbf{C}) \end{aligned} \tag{13.16}$$

Dengan penjelasan ini, mungkin pembaca berpikir Gambar. 13.18 tidak lengkap. Kamu benar! Penulis sengaja memberikan gambar simplifikasi. Gambar lebih lengkap (dan lebih nyata) diilustrasikan pada Gambar. 13.19.



Gambar 13.19: Konsep *encoder-decoder* (*full*).

Kotak berwarna ungu dan hijau dapat disebut sebagai *lookup matrix* atau *lookup table*. Tugas mereka adalah mengubah *input* x_t menjadi bentuk *feature vector*-nya (e.g., *word embedding*) dan mengubah $\mathbf{e}'_t$ menjadi y_t (e.g., *word embedding* menjadi kata). Komponen "*Beam Search*" dijelaskan pada subbab berikutnya.

13.4.3 Beam Search

Kita ingin mencari sekuens *output* yang memaksimalkan nilai probabilitas pada persamaan 13.13. Artinya, kita ingin mencari *output* terbaik. Pada su-

atu tahapan *decoding*, kita memiliki beberapa macam kandidat objek untuk dijadikan *output*. Kita ingin mencari sekuens objek sedemikian sehingga probabilitas akhir sekuens objek tersebut bernilai terbesar sebagai *output*. Hal ini dapat dilakukan dengan algoritma *Beam Search*.[14]

```
beamSearch(problemSet, ruleSet, memorySize)
        openMemory = new memory of size memorySize
        nodeList = problemSet.listOfNodes
        node = root or initial search node
        add node to OpenMemory;
        while(node is not a goal node)
                delete node from openMemory;
                expand node and obtain its children, evaluate those children;
                if a child node is pruned according to a rule in ruleSet, delete it;
                place remaining, non-pruned children into openMemory;
                if memory is full and has no room for new nodes, remove the worst
                        node, determined by ruleSet, in openMemory;
        node = the least costly node in openMemory;
```

Gambar 13.20: *Beam Search*.[15]

Secara sederhana, algoritma *Beam Search* mirip dengan algoritma Viterbi yang sudah dijelaskan pada bab 8, yaitu algoritma untuk mencari sekuens dengan probabilitas tertinggi. Perbedaannya terletak pada *heuristic*. Untuk menghemat memori komputer, algoritma *Beam Search* melakukan ekspansi terbatas. Artinya mencari hanya beberapa (B) kandidat objek sebagai sekuens berikutnya, dimana beberapa kandidat objek tersebut memiliki probabilitas $P(y_t \mid y_{t-1})$ terbesar. B disebut sebagai *beam-width*. Algoritma *Beam Search* bekerja dengan prinsip yang mirip dengan *best-first search* (*best-B search*) yang sudah kamu pelajari di kuliah algoritma atau pengenalan kecerdasan buatan.[16] Pseudo-code *Beam Search* diberikan pada Gambar. 13.20 (*direct quotation*).

13.4.4 Attention-based Mechanism

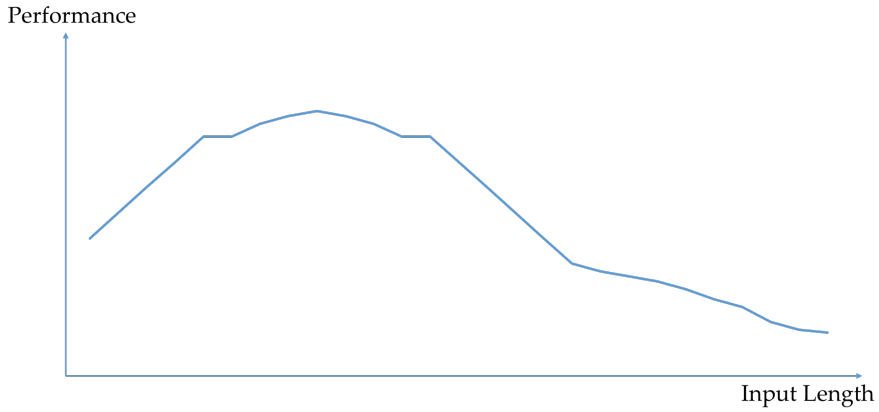
Seperti yang sudah dijelaskan sebelumnya, model *encoder-decoder* memiliki masalah saat diberikan sekuens yang panjang (*vanishing* atau *exploding gradient problem*). Kinerja model dibandingkan dengan panjang *input* kurang lebih dapat diilustrasikan pada Gambar. 13.21. Secara sederhana, kinerja model menurun seiring sekuens input bertambah panjang. Selain itu, representasi **c** yang dihasilkan *encoder* harus memuat informasi keseluruhan *input* walaupun sulit dilakukan. Ditambah lagi, *decoder* menggunakan representasinya **c** saja tanpa boleh melihat bagian-bagian khusus *input* saat *decoding*. Hal ini tidak sesuai dengan cara kerja manusia, misalnya pada kasus

[14] https://en.wikipedia.org/wiki/Beam_search

[15] https://en.wikibooks.org/wiki/Artificial_Intelligence/Search/Heuristic_search/Beam_search

[16] https://www.youtube.com/watch?v=j1H3jAAGlEA&t=2131s

translasi bahasa. Ketika mentranslasi bahasa, manusia melihat bolak-balik bagian mana yang sudah ditranslasi dan bagian mana yang sekarang (difokuskan) untuk ditranslasi. Artinya, manusia berfokus pada suatu bagian *input* untuk menghasilkan suatu translasi.

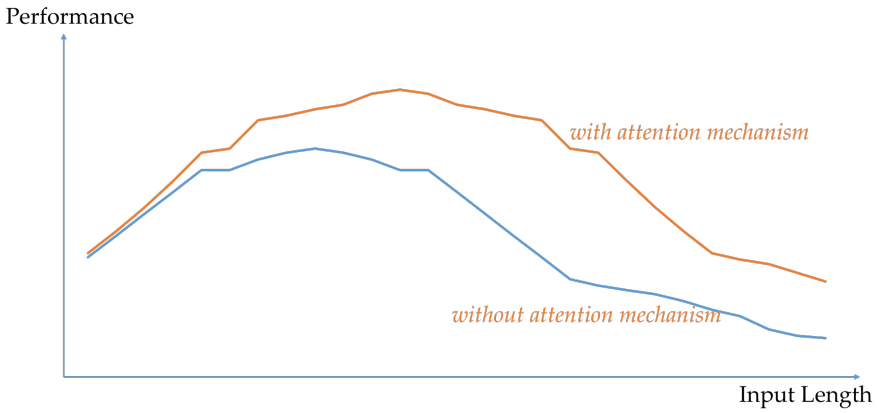


Gambar 13.21: Permasalahan *input* yang panjang.

Sudah dijelaskan sebelumnya bahwa representasi sekuens *input* $\mathbf{c}$ adalah sebuah *weighted sum*. $\mathbf{c}$ yang sama digunakan sebagai *input* bagi *decoder* untuk menentukan semua *output*. Akan tetapi, untuk suatu tahapan *decoding* (untuk *hidden state* $\mathbf{h}'_t$ tertentu), kita mungkin ingin model lebih berfokus pada bagian *input* tertentu daripada *weighted sum* yang sifatnya generik. Ide ini adalah hal yang mendasari ***attention mechanism*** [57, 58]. Ide ini sangat berguna pada banyak aplikasi pemrosesan bahasa alami. *Attention mechanism* dapat dikatakan sebagai suatu *soft alignment* antara *input* dan *output*. Mekanisme ini dapat membantu mengatasi permasalahan *input* yang panjang, seperti diilustrasikan pada Gambar. 13.22.

Dengan menggunakan *attention mechanism*, kita dapat mentransformasi persamaan 13.16 pada *decoder* menjadi persamaan 13.17, dimana $\mathbf{k}_t$ merepresentasikan seberapa (*how much*) *decoder* harus memfokuskan diri ke *hidden state* tertentu pada *encoder* untuk menghasilkan *output* saat ke-t. $\mathbf{k}_t$ dapat dihitung pada persamaan 13.18, dimana T merepresentasikan panjang *input*, $\mathbf{h}_i$ adalah *hidden state* pada *encoder* pada saat ke-i, $\mathbf{h}'_{t-1}$ adalah *hidden state* pada *decoder* saat ke $t-1$.

$$\mathbf{h}'_t = f'(\mathbf{h}'_{t-1}, \mathbf{e}'_{t-1}, \mathbf{c}, \mathbf{k}_t) \tag{13.17}$$

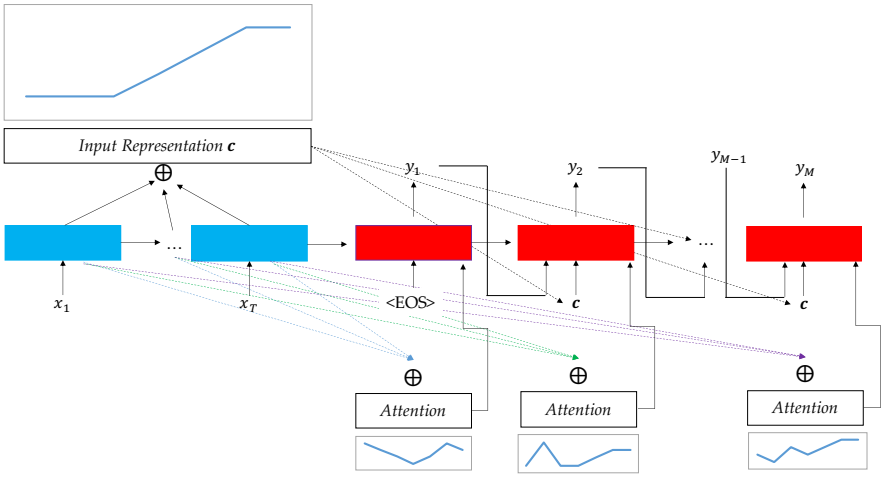


Gambar 13.22: Menggunakan vs tidak menggunakan *attention*.

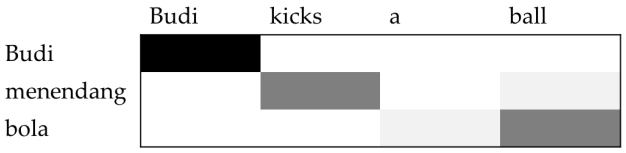
$$\begin{aligned} \mathbf{k}_t &= \sum_{i=1}^{T} \alpha_{t,i} \mathbf{h}_i \\ \alpha_{t,i} &= \frac{\exp(\mathbf{h}_i \cdot \mathbf{h}'_{t-1})}{\sum_{z=1}^{T} \exp(\mathbf{h}_z \cdot \mathbf{h}'_{t-1})} \end{aligned} \tag{13.18}$$

Sejatinya $\mathbf{k}_t$ adalah sebuah *weighted sum*. Berbeda dengan $\mathbf{c}$ yang bernilai sama untuk setiap tahapan *decoding*, *weight* atau bobot ($\alpha_{t,i}$) masing-masing *hidden state* pada *encoder* berbeda-beda untuk tahapan *decoding* yang berbeda. Perhatikan Gambar. 13.23 sebagai ilustrasi (lagi-lagi, bentuk *encoder-decoder* yang disederhanakan). Terdapat suatu bagian grafik yang menunjukkan distribusi bobot pada bagian *input representation* dan *attention*. Distribusi bobot pada *weighted sum* $\mathbf{c}$ adalah pembobotan yang bersifat generik, yaitu berguna untuk keseluruhan (rata-rata) kasus. Masing-masing *attention* (semacam *layer* semu) memiliki distribusi bobot yang berbeda pada tiap tahapan *decoding*. Walaupun *attention mechanism* sekalipun tidak sempurna, ide ini adalah salah satu penemuan yang sangat penting.

Seperti yang dijelaskan pada bab 11 bahwa *neural network* susah untuk dimengerti. *Attention mechanism* adalah salah satu cara untuk mengerti *neural network*. Contoh yang mungkin lebih mudah dipahami diberikan pada Gambar. 13.24 yang merupakan contoh kasus mesin translasi [57]. *Attention mechanism* mampu mengetahui *soft alignment*, yaitu kata mana yang harus difokuskan saat melakukan translasi bahasa (bagian *input* mana berbobot lebih tinggi). Dengan kata lain, *attention mechanism* memberi interpretasi kata pada *output* berkorespondensi dengan kata pada *input* yang mana. Sebagai informasi, menemukan cara untuk memahami (interpretasi) ANN adalah salah satu tren riset masa kini [56].



Gambar 13.23: *Encoder-decoder with attention.*



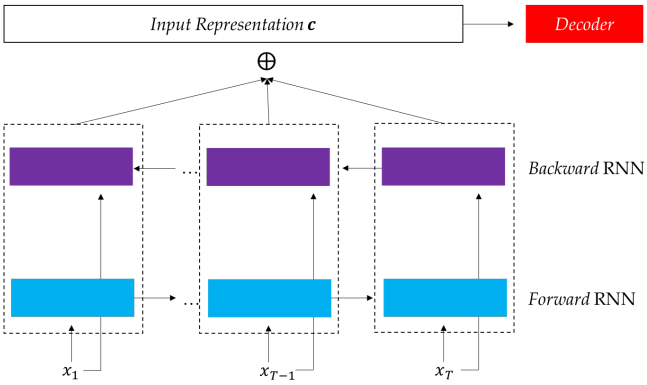
Gambar 13.24: *Attention mechanism* pada translasi bahasa [57]. Warna lebih gelap merepresentasikan bobot (fokus/*attention*) lebih tinggi. Sebagai contoh, kata "menendang" berkorespondensi paling erat dengan kata "kicks".

13.4.5 Variasi Arsitektur Sequence to Sequence

Selain RNN, kita juga dapat menggunakan *bidirectional* RNN (BiRNN) untuk mengikutsertakan pengaruh baik *hidden state* sebelum $(\mathbf{h}_1, \cdots, \mathbf{h}_{t-1})$ dan setelah $(\mathbf{h}_{t+1}, \cdots, \mathbf{h}_T)$ untuk menghitung *hidden state* sekarang $(\mathbf{h}_t)$ [85, 86, 87]. BiRNN menganggap $\mathbf{h}_t$ sebagai gabungan (*concatenation*) *forward hidden state* $\mathbf{h}_t^{\rightarrow}$ dan *backward hidden state* $\mathbf{h}_t^{\leftarrow}$, ditulis sebagai $\mathbf{h}_t = \mathbf{h}_t^{\rightarrow} + \mathbf{h}_t^{\leftarrow}$.[17] *Forward hidden state* dihitung seperti RNN biasa yang sudah dijelaskan pada subbab *encoder*, yaitu $\mathbf{h}_t^{\rightarrow} = f(\mathbf{h}_{t-1}^{\rightarrow}, \mathbf{e}_t)$. *Backward hidden state* dihitung dengan arah terbalik $\mathbf{h}_t^{\leftarrow} = f(\mathbf{h}_{t+1}^{\leftarrow}, \mathbf{e}_t)$. Ilustrasi *encoder-decoder* yang menggunakan BiRNN dapat dilihat pada Gambar. 13.25.

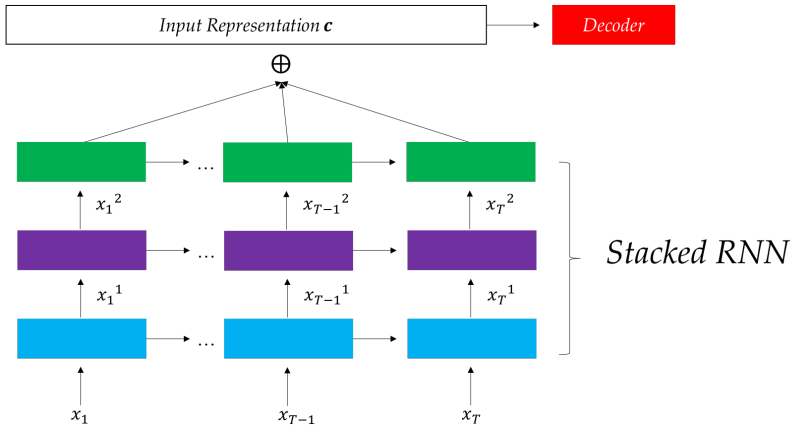
Selain variasi RNN menjadi BiRNN kita dapat menggunakan *stacked RNN* seperti pada Gambar. 13.26 dimana *output* pada RNN pertama bertindak sebagai *input* pada RNN kedua. *Hidden states* yang digunakan untuk menghasilkan representasi *encoding* adalah RNN pada tumpukan paling atas. Kita

[17] Perhatikan! + disini dapat diartikan sebagai penjumlahan atau konkatenasi



Gambar 13.25: *Encoder-decoder* dengan Bidirectional Recurrent Neural Network.

juga dapat menggunakan variasi *attention mechanism* seperti *neural checklist model* [88] atau *graph-based attention* [89]. Selain yang disebutkan, masih banyak variasi lain yang ada, silahkan eksplorasi lebih lanjut sendiri.



Gambar 13.26: *Encoder-decoder* dengan *stacked* Recurrent Neural Network.

13.4.6 Rangkuman

Sequence to sequence adalah salah satu bentuk *conditioned generation*. Artinya, menggunakan RNN untuk menghasilkan (*generate*) suatu sekuens *output* yang dikondisikan oleh variabel tertentu. Diktat ini memberikan contoh bagaimana menghasilkan suatu sekuens *output* berdasarkan sekuens *input* (*conditioned*

on a sequence of input). Selain *input* berupa sekuens, konsep ini juga dapat diaplikasikan pada bentuk lainnya. Misalnya, menghasilkan *caption* saat input yang diberikan adalah sebuah gambar [90]. Kita ubah *encoder* menjadi sebuah CNN (ingat kembali subbab 13.1) dan *decoder* berupa RNN [90]. Gabungan CNN-RNN tersebut dilatih bersama menggunakan metode *backpropagation*.

Perhatikan, walaupun memiliki kemiripan dengan *hidden markov model*, *sequence to sequence* bukanlah *generative model*. Pada *generative model*, kita ingin memodelkan *joint probability* $p(x, y) = p(y \mid x)p(x)$ (walaupun secara tidak langsung, misal menggunakan teori Bayes). *Sequence to sequence* adalah *discriminative model* walaupun *output*-nya berupa sekuens, ia tidak memodelkan $p(x)$, berbeda dengan *hidden markov model*. Kita ingin memodelkan *conditional probability* $p(y \mid x)$ secara langsung, seperti *classifier* lainnya (e.g., *logistic regression*). Jadi yang dimodelkan antara *generative* dan *discriminative model* adalah dua hal yang berbeda.

Pada subbab ini, penulis memberikan contoh *attention mechanism* yang beroperasi antara *encoder* dan *decoder*. Masih banyak variasi lainnya seperti *self-attention*, *multi-head attention* dan *hierarchical-attention* [91, 92]. Walaupun motivasi dan penggunaan variasi *attention mechanism* berbeda-beda, konsep dasarnya sama yaitu mengekstrak (atau mengambil) informasi dari bagian *network* lainnya.

13.5 Arsitektur Lainnya

Selain arsitektur yang sudah dipaparkan, masih banyak arsitektur lain baik bersifat generik (dapat digunakan untuk berbagai karakteristik data) maupun spesifik (cocok untuk data dengan karakteristik tertentu atau permasalahan tertentu) sebagai contoh, *Restricted Boltzman Machine*[18] dan *Generative Adversarial Network* (GAN).[19] Saat buku ini ditulis, GAN dan *adversarial training* sedang populer.

13.6 Architecture Ablation

Pada bab 9, kamu telah mempelajari *feature ablation*, yaitu memilih-milih elemen pada *input* (untuk dibuang), sehingga model memiliki kinerja optimal. Pada *neural network*, proses *feature engineering* mungkin tidak sepenting pada model-model yang sudah kamu pelajari sebelumnya (e.g., model linear) karena ia dapat memodelkan interaksi yang kompleks dari seluruh elemen *input*. Pada *neural network*, masalah yang muncul adalah memilih arsitektur yang tepat. Untuk menyederhanakan pencarian arsitektur, pada umumnya kita dapat mengganggap sebuah *neural network* tersusun atas beberapa

[18] https://deeplearning4j.org/restrictedboltzmannmachine
[19] https://deeplearning4j.org/generative-adversarial-network

"modul". Pembagian *neural network* menjadi modul adalah hal yang relatif. Untuk mencari tahu konfigurasi arsitektur yang memberikan performa maksimal, kita dapat melakukan *architecture ablation*. Idenya mirip dengan *feature ablation*, dimana kita mencoba mengganti-ganti bagian (modul) neural network. Sebagai contoh, ingat kembali arsitektur *sequence to sequence* dimana kita memiliki *encoder* dan *decoder*. Kita dapat menggunakan RNN, Bidirectional RNN, ataupun Stacked RNN sebagai *encoder*. Hal ini adalah salah satu contoh *architecture ablation*. Akan tetapi, bisa jadi kita mengasumsikan modul yang lebih kecil. Sebagai contoh, menggunakan RNN *encoder* pada *sequence to sequence* dan kita coba mengganti-ganti fungsi aktivasi.

Architecture ablation ini bisa menjadi semakin rumit tergantung persepsi kita tentang definisi modul pada *neural network*, seperti sampai menentukan jumlah *hidden layers* dan berapa jumlah unit pada masing-masing *layer*. Contoh lain adalah memilih fungsi aktivasi yang cocok untuk setiap *hidden layer*. Pada kasus ini "modul" kita adalah sebuah *layer*. Walaupun *neural network* memberikan kita kemudahan dari segi pemilihan fitur, kita memiliki kesulitan dalam menentukan arsitektur. Terlebih lagi, alasan pemilihan banyaknya *units* pada suatu *layer* (e.g., 512 dibanding 256 *units*) mungkin tidak dapat dijustifikasi dengan akurat. Pada *feature ablation*, kita dapat menjustifikasi alasan untuk menghilangkan suatu fitur. Pada *neural network*, kita susah menjelaskan alasan pemilihan arsitektur (dan konfigurasi parameter) karena *search space*-nya jauh lebih besar.

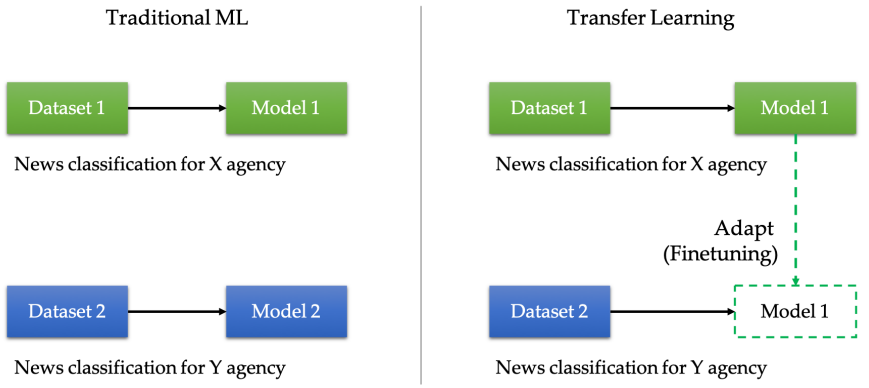
13.7 Transfer Learning

Walau konsep *transfer learning* (TL) tidak terbatas pada *neural network*, subbab ini membahas pemanfaatan TL paling umum pada *neural network*. Pembaca dipersilahkan mengeksplorasi lebih lanjut.

Bayangkan kondisi berikut. Ada dua orang, Haryanto dan Wira. Saat masih kecil, Wira pernah belajar cara memainkan Ukulele, sedangkan Haryanto tidak. Ketika kedua orang tersebut belajar memainkan gitar, menurutmu siapa yang bisa menguasai gitar lebih cepat?

Pada TL, kita ingin menggunakan suatu pengetahun (*knowledge*) pada suatu *task* T_1, untuk menyelesaikan permasalahan *task* T_2 [93, 94]. Kita memiliki asumsi bahwa T_1 memiliki kaitan dengan T_2, sedemikian sehingga fasih pada T_1 akan menyebabkan kita fasih pada T_2 (atau lebih fasih dibandingan tidak menguasai T_1 sama sekali). Perhatikan Gambar 13.27 yang mengilustrasikan perbedaan pembelajaran mesin biasa dan penggunaan TL. Pada pembelajaran mesin biasa, kita melatih model untuk masing-masing *task*. Pada TL, kita menggunakan model yang sudah ada, disebut ***pretrained model***, untuk *task* baru. Selain dimotivasi oleh kemiripan kedua *tasks*, TL juga dimotivasi oleh ketersediaan data. Misal dataset untuk *task* T_1 banyak, sedangkan untuk *task* T_2 sedikit. Berhubung T_1 dan T_2 memiliki kemiripan, model untuk

T_1 yang diadaptasi untuk T_2 akan konvergen lebih cepat dibanding melatih model dari awal untuk T_2.



Gambar 13.27: Pembelajaran mesin tradisional vs. menggunakan *transfer learning*.

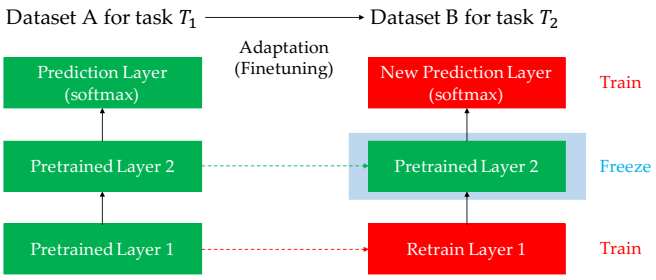
Gambar 13.28: Proses *transfer learning*.

Proses mengadaptasi suatu *pretrained model* disebut ***finetuning*** (Gambar 13.28). Pertama-tama kita ganti *layer* terakhir (*prediction layer*) pada *pretrained model* menggunakan *layer* baru yang diinisialisasi secara *random*.[20] Kemudian, kita latih kembali model yang sudah ada menggunakan data untuk T_2.

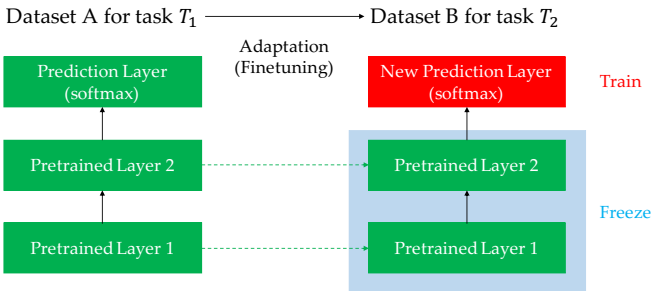
Secara umum, ada tiga cara untuk melakukan *finetuning*.

1. **Freeze some layers**. Kita *freeze* beberapa *layer* (parameternya tidak diperbaharui saat *finetuning*), kemudian latih *layer* lainnya. Ilustrasi diberikan pada Gambar 13.29.
2. **Train only new last layer**. Kita *freeze* semua *layer*, kecuali *layer* terakhir untuk task T_2. Ilustrasi diberikan pada Gambar 13.30
3. **Train all layers**. Setelah menggantu *layer terakhir*, kita latih semua *layer* untuk task T_2. Ilustrasi diberikan pada Gambar 13.31.

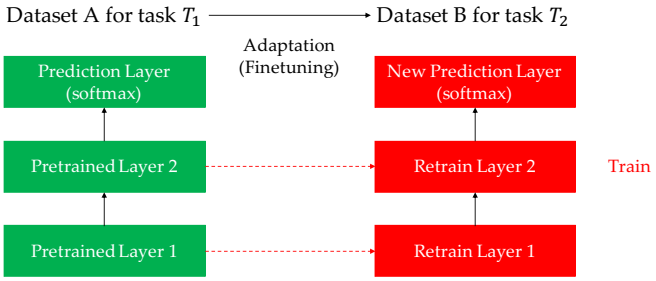
[20] Penulis rasa, hal ini hampir wajib hukumnya



Gambar 13.29: *Freeze some layers.*



Gambar 13.30: *Train only new last layer.*



Gambar 13.31: *Train all layers.*

Selain alasan yang sudah disebutkan, TL juga digunakan untuk mempercepat *training*. Konon *pre-trained* model pada umumnya membutuhkan waktu *training* yang lebih cepat (lebih sedikit iterasi) dibanding melatih model baru. Dengan demikian, kita dapat menghemat listrik dan mengurangi polusi CO_2. TL juga berkaitan erat dengan *successive learning* (bab 11), dimana

kita melatih arsitektur lebih kecil kemudian menggunakannya pada arsitektur yang lebih besar.

Demikian konsep paling dasar TL. Selebihkan, penulis menyarankan untuk membaca *paper* atau tutorial terkait, seperti dibawah berikut.

- https://www.cs.uic.edu/ liub/Lifelong-Learning-tutorial-slides.pdf
- https://www.aclweb.org/anthology/attachments/N19-5004.Presentation.pdf

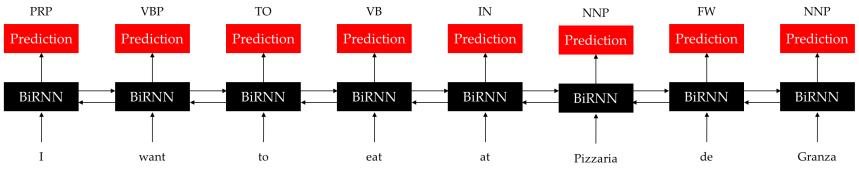
13.8 Multi-task Learning

Subbab ini akan menjelaskan *framework* melatih model pembelajaran mesin menggunakan *multi-task learning* (MTL). Walaupun konsep MTL tidak terbatas pada *neural network*, bab ini membahas konsep tersebut menggunakan arsitektur *neural network* sebagai contoh (karena itu dimasukkan ke dalam bab ini). Kami hanya memberikan penjelasan paling inti MTL menggunakan contoh yang sederhana.

Pada MTL, kita melatih model untuk mengerjakan beberapa hal yang mirip atau berkaitan, secara bersamaan. Misalnya, melatih model POS *tagging* dan *named-entity recognition* [95], mesin penerjemah untuk beberapa pasangan bahasa [96], klasifikasi teks [97] dan *discourse parsing* [98]. Karena model dilatih untuk beberapa permasalahan yang mirip (sejenis), kita berharap agar model mampu mendapatkan "intuisi" dasar yang dapat digunakan untuk menyelesaikan semua permasalahan. Perbedaan TL (dalam konteks pembahasan sebelumnya) dan MTL terletak pada *timing* pelatihan. Apabila pada TL, model untuk *task* T_1 dan T_2 dilatih pada waktu yang berbeda, sedangkan untuk MTL, dilatih bersamaan.

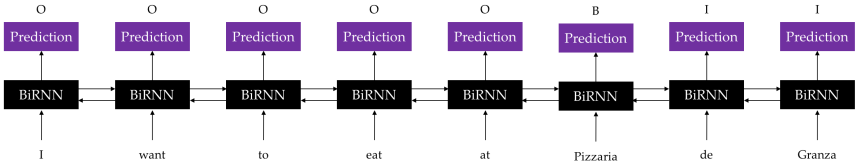
Perhatikan Gambar 13.32 yang merupakan ilustrasi permasalahan POS *tagging*. Diberikan *input* sekuens kata $\mathbf{x}$, kita ingin mencari sekuens *tag* $\mathbf{y}$ terbaik untuk melambangkan kelas tiap kata. Kami harap kamu masih ingat definisi permasalahan tersebut karena sudah dibahas pada bab-bab sebelumnya. Kita ingin memodelkan *conditional probability* $p(\mathbf{y} \mid \mathbf{x}, \theta)$. POS *tagging* adalah salah satu *sequence tagging task*, dimana setiap elemen *input* berkorespondensi dengan elemen *output*. Kita dapat melatih model BiRNN ditambah dengan MLP untuk melakukan prediksi kelas kata. Sebelumnya, telah dijelaskan bahwa BiRNN mungkin lebih intuitif untuk POS *tagging* dibanding RNN biasa. Hal ini karena kita dapat memodelkan "konteks" kata (*surrounding words*) dengan lebih baik, yaitu informasi dari kata sebelum dan sesudah (BiRNN), dibanding hanya mendapat informasi dari kata sebelum (RNN).

Sekarang kamu perhatikan Gambar 13.33 yang mengilustrasikan *named entity recognition task* (NER). *Named entity* secara sederhana adalah objek yang bernama, misal lokasi geografis, nama perusahaan, dan nama orang. Pada NER, kita ingin mengekstrasi *named entity* yang ada pada *input*. *Task* ini biasanya direpresentasikan dengan BIO *coding scheme*. Artinya, *output*



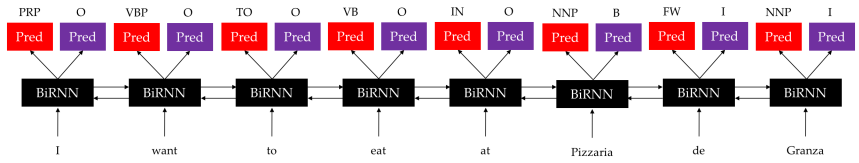
Gambar 13.32: POS *tagger*.

untuk NER adalah pilihan B (*begin*), I (*inside*) dan O (*outside*). Apabila suatu kata adalah kata pertama dari suatu *named entity*, kita mengasosiasikannya dengan *output* B. Apabila suatu kata adalah bagian dari *named entity*, tetapi bukan kata pertama, maka diasosiasikan dengan *output* I. Selain keduanya, diasosiasikan dengan *output* O. Seperti POS *tagging*, NER juga merupakan *sequence tagging* karena kita ingin memodelkan $p(\mathbf{y} \mid \mathbf{x}, \theta)$ untuk $\mathbf{x}$ adalah *input* dan $\mathbf{y}$ adalah *output* (BIO).



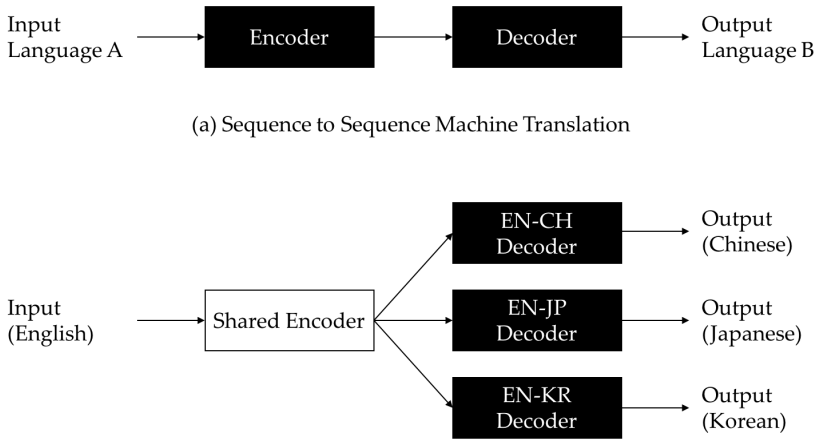
Gambar 13.33: *Named Entity Recognition*.

POS *tagging* dan NER dianggap sebagai *task* yang "mirip" karena keduanya memiliki cara penyelesaian masalah yang mirip. Selain dapat diselesaikan dengan cara yang mirip, kedua *task* tersebut memiliki *nature* yang sama. Dengan alasan ini, kita dapat melatih model untuk POS *tagging* dan NER dengan kerangka *multi-task learning*. Akan tetapi, menentukan apakah dua *task* memiliki *nature* yang mirip ibarat sebuah seni (butuh *sense*) dibanding *hard science* [1].



Gambar 13.34: *Multi-task Learning* untuk POS *tagging* dan *Named Entity Recognition*.

Ide utama MTL adalah melatih ***shared representation***. Sebagai ilustrasi, perhatikan Gambar 13.34. Sebelumnya, kita melatih dua model dengan BiRNN yang dilewatkan pada MLP. Pada saat ini, kita melatih BiRNN yang dianggap sebagai *shared representation*. BiRNN diharapkan memiliki "intuisi" untuk menyelesaikan kedua permasalahan, berhubung keduanya memiliki *nature* yang sama. Setiap *hidden layer* pada BiRNN dilewatkan pada MLP untuk melakukan prediksi pada masing-masing *task*. Tujuan utama MTL adalah untuk meningkatkan kinerja. Kita melatih model untuk *task* X dengan meminjam "intuisi" penyelesaikan dari *task* Y dengan harapan "intuisi" yang dibawa dari *task* Y dapat memberikan informasi tambahan untuk penyelesaian *task* X.



Gambar 13.35: *Multi-task Learning* pada mesin translasi.

Perhatikan contoh berikutnya tentang MTL pada mesin translasi (Gambar 13.35). Pada permasalahan mesin translasi, kita melatih model menggunakan data paralel kombinasi pasangan bahasa X-Y.

Penggunaan MTL pada mesin mesin translasi pada umumnya dimotivasi oleh dua alasan.

- Pada kombinasi pasangan bahasa tertentu, tersedia *dataset* dengan jumlah yang banyak. Tetapi, bisa jadi kita hanya memiliki *dataset* berukuran kecil untuk bahasa tertentu. Sebagai contoh, data mesin translasi untuk pasangan English-France lebih besar dibanding English-Indonesia. Karena kedua kombinasi pasangan bahasa memiliki *nature* yang cukup sama, kita dapat menggunakan MTL sebagai kompensasi data English-Indonesia

yang sedikit agar model pembelajaran bisa konvergen. Dalam artian, *encoder* yang dilatih menggunakan sedikit data kemungkinan memiliki performa yang kurang baik. Dengan ini, kita latih suatu *encoder* menggunakan data English-France dan English-Indonesia agar model bisa konvergen. Pada kasus ini, transfer learning juga dapat digunakan. Kita melatih model English-France, kemudian memasangkan *encoder* yang sudah dilatih dengan *decoder* baru untuk bahasa Indonesia.

- Seperti yang sudah dijelaskan sebelumnya, kita ingin menggunakan "intuisi" penyelesaian suatu permasalahan untuk permasalahan lainnya, berhubung solusinya keduanya mirip. Dengan hal ini, kita harap kita mampu meningkatkan kinerja model. Sebagai contoh, kombinasi pasangan bahasa English-Japanese dan English-Korean, berhubung kedua bahasa target memiliki struktur yang mirip.

Pada kerangka MTL, *utility function* atau objektif *training* adalah meminimalkan *joint loss* semua *tasks* (hal ini juga membedakan TL dan MTL), diberikan pada persamaan 13.19. Kita dapat mendefinisikan *loss* pada kerangka MTL sebagai penjumlahan *loss* pada masing-masing *task*, seperti pada persamaan 13.20. Apabila kita mengganggap suatu *task* lebih penting dari *task* lainnya, kita dapat menggunakan *weighted sum*, seperti pada persamaan 13.21. Kita juga dapat menggunakan *dynamic weighting* untuk memperhitungkan *uncertainty* pada tiap *task*, seperti pada persamaan 13.22 [99], dimana σ melambangkan varians *task-specific loss*.

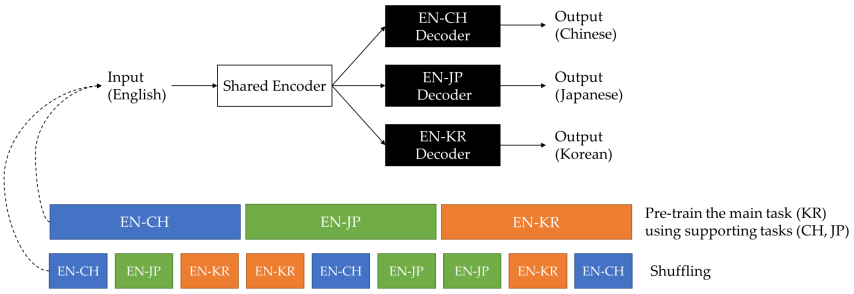
$$\mathcal{L}_{\text{MTL}} = q(\mathcal{L}_{T_1}, ...\mathcal{L}_{T_D}) \tag{13.19}$$

$$q(\mathcal{L}_{T_1}, ...\mathcal{L}_{T_D}) = \sum_{i}^{D} \mathcal{L}_{T_i} \tag{13.20}$$

$$q(\mathcal{L}_{T_1}, ...\mathcal{L}_{T_D}) = \sum_{i}^{D} \alpha_i \mathcal{L}_{T_i} \tag{13.21}$$

$$q(\mathcal{L}_{T_i}, ...\mathcal{L}_{T_D}) = \sum_{i}^{D} \frac{1}{2\sigma_i^2} \mathcal{L}_{T_i} + \ln(\sigma_i) \tag{13.22}$$

Saat melatih MTL, tujuan *training* dapat mempengaruhi proses penyajian data. Seumpama saat melatih mesin translasi untuk English-{Chinese, Japanese, Korean}, kita ingin menganggap English-Korean sebagai *main task* sementara sisanya sebagai *supporting task*, kita dapat melakukan *pre-training* menggunakan data English-Chinese dan English-Japanese terlebih dahulu, diikuti oleh English-Korean (tetapi *loss* tetap *joint loss*). Pada kasus ini, penggunaan *joint weighted loss* dapat dijustifikasi. Di lain pihak, apabila kita mengganggap semua *tasks* penting, kita dapat melakukan data *shuffling* sehingga urutan *training data* tidak bias pada *task* tertentu. Pada kasus ini,



Gambar 13.36: *Multi-task Learning setup.*

penggunaan *joint loss–sum* dapat dijustifikasi. Ilustrasi diberikan pada Gambar 13.36.

Soal Latihan

13.1. POS *tagging*
Pada subbab 13.3, disebutkan bahwa *bidirectional recurrent neural network* lebih cocok untuk persoalan POS *tagging*. Jelaskan mengapa! (hint pada bab 8)

13.2. Eksplorasi
Jelaskanlah pada teman-temanmu apa dan bagaimana prinsip kerja:

(a) *Boltzman Machine*
(b) *Restricted Boltzman Machine*
(c) *Generative Adversarial Network*

Bagian IV

Aplikasi dan Topik Tambahan

14

Penerapan Pembelajaran Mesin

> "Leading is not the same as being the leader. Being the leader means you hold the highest rank, either by earning it, good fortune or navigating internal politics. Leading, however, means that others willingly follow you–not because they have to, not because they are paid to, but because they want to."
>
> Simon Sinek

Bab ini memuat contoh penggunaan *machine learning* untuk dua permasalahan praktis yaitu: (1) sistem rekomendasi dan (2) sistem peringkasan dokumen. Dua domain ini dipilih karena tidak asing (*familiar*) bagi penulis. Seperti yang sudah dideskripsikan pada bab-bab sebelumnya, penerapan *machine learning* pada suatu domain membutuhkan pengetahuan/keahlian pada domain tersebut. Bab ini tidak akan membahas domain secara detail, tetapi secara abstrak (bertujuan memberikan gambaran/pengenalan). Untuk mengerti domain yang dibahas secara mendalam, silakan membaca sumber lainnya. Bab ini akan memuat secara sangat singkat, apa guna *machine learning* dan pada contoh kasus seperti apa teknik *machine learning* diterapkan pada permasalahan spesifik domain. Tujuan bab ini adalah untuk memberikan gambaran, bahwa mengetahui *machine learning* saja mungkin tidak cukup. Sekali lagi penulis ingin menekankan, pembaca harus mengerti domain aplikasi. Selain itu, bab ini mengilustrasikan bahwa model *machine learning* tidak berdiri sendiri. Artinya, model *machine learning* bisa jadi hanyalah sebuah modul (untuk mengeksekusi fungsi tertentu) pada sebuah perangkat lunak.

14.1 Sistem Rekomendasi

Bagian sistem rekomendasi ditulis oleh **Candy Olivia Mawalim** (*Japan Advanced Institute of Science and Technology*). Penulis ingin berterima kasih atas sumbangsih yang diberikan.

Salah satu penerapan pembelajaran mesin adalah sistem rekomendasi. Sistem rekomendasi dimotivasi oleh keinginan pemilik usaha untuk meningkatkan penjualan dengan cara mengerti pola pembelian pengguna. Aplikasi yang memanfaatkan sistem rekomendasi dapat kita temukan dalam kehidupan sehari-hari, misalnya Youtube yang memberikan rekomendasi video berdasarkan riwayat video yang telah kita lihat sebelumnya dan Amazon yang merekomendasikan produknya dengan cara menawarkan produk yang sering dibeli pengguna lain yang memiliki karakteristik yang "mirip" dengan kita. Ada dua komponen penting pada sistem rekomendasi yaitu: pengguna dan *item*. Pengguna adalah sang pengguna sistem, sementara *item* dapat berupa video, buku, dan lain sebagainya (produk/layanan yang ditawarkan sistem).

Secara umum, terdapat dua teknik untuk membangun sistem rekomendasi yaitu: (1) *content-based filtering* dan (2) *collaborative filtering*. Teknik pertama berfokus pada karakteristik pengguna secara spesifik. Teknik kedua berfokus pada selera terhadap suatu *item*. Dalam sistem rekomendasi, teknik *machine learning* dapat digunakan untuk memprediksi *item* yang mungkin disukai pengguna. Dua subbab berikutnya memuat aplikasi teknik *machine learning* untuk kedua teknik sistem rekomendasi(14.1.1 dan 14.1.2). Setelah proses *filtering* (memprediksi *item* yang mungkin disukai pengguna), biasanya ada tahapan *post-processing*. Misalnya menampilkan *item* yang bersesuaian dengan usia pengguna. Contoh lain adalah *post-processing* untuk memberikan rekomendasi yang cukup beranekaragam [100].

14.1.1 Content-based Filtering

Teknik membangun sistem rekomendasi berdasarkan *content-based filtering* memanfaatkan informasi mengenai profil seorang pengguna beserta uraian item yang sangat menarik bagi pengguna tersebut [101]. Profil pengguna diartikan sebagai karakteristik (atribut dan *behavior*) yang dimiliki pengguna. Atribut pengguna misalnya *gender*, kewarganegaraan, umur, dan lain-lain. Informasi mengenai *behavior* mencakup karakteristik *item* yang seorang pengguna sukai. Misalkan *item* adalah film, karakteristik film dapat ditentukan dari aktor yang terlibat dalam film, pembuat film, tahun pembuatan film, dan *genre* dari film (misalnya *action*, *horror*, *comedy*). Dengan karateristik ini, kita dapat menentukan kemiripan antar-film.

Kedua informasi ini dapat direpresentasikan sebagai vektor (ekuivalen dengan *feature vector*) agar mudah untuk dilakukan operasi aritmatika, dikenal dengan istilah ***user embedding*** dan ***item embedding***. Cara melakukan *embedding* untuk profil pengguna dan uraian item mirip dengan cara *word embedding* yang telah dijelaskan pada subbab 12.5.3, *one-hot encoding*.

Rekomendasi *item* yang diberikan pada pengguna adalah *item* yang paling mungkin disukai pengguna berdasarkan karakteristiknya. Agar mendapat gambaran lebih jelas, penulis mengajak pembaca untuk mengikuti tutorial pembuatan sistem rekomendasi film sangat sederhana menggunakan dataset MovieLens[1] berukuran kecil (100K). Dataset ini memuat berbagai informasi sebagai berikut:

1. Informasi *rating* pengguna untuk masing-masing film.
2. Informasi film, misalkan berupa *genre* dan tanggal *release*.
3. Demografik pengguna, misal usia, *gender*, pekerjaan, dan lain lain.

Untuk menyederhanakan tutorial, kami hanya akan menjelaskan contoh penggunaan *genre* film untuk membuat sistem rekomendasi. Pertama-tama, kita bangun representasi *item embedding*, yaitu *genre* untuk masing-masing film (Tabel 14.1). Setiap baris pada tabel tersebut merepresentasikan *item embedding* untuk suatu film. Perhatikan! Setiap sel pada tabel diisi nilai "1" apabila film memiliki *genre* yang tertera pada kolom bersesuaian, "0" apabila tidak.

MovieId	Adventure	Animation	Children	Comedy
6	1	1	1	1
22	1	0	1	0
50	0	0	0	1

Tabel 14.1: Representasi *item embedding* untuk film berdasarkan *genre*.

Kedua, kita bangun representasi *user embedding*, yaitu apakah pengguna menyukai suatu film atau tidak (*binary*). Suka atau tidaknya pengguna terhadap suatu film dapat dilihat berdasarkan *rating* yang ia berikan. Sebagai contoh sederhana, kita anggap apabila pengguna menyukai suatu film apabila memberi nilai *rating* lebih dari atau sama dengan 4. Sebagai contoh, perhatikan Tabel 14.2. Apabila user menyukai suatu film, nilai "1" diisi pada kolom "Like or Not", dan "0" apabila sebaliknya.

UserId	MovieId	Like or Not
1	6	0
1	22	0
1	50	1

Tabel 14.2: Representasi *user embedding* berdasarkan *rating* yang diberikan pengguna.

[1] https://grouplens.org/datasets/movielens/

Berdasarkan *item embedding* dan *user embedding* yang kita punya, kita ganti kolom MovieId pada Tabel 14.2 menggunakan baris pada *item embedding*. Sekarang, kita memiliki dataset *behavior* pengguna dengan UserId=1 (Tabel 14.3). Perhatikan! Tabel tersebut seperti dataset *machine learning* yang sudah kamu pelajari pada bab-bab sebelumnya. Diberikan *feature vector* dan kelas (Like or Not) yang berkorespondensi. Menggunakan data seperti

MovieId	Adventure	Animation	Children	Comedy	Like or Not
6	1	1	1	1	0
22	1	0	1	0	0
50	0	0	0	1	1

Tabel 14.3: Dataset *behavior* pengguna dengan UserId=1.

pada Tabel 14.3 yang dirata-ratakan, kita dapat menggunakan teknik *machine learning* untuk memprediksi apakah suatu pengguna akan menyukai film tertentu, berdasarkan *genre* yang dimuat oleh film tersebut. Sederhananya, bisa menggunakan K-*nearest-neighbor* dengan menghitung *cosine similarity* antara *item embedding* dan *user embedding*.

Teknik *content-based filtering* memiliki keunggulan dimana kita tidak memerlukan banyak informasi tentang pengguna lain. Kita hanya memerlukan informasi uraian *item* dan informasi karakteristik suatu pengguna. Hal ini mengakibatkan rekomendasi yang diberikan sangat bergantung pada kepribadian pengguna. Apabila pengguna tidak konsisten, sistem rekomendasi juga bingung.

14.1.2 Collaborative Filtering

Teknik ini diperkenalkan oleh Paul Resnick dan Hal Varian pada 1997 [102]. Prinsip *collaborative filtering* adalah asumsi bahwa selera penggunaan terhadap suatu *item* cenderung sama dari waktu ke waktu [103]. Pada contoh kasus sederhana untuk sistem rekomendasi film, teknik ini memanfaatkan informasi *rating* dari banyak pengguna. Kita dapat merepresentasikan tingkah laku (*behaviour*) semua pengguna menggunakan matriks utilitas dimana baris merepresentasikan profil pengguna dan kolom merepresentasikan *item*. Sebagai contoh, perhatikanlah Tabel 14.4.

Ada dua metode varian *collaborative filtering* yaitu: (1) *neighborhood-based collaborative filtering* (*memory-based method*) dan (2) *model-based collaborative filtering*. Metode *neighborhood-based collaborative filtering* bekerja dengan fakta bahwa pengguna yang "mirip" memiliki pola yang "mirip" dalam memberikan *rating* untuk *item* [104] (pada Tabel 14.4, pengguna yang mirip memiliki baris yang mirip). Selain itu, *item* yang memiliki kemiripan, akan memiliki pola *rating* yang mirip (pada Tabel 14.4, *item* yang mirip memiliki

	$item_1$	$item_2$	...	$item_N$
$user_1$	2	3	...	4
$user_2$	5	3	...	1
...				
$user_3$	4	1	...	2

Tabel 14.4: Matriks utilitas.

kolom yang mirip). Dengan itu, kita dapat menggunakan perhitungan kemiripan vektor untuk menghitung pengguna mana yang mirip dengan suatu pengguna p. Saat memberikan rekomendasi film bagi pengguna p, kita tunjukkan film-film yang pernah ditonton pengguna yang mirip dengannya, atau kita tunjukkan film-film yang mirip dengan film-film yang pernah ditonton oleh pengguna p.

Untuk *model-based collaborative filtering*, prediksi dibangun dengan menggunakan teknik *machine learning* [104]. Untuk suatu sistem dengan data yang *sparse*, matriks utilitas seperti Tabel 14.4 tidaklah efisien secara memori. Kita dapat memanfaatkan teknik-tenik seperti *matrix factorization*/*principal component analysis* dan *autoencoder* untuk mengurangi ukuran matriks. Silakan membaca lebih lanjut materi-materi tersebut (ingat kembali materi bab 12).

14.2 Peringkasan Dokumen

Meringkas dokumen berarti mengerti keseluruhan isi teks/dokumen, kemudian mampu menyampaikan kembali **sebanyak/seakurat mungkin** maksud dokumen asli, ke dalam bentuk yang lebih singkat [105, 106, 57]. Suatu ringkasan harus lebih pendek dibanding dokumen asli. Dengan demikian, sulit untuk dikatakan bahwa suatu ringkasan dapat memuat keseluruhan isi dokumen. Karena itu, ringkasan hanya memuat **sebanyak/seakurat mungkin** maksud dokumen asli, diberikan *constraint* jumlah kata maksimum pada ringkasan. Ada beberapa jenis peringkasan dokumen dilihat dari berbagai sudut pandang, misal:

1. Banyaknya dokumen yang diringkas, meringkas satu dokumen (*single-document summarization*) [107] atau banyak dokumen (*multi-document summarization*) [108] menjadi satu ringkasan.
2. Indikatif atau Informatif. Indikatif berarti menyediakan *pointer* ke bagian dokumen (misal membuat daftar isi). Informatif berarti meyampaikan sebanyak/seakurat mungkin maksud dokumen asli ke dalam bentuk lebih singkat. Pada masa sekarang, hampir seluruh riset peringkasan dokumen mengacu untuk menyediakan ringkasan yang informatif.
3. Domain. Hasil ringkasan bergantung pada domain dokumen, misalkan berita atau novel. Pada domain berita, hal yang penting untuk dimuat

dalam ringkasan adalah 5W1H (*what, who, when, whom, where, how*) [109], sementara pada novel mungkin kita ingin tahu kejadian-kejadian yang ada (beserta urutannya).

4. Generik, *query-based*, atau *tailored*. Generik berarti menghasilkan ringkasan untuk umum, dalam artian tidak ada *target user* secara spesifik, e.g., *review* buku [105]. *Query-based* artinya menghasilkan ringkasan dokumen berdasarkan *query* yang diberikan *user*, i.e., ringkasan adalah informasi spesifik yang dibutuhkan oleh *user* [110]. *Tailored* berarti menyajikan informasi dengan tipe spesifik. Misalkan pada berita terdapat informasi 5W1H, kita hanya ingin mencari tahu informasi *how*.
5. Ekstraktif [111] atau abstraktif [57]. Ekstraktif berarti ringkasan hanya memuat unit informasi yang ada di dokumen asli. Analoginya seperti memilih potongan dari dokumen asli sebagai ringkasan (*copy*). Abstraktif berarti ringkasan dapat memuat unit informasi yang mungkin tidak ada di dokumen asli. Analoginya seperti mengerti dokumen kemudian menulis ulang (parafrase).

Hal terpenting untuk diingat adalah peringkasan dokumen (apa yang diringkas, ringkasan harus memuat apa, dsb) sangat bergantung pada **tujuan ringkasan** (siapa pembaca ringkasan, untuk apa ringkasan dihasilkan). Diktat ini akan membahas *framework* (kerangka kerja/berpikir) peringkasan dokumen dan bagaimana *machine learning* membantu peringkasan dokumen.

Secara umum, ada beberapa tahap pada proses peringkasan dokumen secara otomatis, terlepas dari tipe peringkasan dokumen yang dijabarkan [105, 112, 113, 106, 114, 109, 108, 107, 57, 115]:

1. *Representation* – Melakukan pemisahan dan representasi unit informasi; pada umumnya adalah kalimat, kata, atau frase. Dalam artian, kita menganggap dokumen tersusun atas sekuens kalimat, kata, atau frase. Kita potong-potong unit informasi pada dokumen lalu, unit informasi dapat direpresentasikan sebagai vektor untuk tiap unitnya [116]. Kita pun dapat menggali hubungan antara unit informasi (misal direpresentasikan sebagai graf) untuk kemudian mencari unit mana yang sentral [109, 117].
2. *Filtering* – Menyaring informasi. Unit manakah yang penting/tidak penting. Unit mana yang dibutuhkan/tidak dibutuhkan. Unit mana yang relevan/tidak relevan. Unit mana yang representatif/tidak representatif. Teknik menyaring informasi sangat bergantung pada representasi unit pada tahap sebelumnya (e.g., vektor atau graf). Secara singkat, tahap ini memilih unit informasi berdasarkan tujuan. Model pembelajaran mesin biasanya paling sering digunakan pada tahap ini.
3. *Generation* – Menghasilkan (*generate*) ringkasan. Bagian ini membutuhkan pengetahuan mengenai bidang pemrosesan bahasa alami (*natural language processing*) (NLP). Pada diktat ini, tidak akan dibahas.

Teknik *machine learning* dapat berperan penting pada keseluruhan tahapan diatas, khususnya tahap kedua (*filtering*). Kami akan memberikan studi

kasus pada subbab 14.2.1 agar lebih jelas. Sistem peringkasan dokumen yang membagi-bagi proses menjadi tahapan-tahapan tersebut disebut *pipelined approach*. Artinya kita melakukan proses diatas sebagai tahap-tahap berbeda (independen satu sama lain). Selain *pipelined approach*, kita juga dapat memandang peringkasan dokumen dengan *single-view approach* (subbab 14.2.2), artinya keseluruhan proses tersebut terjadi bersamaan.

14.2.1 Pipelined Approach

Subbab ini akan memuat cerita singkat tentang peringkasan *paper* [112, 115]. Berdasarkan teori *argumentative zoning* [112], *paper* terdiri dari zona-zona dengan tujuan komunikatif yang berbeda. Dengan bahasa yang lebih mudah, tiap kalimat (unit informasi) pada *paper* memiliki tujuan komunikasi yang berbeda. Misalnya ada kalimat yang menyatakan tujuan penelitian, latar belakang penelitian, atau hasil penelitian. Tujuan komunikasi kalimat disebut *rhetorical categories* [118].

Ringkasan, dalam bentuk abstrak atau judul *paper* memiliki pola [119, 115]. Dalam artian, suatu ringkasan bisa jadi hanya memuat informasi dengan *rhetorical categories* tertentu (*tailored summary*). *Machine learning* dapat digunakan untuk mengklasifikasikan kalimat (unit informasi) ke kelas masing-masing [118, 115]. Pertama-tama setiap kalimat direpresentasikan menjadi *feature vector* (ingat kembali materi bab 3 dan bab 4). Sebagai contoh, *paper* [115] merepresentasikan kalimat pada *paper* sebagai *feature vector* berdasarkan fitur-fitur sebagai berikut:

1. Posisi. Lokasi kalimat dibagi dengan panjang teks (numerik).
2. Kata kunci (*lexicon*). Apakah kalimat memuat kata kunci yang eksklusif untuk *rhetorical category* tertentu (*binary* – ya atau tidak).
3. Bobot kalimat, i.e., seberapa "penting" kalimat tersebut. Hal ini dapat dihitung menggunakan TF-IDF (ingat bab 12) kata pada kalimat (numerik).
4. Kategori sebelumnya. *Rhetorical category* kalimat sebelumnya (nominal).

Setelah diubah menjadi *feature vector*, teknik *machine learning* digunakan untuk mengklasifikasikan kalimat menjadi kelas *rhetorical categories* yang sesuai. Dengan demikian, kita dapat menyaring kalimat berdasarkan tipe informasi mana yang kita inginkan, berdasarkan tujuan peringkasan. Selanjutnya, teknik NLP digunakan untuk memproses informasi yang disaring untuk menghasilkan ringkasan. Materi tersebut diluar bahasan diktat ini.

14.2.2 Single-view Approach

Pada *pipelined approach*, setiap tahapan peringkasan dokumen (*representation*, *filtering*, dan *generation*) dianggap sebagai tahap yang independen satu sama lain. Permasalahannya adalah kesalahan pada suatu tahap akan mempengaruhi tahap berikutnya. Opsi lain adalah dengan melakukan keseluruhan

proses sebagai satu tahapan yang utuh (*end-to-end*). Metode ini memiliki kaitan yang erat dengan teknik *machine translation*, yang pada masa sekarang populer didekati dengan teknik *deep learning* (*sequence to sequence encoder–decoder*) [57, 81, 80, 89, 120, 121]. Dokumen tersusun atas sejumlah T sekuens unit informasi (e.g., kata, frase) $\mathbf{u} = u_1, u_2, \ldots, u_T$. Sebuah ringkasan adalah M buah sekuens unit informasi $\mathbf{r} = r_1, r_2, \ldots, r_M$ dimana $M < T$. Tujuan peringkasan adalah untuk mencari $\mathbf{r}$ terbaik, sedemikian sehingga dapat memenuhi persamaan 14.1.

$$\arg\max_{\mathbf{r}} p(\mathbf{r} \mid \mathbf{u}) \tag{14.1}$$

Pada umumnya, terdapat suatu variabel tambahan θ yang mengendalikan (*govern*) probabilitas tersebut. Sehingga secara lebih tepat, persamaan 14.1 diubah menjadi persamaan 14.2.

$$\arg\max_{\mathbf{r}} p(\mathbf{r} \mid \mathbf{u}, \theta) \tag{14.2}$$

Perhatikan, persamaan 14.2 adalah bentuk yang dapat dimodelkan menggunakan *sequence to sequence encoder–decoder*. Selain peringkasan dokumen, banyak permasalahan pada bidang *natural language processing* memiliki bentuk yang "mirip", seperti: *part-of-speech tagging* [122], *named entity recognition* [123], mesin translasi [81] dan rekonstruksi paragraf [124].

14.3 Konklusi

Teknik *machine learning* sangatlah berguna untuk berbagai macam permasalahan. Tetapi perlu dipahami bahwa teknik yang ada belum mampu memodelkan proses berpikir manusia dengan benar. Proses berpikir manusia sangatlah kompleks, dan model yang ada sekarang ini adalah bentuk simplifikasi. Sebagai contoh, seorang bayi sekali melihat kucing mungkin akan mengetahui kucing lainnya walaupun tidak memiliki rupa yang sama persis. Sementara itu, model *machine learning* harus diberikan banyak sampel. Manusia memiliki otak yang luar biasa hebat karena mampu belajar dengan sedikit contoh.

Persoalan-persoalan yang diberikan pada buku ini secara umum mencakup *supervised* dan *unsupervised learning* saja. Terdapat satu pemodelan masalah penting lainnya yaitu *reinforcement learning* dimana kita ingin memaksimalkan hasil untuk sekuens aksi (sekuens keputusan). Setiap keputusan dapat diberikan bobot yang berbeda. Kemudian, agen memaksimalkan nilai untuk sekuens keputusan (*cummulative reward*).[2] Pada *supervised learning*, kita hanya perlu membuat satu keputusan saja (menggolongkan data ke kelas mana). Pada konferensi-konferensi, para *master* banyak menyebutkan

[2] https://en.wikipedia.org/wiki/Reinforcement_learning

masa depan *machine learning* berada pada *unsupervised* atau *reinforcement learning*. Aplikasi mutakhir yang ada sekarang ini didominasi oleh *supervised learning*. Pada kenyataannya, sebagian besar (hampir semua) data yang ada di dunia ini tidak berlabel. Karena itu, representasi data secara *unsupervised* menjadi topik riset hangat (*representation learning*).

Buku ini telah menjelaskan berbagai macam teknik, tetapi perlu diketahui bahwa materi yang diberikan adalah simplifikasi agar mampu dipahami secara intuitif. Selain itu, buku ini di desain sebagai materi pengantar saja. Untuk memahami teknik *machine learning* secara lebih jauh dan sempurna, penulis menyarankan untuk membaca buku referensi [8, 11].

Materi yang disajikan pada buku ini adalah persoalan-persoalan *machine learning* dari sudut pandang optimisasi. Yaitu mencari himpunan (*set of*) parameter agar model pembelajaran yang dibangun mampu memberi keputusan yang optimal. Kami menyarankan pembaca untuk mencari referensi lebih jauh tentang *machine learning* dari sudut pandang eksplorasi, yaitu mencari *concept space* dimana model dapat bekerja dengan "baik". Sebagai contoh, kita memiliki sebuah mesin (fisik) dengan berbagai macam konfigurasi (parameter). Apabila kita menjalankan mesin dengan konfigurasi parameter tertentu, mesin akan rusak. Tugas kita adalah mencari *parameters space* dimana mesin dapat berjalan dengan optimal dan tidak rusak. Sedangkan, pada persoalan optimisasi, kita mencari satu konfigurasi terbaik.

Machine learning adalah bidang yang sangat luas (lebih luas dari apa yang diceritakan pada buku ini) dan berkembang pesat. Penulis menyarankan pembaca untuk membaca makalah dari konferensi[3] *top-tier* untuk mengetahui perkembangan terkini. Sebagai contoh (diurutkan berdasarkan abjad):[4]

- AAAI. AAAI Conference on Artificial Intelligence
- ACL. Annual Meeting of Association for Computational Linguistics
- CVPR. IEEE Conference on Computer Vision and Pattern Recognition
- EMNLP. Empirical Methods in Natural Language Processing
- ICCV. IEEE International Conference on Computer Vision
- ICLR. International Conference on Learning Representation
- ICML. International Conference on Machine Learning
- IJCAI. International Conference on Artificial Intelligence
- INTERSPEECH. Conference of the International Speech Association
- NeurIPS (dahulu disebut NIPS). Neural Information Processing System
- SIGIR. ACM Special Interest Group in Information Retrieval
- SIGKDD. ACM Special Interest Group in Knowledge Discovery and Data Mining

[3] Kamu lebih mudah mengetahui informasi terkini dari konferensi dibanding jurnal karena proses *peer-review* yang lebih cepat. Pada bidang keilmuan ilmu komputer/teknik informatika, konferensi internasional lebih penting dibanding jurnal.

[4] Konferensi penting untuk bidang pemrosesan bahasa alami terdaftar pada https://aclanthology.coli.uni-saarland.de/

14.4 Saran Buku Lanjutan

Penulis ingin memberikan beberapa rekomendasi bacaan pembelajaran mesin selanjutnya (teoritis). Buku-buku berikut (diurutkan secara subjektif berdasarkan kualitas konten dan kemudahan dimengerti):

1. *Deep Learning* oleh Ian Goodfellow et al. [11]. Banyak yang menggganggap buku ini sebagai "kitab" *deep learning* modern. Buku ini juga mencakup materi matematika dasar (e.g., aljabar linier) yang dibutuhkan untuk mengerti *deep learning*.
2. *Neural Network Methods in Natural Language Processing* oleh Goldberg [1]. Buku ini ditujukan sebagai bahan transisi bagi peneliti bidang pemrosesan bahasa alami untuk menggunakan metode *neural network*. Apabila kamu tidak tertarik dengan pemrosesan bahasa alami, kamu bisa membaca bab 1-5 pada buku ini sebagai pengenalan *neural network*.
3. *Pattern Recognition and Machine Learning* oleh Bishop [8]. Menurut penulis, banyak yang tahu buku ini karena dianggap sebagai "kitab". Penjelasan buku ini sangat matematis dan relatif berat untuk dimengerti. Tetapi, kamu dapat menjadi *master* apabila memahami seluruh materi pada buku ini. Algoritma *machine learning* yang disajikan juga relatif lebih "klasik" dibanding rekomendasi pertama.
4. *Machine Learning* oleh Tom Mitchel [4]. Buku ini memuat materi *machine learning* yang cukup "klasik". Buku ini cocok sebagai materi pengenalan, tapi relatif kurang dalam.

Sedikit tambahan pesan sponsor karena penulis berlatar belakang dari bidang pemrosesan bahasa alami, penulis menyarankan membaca buku-buku (teoritis) berikut sebagai dasar pengetahuan pemrosesan bahasa alami (diurutkan dari konten paling dasar):

1. *Foundations of Statistical Natural Language Processing* oleh Christopher D. Mannning dan Hinrich Schutze [64]. Buku ini dapat dideskripsikan dengan satu kata, **TOP**. Buku ini memuat materi pemrosesan bahasa alami dengan sudut pandang matematis, dapat dianggap sebagai *framework* berpikir yang modern.
2. *Speech and Language Processing* oleh Daniel Jurafsky dan James H. Martin [12]. Buku ini jauh lebih tebal dari buku pertama dan memuat materi yang lebih luas. Penulis merekomendasikan buku ini untuk mengetahui permasalahan-permasalahan pemrosesan bahasa alami.
3. *An introduction to Information Retrieval* oleh Manning et al. [65]. Walaupun berjududul *information retrieval*, penulis pertama kali mengenal konsep *embedding* dari buku ini. Buku ini ditulis dengan apik.
4. *Neural Network Methods in Natural Language Processing* oleh Goldberg [1]. Apabila kamu telah membaca buku-buku yang disebutkan sebelumnya, kamu dapat membaca buku ini untuk transisi ke metode *neural network*.

Pada saat menulis buku ini, penulis berharap bisa menulis pada level diantara buku Tom Mitchel [4] dan Bishop [8] yaitu cukup matematis, lumayan mudah dipahami, dan lumayan dalam. Mudah-mudahan pembaca merasa tujuan ini tercapai dan menikmati membaca buku ini. Sebagai kalimat penutup, terimakasih sudah membaca sampai tahap ini.

Soal Latihan

14.1. Cold Start Problem

(a) Jelaskan apa itu *cold start problem* pada sistem rekomendasi, serta bagaimana cara menangani permasalahan tersebut!
(b) Pada teknik sistem rekomendasi manakah (*content-based filtering* atau *collaborative filtering*) *cold start problem* mungkin muncul? Mengapa?

14.2. Eksplorasi Sistem Rekomendasi

Bangunlah suatu sistem rekomendasi dengan teknik *collaborative filtering* pada dataset MovieLens dengan memanfaatkan library `recommenderlab`[5]!

14.3. Peringkasan Dokumen

(a) Presentasikanlah di kelasmu, *paper* sistem peringkasan dokumen otomatis oleh Kupiec et al. (1995) [116]![6] yang menggunakan *pipelined approach*!
(b) Presentasikanlah di kelasmu, *paper* sistem peringkasan dokumen otomatis oleh Cheng and Lapata [121]![7] yang menggunakan *single-view approach*!
(c) Jelaskan perbedaan, kelebihan, dan kelemahan pendekatan-pendekatan sistem peringkasan dokumen!

[5] https://cran.r-project.org/web/packages/recommenderlab/index.html
[6] https://dl.acm.org/citation.cfm?id=215333
[7] http://www.aclweb.org/anthology/P16-1046

Referensi

[1] Yoav Goldberg. *Neural Network Methods in Natural Language Processing*. Morgan & Claypool Publishers, 2017.

[2] Peter Linz. *An Introduction to Formal Language and Automata*. Jones and Bartlett Publishers, Inc., USA, 2006.

[3] Ronald Brachman and Hector Levesque. *Knowledge Representation and Reasoning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[4] Tom Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[5] Stuart Russel and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

[6] Jonathan Gratch and Stacell Marsella. Computationally modelling human emotion. *Communications of the ACM*, 57(12):71–99, 2014.

[7] Masayu Leylia Khodra and Dessi Puji Lestari. Odd semester lecture on machine learning. Lecture of Institute Teknologi Bandung, 2015.

[8] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[9] Sumio Watanabe and Hidetoshi Nishimori. Fall lecture note on statistical learning theory. Lecture note for Tokyo Institute of Technology, 2016.

[10] Brian Caffo. *Statistical Inference for Data Science*. Lean Publishing, 2015.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[12] Daniel Jurafsky and James H. Martin. *Speech and Language Processing Second Edition*. Prentice Hall, 2009.

[13] Nils Reimers and Iryna Gurevych. Reporting score distributions makes a difference: Performance study of LSTM-networks for sequence tagging. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 338–348, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

[14] Rotem Dror, Gili Baumer, Segev Shlomov, and Roi Reichart. The hitchhiker's guide to testing statistical significance in natural language processing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1383–1392, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[15] Rotem Dror, Segev Shlomov, and Roi Reichart. Deep dominance - how to properly compare deep neural models. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2773–2785, Florence, Italy, July 2019. Association for Computational Linguistics.

[16] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley, 1991.

[17] Hidetoshi Nishimori. *Statistical Physics of Spin Glasses and Information Processing: An Introduction*. Clarendon Press, 2001.

[18] Sharon L. Myres Ronald E. Walpole, Raymond H. Myers and Keying Ya. *Probability and Statistics for Engineers and Scientists*. Prentice Hall, 2012.

[19] Gilbert Strang. *Linear algebra and its applications*. Thomson, Brooks/Cole, Belmont, CA, 2006.

[20] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning, with applications in R*. Springer, 2013.

[21] Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data Minin gg: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2011.

[22] Jeff Leek. *The Elements of Data Analytic Style*. Leanpub, 2015.

[23] Takao Terano and Tsuyoshi Murata. Spring lecture on machine learning. Lecture of Tokyo Institute of Technology, 2017.

[24] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 311–318, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[25] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Proc. ACL workshop on Text Summarization Branches Out*, page 10, 2004.

[26] Irina Rish. An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46. IBM New York, 2001.

[27] J. A. Hartigan and M. A. Wong. A k-means clustering algorithm. *JSTOR: Applied Statistics*, 28(1):100–108, 1979.

[28] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theor.*, 13(1):21–27, September 2006.

[29] Mohammad S. Sorower. A literature survey on algorithms for multi-label learning. 2010.

[30] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization, 2010.

[31] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, Sep 1995.

[32] Marti A. Hearst. Support vector machines. *IEEE Intelligent Systems*, 13(4):18–28, July 1998.

[33] J. R. Quilan. *Discovering rules by induction from large collections of examples*. Edinburgh University Press, 1979.

[34] J.R. Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106, March 1986.

[35] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

[36] Takao Terano and Tsuyoshi Murata. Spring lecture on machine learning. Lecture of Tokyo Institute of Technology, 2017.

[37] L. R. Rabiner and B. H. Juang. An introduction to hidden markov models. *IEEE ASSp Magazine*, 1986.

[38] James Allen. *Natural Language Understanding*. Benjamin-Cummings Publishing Co., Inc., 1995.

[39] Eliyahu Kiperwasser and Yoav Goldberg. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327, 2016.

[40] Vishal M. Patel, Raghuraman Gopalan, Ruonan Li, and Rama Chellappa. Visual domain adaptation: A survey of recent advances. *IEEE Signal Process. Mag.*, 32(3):53–69, 2015.

[41] George Karypis Michael Steinbach and Vipin Kumar. A comparison of document clustering techniques. In *KDD Workshop on Text Mining*, pages 525 – 526, 2000.

[42] Omer I. E. Mohamed Fathi H. Saad and Rafa E. Al-Qutaish. Comparison of hierarchical agglomerative algorithms for clustering medical documents. *International Journal of Software Engineering and Applications (IJSEA)*, 3(3), 2012.

[43] Rajeev Rastogi Sudipto Guha and Kyuseok Shim. Cure: An efficient clustering algorithm for large databases. In *Proceedings of ACMSIGMOD International Conference on Management of Data*, pages 73 – 84, 1998.

[44] Jack D. Cowan. Neural networks: The early days. In *Proceedings of Advances in Neural Information Processing Systems 2*, 1989.

[45] Amir Atiya. *Learning Algorithms for Neural Network*. PhD thesis, California Institute of Technology, 1994.

[46] Al. Cripps. Using artificial neural nets to predict academic performance. In *Proceedings of the 1996 ACM Symposium on Applied Computing*, pages 33–37, 1996.

[47] Thomas Mikolov. *Statistical Language Models Based on Neural Networks*. PhD thesis, Brno University of Technology, 2012.

[48] Kai Chen Greg Corrado Thomas Mikolov, Ilya Sutskever and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of CoRR*, 2013.

[49] Gred Corrado Thomas Mikolov, Kai Chen and Jeffrey Dean. Efficient estimation of word representations in vector space. In *Proceedings of CoRR*, 2013.

[50] Kai Yu. Large-scale deep learning at baidu. In *Proceedings of the 22nd ACM International Conference on Information and Knowledge Management*, pages 2211–2212, 2013.

[51] M. A. Aizerman, E. A. Braverman, and L. Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning. In *Automation and Remote Control,*, number 25, pages 821–837, 1964.

[52] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

[53] Marvin L. Minsky and Seymour A. Papert. *Perceptrons: Expanded Edition*. MIT Press, Cambridge, MA, USA, 1988.

[54] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter Learning Internal Representations by Error Propagation, pages 318–362. MIT Press, Cambridge, MA, USA, 1986.

[55] Tao Lei, Regina Barzilay, and Tommi Jaakkola. Rationalizing neural predictions. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 107–117, Austin, Texas, November 2016. Association for Computational Linguistics.

[56] David Alvarez-Melis and Tommi Jaakkola. A causal framework for explaining the predictions of black-box sequence-to-sequence models. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 412–421, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

[57] Dzmitry Bahdanau, KyungHyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *Proceedings of the International Conference on Learning and Representation (ICLR)*, 2015.

[58] Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[59] Finale Doshi-Velez and Been Kim. A roadmap for a rigorous science of interpretability. In *ArXiv e-prints*.

[60] Jeffrey L. Elman. Learning and development in neural networks: The importance of starting small. *Journal of Cognition*, (48):71–99, 1993.

[61] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International*

Conference on Machine Learning, ICML '09, pages 41–48, New York, NY, USA, 2009. ACM.

[62] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014.

[63] Dzmitry Bahdanau Kyunghyun Cho, Bart van Merrienboer and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, Doha, Qatar, October 2014. Association for Computational Linguistics.

[64] Christopher D. Manning and Hinrich Schutze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[65] Prabhakar Raghavan Christopher D. Manning and Hinrich Schutze. *An Introduction to Information Retrieval*. Cambridge UP, 2009.

[66] Andrew Y. Ng Richard Socher, Cliff Chiung-Yu Lin and Christopher D. Manning. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th International Conference on Machine Learning*, 2011.

[67] Jean Y. Wu Jason Chuang Richard Socher, Alex Perelygin and Christopher D. Manning. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the Emperical Methods in Natural Language Processing*, 2013.

[68] Erhc H. Huang Andrew Y. Ng Richard Socher, Jeffrey Pennington and Christoper D. Manning. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the Emperical Methods in Natural Language Processing*, 2011.

[69] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31$_{st}$ International Conference on Machine Learning*, 2014.

[70] Richard Socher Jeffrey Pennington and Christopher D. Manning. Glove: Global vectors for word representation. In *Proceedings of the Emperical Methods in Natural Language Processing*, pages 1532 – 1543, 2014.

[71] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, March 2003.

[72] Omer Levy, Yoav Goldberg, and Ido Dagan. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225, 2015.

[73] Guoqiang Zhong, Li-Na Wang, Xiao Ling, and Junyu Dong. An overview on data representation learning: From traditional feature learning to recent deep learning. *The Journal of Finance and Data Science*, 2(4):265 – 278, 2016.

[74] Peter D. Turney and Patrick Pantel. From frequency to meaning: Vector space models of semantics. *Journal of Artificial Intelligence Research*, (37):141–188, 2010.

[75] Jan Wira Gotama Putra and Takenobu Tokunaga. Evaluating text coherence based on semantic similarity graph. In *Proceedings of TextGraphs-11: the Workshop on Graph-based Methods for Natural Language Processing*, pages 76–85, Vancouver, Canada, August 2017. Association for Computational Linguistics.

[76] Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time-series. In M. A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*. MIT Press, 1995.

[77] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[78] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[79] Paul J. Werbos. Backpropagation through time: what does it do and how to do it. In *Proceedings of IEEE*, volume 78, pages 1550–1560, 1990.

[80] Caglar Gulcehre Dzmitry Bahdanau Fethi Bougares HolgerSchwenk Kyunghyun Cho, Bart van Merrienboer and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics.

[81] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems*, NIPS'14, pages 3104–3112, Cambridge, MA, USA, 2014. MIT Press.

[82] Yan Shao, Christian Hardmeier, Jörg Tiedemann, and Joakim Nivre. Character-based joint segmentation and pos tagging for chinese using bidirectional rnn-crf. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 173–183, Taipei, Taiwan, November 2017. Asian Federation of Natural Language Processing.

[83] Tobias Horsmann and Torsten Zesch. Do lstms really work so well for pos tagging? – a replication study. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 727–736, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

[84] Barbara Plank, Anders Søgaard, and Yoav Goldberg. Multilingual part-of-speech tagging with bidirectional long short-term memory models and auxiliary loss. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 2: Short Papers*, 2016.

[85] Ryohei Sasano Hiroya Takamura Yuta Kikuchi, Graham Neubig and Manabu Okumura. Controlling output length in neural encoder-

decoders. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1328–1338, Austin, Texas, November 2016. Association for Computational Linguistics.

[86] Ramesh Nallapati, Bowen Zhou, Cícero Nogueira dos Santos, and aglar Gülehre and Bing Xiang. Abstractive text summarization using sequence-to-sequence rnns and beyond. In *CoNLL*, 2016.

[87] Yan-Kai Lin Cun-Chao Tu Yu Zhao Zhi-Yuan Liu Ayana, Shi-Qi Shen and Mao-Song Sun. Recent advances on neural headline generation. *Journal of Computer Science and Technology*, 32(4):768–784, Jul 2017.

[88] Chloé Kiddon, Luke Zettlemoyer, and Yejin Choi. Globally coherent text generation with neural checklist models. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 329–339, Austin, Texas, November 2016. Association for Computational Linguistics.

[89] Xiaojun Wan Jiwei Tan and Jianguo Xiao. Abstractive document summarization with a graph-based attentional neural model. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1171–1181, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[90] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3156–3164, 2015.

[91] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489, San Diego, California, June 2016. Association for Computational Linguistics.

[92] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017.

[93] Karl Weiss, Taghi M. Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big Data*, 3(1):9, May 2016.

[94] Sebastian Ruder, Matthew E. Peters, Swabha Swayamdipta, and Thomas Wolf. Transfer learning in natural language processing. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Tutorials*, pages 15–18, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[95] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *J. Mach. Learn. Res.*, 12:2493–2537, November 2011.

[96] Daxiang Dong, Hua Wu, Wei He, Dianhai Yu, and Haifeng Wang. Multi-task learning for multiple language translation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1723–1732. Association for Computational Linguistics, 2015.

[97] Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. Adversarial multi-task learning for text classification. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1–10. Association for Computational Linguistics, 2017.

[98] Anne Lauscher, Goran Glavaš, Simone Paolo Ponzetto, and Kai Eckert. Investigating the role of argumentation in the rhetorical analysis of scientific publications with neural multi-task learning models. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3326–3338. Association for Computational Linguistics, 2018.

[99] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 7482–7491, 2018.

[100] Arda Antikacioglu and R. Ravi. Post processing recommender systems for diversity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD 17, page 707716, New York, NY, USA, 2017. Association for Computing Machinery.

[101] Daniel Billsus and Michael J. Pazzani. The adaptive web. chapter Adaptive News Access, pages 550–570. Springer-Verlag, Berlin, Heidelberg, 2007.

[102] Paul Resnick and Hal R. Varian. Recommender systems. *Commun. ACM*, 40(3):56–58, March 1997.

[103] Daniar Asanov. Algorithms and methods in recommender systems. Berlin Institute of Technology, 2011.

[104] Charu C. Aggrawal. *Recommender Systems: The Textbook*. Springer International Publishing Switzerland, 2016.

[105] Eduard Hovy and Chin-Yew Lin. Automated text summarization and the summarist system. In *Proceedings of a Workshop on Held at Baltimore, Maryland: October 13-15, 1998*, TIPSTER '98, pages 197–214, Stroudsburg, PA, USA, 1998. Association for Computational Linguistics.

[106] Liang Zhou and Eduard Hovy. Template-filtered headline summarization. In *In the Proceedings of the ACL workshop, Text Summarization Branches Out*, pages 56–60, 2004.

[107] Amin Mantrach Carlos A. Colmenares, Marina Litvak and Fabrizio Silvestri. Heads: Headline generation as sequence prediction using an abstract feature-rich space. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 133–142, Denver, Colorado, May–June 2015. Association for Computational Linguistics.

[108] Daniele Pighin Enrique Alfonseca and Guillermo Garrido. Heady: News headline abstraction through event pattern clustering. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1243–1253, Sofia, Bulgaria, August 2013. Association for Computational Linguistics.

[109] Pierre-Etienne Genest and Guy Lapalme. Framework for abstractive summarization using text-to-text generation. In *Proceedings of the Workshop on Monolingual Text-To-Text Generation*, pages 64–73, Portland, Oregon, June 2011. Association for Computational Linguistics.

[110] Shufeng Xiong and Donghong Ji. Query-focused multi-document summarization using hypergraph-based ranking. *Inf. Process. Manage.*, 52(4):670–681, July 2016.

[111] David Zajic, Bonnie J. Dorr, and Richard Schwartz. Bbn/umd at duc-2004: Topiary. In *Proceedings of the North Americal Chapter of the Association for Computational Linguistics Workshop on Document Understanding*, pages 112–119, 2004.

[112] Simone Teufel and Marc Moens. Argumentative classification of extracted sentences as a first step towards flexible abstracting. In *Advances in automatic Text Summarization*, pages 155–171. MIT Press, 1999.

[113] Bonnie J. Dorr, David Zajic, and Richard Schwartz. Hedge trimmer: A parse-and-trim approach to headline generation. In Dragomir Radev and Simone Teufel, editors, *Proceedings of the HLT-NAACL 03 Text Summarization Workshop*, pages 1–8, 2003.

[114] Jurij Leskovec, Natasa Milic-Frayling, and Marko Grobelnik. Extracting summary sentences based on the document semantic graph. *Microsoft Research*, 2005.

[115] Jan Wira Gotama Putra. Rhetorical sentence classification for automatic title generation in scientific article. *TELKOMNIKA*, 15(2):656–664, 2017.

[116] Jan Pedersen Julian Kupiec and Francine Chen. A trainable document summarizer. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '95, pages 68–73, New York, NY, USA, 1995. ACM.

[117] Hans-Martin Ramsl Daraksha Parveen and Michael Strube. Topical coherence for graph-based extractive summarization. In *Conference on*

Empirical Methods in Natural Language Processing, pages 1949–1954. The Association for Computational Linguistics, 2015.

[118] Simone Teufel and Marc Moens. Summarizing scientific articles: Experiments with relevance and rhetorical status. *Comput. Linguist.*, 28(4):409–445, December 2002.

[119] Diarmuid Ó Séaghdha and Simone Teufel. Unsupervised learning of rhetorical structure with un-topic models. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 2–13, Dublin, Ireland, August 2014. Dublin City University and Association for Computational Linguistics.

[120] Vibhu O. Mittal Michele Banko and Michael J. Witbrock. Headline generation based on statistical translation. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, ACL '00, pages 318–325, Stroudsburg, PA, USA, 2000. Association for Computational Linguistics.

[121] Jianpeng Cheng and Mirella Lapata. Neural summarization by extracting sentences and words. *CoRR*, abs/1603.07252, 2016.

[122] Christopher D. Manning. Part-of-speech tagging from 97% to 100%: Is it time for some linguistics? In *Proceedings of the 12th International Conference on Computational Linguistics and Intelligent Text Processing - Volume Part I*, CICLing'11, pages 171–189, Berlin, Heidelberg, 2011. Springer-Verlag.

[123] Lev Ratinov and Dan Roth. Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, CoNLL '09, pages 147–155, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[124] Jiwei Li and Dan Jurafsky. Neural net models of open-domain discourse coherence. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 198–209, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

Biografi Penulis

Jan **Wira** Gotama Putra adalah PhD Candidate (JSPS Research Fellow) di Computational Linguistics laboratory, Artificial Intelligence Major, Department of Computer Science, Tokyo Institute of Technology. Sebelumnya, penulis mendapatkan gelar sarjana di jurusan Teknik Informatika, Institut Teknologi Bandung. Kemudian, melanjutkan magister di Tokyo Institute of Technology dengan pendanaan MEXT scholarship. Penulis memiliki pengalaman menulis makalah ilmiah pada bidang pemrosesan bahasa alami dan menerapkan teknik pembelajaran mesin untuk perusahaan IT (saat *internship*), e.g., *discourse processing*, klasifikasi teks, peringkasan teks, *automatic hashtag-ing*, dan *content filtering* (*moderation*). Penulis pernah mendapatkan *best paper award* di workshop internasional dan memenangkan lomba *data mining* (terbuka untuk publik) di Indonesia. Penulis masih terus mempelajari teknik *machine learning*. Buku ini adalah catatan yang ingin ia bagikan.

https://wiragotama.github.io/