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 [23]. 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 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 **instans**/*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]

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

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=$\{0.7, 0.6\}$, sehingga probabilitas kelas menjadi=$\{0.538, 0.461\}$. Karena perbedaan probabilitas kelas relatif tidak terlalu besar (contoh ini adalah penyederhanaan), kita mungkin harus berpikir kembali untuk mengklasifikasikan instans ke kelas pertama. Hal ini berkaitan dengan seberapa yakin kamu mengklasifikasikan suatu instans ke

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

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** [24], 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.

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

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

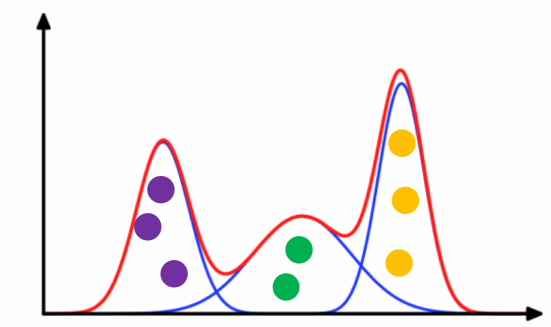
Apabila perbedaan suatu data terhadap kedua centroid bernilai sama, kita masukkan ke kelas dengan nomor urut lebih kecil.

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 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.

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



Gambar 4.1. Ilustrasi hubungan Clustering, kelas, dan Gaussian

4.3 K-nearest-neighbor

Ide **K-nearest-neighbor** (KNN) adalah mengelompokkan data ke kelompok yang memiliki sifat termirip dengannya [25]. 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 anggotanya lebih banyak menjadi *nearest neighbor*[3]. Dengan strategi tersebut, kita mengklasifikasikan data baru ke kelompok pertama.

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*!

[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

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

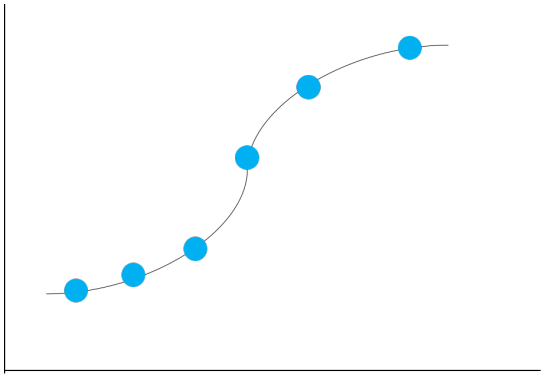
Bab ini membahas tentang model linear (algoritma parametrik), sebagai contoh pembelajaran yang sederhana karena cukup mudah dipahami idenya. Bab ini juga membahas *error function* dan *stochastic gradient descent*. Dimulai dari bab ini, kita akan memasuki materi lebih serius.

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.

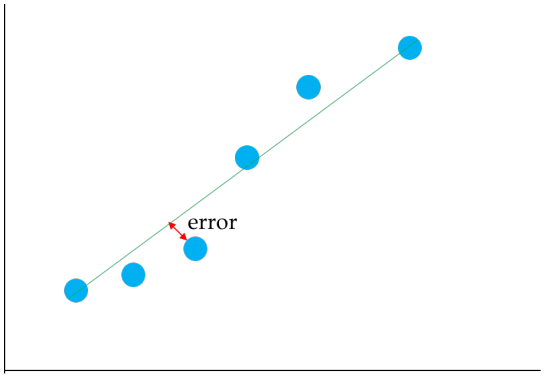
Error function paling mudah dijelaskan dalam permasalahan regresi. Diberikan $(x, y) \in \mathbb{R}$ sebagai *random variable*. Terdapat sebuah fungsi $f(x) \rightarrow y$, yang memetakan x ke y, berbentuk seperti pada Gambar 5.1. sekarang fungsi $f(x)$ tersebut disembunyikan (tidak diketahui), diberikan contoh-contoh pasangan (x_i, y_i); $i = 1, 2, ..., 6$ adalah titik pada dua dimensi (titik sampel), seperti lingkaran berwarna biru. 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 nilai sedemikian sehingga $g(x)$ mirip dengan $f(x)$. w adalah parameter sementara b adalah *bias*. Anggap kamu sudah berhasil melakukan pendekatan dan



Gambar 5.1. Contoh fungsi Sigmoid

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. (x_i, y_i) adalah pasangan *training data* (*input* - *desired output*). Nilai *squared error* dapat menjadi tolak ukur untuk membandingkan kinerja 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].

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

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

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

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

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

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. (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 testing data. Tujuan dari regresi/*machine learning* secara umum adalah untuk meminimalkan nilai *loss* atau *error* baik pada *training* maupun *unseen instances*. Secara umum, kita lebih ingin meminimalkan *loss*, dimana *error* dapat menjadi *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*[2] b. 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.

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

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

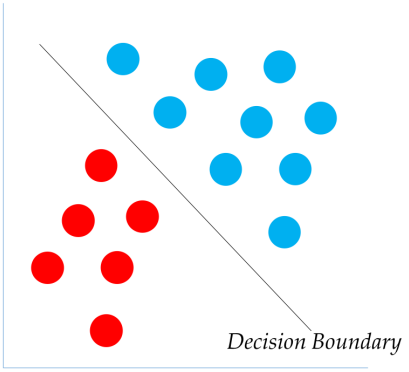
$$\text{output} = \text{sgn}(f(\mathbf{x}) \tag{5.8}$$

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*.

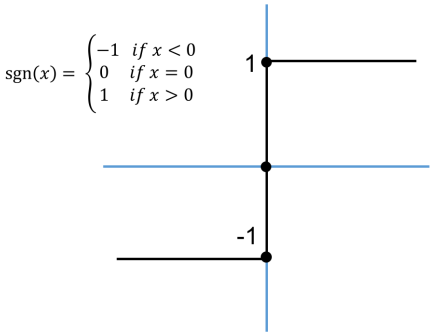
5.3 Log-linear Binary Classification

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,

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



Gambar 5.3. Contoh *decision boundary*



Gambar 5.4. Fungsi *sign*

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 data diklasifikasikan sebagai kelas tertentu (1 - nilai peluang, 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[3]. Hasil fungsi sigmoid, apabila di-*plot* maka akan berbentuk seperti Gambar 5.1 (berbentuk karakter "S").

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

[3] Menggunakan nilai peluang untuk klasifikasi

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

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

Nilai y ini diasosiasikan dengan suatu kelas. 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 $P(c_i)$ melambangkan probabilitas *input* diklasifikasikan ke kelas c_i dan N melambangkan banyaknya kelas. Untuk *binary classification*, $P(c_1) = 1 - P(c_2)$.

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

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. Dengan analogi, kita ingin fungsi logistik kita berbentuk semirip mungkin dengan fungsi sign untuk *high confidence*. *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 representasikan *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 $b \in \mathbb{R}^{1 \times 3}$ 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.

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

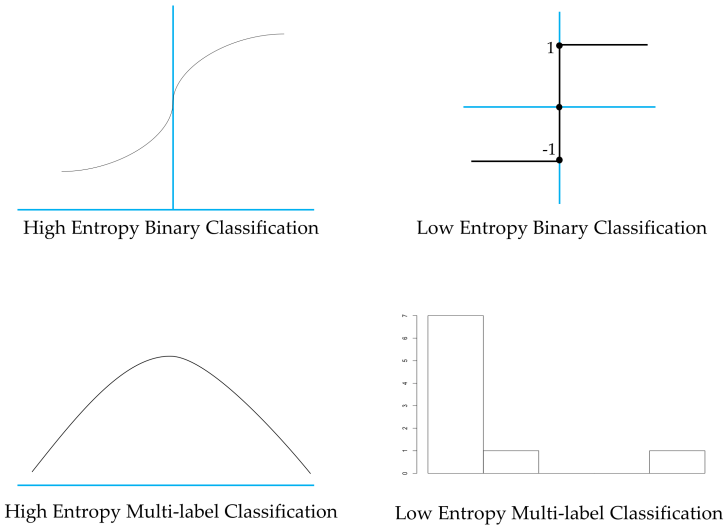
$$\hat{c} = \arg\max_{c_i \in \mathbf{c}} c_i \tag{5.14}$$

Seperti yang diceritakan pada subbab berikutnya, kita mungkin juga tertarik dengan probabilitas masing-masing kelas, bukan hanya *likelihood*-nya. Kita dapat menggunakan fungsi *softmax*[4] 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.

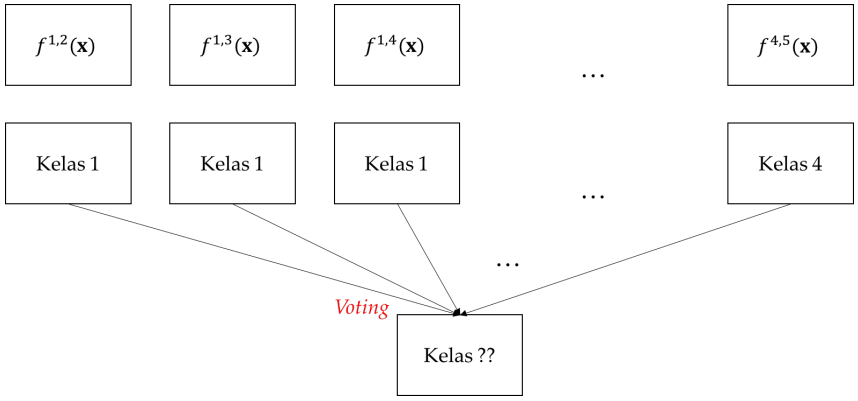
$$c_i = \frac{e^{(\mathbf{x} \cdot \mathbf{W} + \mathbf{b})_{[i]}}}{\sum_j e^{(\mathbf{x} \cdot \mathbf{W} + \mathbf{b})_{[j]}}} \tag{5.15}$$

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* sekecil 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.

[4] `https://en.wikipedia.org/wiki/Softmax_function`



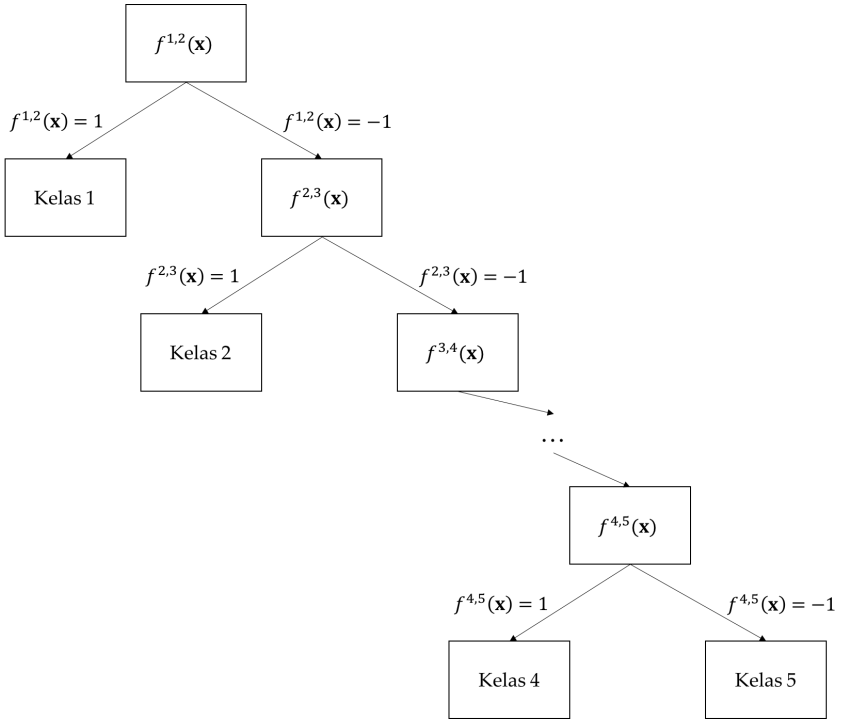
Gambar 5.5. *Classification Entropy*



Gambar 5.6. *One versus one*

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



Gambar 5.7. *One versus all*

semua model. Hal ini diilustrasikan pada Gambar 5.6 (untuk lima kelas). $f^{i,j}$ 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 probabilitas suatu instans 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 Gambar 5.8 sebagai ilustrasi, dimana "1" melambangkan *class assignment*.

Instans	Apel	Mangga	Sirsak
Gambar-1	1	0	0
Gambar-2	0	1	0
Gambar-3	0	0	1
...			

Multi-class Classification

Instans	Agama	Politik	Hiburan
Berita-1	1	1	0
Berita-2	0	1	1
Berita-3	1	0	1
...			

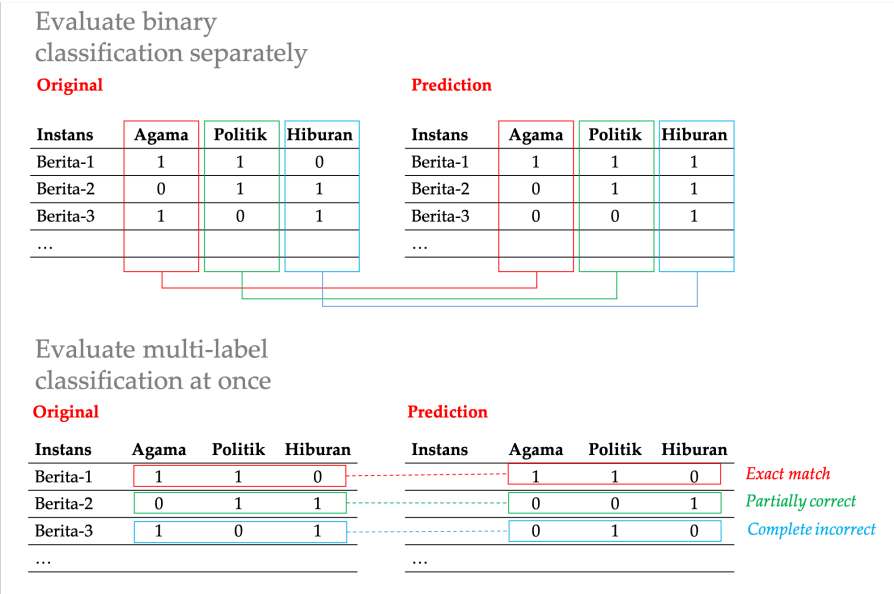
Multi-label Classification

Gambar 5.8. *Multi-class* vs. *multi-label classification*

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 classifiers* untuk melakukan klasifikasi. Kedua, kita dapat mengevaluasi kinerja *multi-label classification* itu sendiri. Perhatikan Gambar 5.9! 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 mendapatkan 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 jumlah instans (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



Gambar 5.9. Cara mengevaluasi *multi-label classifier*

dapat membaca [26] 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 persamaan 5.17, dimana θ adalah *learning parameter*[5], dan $\mathcal{L}$ adalah *loss function*. Perubahan parameter dapat menyebabkan perubahan *loss*. Karena itu, *loss function* memiliki θ sebagai parameternya.

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

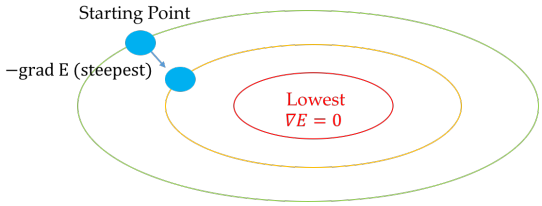
dimana $\hat{\theta}$ adalah nilai parameter paling optimal. Perhatikan, "arg min" dapat juga diganti dengan "arg max" tergantung **optimisasi** apa yang ingin dilakukan.

[5] Umumnya dapat berupa skalar, vektor, atau matriks. Demi istilah yang lebih generik, kita gunakan θ.

Sekarang, mari kita hubungkan dengan contoh yang sudah diberikan pada subbab sebelumnya. Kita coba melakukan estimasi *minimum squared error*, dengan mencari nilai *learning parameters* $\mathbf{w}$ yang meminimalkan nilai *error* pada model linear (persamaan 5.18)[6]. 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*[7]. 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 5.10!



Gambar 5.10. *Stochastic Gradient Descent*

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.

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

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

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

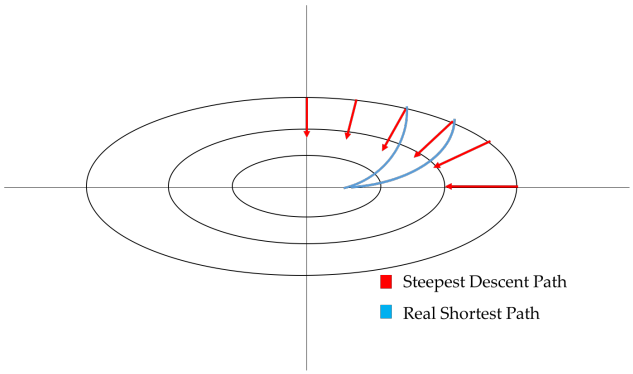
[7] Konsep *Hill Climbing* dapat digunakan untuk memaksimalkan *utility function*. Konsep tersebut sangat mirip dengan *gradient descent* `https://en.wikipedia.org/wiki/Hill_climbing`.

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 [27].

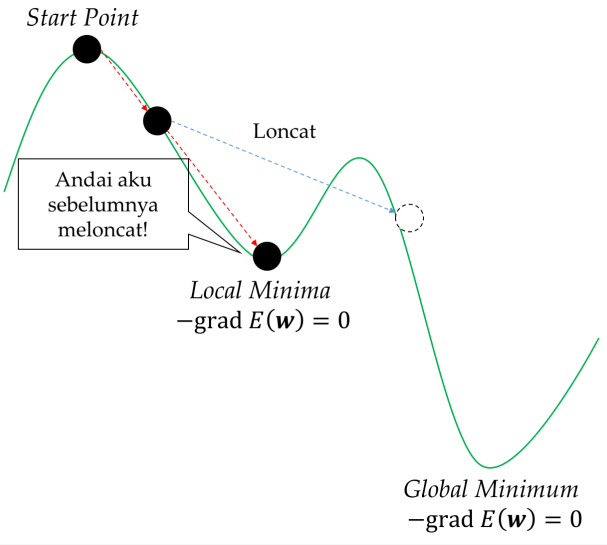
Walaupun kamu berharap bisa menuju titik terendah dengan menelusuri jalan terdekat dengan kemiringan paling tajam, tapi kenyataanya hal tersebut bisa jadi bukanlah jalan tercepat, seperti yang diilustrasikan pada Gambar 5.11. Warna merah melambangkan jalan yang dilalui *gradient descent*, sementara warna biru melambangkan jalanan terbaik (tercepat).



Gambar 5.11. *Stochastic Gradient Descent* 2

Pandangan kita yang terbatas layaknya kita tidak bisa melihat keseluruhan pengunungan secara keseluruhan, 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



Gambar 5.12. *Stuck at local minima*

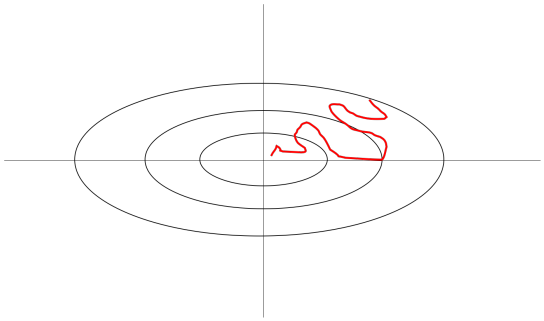
macet/berhenti saat sudah mencapai *local minima*, yaitu nilai minimum pada suatu daerah lokal saja. Sebagai ilustrasi, perhatikan Gambar 5.12. Kamu berada di puncak pegunungan kemudian turun bertahap. Kemudian, kamu sampai di suatu daerah landai ($-\text{grad } E(\mathbf{w}) = 0$). Kamu pikir daerah landai 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 5.13. Goyangan tersebut ibarat "meloncat-loncat" pada ilustrasi Gambar 5.12.

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



Gambar 5.13. *Swing*

mentara 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 [17]: (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.

$$f(\mathbf{x}) = x_1 w_1 + x_1 w_2 + x_1 x_2 w_3 + b \tag{5.24}$$

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 delta x_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*.

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

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 [17]. 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.

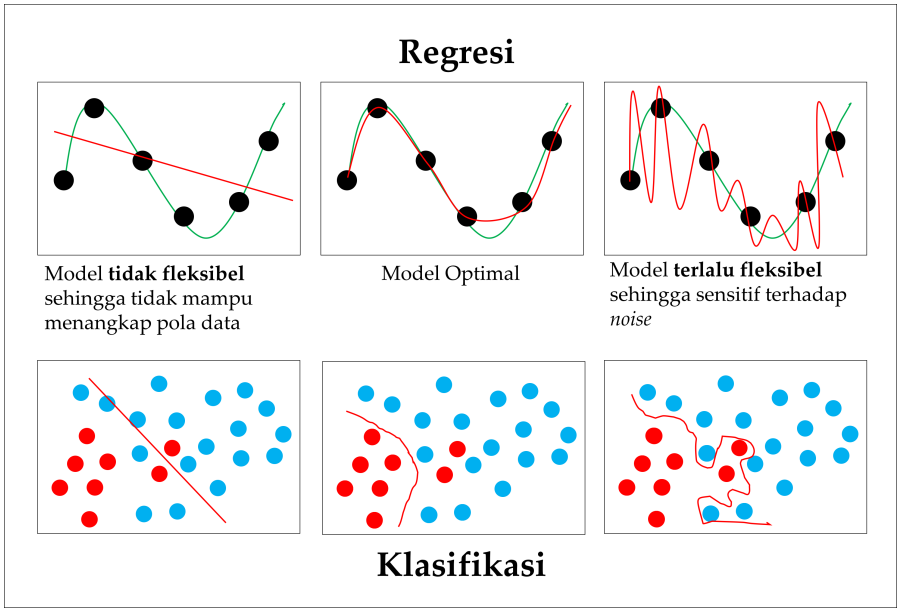
5.8 Overfitting dan Underfitting

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*, *development*, 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[8]. Dengan demikian, kita ingin *loss* atau *error* pada *training*, *development*, 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.

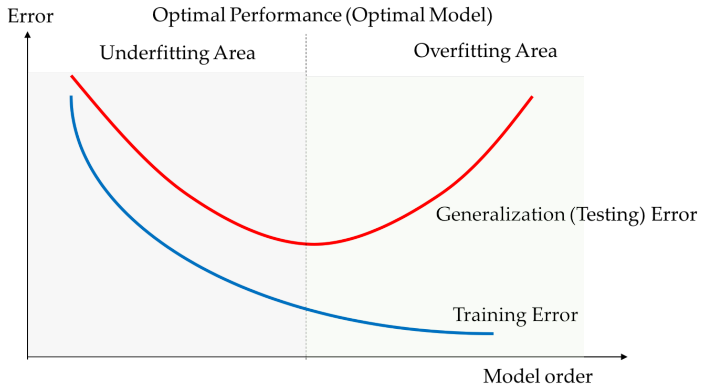
Underfitting adalah keadaan ketika kinerja model bernilai buruk baik pada *training* atau *development* maupun *testing data*. *Overfitting* adalah keadaan ketika kinerja model bernilai baik untuk *training* tetapi buruk pada *unseen data*. Hal ini diilustrasikan pada Gambar 5.14. *Underfitting* terjadi akibat model yang terlalu tidak fleksibel, yaitu memiliki kemampuan yang rendah untuk mengestimasi variasi fungsi. Sedangkan, *overfitting* terjadi ketika model terlalu fleksibel, yaitu memiliki kemampuan yang terlalu tinggi untuk mengestimasi banyak fungsi atau terlalu mencocokkan diri terhadap *training* data. Perhatikan kembali Gambar 5.14, 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 5.15. Model yang ideal adalah model yang memiliki kinerja yang

[8] Baca teknik *sampling* pada buku statistika.
[9] Rekonstruksi `https://www.inf.ed.ac.uk/teaching/courses/iaml/slides/eval-2x2.pdf`



Gambar 5.14. *Underfitting* vs. *Overfitting*[9]



Gambar 5.15. *Selection Error*

baik pada *training*, *development*, 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 *development* 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 jumlah instans yang relatif sedikit. Kamu harus hati-hati terhadap *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/error* dapat diminimalisir. 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[10]. 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.

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

Pilihan umum untuk fungsi *regularization* pada umumnya adalah L_2 dan L_1 norm. L_2 *regularization* menggunakan jumlah kuadrat dari *Euclidean norm*[11] 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

[10] Baca buku Yoav Goldberg [1] untuk mendapat penjelasan lebih baik.
[11] `https://en.wikipedia.org/wiki/Norm_(mathematics)`

diturunkan pada seluruh poin, sementara L_1 tidak dapat diturunkan pada semua poin[12]. 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.

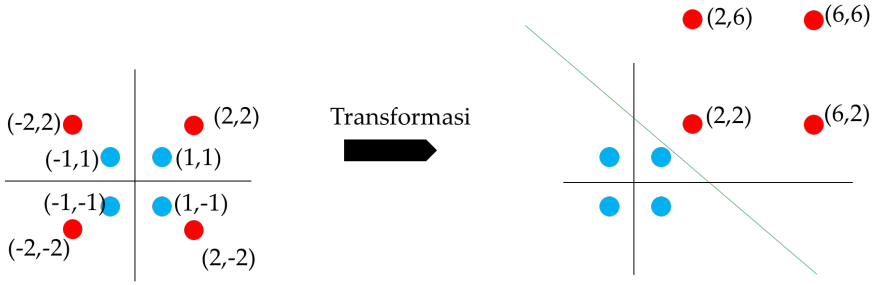
5.10 Transformasi Data

Seperti yang sudah dijelaskan sebelumnya, alangkah baik apabila semua data memiliki hubungan secara linear atau bersifat *linearly separable*. Kenyataannya, 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 5.16. 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.

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

Secara umum, fungsi transformasi tidaklah sesederhana contoh yang diberikan. Fungsi transformasi pada umumnya menambah dimensi data (misal dari dua

[12] `https://stats.stackexchange.com/questions/136895/why-is-the-l1-norm-in-lasso-not-differentiable`



Gambar 5.16. Contoh transformasi [7]

dimensi, menjadi tiga dimensi). Beberapa fungsi transformasi (dikenal juga dengan istilah *kernel*) yang terkenal diantaranya[13]:

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) [28, 29]. 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 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:

[13] `https://en.wikipedia.org/wiki/Kernel_method`

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. [17])

5.5. Linear Discriminant Analysis
Jelaskan prinsip dan cara kerja *linear discriminant analysis*! (Hint: baca buku oleh James et al. [17])

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 [30, 31] yang terinspirasi oleh teori informasi [32]. Algoritma ini sudah cukup tua, tetapi layak dimengerti. ID3 adalah salah satu varian dari *supervised learning*.

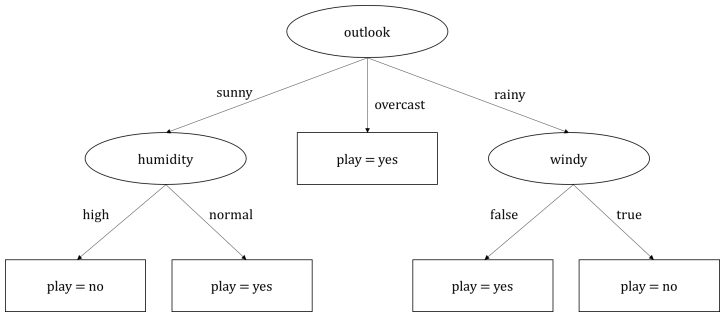
6.1 Inductive Learning

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 *instans* 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

Seperti yang dijelaskan pada subbab sebelumnya, *decision tree* adalah varian dari *inductive learning*. ID3 adalah salah satu algoritma varian *decision tree* [31]. *Decision tree* dibangun berdasarkan asumsi bila atribut yang ada memberikan informasi yang cukup memadai maka kita mampu membangun *decision tree* yang mampu mengklasifikasikan seluruh instans di *training data* [31]. Akan tetapi, kita tentunya ingin melakukan generalisasi, yaitu *decision tree* yang juga mampu mengklasifikasikan objek dengan benar untuk instans 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 instans 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*

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.

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

dimana c_i adalah jumlah instans diklasifikasikan sebagai kelas ke-i (atau secara lebih umum, ke *node-i*). *Information gain* dihitung sebagai persamaan 6.3.

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

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

Sebagai contoh, mari kita hitung *Information gain* untuk atribut *outlook* sebagai *root*. Dari keseluruhan data terdapat 9 instans untuk $play = yes$ dan 5 instans 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 instans dengan $play = yes$ dan tiga instans 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 instans dengan $play = yes$ dan tidak ada instans 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 instans dengan $play = yes$ dan dua instans 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.

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

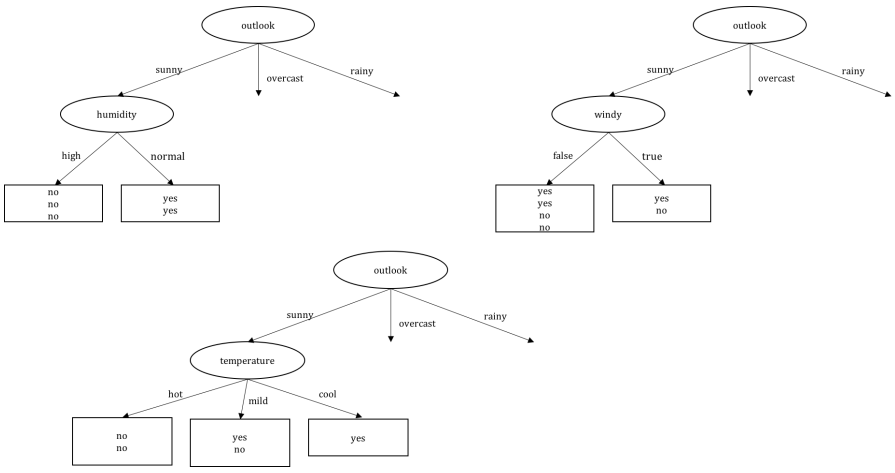
Dengan demikian, kita memilih atribut *outlook* sebagai *root*.

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 instans dengan kelas $play = yes$ dan tiga instans 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

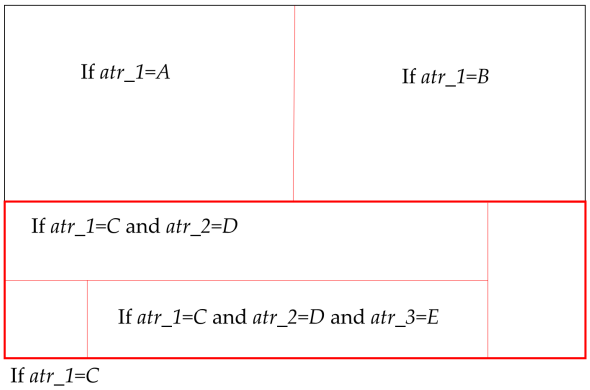
6.3 Isu pada ID3

Pada algoritma *decision tree* secara umum, terdapat beberapa isu diantara lain [7, 33]:

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

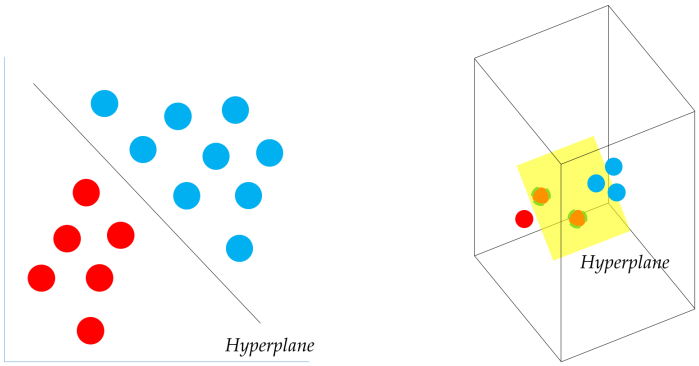
> "More data beats clever algorithms, but better data beats more data."
>
> Peter Norvig

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 [17]. *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. [17].

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.

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

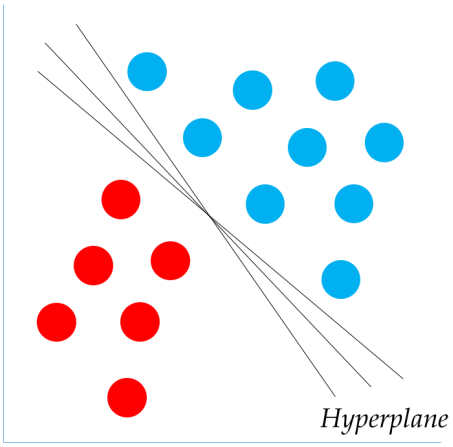
$$\mathbf{x} \cdot \mathbf{w} + b = 0 \tag{7.2}$$

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*.

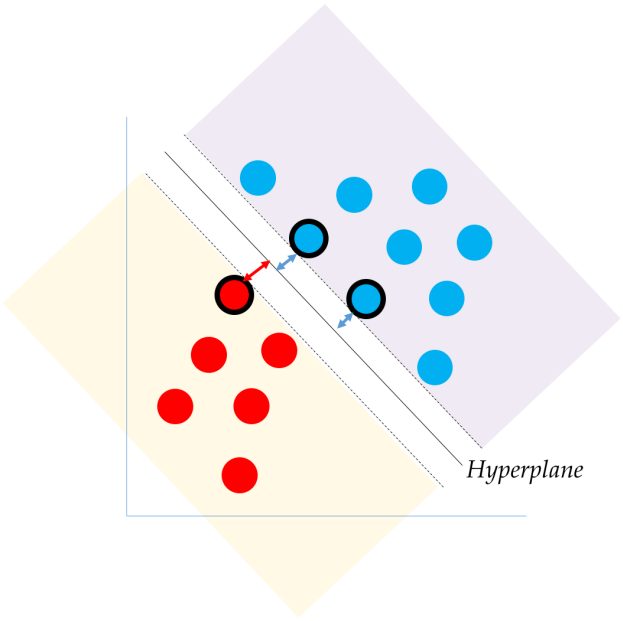
$$\text{if } \mathbf{x} \cdot \mathbf{w} + b > 0, \text{then class } A \tag{7.3}$$

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

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 7.4 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*).



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

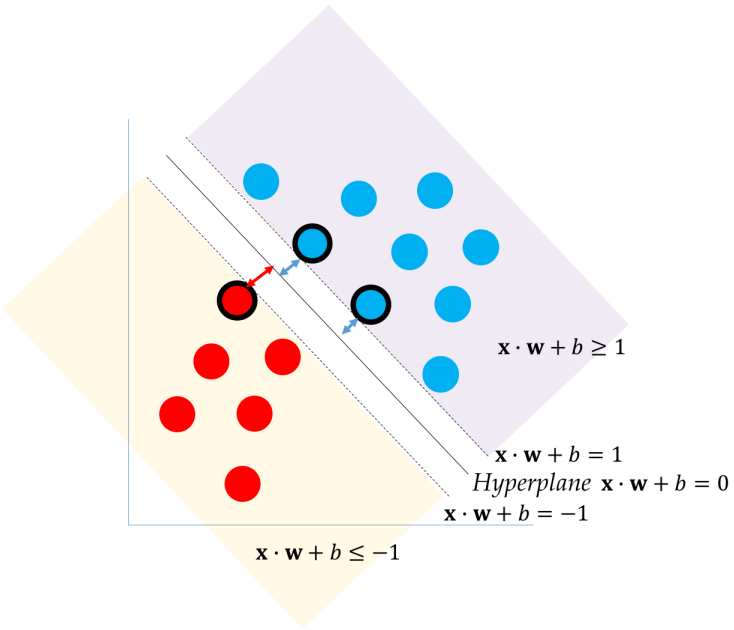


Gambar 7.3. *Maximal Margin Hyperplane*. Bentuk lingkaran yang memiliki *border* berwarna hitam menandakan data terluar pada masing-masing kelas, dikenal sebagai *support vectors*. Garis putus-putus melambangkan garis yang dibentuk oleh masing-masing *support vectors* untuk masing-masing kelas (*margin*)

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!

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

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



Gambar 7.4. *Maximal Margin Classifier*

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.

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

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

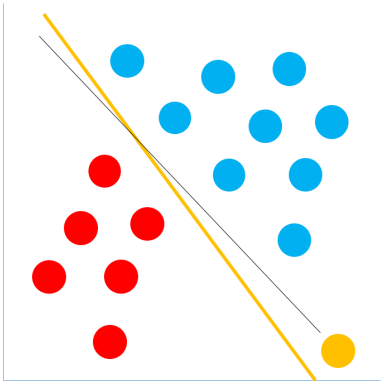
Sekarang, kita formalisasi *maximal margin classifier* secara matematis. Objektifnya adalah memaksimalkan *margin* (persamaan 7.9) dengan menjaga setiap *training data* diklasifikasikan dengan benar (persamaan 7.10).

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

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

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 7.5. *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)

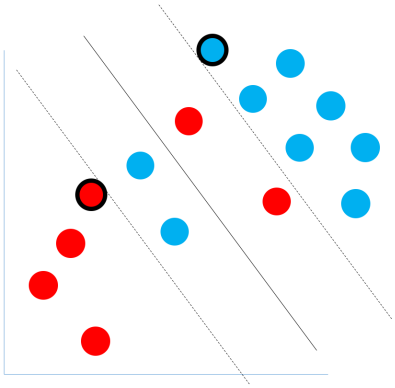
7.2 Support Vector Classifier

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.

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

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

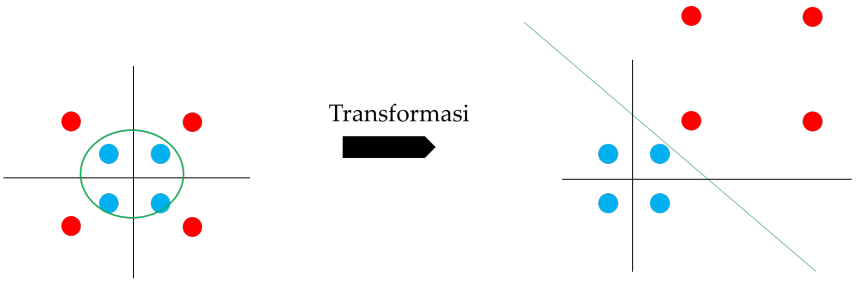
Ilustrasi *support vector classifier* diberikan pada Gambar 7.6. Disini, kita sedikit modifikasi definisi *support vectors* sebagai observasi yang tepat jauh pada *margin* atau pada daerah yang tidak sesuai dengan kelasnya [17].



Gambar 7.6. *Support Vector Classifier*. Lingkaran dengan *border* 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 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)
2. Radial Basis Function Kernel (persamaan 7.14, σ melambangkan varians)

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

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

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

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 $\mathbf{x}'$

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

melambangkan suatu data baru, $\mathbf{x}_i$ adalah suatu instans 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 instans 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 instans 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.

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

Ketika menggunakan fungsi *kernel*, kita menghitung kemiripan (*inner product*) antara dua vektor pada dimensi transformasi. Kita mengghanti *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 [34]. Tetapi algoritma ini penting untuk diketahui karena digunakan sebagai teknik dasar untuk *automatic speech recognizer* (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, 35, 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 pembaca membaca dengan sabar.

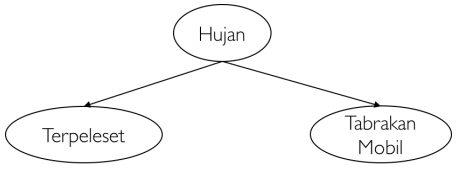
8.1 Probabilistic Reasoning

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.

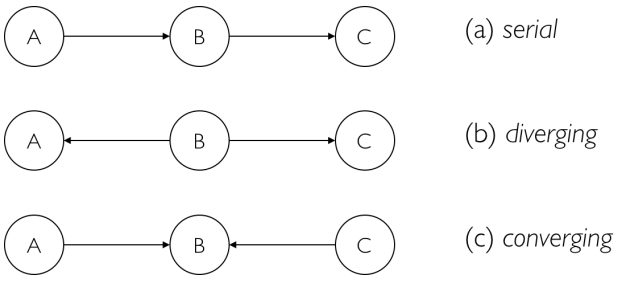
jadi, maka "terpeleset" juga pasti akan terjadi. Tetapi tidak sebaliknya, apabila kita "terpeleset", belum tentu "hujan" juga terjadi. Pada *probabilistic reasoning* kita mengkuantifikasi kepastian atau ketidakpastian itu. Apabila "hujan" terjadi, berapa besar kemungkinan "terpeleset" juga terjadi (dan 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 hujan, mungkin seseorang terpeleset dan kecelakaan juga terjadi". Tetapi, apabila ada cerita lain bahwa seseorang "terpeleset" dan ada juga "kecelakaan"; belum tentu keduanya disebabkan oleh "hujan" (kedua kejadian tidak terhubung).



Gambar 8.1. Contoh *Bayesian Network*

Terdapat beberapa kerangka berpikir yang berbeda-beda, tergantung hubungan kausalitas antara *events*: *serial*, *diverging*, dan *converging* (diilustrasikan pada Gambar 8.2.



Gambar 8.2. Tipe jaringan kausalitas

Berdasarkan hubungan/jaringan kausalitas antara *events*, ada beberapa kerangka berpikir yang dapat digunakan: *serial*, *diverging*, dan *converging* [5]. 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.

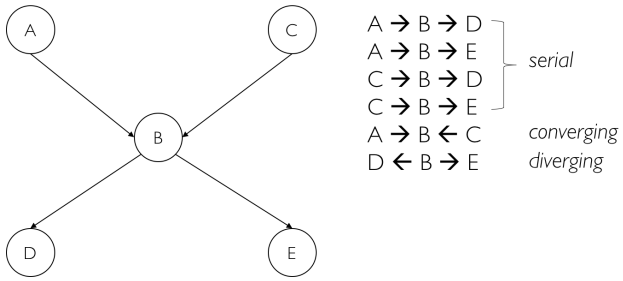
(b) *Diverging*. Bila kita mengetahui A maka kita bisa mengetahui sesuatu tentang C, dan sebaliknya. Tetapi apabila B diketahui, maka hubungan antara A dan C menjadi terputus. Dengan bahasa lebih matematis, A dan C independen kondisional ketika B diketahui.

(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.

$$P(A, B, C, D, E) = P(D \mid B)\ P(E \mid B)\ P(B \mid A, C)\ P(A)\ P(C) \qquad (8.1)$$

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.

$$P(E \mid A) = P(E \mid B)\ P(B \mid A)\ P(A) + P(E \mid \neg B)\ P(\neg B \mid A)\ P(A) \quad (8.2)$$

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

$$P(E \mid B, A) = P(E \mid B)\ P(B) \quad (8.3)$$

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.

$$P(C \mid B, A) = P(C \mid B)\ P(B \mid A)P(A) \quad (8.4)$$

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[3]. 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[4]. *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

[2] Parameter $\mathbf{w}$ dihilangkan untuk menyederhanakan penjelasan.
[3] Karena x adalah *fixed input* dan y terbaiklah yang ingin kita temukan.
[4] x_i dan y_i dapat berupa vektor

pemaparan subbab 8.1, apabila suatu *event* dikondisikan variabel lain dengan 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*". 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* [5], diberikan pada Tabel 8.1.

POS *tagging* adalah salah satu bentuk pekerjaan ***sequential classification***. Diberikan sebuah sekuens (dalam hal ini 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, 35].

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

Dimana C adalah daftar kelas kata. Akan tetapi, menghitung persamaan 8.6 sangatlah sulit karena dibutuhkan data yang sangat banyak (kombinasi sekuens kata sangat sulit untuk didaftar/sangat banyak). Teori Bayes digunakan untuk melakukan aproksimasi permasalahan ini. Ingat kembali teori Bayes seperti pada persamaan 8.7.

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

[5] `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

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.

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

dimana 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*[6] 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).

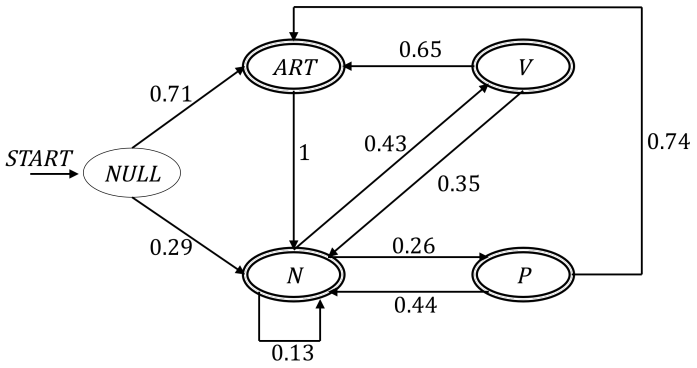
Sebagai contoh, perhatikan Tabel 8.2. *ART* adalah *article*, *N* adalah *noun*, *V* adalah *verb* dan *P* adalah *preposition*. Mereka adalah contoh kelas kata yang disederhanakan demi membuat contoh yang mudah. Tabel 8.2 yang merepresentasikan probabilitas transisi kelas kata, ketika dikonversi menjadi *weighted automaton*, akan menjadi Gambar 8.4.

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

Tabel 8.2. Probabilitas bigram [35]

Tabel 8.2 dan Gambar 8.4 telah merepresentasikan probabilitas *prior*, sekarang kita ingin model yang kita punya juga mencakup ***lexical emission probabilities***, yaitu *likelihood* pada persamaan 8.12.

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



Gambar 8.4. *Weighted automaton* [35]

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

Tabel 8.3. *Lexical emission probabilities* [35]

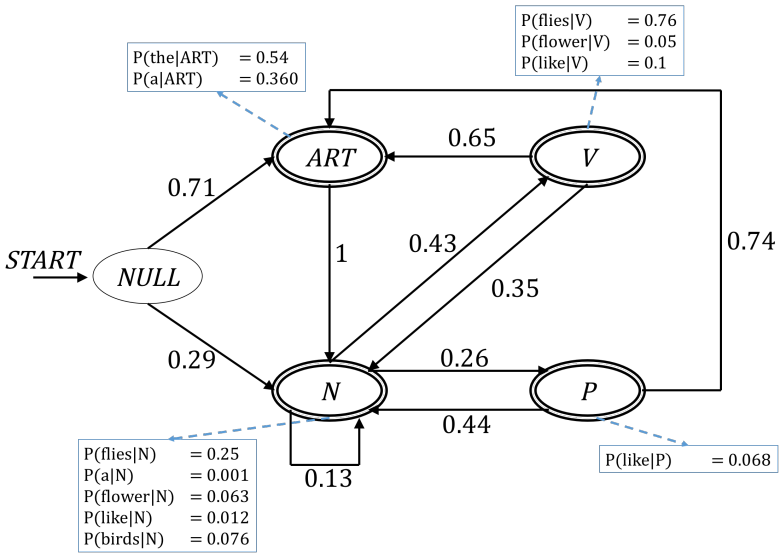
Seumpama kita mempunyai *lexical emission probabilities* seperti pada Tabel 8.3. Setiap *state* pada automaton, dapat menghasilkan/meng-*output*-kan suatu kata (*word*) dengan probabilitas pada Tabel 8.3. Kita kembangkan lagi Gambar 8.4 dengan tambahan informasi *lexical emission probabilities* menjadi Gambar 8.5. 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*) [35].

Diberikan kalimat "*flies like a flower*", untuk menghitung sekuens kelas kata untuk kalimat tersebut, kita menyelusuri automaton Gambar 8.5. Hasil penelusuran memberikan kita kombinasi sekuens yang mungkin seperti pada Gambar 8.6. Pekerjaan berikutnya adalah, dari seluruh kombinasi sekuens yang mungkin ($\prod_{i=1}^{T} P(w_i|c_i)P(c_i|c_{i-1})$), bagaimana cara kita menentukan sekuens terbaik (paling optimal), yaitu sekuens dengan probabilitas tertinggi (diberikan pada subbab berikutnya).

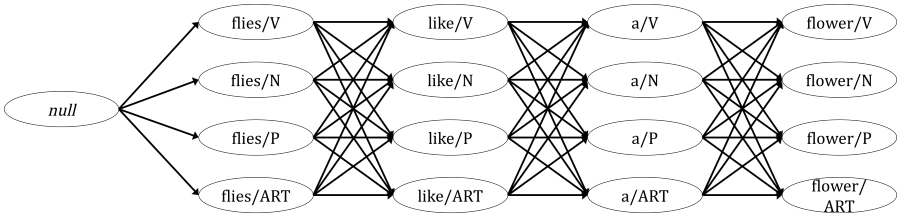
Secara formal, ***hidden markov model tagger*** didefinisikan oleh beberapa komponen [12]:

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)**.



Gambar 8.5. *Hidden Markov Model*



Gambar 8.6. Sekuens yang mungkin (*brute force*)

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* [7]. 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

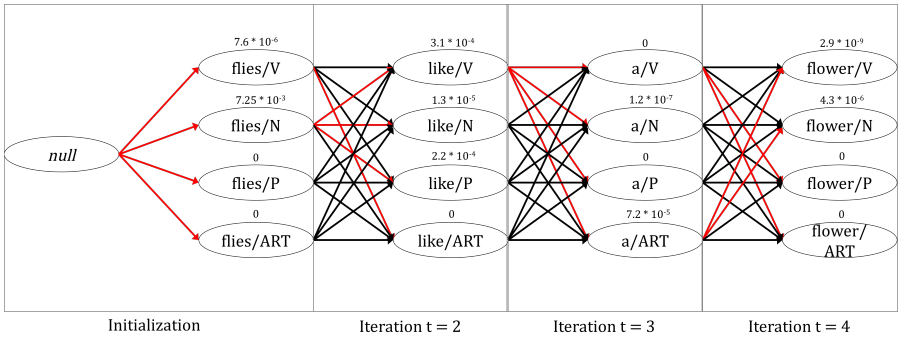
Sequence Identification Step
$c_T \leftarrow i$ that maximizes viterbi[i, T]
for $i = T-1$ **to** 1 **do**
  $c_i \leftarrow$ backpointer[c_{i+1},$i+1$]

Gambar 8.7. Algoritma Viterbi [12, 35]

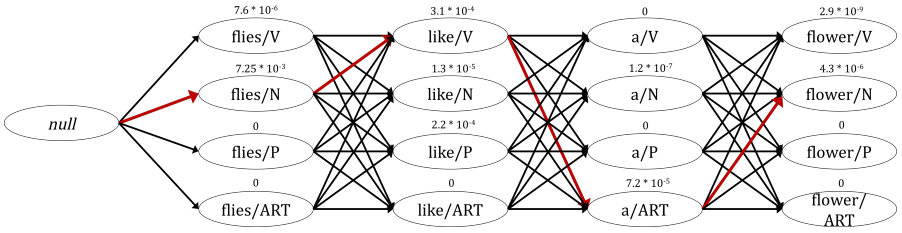
Perhatikan Gambar 8.7 yang menunjukkan *pseudo-code* untuk algoritma Viterbi. Variabel c berarti kelas kata, a adalah *transition probability*, dan b adalah *lexical-generation probability*. Pertama-tama, algoritma tersebut membuat suatu matriks berukuran SxT dengan S adalah banyaknya *states* (tidak termasuk *start state*) dan T (*time*) adalah panjang sekuens. Pada setiap iterasi, kita pindah ke observasi kata lainnya. Gambar 8.8 adalah ilustrasi algoritma Viterbi untuk kalimat *input* (*observed sequence*) "*flies like a flower*"

[7] `https://en.wikipedia.org/wiki/Edit_distance`

dengan *lexical generation probability* pada Tabel 8.3 dan *transition probabilities* pada Tabel 8.2 (bigram) [35]. 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 8.9. Dengan itu, kita mendapatkan sekuens "flies/N like/V a/ART flower/N".



Gambar 8.8. Ilustrasi algoritma Viterbi per iterasi



Gambar 8.9. 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].

[8] Walau kami mengakui penjelasannya kurang baik. Mohon maaf tetapi kami sudah berusaha.

8.6 Proses Training Hidden Markov Model

Hidden Markov Model (HMM) adalah salah satu varian *supervised learning*[9], 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. Sebagai contoh, *emission probability* suatu kata untuk setiap kelas kata diberikan pada persamaan 8.14 (bentuk *softmax function*), dimana N melambangkan banyaknya data (banyaknya pasangan sekuens *input-output*)

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

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.

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

dimana $\alpha_i(t)$ adalah probabilitas untuk menghasilkan kata $w_1, w_2, \ldots, w_t$ dengan w_t dihasilkan (*emitted*) oleh c_i.

Pseudo-code perhitungan kemunculan kelas c_i sebagai kategori pada posisi tertentu diberikan oleh Gambar 8.10, 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 (t ke T), dari posisi ke-t sampai ujung akhir. Anda

[9] 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 8.10. Algoritma *forward* [35]

dapat melihat *pseudo-code* pada Gambar 8.11, dengan a adalah *transition probability* dan b adalah *emission probability*.

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 8.11. Algoritma *backward* [35]

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 it-

erasi, 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 8.12. 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 8.12. Walaupun HMM menggunakan *independence assumption*, tetapi kita dapat mengikutsertakan pengaruh probabilitas keseluruhan sekuens untuk perhitungan probabilitas keberadaan kita pada suatu *state* pada saat tertentu. Metode ini dapat

dianggap sebagai suatu cara optimalisasi menggunakan *smoothing* [10] terhadap nilai parameter (**A** dan **b**). Kita mencapai titik *local optimal* apabila tidak ada perubahan parameter.

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!

[10] `https://en.wikipedia.org/wiki/Smoothing`

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 5. Hanya saja, beberapa hal lebih baik dijelaskan secara lebih mendalam ketika sudah mempelajari beberapa algoritma pembelajaran mesin, yaitu *feature engineering*, *feature selection* dan penjelasan lebih lanjut *cross validation*. Kamu dapat menganggap bab ini sebagai kelanjutan tips dan trik pembelajaran mesin lanjutan 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 variabel-variabel 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* (instans) 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

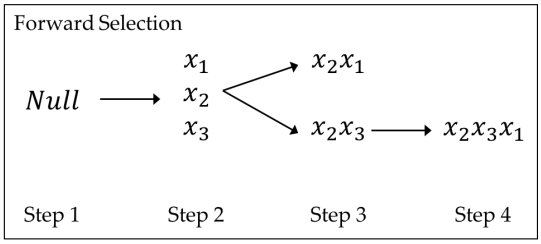
[1] `Curse of Dimensionality`

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***.

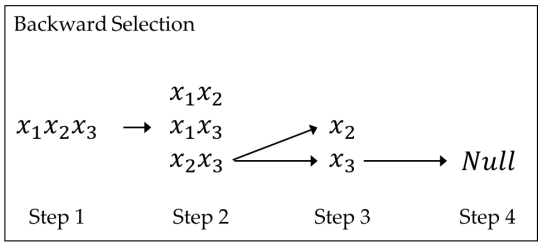
Pada *forward selection*, kita mulai dengan suatu model yang tidak menggunakan fitur apapun sama sekali, yaitu meng-*assign* kelas yang paling sering muncul di dataset, pada *input*. Setelah itu, kita tambahkan satu per satu fitur pada setiap langkah. Langkah berikutnya, kita gunakan satu fitur. Diantara 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 $1+F+(F-1)+\cdots+1 = 1+F(F+1)/2$. Apabila kita memiliki fitur sebanyak $F = 10$, kombinasi *brute force* menghasilkan $1,024$ kombinasi, sementara *forward selection* hanya sebanyak 56.



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 selection* mengurangi satu fitur pada tiap langkah. Ilustrasi diberikan pada Gam-

bar 9.2. Seperti *forward selection*, *backward selection* juga hanya mencoba sebanyak $1 + F(F + 1)/2$ 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 [17].



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 [17]. 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 9.4).

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

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

[2] baca buku [17] 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 (Dimension 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 $M < F$ (*dimension 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 *dimension 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 *interpretable* secara jumlah parameter, tetapi kita membutuhkan usaha ekstra untuk mengerti representasi fitur-fitur pada dimensi yang baru.

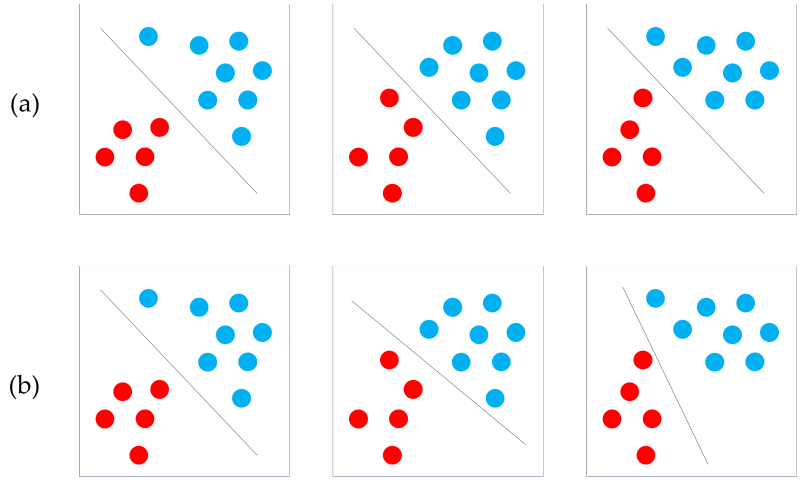
Teknik yang digunakan untuk mengurangi dimensi adalah *principal component analysis*. Ide dasarnya adalah mencari bentuk data (pada dimensi lebih kecil) yang memiliki nilai varians tinggi untuk tiap fiturnya, karena varians yang tinggi berarti kemampuan fitur yang tinggi dalam melakukan diskriminasi (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 grup 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 [17]. 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.

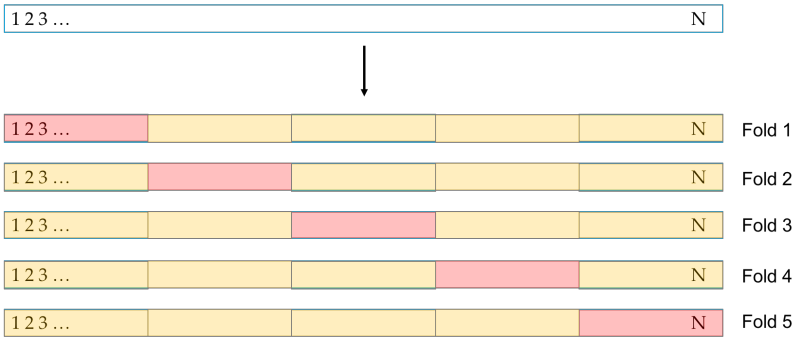
9.4 Cross Validation

Ingat kembali materi bab-bab sebelumnya bahwa pada umumnya kita membagi dataset menjadi tiga grup: *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 grup 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*. Walaupun tidak boleh ada data yang sama (*overlap*) pada ketiga grup tersebut, memiliki *validation* dan *testing data* yang dibedakan adalah hal yang cukup bijaksana.



Gambar 9.3. Model yang stabil (a) memberikan *decision boundary* 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 grup 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*). Tetapi ada masalah apabila kita hanya menguji model pada satu set *validation data*, yaitu *bias*[3]. *Validation data* yang tetap menyebabkan kita tidak mengetahui variasi kinerja model [17]. Sebisanya, kita ingin mengetes kinerja model pada beberapa dataset, kemudian mengevaluasi bagaimana variasi kinerja model. Kegiatan semacam ini membuat kita mengetahui apakah suatu model cukup stabil[4] dan fleksibel. Stabil berarti kinerja model tidak begitu berubah, diberikan *input* yang sedikit berubah (ilustrasi diberikan pada Gambar 9.3). 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 telah dijelaskan pada bab 5.



Gambar 9.4. 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

[3] *statistical bias*

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

membagi-bagi *dataset* yang kita miliki menjadi K bagian. Kita latih model menggunakan $K-1$ bagian, kemudian menguji model dengan menggunakan satu bagian lainnya sebagai *validation data* yang mengaproksimasi kinerja pada *testing data* (Ilustrasi pada Gambar 9.4). Hal ini disebut sebagai K-***fold-cross-validation***. Untuk $K = N$ yaitu jumlah data, kita sebut teknik tersebut sebagai ***leave-one-out-cross-validation***. Perlu diperhatikan, distribusi tiap-tiap subset data haruslah sama (*stratified sampling*). Dengan 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 9.4. 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).

$$\mathrm{CV}_{(K)} = \frac{1}{K}\sum_{i=1}^{K}\mathrm{KinerjaModel}_i \tag{9.4}$$

9.5 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*[5] 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* [36].

[5] `Pranala Post Yoav Goldberg`

Soal Latihan

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

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

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

9.4. 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 [37], walaupun relatif lama, tetapi paper tersebut memberikan penjelasan yang mudah dimengerti tentang *clustering*. Selain itu, kamu juga dapat membaca *paper* oleh Saad et al. [38].

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 [39]: (1) *partitional*, yaitu menentukan partisi sejumlah 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 sejumlah 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 sejumlah 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 instans 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* [37]:

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.

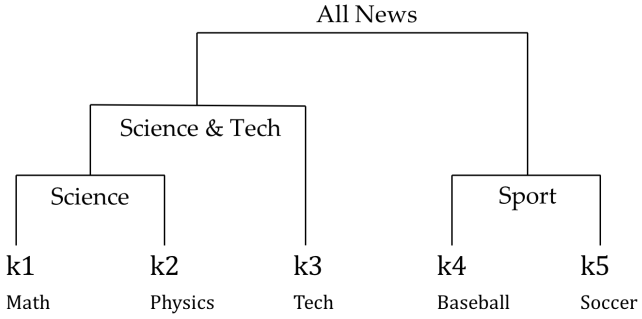
Sebagai contoh, algoritma *hierarchical clustering* menghasilkan struktur hirarkis seperti pada gambar 10.1 yang disebut **dendogram**. Dendogram melambangkan taksonomi, sebagai contoh taksonomi dokumen berita *sport*, dipecah menjadi *baseball* dan *soccer*.

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 sejumlah K *clusters*. Kamu dapat menganggap satu instans data sebagai suatu *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 [39] (untuk data numerik):

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



Gambar 10.1. Ilustrasi *hierarchical clustering*

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

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

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

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

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* $\mathbf{U}$.

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:

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

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 [38].

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

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*.

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 [37])
(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!