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

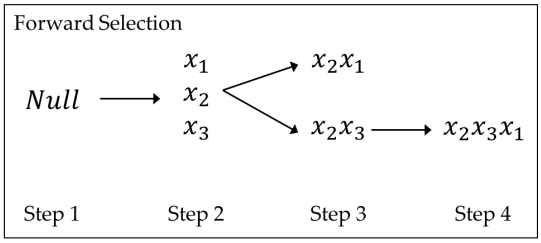
[1] Curse of Dimensionality

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***.

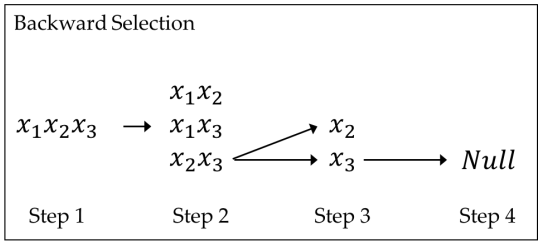
Pada *forward selection*, kita mulai dengan suatu model yang tidak menggunakan fitur apapun sama sekali, yaitu meng-*assign* kelas yang paling sering muncul di dataset, pada *input*. Setelah itu, kita tambahkan satu per satu fitur pada setiap langkah. Langkah berikutnya, kita gunakan satu fitur. Diantara 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 $\frac{5}{10} = 0.5$.

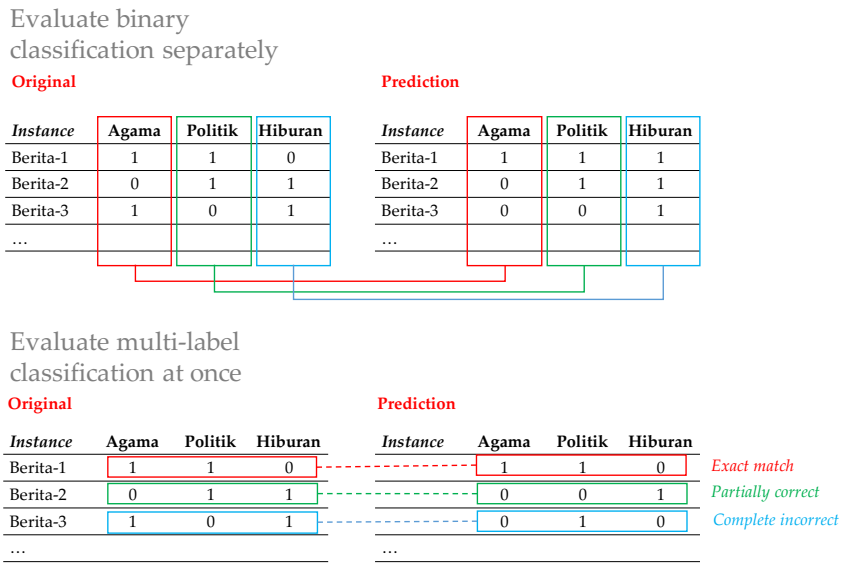
Perhitungan akurasi sama halnya untuk *multi-class classification*, tetapi menjadi semakin rumit untuk *multi-label classification*. Ingat, pada *multi-label classification*, suatu *input* dapat diklasifikasikan menjadi beberapa kelas. *Multi-label classification* dapat dievaluasi dengan dua cara. Pertama, 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.

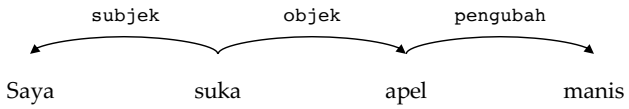
Graf

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".

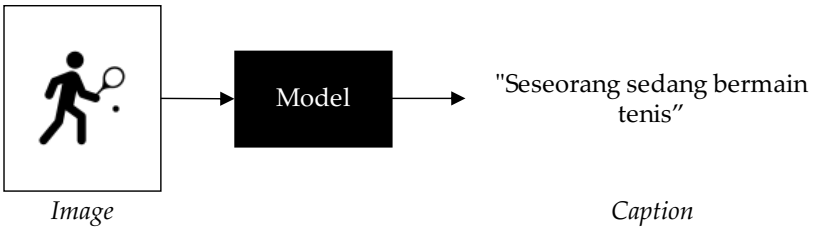
Kita dapat mengevaluasi suatu prediksi struktur dengan cara memfaktorkan struktur tersebut menjadi unit lebih kecil, kemudian mengevaluasi performa model pada satuan unit yang lebih kecil. Kinerja model pada satuan unit diagregat atau digabungkan menjadi satu skor untuk merepresentasikan keseluruhan 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."

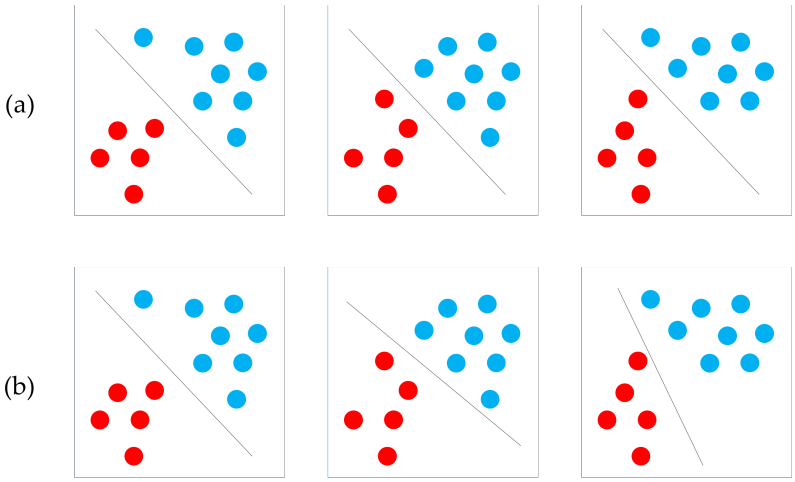
Hal serupa juga harus diperhatikan pada persoalan mesin translasi bahasa. Apakah konten translasi memang benar sama dengan konten aslinya? Bagaimana dengan gaya penulisan? Pada sistem peringkasan teks, apakah hasil ringkasan memang benar sesuai dengan teks aslinya, bukan menambah informasi atau mengada-ada?

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.



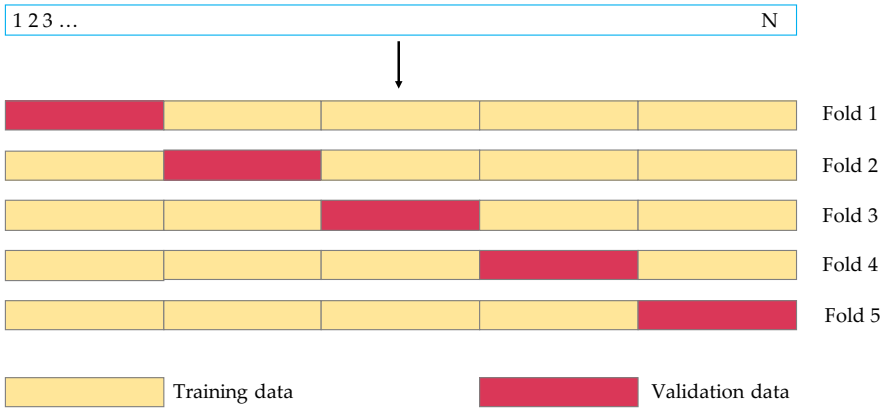
Gambar 9.7: Model yang stabil (a) memberikan *decision boundary* (garis hitam) yang serupa walaupun *input* sedikit diubah-ubah (variasi kinerja yang kecil, untuk set *input* yang berbeda). Model yang kurang stabil (b) memberikan *decision boundary* yang berbeda untuk *input* yang sedikit diubah-ubah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