# 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 [22]. 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 (*variabel*), $t$ adalah fitur (satu fitur, bukan *feature vector*) dan $F$ adalah banyaknya fitur. Kita memprediksi kelas berdasarkan probabilitas kemunculan nilai fitur pada kelas tersebut. Hal ini didefinisikan pada persamaan 4.1 dan 4.2. Interpretasi kedua persamaan itu sangatlah sederhana. Kita menetapkan kelas untuk suatu data, berdasarkan bagaimana probabilitas korespondisi fitur-fiturnya terhadap kelas tersebut.

$$likelihood(c_i) = \arg\max_{c_i \epsilon C} P(c_i) \prod_{f=1}^{F} P(t_f|c_i) \tag{4.1}$$

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

| 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.

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

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

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*.

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

Karena $P(play = no) > P(play = yes)$ 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.

## 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** [23], kita harus mengikuti langkah-langkah sebagai berikut:

| 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.

| 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.

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). 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 *assign-*

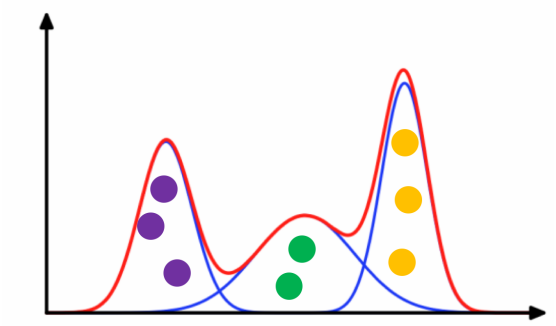
[1] Cara lain memilih akan dijelaskan pada bab 8.

| 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.

*ment* 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).



**Gambar 4.1.** Ilustrasi Hubungan Clustering, Kelas, dan Gaussian.

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.

## 4.3 K-nearest-neighbor

Ide **K-nearest-neighbor** (KNN) adalah mengelompokkan data ke kelompok yang memiliki sifat termirip dengannya [24]. 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 upabab 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.

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

**Tabel 4.8.** Perbedaan data baru vs data orang kaya.

## Soal Latihan

### 4.1. Data numerik

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

### 4.2. K-means, KNN, GMM, EM

Buktikan bahwa K-means, K-nearest-neighbor, *Gaussian Mixture Model*, dan

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

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*.

## 5.1 Curve Fitting dan Error Function

Pertama, penulis ingin menceritakan tentang 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.

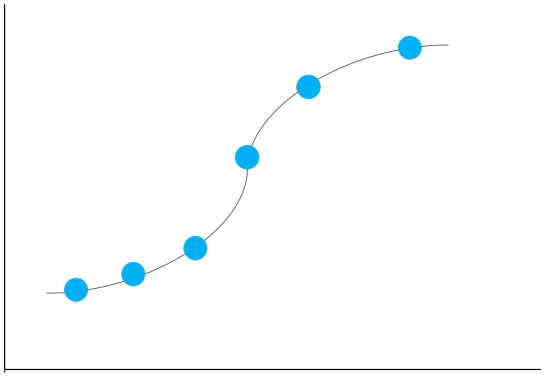
*Error function* paling mudah dijelaskan dalam permasalahan regresi. Masih ingat contoh bab sebelumnya tentang estimasi distribusi *Univariate Gaussian*? Ingat kembali konsep tersebut untuk mengerti bab ini. Diberikan $(x, y)$ sebagai *random variable* berdimensi $R^M$ dan $R^N$ (keduanya berada pada Euclidean Space) yang merupakan parameter[1] bagi *probability density function* $q(x, y)$. Terdapat sebuah fungsi $f(x) \rightarrow y$, yang memetakan $x$ ke $y$. Aproksimasi $f(x)$, sebut saja sebagai $g(x)$ adalah fungsi hasil regresi. Fungsi regresi g: $R^M \rightarrow R^N$ didefinisikan secara konseptual sebagai persamaan 5.1 [9].

$$g(x) = \int y \; q(y|x) dy \tag{5.1}$$

[1] *Which is subject to a simultaneous probability density function*

Persamaan 5.1 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, regresi adalah pendekatan sehingga belum tentu 100% benar (hal ini juga berlaku pada model *machine learning* pada umumnya).

Sebagai ilustrasi *curve fitting problem*, kamu diberikan fungsi $f(x)$ 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 tanda bulat warna biru. Tugasmu adalah untuk mencari tahu $f(x)$!



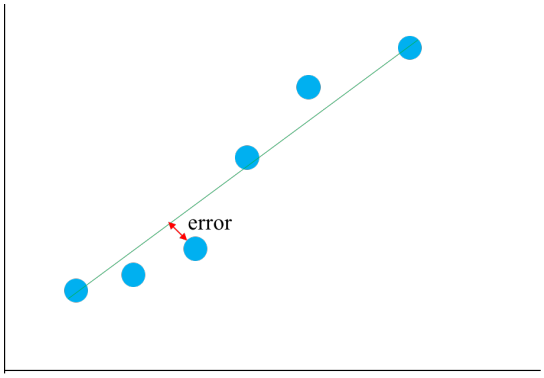
**Gambar 5.1.** Contoh fungsi Sigmoid.

Anggap dengan metode regresi, kamu berhasil melakukan pendekatan dan menghasilkan fungsi polinomial $g(x) = x \cdot \mathbf{w}$; seperti Gambar. 5.2 (garis berwarna hijau). Akan tetapi, fungsi approksimasi ini tidak 100% tepat sesuai dengan fungsi aslinya (ini perlu ditekankan). Jarak antara titik biru terhadap garis hijau disebut *error*.

Salah satu cara menghitung *error* fungsi $g(x)$ adalah menggunakan ***squared error function*** dengan bentuk konseptual pada persamaan 5.2. Estimasi terhadap persamaan tersebut disajikan dalam bentuk diskrit pada persamaan 5.3. $(x_i, y_i)$ 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].

$$E(g) = \int \int \|y - g(x)\|^2 \, q(x, y) \, dx \, dy \tag{5.2}$$

$$E(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} \|y_i - g(x_i, \mathbf{w})\|^2 \tag{5.3}$$




**Gambar 5.2.** Pendekatan fungsi Sigmoid.

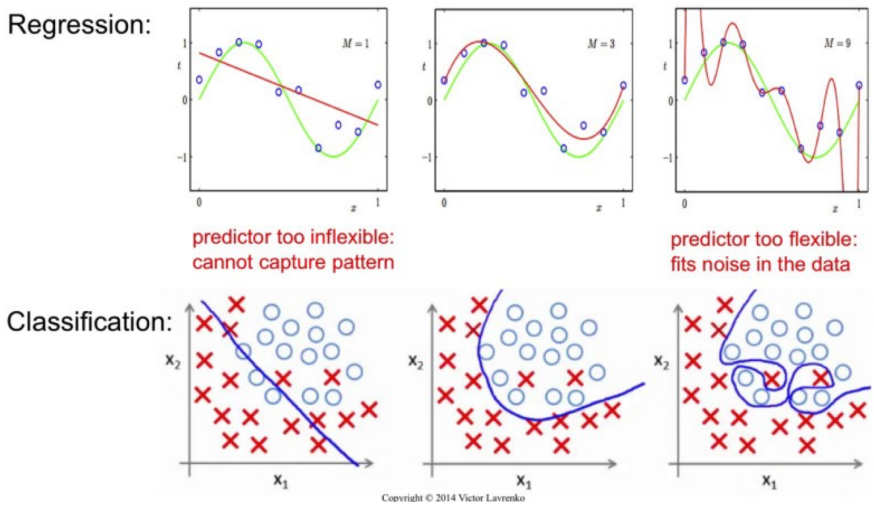
Ingat kembali bab 1, *learning machine* yang direpresentasikan dengan fungsi $g$ bisa diatur kinerjanya dengan parameter *training* $\mathbf{w}$. *Square error* untuk *learning machine* dengan parameter *training* $\mathbf{w}$ diberikan oleh persamaan 5.3. $(x_i, y_i)$ adalah pasangan *input-desired output*. Selain untuk menghitung ***square error*** pada *training data*, persamaan 5.3 juga dapat digunakan untuk menghitung *square 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 Overfitting dan Underfitting

Seperti yang sudah dijelaskan pada bab-bab sebelumnya (dan disinggung kembali pada subbab sebelumnya), 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[2]. 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.

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

*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. 5.3. *Underfitting* terjadi akibat model yang terlalu tidak fleksibel (memiliki kemampuan yang rendah untuk mengestimasi variasi fungsi), sedangkan *overfitting* terjadi ketika model terlalu fleksibel (memiliki kemampuan yang terlalu tinggi untuk mengestimasi banyak fungsi) atau terlalu mencocokkan diri terhadap *training* data. Perhatikan kembali Gambar. 5.3, 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).

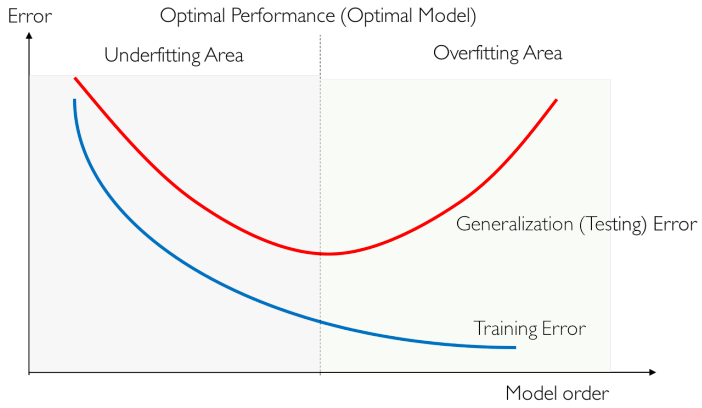



**Gambar 5.3.** *Underfitting* vs. *Overfitting*[3].

Apabila kita gambarkan grafik kinerja terhadap konfigurasi model (*model order*), fenomena *underfitting* dan *overfitting* dapat diilustrasikan seperti Gambar. 5.4. Model yang ideal adalah model yang memiliki kinerja yang 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 5.8). 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

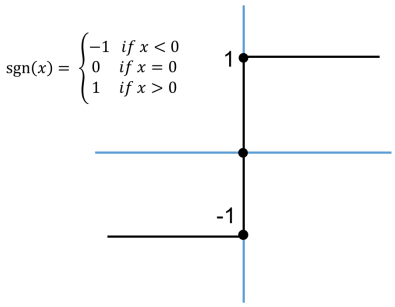
[3] `https://www.inf.ed.ac.uk/teaching/courses/iaml/slides/eval-2x2.pdf`

memastikan kesamaan karakteristik data (e.g., apakah *training* dan *testing* data benar diambil dari distribusi yang sama).




**Gambar 5.4.** *Selection Error.*

## 5.3 Binary Classification




**Gambar 5.5.** Fungsi *sign*.

Kamu sudah mendapat penjelasan pada bab-bab sebelumnya. *Binary classification* adalah mengklasifikasikan data menjadi dua kelas (*binary*). Contoh model linear sederhana untuk *binary classification* diberikan pada persamaan 5.4. Perhatikan, pada persamaan 5.4, suatu data direpresentasikan

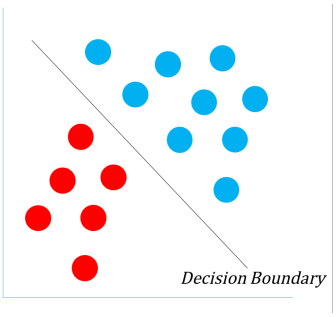
sebagai *feature vector* $\mathbf{x}$, dan terdapat *bias*[4] $b$. Klasifikasi dilakukan dengan melewatkan data pada fungsi yang memiliki parameter. Fungsi tersebut menghitung bobot setiap fitur pada vektor dengan mengalikannya dengan parameter (*dot product*). Persamaan 5.4 dapat ditulis kembali sebagai persamaan 5.5, dimana $x_i$ merupakan elemen ke-$i$ dari vektor $\mathbf{x}$. Fungsi ini memiliki *range* $[-\infty, \infty]$. Pada umumnya, kita menggunakan fungsi *sign* (sgn, Gambar. 5.5) untuk merubah nilai fungsi menjadi $[-1, 1]$ sebagai *output* (persamaan 5.6); dimana $-1$ merepresentasikan *input* dikategorikan ke kelas pertama dan nilai 1 merepresentasikan *input* dikategorikan ke kelas kedua.

$$f(\mathbf{x}) = \mathbf{x} \cdot \mathbf{w} + b \tag{5.4}$$

$$f(\mathbf{x}) = x_1 w_1 + x_1 w_2 + \cdots + x_n w_n + b \tag{5.5}$$

$$\text{output} = \text{sgn}(f(\mathbf{x}) \tag{5.6}$$

Seperti halnya fungsi regresi, kita juga dapat menghitung performa *binary classifier* sederhana ini menggunakan *squared error function*, 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. 5.6 adalah *decision boundary*.




**Gambar 5.6.** Contoh *decision boundary*.

## 5.4 Log-linear Binary Classification

Pada subbab sebelumnya, telah dijelaskan fungsi binary classifier memetakan data menjadi nilai $[-1, 1]$, dengan $-1$ merepresentasikan kelas pertama dan

[4] Berbeda dengan contoh pada subbab sebelumnya karena data adalah skalar. Penulis berharap pembaca dapat menginterpretasikan simbol berdasarkan konteks.

1 merepresentasikan kelas kedua. Tidak hanya kelas yang berkorespondensi, kita juga terkadang ingin tahu seberapa besar peluang data tergolong pada kelas tersebut. Salah satu alternatif adalah dengan menggunakan fungsi sigmoid dibanding fungsi *sign* untuk merubah nilai fungsi menjadi $[0, 1]$ yang merepresentasikan peluang data diklasifikasikan sebagai kelas tertentu (1 - nilai peluang, untuk kelas lainnya). Konsep ini dituangkan menjadi persamaan 5.7, dimana $y$ merepresentasikan probabilitas, $\mathbf{x}$ merepresentasikan data (*feature vector*), dan $b$ merepresentasikan *bias*. Ingat kembali materi bab 4, algoritma *Naive Bayes* melakukan hal serupa. Hasil fungsi sigmoid, apabila di-*plot* maka akan berbentuk seperti Gambar. 5.1 (berbentuk karakter "S").

$$y = \sigma(f(\mathbf{x})) = \frac{1}{1 + e^{-(\mathbf{x} \cdot \mathbf{w} + b)}} \tag{5.7}$$

## 5.5 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 5.8, dimana $\hat{c}$ adalah kelas terpilih (keputusan), yaitu kelas yang memiliki nilai tertinggi.

$$\hat{c} = f(\mathbf{x}) = \arg\max_{c_i \in C} (\mathbf{x} \cdot \mathbf{w}_{c_i} + b_{c_i}) \tag{5.8}$$

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 5.8 dapat ditulis kembali sebagai persamaan 5.9. $\hat{\mathbf{c}}$ adalah vektor yang memuat nilai fungsi terhadap seluruh kelas. Kita memprediksi kelas berdasarkan indeks elemen $\hat{\mathbf{c}}$ yang memiliki nilai terbesar (Persamaan 5.10).

$$\hat{\mathbf{c}} = f(\mathbf{x}) = \mathbf{x} \cdot \mathbf{W} + \mathbf{b} \tag{5.9}$$

$$\text{kelas prediksi} = \arg\max_{\hat{c}_i \in \hat{\mathbf{c}}} \hat{c}_i \tag{5.10}$$

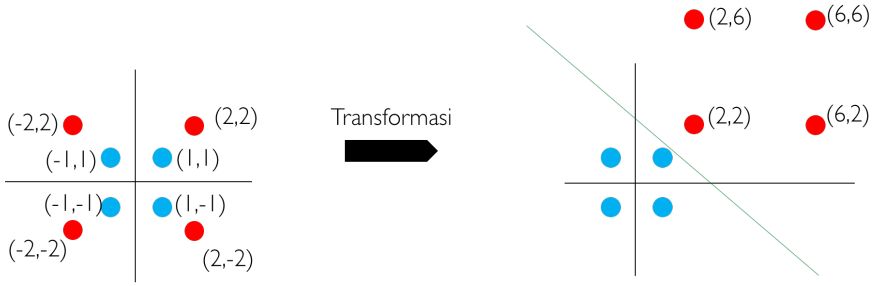
Seperti yang diceritakan pada subbab berikutnya, kita mungkin juga tertarik dengan probabilitas masing-masing kelas. Kita dapat menggunakan

fungsi *softmax*[5] untuk hal ini. Fungsi *softmax* mentransformasi $\hat{\mathbf{c}}$ agar jumlah semua nilainya berada pada range $[0, 1]$. Dengan itu, $\hat{\mathbf{c}}$ dapat diinterpretasikan sebagai distribusi probabilitas. Konsep ini dituangkan pada persamaan 5.11, dimana $\hat{c}_i$ adalah elemen vektor ke-$i$, melambangkan probabilitas masuk ke kelas ke-$i$.

$$\hat{c}_i = \frac{e^{(\mathbf{x}\cdot\mathbf{W}+\mathbf{b})_{[i]}}}{\sum_j e^{(\mathbf{x}\cdot\mathbf{W}+\mathbf{b})_{[j]}}} \tag{5.11}$$

## 5.6 Transformasi

Seperti yang sudah dijelaskan pada bab-bab sebelumnya. Alangkah baik apabila semua data bersifat *linearly separable*. Kenyataannya, kebanyakan data bersifat *non-linearly separable*. Dengan demikian, kita membutuhkan suatu fungsi untuk mentransformasi data, sebelum menggunakan model linear untuk mengklasifikasikan data. Sebagai contoh, perhatikan Gambar. 5.7. 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 5.12.




**Gambar 5.7.** Contoh transformasi [7].

$$\phi(x, y) = \begin{cases} \sqrt{x^2 + y^2} \geq 2 \rightarrow (4 - x + \|x - y\|, 4 - x + \|x - y\|) \\ \sqrt{x^2 + y^2} \leq 2 \rightarrow (x, y) \end{cases} \tag{5.12}$$

Secara umum, fungsi transformasi tidaklah sesederhana contoh yang diberikan. Fungsi transformasi pada umumnya menambah dimensi data (misal dari dua dimensi, menjadi tiga dimensi). Beberapa fungsi transformasi (dikenal juga dengan istilah *kernel*) yang terkenal diantaranya[6]:

[5] `https://en.wikipedia.org/wiki/Softmax_function`
[6] `https://en.wikipedia.org/wiki/Kernel_method`

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) [25]. Perhatikan, algorithma SVM sebenarnya sangatlah penting. Akan tetapi, perlu kami informasikan bahwa buku ini tidak memuat materi SVM karena penulis belum mengerti sampai seluk beluk SVM[7]. Dengan demikian, kami harap pembaca dapat mencari referensi lain tentang SVM.

## 5.7 Pembelajaran sebagai Permasalahan Optimisasi

Salah satu tujuan dari pembelajaran (*training*) adalah untuk meminimalkan *error* sehingga kinerja *learning machine* (model) diukur oleh *square error*. Dengan kata lain, *utility function* adalah meminimalkan *square error*. Secara lebih umum, kita ingin meminimalkan *loss* (*squared error function* adalah salah satu *loss function*), diilustrasikan pada persamaan 5.13, dimana $\theta$ adalah *learning parameter*[8], dan $\mathcal{L}$ adalah *loss function*. Perubahan parameter dapat menyebabkan perubahan *loss*. Karena itu, *loss function* memiliki $\theta$ sebagai parameternya.

$$\hat{\theta} = \arg\min_{\theta} \mathcal{L}(\theta) \tag{5.13}$$

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

Sekarang, mari kita hubungkan dengan contoh yang sudah diberikan pada subbab sebelumnya. Kita coba melakukan estimasi *minimum square error*, dengan mencari nilai *learning parameters* $\mathbf{w}$ yang meminimalkan nilai *error* pada model linear (persamaan 5.14)[9]. Terdapat beberapa cara untuk memi-

[7] Penulis memutuskan tidak membahas SVM, dibanding pembahasan pada level *superficial*.

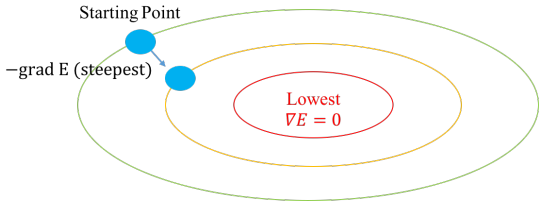
[8] Umumnya dapat berupa skalar, vektor, atau matriks. Demi istilah yang lebih generik, kita gunakan $\theta$.

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

malkan *square error*. Yang penulis akan bahas adalah *gradient-based method*. ***Stochastic gradient descent*** akan dibahas untuk meminimalkan *error*[10].

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} E(\mathbf{w}) \tag{5.14}$$

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 sejauh mata memandang, kemudian menuju titik tersebut dan menganggapnya sebagai titik terendah. 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. 5.8!




**Gambar 5.8.** *Stochastic Gradient Descent.*

Jalanan dengan kemiringan paling tajam adalah $-grad\ E(\mathbf{w})$, dimana $E(\mathbf{w})$ adalah nilai *error* saat model memiliki parameter $\mathbf{w}$. Dengan definisi $grad\ E(\mathbf{w})$ diberikan pada persamaan 5.15 dan persamaan 5.16, dimana $w_i$ adalah nilai elemen vektor ke-$i$.

$$grad\ E(\mathbf{w}) = \left( \frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, ..., \frac{\partial E}{\partial w_3} \right) \tag{5.15}$$

$$\frac{\mathrm{d}\mathbf{w}}{\mathrm{d}t} = -grad\ E(\mathbf{w}); t = time \tag{5.16}$$

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.

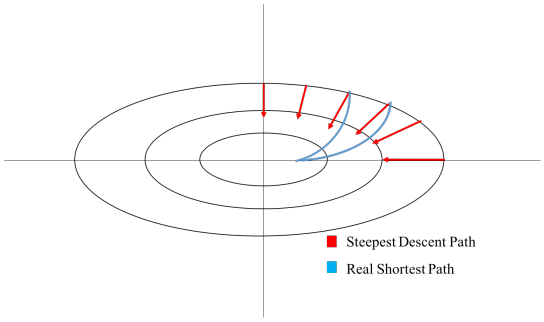
Tentunya seiring berjalannya waktu, kita mengubah-ubah parameter $\mathbf{w}$ agar kinerja model optimal. Nilai optimal diberikan oleh turunan $\mathbf{w}$ terhadap waktu, yang bernilai sama dengan $-grad\ E(\mathbf{w})$. Bentuk diskrit persamaan 5.16 diberikan pada persamaan 5.17

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

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \eta \; grad \; E(\mathbf{w}(t)) \tag{5.17}$$

dimana $\eta$ 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, $\eta$ juga sering diubah-ubah nilainya sepanjang waktu. Semakin kita sudah dekat dengan tujuan (titik *loss* terendah), kita mengurangi nilai $\eta$ (ibaratnya seperti mengerem) [26].

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. 5.9. Warna merah melambangkan jalan yang dilalui *gradient descent*, sementara warna biru melambangkan jalanan terbaik (tercepat).

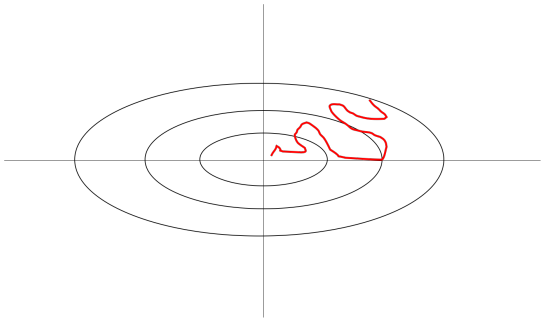



**Gambar 5.9.** *Stochastic Gradient Descent* 2.

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 macet/berhenti saat sudah mencapai *local minima*, yaitu nilai minimum pada suatu daerah lokal saja. Untuk menghindari hal tersebut, kita menggunakan *learning rate* ($\eta$). Apabila nilai *learning rate* ($\eta$) pada persamaan 5.17 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. 5.10.

Untuk mengontrol *learning parameter* $\mathbf{w}$ sehingga memberikan nilai $E(\mathbf{w})$ terendah, persamaan *steepest gradient descent* dapat ditambahkan dengan momentum ($\alpha$) pada persamaan 5.18. Alfa adalah momentum karena dika-



**Gambar 5.10.** *Swing.*

likan dengan hasil perbedaan descent pada tahap sebelumnya. Alfa adalah parameter kontrol tambahan untuk mengendalikan *swing* yang sudah dibahas sebelumnya.

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \eta \; grad \; E(\mathbf{w}(t)) + \alpha(\Delta \mathbf{w}) \tag{5.18}$$

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

## 5.8 Regularization

*Gradient-based method* mengubah-ubah parameter model **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 **w** berubah-ubah seiring waktu, bisa jadi urutan *training data* menjadi penting[11]. Pada umumnya, kita menggunakan *batch method*. Hal ini akan kamu lebih mengerti setelah mambaca bab 9.

Pada kenyataan, model yang kita hasilkan bisa jadi *overfitting*. 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 5.13 dapat kita ubah menjadi persamaan 5.19, dimana $\mathrm{R}(\theta)$ adalah fungsi *regularization* dan $\lambda$ adalah parameter kontrol.

$$\theta = \arg\min_{\theta} \mathcal{L}(\theta) + \lambda \mathrm{R}(\theta) \tag{5.19}$$

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

Pilihan umum untuk fungsi *regularization* pada umumnya adalah $L_2$ dan $L_1$ norm. $L_2$ *regularization* menggunakan jumlah kuadrat dari *Euclidean norm*[12] seluruh parameter, seperti pada persamaan 5.20. Sedangkan, $L_1$ *regularization* menggunakan jumlah dari nilai *absolute-value norm* seluruh parameter, diberikan pada persamaan 5.21.

$$\mathrm{R}(\theta) = \|\mathbf{w}\|_2^2 = \sum_i (w_i)^2 \tag{5.20}$$

$$\mathrm{R}(\theta) = \|\mathbf{w}\| = \sum_i (w_i) \tag{5.21}$$

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.

## 5.9 Bacaan Lanjutan

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

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

[12] `https://en.wikipedia.org/wiki/Norm_(mathematics)`

## 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!

# 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 [27, 28] yang terinspirasi oleh teori informasi [29]. Algoritma ini sudah cukup tua, tetapi layak dimengerti. ID3 adalah salah satu varian dari *supervised learning*.

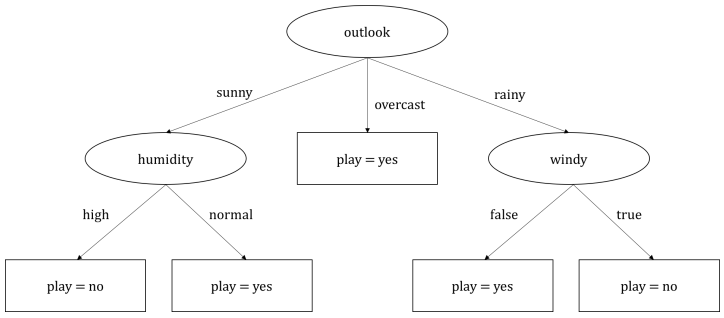
## 6.1 Inductive Learning

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.

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* [28]. *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* [28]. 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*, kami akan memperkenalkan **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.

$$entropy(a, b) = -a\ log\ a - b\ log\ b \tag{6.1}$$

Kita juga definisikan **Info** sebagai persamaan 6.2.

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

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.

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

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)

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

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.

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

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

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

  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 *informaiton gain* untuk atribut *outlook* sebagai

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

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

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

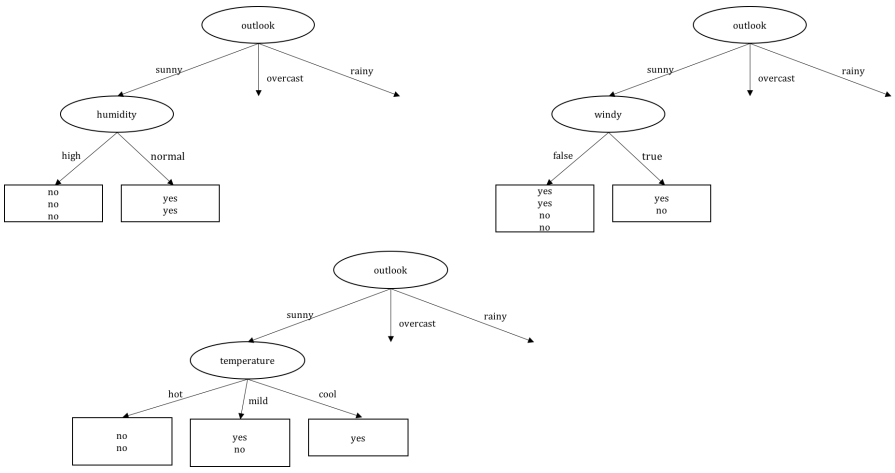
Dengan demikian, kita memilih atribut *outlook* sebagai *root*.

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*.

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

Proses ini dilanjutkan sampai kita tidak bisa mencabang lagi.




**Gambar 6.2.** Percabangan.

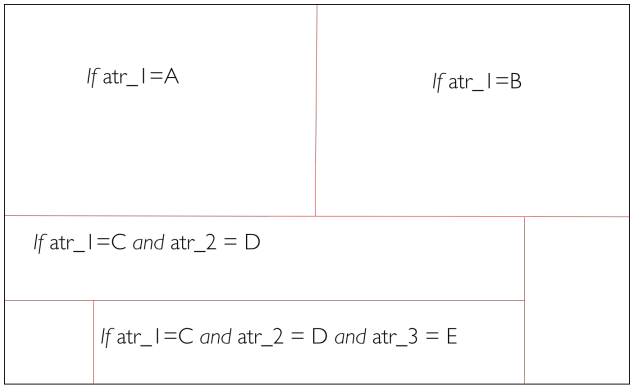
## 6.3 Isu pada ID3

Pada algoritma *decision tree* secara umum, terdapat beberapa isu diantara lain [7, 30]:

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*
5. Data dengan *unseen value*

## 6.4 Hubungan Decision Tree dan Model Linear

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 linier. 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.** 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!

# 7

# 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 [31]. 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, 32, 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.

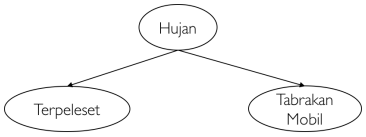
## 7.1 Probabilistic Reasoning

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 mungkin perlu dijelaskan juga cara transformasi sinyal suara menjadi data diskrit.

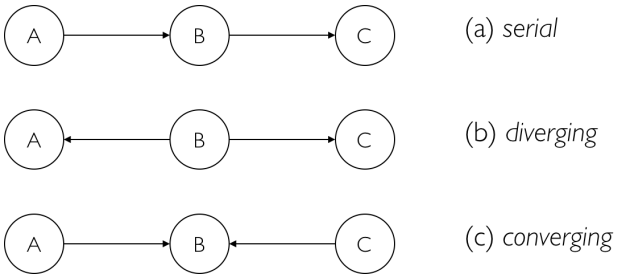
jadi, maka "terpeleset" juga pasti akan terjadi. Tetapi tidak sebaliknya, apabila kita "terpeleset", belum tentu "hujan" juga terjadi. Pada *probabilistic reasoning* kita mengkuantifikasi kepastian atau ketidakpastian itu. Apabila "hujan" terjadi, berapa besar kemungkinan "terpeleset" juga terjadi (dan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

Perhatikan Gambar. 7.1! 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 7.1.** Contoh *Bayesian Network*.

Terdapat beberapa kerangka berpikir yang berbeda-beda, tergantung hubungan kausalitas antara *events*: *serial*, *diverging*, dan *converging* (diilustrasikan pada Gambar. 7.2.




**Gambar 7.2.** Tipe Jaringan Kausalitas.

Berdasarkan hubungan/jaringan kausalitas antara *events*, ada beberapa kerangka pikir 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 dan hanya jika kondisi ($B$ diketahui) terpenuhi.

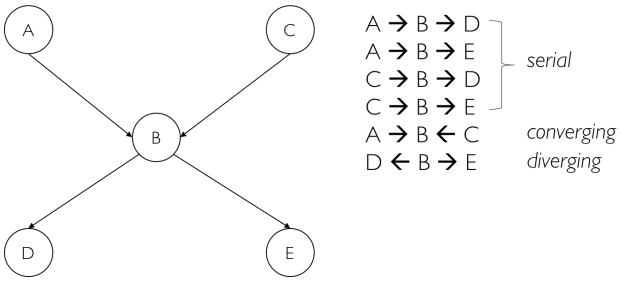
(b) *Diverging*. Bila kita mengetahui $A$ maka kita bisa mengetahui sesuatu tentang $C$, dan sebaliknya. Tetapi apabila $B$ diketahui, maka hubungan antara $A$ dan $C$ menjadi terputus. Dengan bahasa lebih matematis, $A$ dan $C$ independen kondisional ketika $B$ diketahui.

(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 7.3.** Contoh Inferensi.

Perhatikan Gambar. 7.3! Dengan konsep yang sudah dipahami, mari kita coba lakukan inferensi pada jaringan tersebut. *Joint distribution* untuk seluruh *event* diberikan pada persamaan 7.1.

$$P(A, B, C, D, E) = P(D \mid B)\ P(E \mid B)\ P(B \mid A, C)\ P(A)\ P(C) \qquad (7.1)$$

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 7.2.

$$P(E \mid A) = P(E \mid B)\ P(B \mid A)\ P(A) + P(E \mid \neg B)\ P(\neg B \mid A)\ P(A) \quad (7.2)$$

Tetapi, apabila $A$ dan $B$ diketahui, maka inferensi terhadap $E$ dilakukan seperti pada persamaan 7.3.

$$P(E \mid B, A) = P(E \mid B)\ P(B) \quad (7.3)$$

Operasi serupa dapat diaplikasikan pada koneksi tipe *converging* dan *diverging*. Perhatikan subgraf $\{A, B, C\}$ dengan tipe koneksi *converging* pada Gambar. 7.3. Apabila $B$ tidak diketahui, berarti $A$ dan $C$ terpisah (independen). Apabila $B$ dan $A$ diketahui, maka hubungan $A$ dan $C$ dapat dihitung sebagai persamaan 7.4.

$$P(C \mid B, A) = P(C \mid B)\ P(B \mid A)P(A) \quad (7.4)$$

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

## 7.2 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. 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. 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*. Diberikan *input* $\mathbf{x}$ berupa sekuens (sekuens sinyal, sekuens kata, dsb). Kita ingin memodelkan sekuens *output* terbaik $\mathbf{y}$ untuk input tersebut. *Output* ke-$i$ bergantung pada *input* dari awal sampai ke-$i$ dan *output* dari awal sampai sebelumnya $p(y_i \mid y_1, \ldots, y_{i-1}, x_1, \ldots, x_i)$. Berdasarkan pemaparan subbab 7.1, apabila suatu *event* dikondisikan variabel lain dengan tipe koneksi *serial*, maka kita dapat mengabaikan banyak variabel historis. Hal ini dituangkan dalam persamaan 7.5,

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

$$p(y_i \mid y_1, \ldots, y_{i-1}, x_1, \ldots, x_i) = p(y_i \mid y_{i-1}, x_i) \tag{7.5}$$

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.

## 7.3 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* [4] seperti pada Tabel. 7.1.

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 $w_1, w_2, \ldots, w_N$, kita ingin mencari sekuens kelas kata $c_1, c_2, \ldots, c_N$ sedemikian sehingga kita memaksimalkan nilai probabilitas 7.6 [12, 32].

$$\hat{c_1}, \hat{c_2}, \ldots, \hat{c_N} = \underset{c_1, c_2, \ldots, c_N; c_i \epsilon C}{\arg\max} P(c_1, c_2, \ldots, c_N | w_1, w_2, \ldots, w_N) \tag{7.6}$$

Dimana $C$ adalah daftar kelas kata. Akan tetapi, menghitung persamaan 7.6 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 7.7.

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)} \tag{7.7}$$

Dengan menggunakan teori Bayes, kita dapat mentransformasi persamaan 7.6 menjadi persamaan 7.8.

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

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

**Tabel 7.1.** Penn Treebank POS Tag.

$$\hat{c_1}, \hat{c_2}, \ldots, \hat{c_N} = \underset{c_1, c_2, \ldots, c_N; c_i \epsilon C}{\arg\max} \frac{P(w_1, w_2, \ldots, w_N | c_1, c_2, \ldots, c_N) P(c_1, c_2, \ldots, c_N)}{P(w_1, w_2, \ldots, w_N)} \tag{7.8}$$

Untuk suatu sekuens input, $P(w_1, w_2, \ldots, w_N)$ akan selalu sama sehingga

dapat diabaikan (karena operasi yang dilakukan adalah perbandingan yaitu berbentuk arg max). Oleh karena itu, persamaan 7.8 dapat disederhanakan menjadi 7.9.

$$\hat{c_1}, \hat{c_2}, \ldots, \hat{c_N} = \underset{c_1, c_2, \ldots, c_N ; c_i \epsilon C}{\arg\max} P(w_1, w_2, \ldots, w_N | c_1, c_2, \ldots, c_N) P(c_1, c_2, \ldots, c_N) \tag{7.9}$$

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

Persamaan 7.9 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 7.10. Asumsi kedua adalah suatu kemunculan kategori kata hanya bergantung pada kelas kata sebelumnya, seperti pada persamaan 7.11.

$$P(w_1, w_2, \ldots, w_N | c_1, c_2, \ldots, c_N) = \prod_{i=1}^{N} P(w_i | c_i) \tag{7.10}$$

$$P(c_1, c_2, \ldots, c_N) = \prod_{i=1}^{N} P(c_i | c_{i-1}) \tag{7.11}$$

Dengan demikian, persamaan 7.9 disederhanakan kembali menjadi persamaan 7.12 yang disebut ***bigram assumption*** atau ***first-order markov chain***.

$$\hat{c_1}, \hat{c_2}, \ldots, \hat{c_N} = \underset{c_1, c_2, \ldots, c_N ; c_i \epsilon C}{\arg\max} \prod_{i=1}^{N} P(w_i | c_i) P(c_i | c_{i-1}) \tag{7.12}$$

Kita dapat membuat ekstensi persamaan 7.12 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 11.2). 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.

$$\begin{aligned} P(noun, verb, noun) = {} & P(budi \mid noun) P(noun \mid null) P(menendang \mid verb) \\ & P(verb \mid noun) P(bola \mid noun) P(noun \mid verb) \end{aligned} \tag{7.13}$$

| 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 7.2.** Probabilitas Bigram [32].

## 7.4 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*.

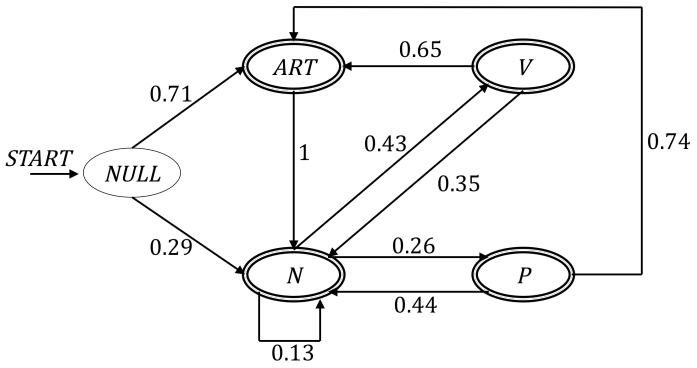
Ingat kembali persamaan 7.12 untuk POS tagging. $P(w_i|c_i)$ disebut *likelihood* dan $P(c_i|c_{i-1})$ disebut *prior*, *multiplication of probabilities* ($\prod$) melambangkan *markov chain*. ***Markov chain*** adalah kasus spesial *weighted automaton*[5] yang mana sekuens *input* menentukan *states* yang akan dilewati oleh automaton. Sederhananya, automaton mencapai *goal state* setelah mengunjungi berbagai *states*. Total bobot *outgoing edges* untuk masing-masing *state* pada automaton haruslah 1 apabila dijumlahkan. Kasus spesial yang dimaksud adalah *emission* (dijelaskan kemudian).

Sebagai contoh, perhatikan Tabel. 7.2. *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 ini, ketika dikonversi menjadi *weighted automaton* akan menjadi Gambar. 7.4.

Tabel. 7.2 dan Gambar. 7.4 telah merepresentasikan probabilitas *prior*, sekarang kita ingin model yang kita punya juga mencakup ***lexical emission probabilities***, yaitu *likelihood* pada persamaan 7.12.

Seumpama kita mempunyai *lexical emission probabilities* seperti pada Tabel. 7.3. Setiap *state* pada automaton, dapat menghasilkan/meng-*output*-kan suatu kata (*word*) dengan probabilitas pada Tabel. 7.3. Kita kembangkan lagi Gambar. 7.4 dengan tambahan informasi *lexical emission probabilities* menjadi Gambar. 7.5. 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*) [32].

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




**Gambar 7.4.** *Weighted Automaton* [32].

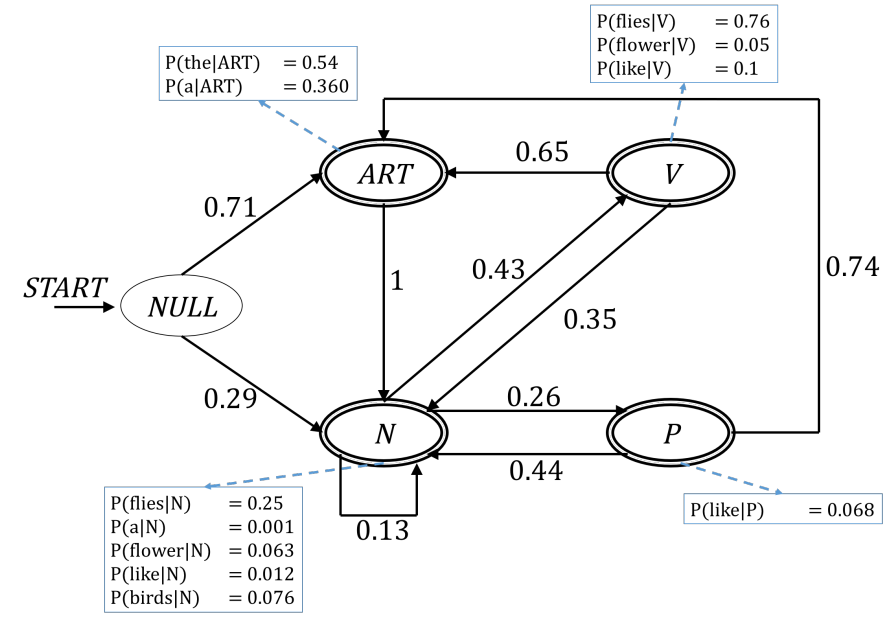
| | | | |
|---|---|---|---|
| $P(the\|ART)$ | 0.54 | $P(a\|ART)$ | 0.360 |
| $P(flies\|N)$ | 0.025 | $P(a\|N)$ | 0.001 |
| $P(flies\|V)$ | 0.076 | $P(flower\|N)$ | 0.063 |
| $P(like\|V)$ | 0.1 | $P(flower\|V)$ | 0.05 |
| $P(like\|P)$ | 0.068 | $P(birds\|N)$ | 0.076 |
| $P(like\|N)$ | 0.012 | | |

**Tabel 7.3.** *Lexical Emission Probabilities* [32].

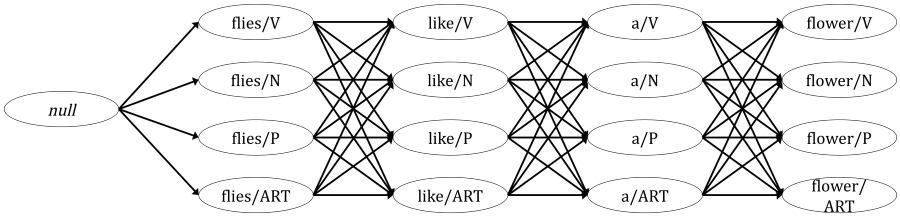
Diberikan kalimat "*flies like a flower*", untuk menghitung sekuens kelas kata untuk kalimat tersebut, kita menyelusuri automaton Gambar. 7.5. Hasil penelusuran memberikan kita kombinasi sekuens yang mungkin seperti pada Gambar. 7.6. Pekerjaan berikutnya adalah, dari seluruh kombinasi sekuens yang mungkin ($\prod_{i=1}^{N} P(w_i|c_i)P(c_i|c_{i-1})$), 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. $Q = \{q_1, q_2, \ldots, q_N\}$ yaitu himpunan ***states***; $N$ menunjukkan banyaknya *states*.
2. $\mathbf{A} = \alpha_{0,0}, \alpha_{1,1}, \alpha_{2,2}, \ldots, \alpha_{N,N}$ yaitu **transition probability matrix** dari suatu *state* $i$ menuju *state* $j$; dimana $\sum_{j=0}^{N} \alpha_{i,j} = 1$. Indeks 0 merepresentasikan *start state* (*null state*).
3. $\mathbf{o} = o_1, o_2, \ldots, o_T$ yaitu sekuens observasi kata; $T$ 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 7.5.** *Complete Hidden Markov Model.*




**Gambar 7.6.** Sekuens yang mungkin (*brute force*).

## 7.5 Algoritma Viterbi

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

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

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

**Initialization Step**
create a path of probability matrix viterbi[$N$,$T$]
**for** each state $s$ **from** 1 **to** $N$ **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** $N$ **do**
        viterbi[$s$,$t$] $\leftarrow \arg\max_{j=1,N}$ *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

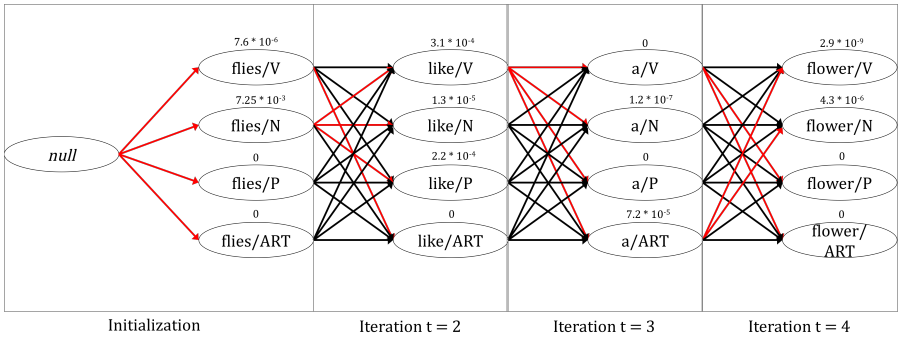
**Sequence Identification Step**
$c_T \leftarrow i$ that maximizes viterbi[$i$, $T$]
**for** $i = T - 1$ **to** 1 **do**
    $c_i \leftarrow$ backpointer[$c_{i+1}$,$i+1$]

**Gambar 7.7.** Algoritma Viterbi [12, 32].

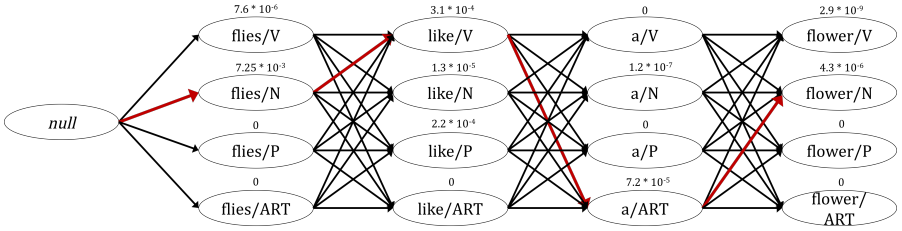
Perhatikan Gambar. 7.7 yang menunjukkan *pseudo-code* untuk algoritma Viterbi. Variabel $c$ berarti kelas kata, $a$ adalah *transition probability*, dan $b$ adalah *lexical-generation probability*. Pertama-tama, algoritma tersebut membuat suatu matriks berukuran $N$x$T$ dengan $N$ adalah banyaknya *states* (tidak termasuk *start state*) dan $T$ adalah panjang sekuens. Pada setiap iterasi, kita pindah ke observasi kata lainnya. Gambar. 7.8 adalah ilustrasi algoritma Viterbi untuk kalimat *input* (*observed sequence*) "*flies like a flower*" dengan *lexical generation probability* pada Tabel. 7.3 dan *transition probabilities* pada Tabel. 7.2 (bigram) [32]. 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. 7.9. Dengan itu, kita mendapatkan sekuens "flies/N like/V a/ART flower/N".

Apabila kamu hanya ingin mengetahui HMM non-parametrik (tanpa variabel yang perlu dilatih/diestimasi), kamu dapat berhenti membaca sampai subbab ini. Apabila kamu ingin mengetahui bagaimana HMM dapat dianggap sebagai model parametrik, kamu dapat melanjutkan membaca subbab berikutnya[7].

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




**Gambar 7.8.** Ilustrasi Algoritma Viterbi per Iterasi.




**Gambar 7.9.** Viterbi *Backtrace*.

## 7.6 Proses Training Hidden Markov Model

Hidden Markov Model (HMM) adalah salah satu varian *supervised learning*[8], 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 7.4). 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 7.14 (bentuk *softmax function*).

$$P(w_i|c_i) = \frac{count(w_i, c_i)}{\sum_{j=1}^{N} count(w_i\, c_j)} \tag{7.14}$$

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

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 7.15.

$$\alpha_i(t) = P(w_t/c_i \mid w_1, w_2, \ldots, w_T) \tag{7.15}$$

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. 7.10, dengan $c_i$ adalah kelas kata ke-$i$ dan $w_i$ adalah kata ke-$i$, $a$ adalah *transition probability*, $b$ adalah *emission probability*, dan $N$ melambangkan banyaknya *states*.

---

**Initialization Step**
**for** $i = 1$ **to** $N$ **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** $N$ **do**
    $\alpha_i(t) \leftarrow \sum_{j=1}^{N}(a_{ji} \times \alpha_j(t-1)) \times b_i(o_t)$

---

**Gambar 7.10.** Algoritma *forward* [32].

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 dapat melihat *pseudo-code* pada Gambar. 7.11, dengan $a$ adalah *transition probability* dan $b$ adalah *emission probability*.

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 7.16.

$$\gamma_j(t) = \frac{\alpha_i(t) \times \beta_i(t)}{\sum_{j=1}^{N} \alpha_j(t) \times \beta_j(t)} \tag{7.16}$$

**Initialization Step**
**for** $i = 1$ **to** $N$ **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** $N$ **do**
    $\beta_i(t) \leftarrow \sum_{j=1}^{N}(a_{ji} \times \beta_i(t+1)) \times b_j(o_{j+1})$

**Gambar 7.11.** Algoritma *backward* [32].

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 7.17.

$$\xi_t(i,j) = \frac{\alpha_i(t) \times a_{ij} \times b_j(o_{t+1}) \times \times \beta_j(t+1))}{\alpha_N(T)} \tag{7.17}$$

Dengan $a_{ij}$ adalah *transition probability* dan $b_j(o_{t+1})$ adalah *emission probability* (ingat kembali definisi formal HMM pada subbab 7.4). Pada setiap iterasi, kita ingin memperbaharui kembali parameter HMM yaitu **A** dan **b**. Kita hitung kembali *transition probability* (nilai yang lama di-*update*), diberikan oleh persamaan 7.18.

$$a_{ij}{}' = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \sum_{j=1}^{N} \xi_t(i,j)} \tag{7.18}$$

Kita juga menghitung kembali *emission probability* (nilai yang lama di-*update*), diberikan oleh persamaan 7.19.

$$b_j(o_k)' = \frac{\sum_{t=1,o_k=w_k}^{T} \gamma_j(t)}{\sum_{t=1}^{T} \gamma_j(t)} \tag{7.19}$$

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

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 $\gamma$ dan $\xi$ untuk mengestimasi kembali parameter **A** dan **b**. Hal ini dijelaskan secara formal pada Gambar. 7.12.

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* [9] terhadap nilai parameter (**A** dan **b**). Kita mencapai titik *local optimal* apabila tidak ada perubahan parameter.

---

**Initialize A and b**
**Iterate** until convergence
**E-step**

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

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

**M-step**

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

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

---

**Gambar 7.12.** Algoritma *Forward-Backward* (EM) [12].

Kami ingin mengakui bahwa penjelasan pada subbab 7.6 mungkin kurang baik (kurang memuaskan). Kami harap kamu mencari referensi lain yang lebih baik untuk subbab ini.

## Soal Latihan

**7.1. 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*.

**7.2. Ekstensi Algoritma Viterbi**
Buatlah ekstensi algoritma Viterbi untuk asumsi trigram!

**7.3. Maximum Entropy Markov Model**

(a) Jelaskan konsep *maximum entropy*!

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

(b) Jelaskan *maximum entropy markov model*!

### 7.4. Gibbs Sampling

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

**7.5. 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!

# 8

# 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 [33], walaupun relatif lama, tetapi paper tersebut memberikan penjelasan yang mudah dimengerti tentang *clustering*. Selain itu, kamu juga dapat membaca *paper* oleh Saad et al. [34].

***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 [35]: (1) *partitional*, yaitu menentukan partisi sejumlah $K$ dan (2) *hierarchical*, yaitu mengelompokan data berdasarkan struktur taksonomi. Contoh algoritma *partitional* adalah **K-means** pada subbab 8.1, sementara contoh algoritma *hierarchical* adalah **agglomerative clustering** pada subbab 8.2.

## 8.1 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 $\mathbf{D} = \{\mathbf{d}_1, \mathbf{d}_2, \cdots, \mathbf{d}_N\}$ pada suatu *cluster* dalam bentuk *feature vector*, maka centroid $c$ untuk *cluster* itu dihitung dengan

$$\mathbf{c} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{d}_i \tag{8.1}$$

yaitu nilai rata-rata setiap elemen *feature vector* untuk seluruh anggota *cluster* tersebut, dimana $N$ adalah banyaknya anggota *cluster* dan $\mathbf{d}_i$ adalah anggota ke-$i$ dalam representasi *feature vector*. 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 8.2, yaitu *dot product* antara dua vektor dibagi dengan perkalian *norm* kedua vektor.

$$cosSim(\mathbf{d}_i, \mathbf{d}_j) = \frac{\mathbf{d}_i \mathbf{d}_j}{\|\mathbf{d}_i\| \, \|\mathbf{d}_j\|} \tag{8.2}$$

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

## 8.2 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* [33]:

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.

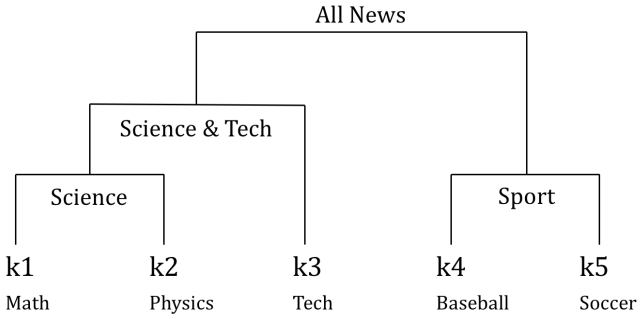
Sebagai contoh, algoritma *hierarchical clustering* menghasilkan struktur hirarkis seperti pada gambar 8.1 yang disebut **dendogram**. Dendogram melambangkan taksonomi, sebagai contoh taksonomi dokumen berita *sport*, dipecah menjadi *baseball* dan *soccer*.

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 8.1). Perhitungan kemiripan *clusters* dapat dihitung dengan tiga metode [35] (untuk data numerik):

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




**Gambar 8.1.** 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].

$$Sim_{single-link}(\mathbf{U}, \mathbf{V}) = \max_{\mathbf{u}_i \in \mathbf{U}, \mathbf{v}_j \in \mathbf{V}} cosSim(\mathbf{u}_i, \mathbf{v}_j) \tag{8.3}$$

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

$$Sim_{complete-link}(\mathbf{U}, \mathbf{V}) = \min_{\mathbf{u}_i \in \mathbf{U}, \mathbf{v}_j \in \mathbf{V}} cosSim(\mathbf{u}_i, \mathbf{v}_j) \tag{8.4}$$

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

$$Sim_{UPGMA}(\mathbf{U}, \mathbf{V}) = \frac{1}{|U|\,|V|} \sum_{\mathbf{u}_i \in \mathbf{U}, \mathbf{v}_j \in \mathbf{V}} cosSim(\mathbf{u}_i, \mathbf{v}_j) = \frac{\mathbf{c}^U \mathbf{c}^V}{|U|\,|V|} \tag{8.5}$$

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

## 8.3 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.

$$I = \frac{1}{N^2} \sum_{\mathbf{d}_i \mathbf{d}_j, i \neq j} cosSim(\mathbf{d}_i, \mathbf{d}_j) \tag{8.6}$$

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

$$I = \frac{1}{N^2} \sum_{\mathbf{d}_i \mathbf{d}_j, i \neq j} cosSim(\mathbf{d}_i, \mathbf{d}_j) = \|\mathbf{c}\|^2 \tag{8.7}$$

Perhitungan ini dapat dinormalisasi sesuai dengan banyaknya anggota cluster

$$I' = \frac{\|\mathbf{c}\|^2}{N} \tag{8.8}$$

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 [34].

$$E = \sum_{k=1}^{K} N_k \frac{\mathbf{c}_k \mathbf{c}^{\mathbf{D}}}{\|\mathbf{c}_k\|} \tag{8.9}$$

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:

$$H = \frac{\sum_{k=1}^{K} I'_k}{E} = \frac{\sum_{k=1}^{K} \frac{\|\mathbf{c}_k\|^2}{N_k}}{\sum_{k=1}^{K} N_k \frac{\mathbf{c}_k \mathbf{c}^{\mathbf{D}}}{\|\mathbf{c}_k\|}} = \sum_{k=1}^{K} \frac{\|\mathbf{c}_k\|}{N_k^2 \mathbf{c}_k \mathbf{c}^{\mathbf{D}}} \tag{8.10}$$

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

**8.1. Intra-cluster Evaluation**
Buktikan kebenaran persamaan 8.7.

**8.2. Entropy**
Bagaimana cara menghitung kualitas algoritma *clustering*, jika diberikan informasi label/ kelas setiap data menggunakan: (hint, baca [33])
(a) Entropy
(b) Purity

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

**8.4. 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.

**8.5. 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)?

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