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.

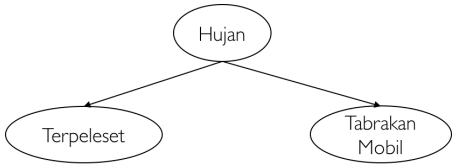
8.1 Probabilistic Reasoning

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

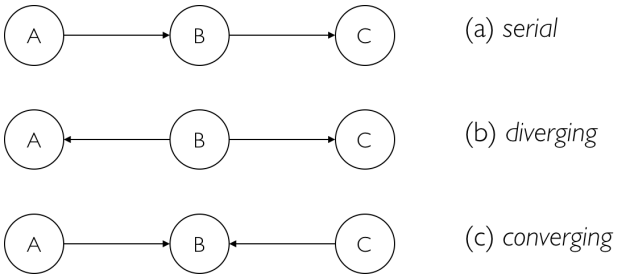
[1] Karena perlu dijelaskan juga cara transformasi sinyal suara menjadi data diskrit.

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.

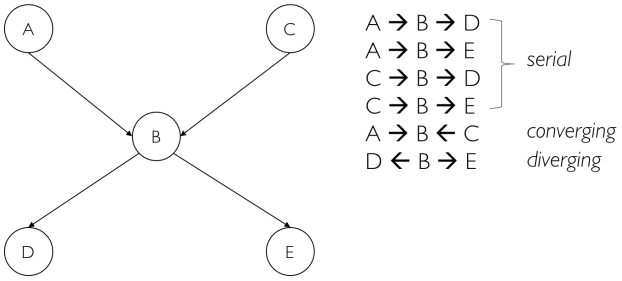
(b) *Diverging*. Bila kita mengetahui A maka kita bisa mengetahui sesuatu tentang C, dan sebaliknya. Tetapi apabila B diketahui, maka hubungan antara A dan C menjadi terputus. Dengan bahasa lebih matematis, A dan C independen kondisional ketika B diketahui.

(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

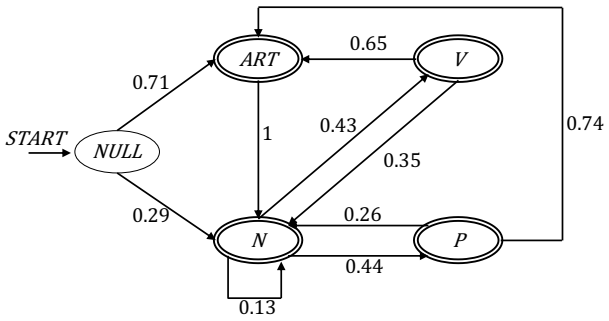
$$P(noun, verb, noun) = P(budi \mid noun)P(noun \mid null)P(menendang \mid verb) \\ P(verb \mid noun)P(bola \mid noun)P(noun \mid verb) \tag{8.13}$$

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].

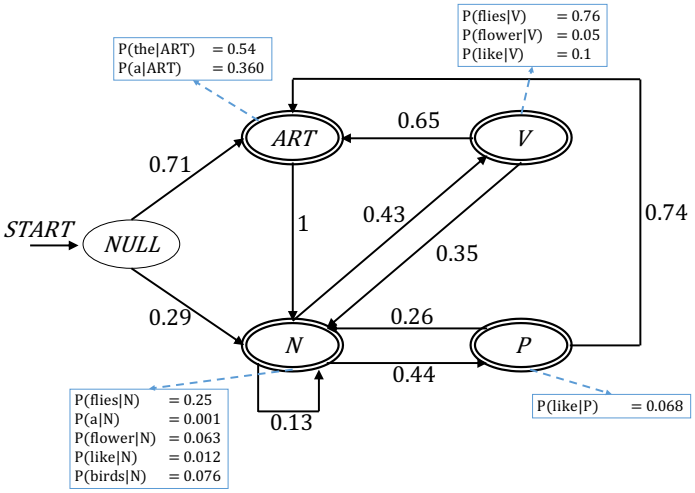
Tabel 8.2 dan Gambar 8.5 telah merepresentasikan probabilitas *prior*, sekarang kita ingin model yang kita punya juga mencakup ***lexical emission probabilities***, yaitu *likelihood* pada persamaan 8.12.

$P(the \mid ART)$	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		

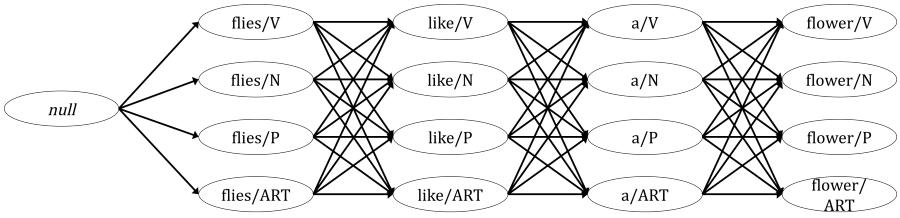
Tabel 8.3: *Lexical emission probabilities* [38]. Tabel ini hanya memuat informasi yang dibutuhkan untuk ilustrasi cerita (penyederhanaan).

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

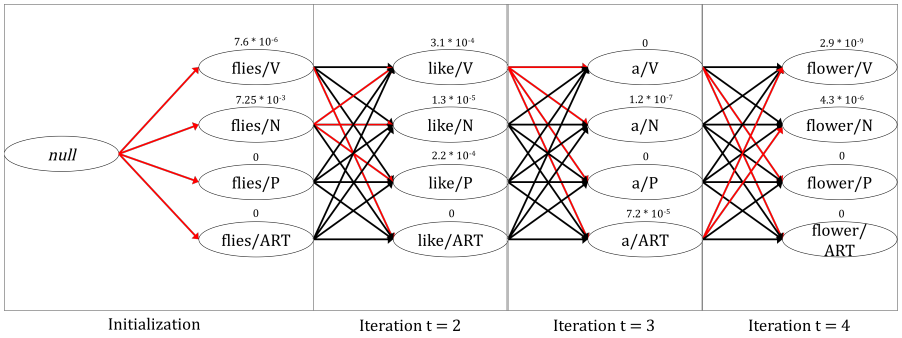
Sequence Identification Step
$c_T \leftarrow i$ that maximizes viterbi[i, T]
for $i = T - 1$ **to** 1 **do**
    $c_i \leftarrow$ backpointer[c_{i+1},$i+1$]

Gambar 8.8: Algoritma Viterbi [12, 38].

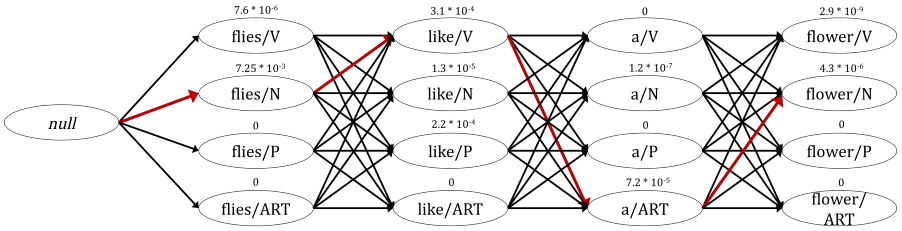
Perhatikan Gambar 8.8 yang menunjukkan *pseudo-code* untuk algoritma Viterbi. Variabel c berarti kelas kata, a adalah *transition probability*, dan b adalah *lexical-generation probability*. Pertama-tama, algoritma tersebut membuat suatu matriks berukuran SxT dengan S adalah banyaknya *states* (tidak

[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!