5

Model Linear

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

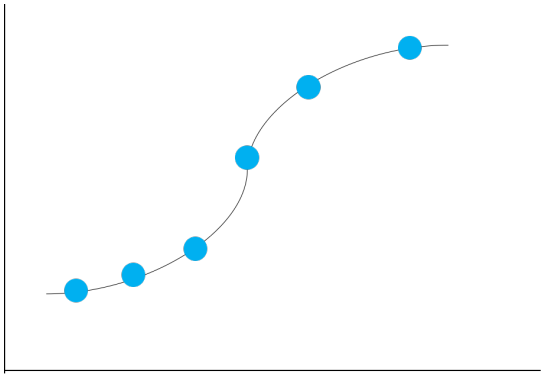
Bab ini membahas tentang model linear (algoritma parametrik), sebagai contoh pembelajaran yang sederhana karena cukup mudah dipahami idenya. Bab ini juga membahas *error function* dan *stochastic gradient descent*. Dimulai dari bab ini, kita akan memasuki materi lebih serius.

5.1 Curve Fitting dan Error Function

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

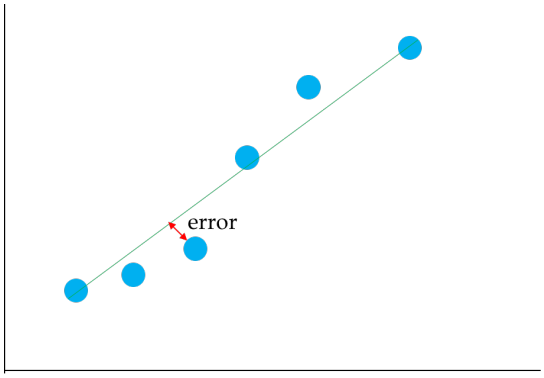
Error function paling mudah dijelaskan dalam permasalahan regresi. Diberikan $(x, y) \in \mathbb{R}$ sebagai *random variable*. Terdapat sebuah fungsi $f(x) \rightarrow y$, yang memetakan x ke y, berbentuk seperti pada Gambar 5.1. sekarang fungsi $f(x)$ tersebut disembunyikan (tidak diketahui), diberikan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

Kamu berasumsi bahwa fungsi $f(x)$ dapat diaproksimasi dengan fungsi linear $g(x) = xw + b$. Artinya, kamu ingin mencari w dan b yang memberikan nilai sedemikian sehingga $g(x)$ mirip dengan $f(x)$. w adalah parameter sementara b adalah *bias*. Anggap kamu sudah berhasil melakukan pendekatan dan



Gambar 5.1. Contoh fungsi Sigmoid

menghasilkan fungsi linear $g(x)$; seperti Gambar 5.2 (garis berwarna hijau). Akan tetapi, fungsi approksimasi ini tidak 100% tepat sesuai dengan fungsi aslinya (ini perlu ditekankan)[1]. Jarak antara titik biru terhadap garis hijau disebut *error*.



Gambar 5.2. Pendekatan fungsi Sigmoid

Salah satu cara menghitung *error* fungsi $g(x)$ adalah menggunakan ***squared error function*** dengan bentuk konseptual pada persamaan 5.1. Estimasi terhadap persamaan tersebut disajikan dalam bentuk diskrit pada persamaan 5.2. (x_i, y_i) adalah pasangan *training data* (*input* - *desired output*). Nilai *squared error* dapat menjadi tolak ukur untuk membandingkan kinerja suatu *learning machine* (model). Secara umum, bila nilainya tinggi, maka kinerja dianggap relatif buruk; sebaliknya bila rendah, kinerja dianggap relatif baik. Hal ini sesuai dengan konsep *intelligent agent* [5].

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

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

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

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

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

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

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

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

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

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

Ingat kembali bab 1, *learning machine* yang direpresentasikan dengan fungsi g bisa diatur kinerjanya dengan parameter *training* $\mathbf{w}$. *Squared error* untuk *learning machine* dengan parameter *training* $\mathbf{w}$ diberikan oleh persamaan 5.2. (x_i, y_i) adalah pasangan *input-desired output*. Selain untuk menghitung ***squared error*** pada *training data*, persamaan 5.2 juga dapat digunakan untuk menghitung *squared error* pada testing data. Tujuan dari regresi/*machine learning* secara umum adalah untuk meminimalkan nilai *loss* atau *error* baik pada *training* maupun *unseen instances*. Secara umum, kita lebih ingin meminimalkan *loss*, dimana *error* dapat menjadi *proxy* untuk *loss*. Selain *error function*, ada banyak fungsi lainnya seperti *Hinge*, *Log Loss*, *Cross-entropy loss*, *Ranking loss* [1].

5.2 Binary Classification

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

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

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

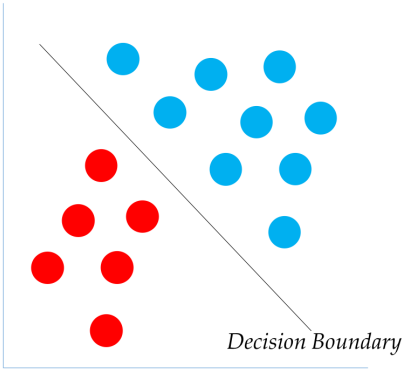
$$\text{output} = \text{sgn}(f(\mathbf{x}) \tag{5.8}$$

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

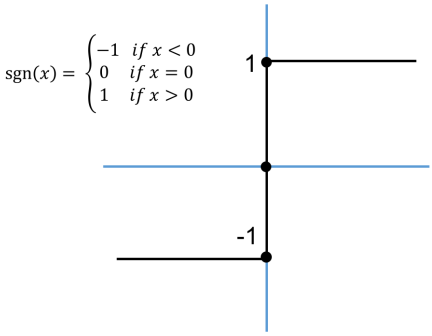
5.3 Log-linear Binary Classification

Pada subbab sebelumnya, telah dijelaskan fungsi binary classifier memetakan data menjadi nilai $[-1, 1]$, dengan -1 merepresentasikan kelas pertama dan 1 merepresentasikan kelas kedua. Tidak hanya kelas yang berkorespondensi,

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



Gambar 5.3. Contoh *decision boundary*



Gambar 5.4. Fungsi *sign*

kita juga terkadang ingin tahu seberapa besar peluang data tergolong pada kelas tersebut. Salah satu alternatif adalah dengan menggunakan fungsi sigmoid dibanding fungsi *sign* untuk merubah nilai fungsi menjadi $[0, 1]$ yang merepresentasikan peluang data diklasifikasikan sebagai kelas tertentu (1 - nilai peluang, untuk kelas lainnya). Konsep ini dituangkan menjadi persamaan 5.9, dimana y merepresentasikan probabilitas *input* $\mathbf{x}$ digolongkan ke kelas tertentu, $\mathbf{x}$ merepresentasikan data (*feature vector*), dan b merepresentasikan *bias*. Ingat kembali materi bab 4, algoritma *Naive Bayes* melakukan hal serupa[3]. Hasil fungsi sigmoid, apabila di-*plot* maka akan berbentuk seperti Gambar 5.1 (berbentuk karakter "S").

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

[3] Menggunakan nilai peluang untuk klasifikasi

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

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

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

Ketika fungsi *machine learning* menghasilkan nilai berbentuk probabilitas, kita dapat menggunakan *cross entropy* sebagai *utility function*. Persamaan 5.11 adalah cara menghitung *cross entropy*, dimana $P(c_i)$ melambangkan probabilitas *input* diklasifikasikan ke kelas c_i dan N melambangkan banyaknya kelas. Untuk *binary classification*, $P(c_1) = 1 - P(c_2)$.

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

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

5.4 Multi-class Classification

Subbab ini akan membahas tentang *multi-class classification*, dimana terdapat lebih dari dua kemungkinan kelas. Terdapat himpunan kelas C beranggotakan $\{c_1, c_2, \ldots, c_K\}$. Untuk suatu data dengan representasikan *feature vector*-nya, kita ingin mencari tahu kelas yang berkorespondesi untuk data tersebut. Contoh permasalahan ini adalah mengklasifikasi gambar untuk tiga kelas: *apel*, *jeruk*, atau *mangga*. Cara sederhana adalah memiliki tiga buah vektor parameter dan *bias* berbeda, $\mathbf{w}_{apel}$, $\mathbf{w}_{jeruk}$, $\mathbf{w}_{mangga}$, dan *bias* $b_{\{apel,jeruk,mangga\}}$. Untuk menentukan suatu data masuk ke kelas mana, kita

dapat memprediksi skor tertinggi yang diberikan oleh operasi *feature vector* terhadap masing-masing vektor parameter. Konsep matematisnya diberikan pada persamaan 5.12, dimana $\hat{c}$ adalah kelas terpilih (keputusan), yaitu kelas yang memiliki nilai tertinggi. C melambangkan himpunan kelas.

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

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

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

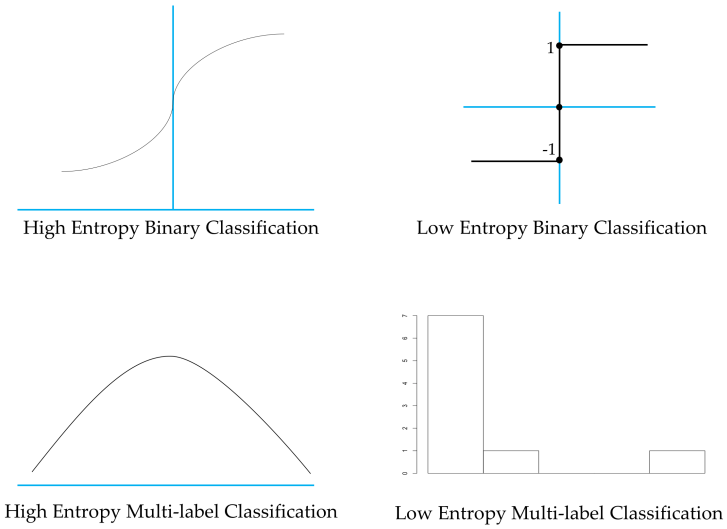
$$\hat{c} = \arg\max_{c_i \in \mathbf{c}} c_i \tag{5.14}$$

Seperti yang diceritakan pada subbab berikutnya, kita mungkin juga tertarik dengan probabilitas masing-masing kelas, bukan hanya *likelihood*-nya. Kita dapat menggunakan fungsi *softmax*[4] untuk hal ini. Fungsi *softmax* mentransformasi $\mathbf{c}$ agar jumlah semua nilainya berada pada range $[0, 1]$. Dengan itu, $\mathbf{c}$ dapat diinterpretasikan sebagai distribusi probabilitas. Konsep ini dituangkan pada persamaan 5.15, dimana c_i adalah elemen vektor ke-i, melambangkan probabilitas masuk ke kelas ke-i.

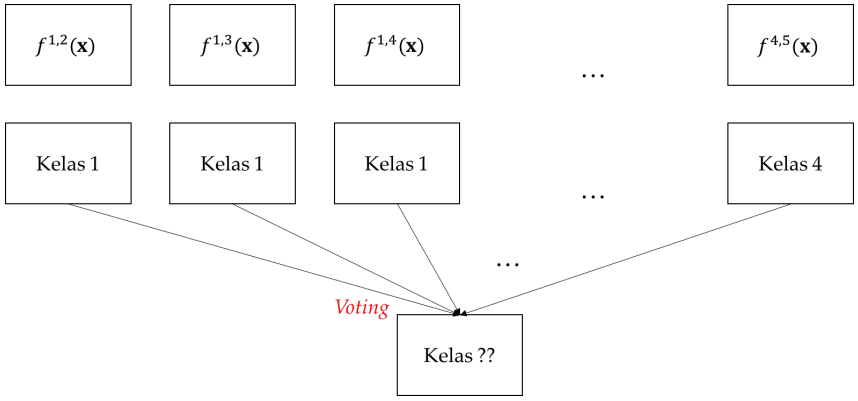
$$c_i = \frac{e^{(\mathbf{x} \cdot \mathbf{W} + \mathbf{b})_{[i]}}}{\sum_j e^{(\mathbf{x} \cdot \mathbf{W} + \mathbf{b})_{[j]}}} \tag{5.15}$$

Karena *output* fungsi berupa distribusi probabilitas, kita juga dapat menghitung *loss* menggunakan *cross entropy*. Seperti yang sudah dijelaskan sebelumnya pada *binary classification*, kita ingin hasil perhitungan *cross entropy* sekecil mungkin karena meminimalkan nilai *cross entropy* meningkatkan *confidence* saat proses klasifikasi. Hal ini dapat dianalogikan dengan jargon "*winner takes it all*". Sebagai ilustrasi, lihatlah Gambar 5.5. Kita ingin agar bentuk distribusi probabilitas *output* kelas ($\mathbf{c}$) condong ke salah satu sisi saja.

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



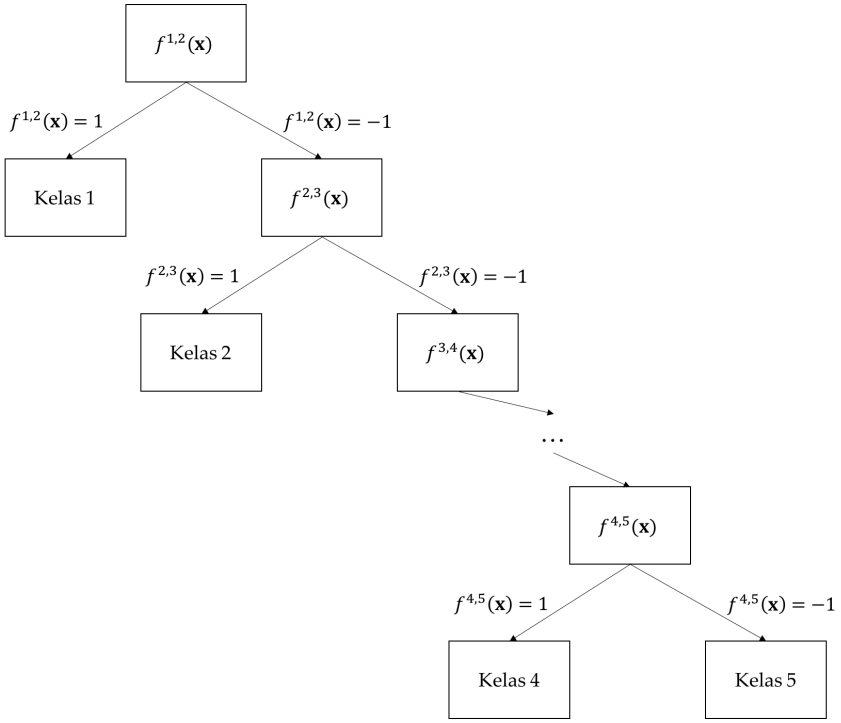
Gambar 5.5. *Classification Entropy*



Gambar 5.6. *One versus one*

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

Pada teknik *one versus one*, kita membuat sejumlah kombinasi pasangan kelas $\binom{N}{2}$, untuk N = banyaknya kelas. Kemudian, kita biarkan masing-masing model mengklasifikasikan *input* ke dalam satu dari dua kelas. Akhirnya, kita memilih kelas klasifikasi berdasarkan kelas yang paling sering muncul dari



Gambar 5.7. *One versus all*

semua model. Hal ini diilustrasikan pada Gambar 5.6 (untuk lima kelas). $f^{i,j}$ melambangkan *binary classifier* untuk kelas i dan kelas j.

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

5.5 Multi-label Classification

Seperti halnya *multi-class classification*, kita dapat mendekomposisi *multi-label classification* menjadi beberapa *binary classifier* (analogi persamaan 5.12). Yang membedakan *multi-class* dan *multi-label* adalah output $\mathbf{c}$. Pada *multi-class classification*, $c_i \in \mathbf{c}$ melambangkan probabilitas suatu instans masuk ke kelas c_i. Keputusan akhir *class assignment* didapatkan dari elemen $\mathbf{c}$ dengan nilai terbesar. Untuk *multi-label classification* nilai $c_i \in \mathbf{c}$ melambangkan apakah suatu kelas masuk ke kelas c_i atau tidak. Bedanya, kita boleh meng-*assign* lebih dari satu kelas (atau tidak sama sekali). Misal $c_i \geq 0.5$, artinya

kita anggap model tersebut layak masuk ke kelas c_i, tanpa harus membandingannya dengan nilai $c_j (i \neq j)$ lain. Inilah yang dimaksud dengan prinsip *mutual exclusivity*. Perhatikan Gambar 5.8 sebagai ilustrasi, dimana "1" melambangkan *class assignment*.

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

Multi-class Classification

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

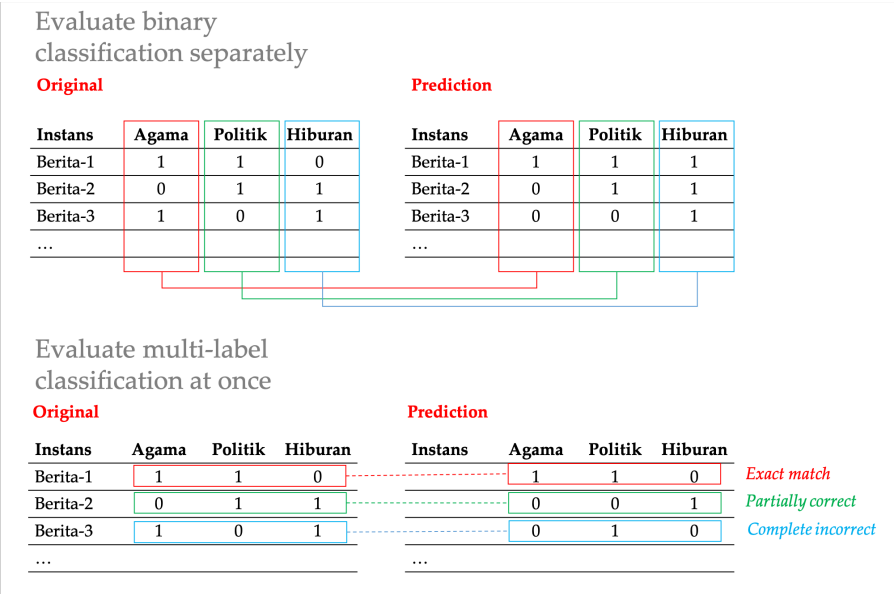
Multi-label Classification

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

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

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

Berhubung cara menginterpretasikan *output* berbeda dengan *multi-class classification*, cara evaluasi kinerja juga berbeda. Ada dua pendekatan evaluasi pada *multi-label classification*. Pertama, kita evaluasi kinerja *binary classification* untuk setiap kelas. Pada pendekatan ini, seolah-olah kita memiliki banyak *binary classifiers* untuk melakukan klasifikasi. Kedua, kita dapat mengevaluasi kinerja *multi-label classification* itu sendiri. Perhatikan Gambar 5.9! Pendekatan pertama ibarat membandingkan kolom *desired output* untuk masing-masing kelas dengan *prediction* yang berkorespondensi. Pada pendekatan kedua, kita membandingkan baris untuk tiap-tiap prediksi (seperti biasa). Saat kita membandingkan tiap-tiap baris prediksi, kita bisa jadi mendapatkan prediksi tipe ***exact match*** (seluruh prediksi sama dengan *desired output*), ***partially correct*** (sebagian prediksi sama dengan *desired output*) atau ***complete incorrect*** (tidak ada prediksi yang sama dengan *desired output*). Dengan ini, evaluasi *multi-label classification* relatif lebih kompleks dibanding *multi-class classification* biasa. Untuk mendapatkan *multi-label classification accuracy*, kita dapat menghitung berapa seberapa banyak *exact match* dibandingkan jumlah instans (walaupun hal ini terlalu ketat). Selain itu, kita dapat menghitung *loss* menggunakan *cross entropy* untuk mengevaluasi kinerja saat melatih *multi-label classifier*, layaknya *multi-class classifier*. Kamu



Gambar 5.9. Cara mengevaluasi *multi-label classifier*

dapat membaca [26] lebih lanjut untuk teknik evaluasi *multi-label classification*.

5.6 Pembelajaran sebagai Permasalahan Optimisasi

Salah satu tujuan dari pembelajaran (*training*) adalah untuk meminimalkan *error* sehingga kinerja *learning machine* (model) diukur oleh *squared error*. Dengan kata lain, *utility function* adalah meminimalkan *squared error*. Secara lebih umum, kita ingin meminimalkan/memaksimalkan suatu fungsi yang dijadikan tolak ukur kinerja (*utility function*), diilustrasikan pada persamaan 5.17, dimana θ adalah *learning parameter*[5], dan $\mathcal{L}$ adalah *loss function*. Perubahan parameter dapat menyebabkan perubahan *loss*. Karena itu, *loss function* memiliki θ sebagai parameternya.

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

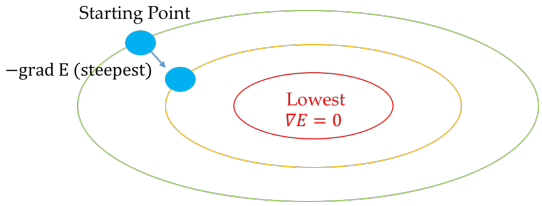
dimana $\hat{\theta}$ adalah nilai parameter paling optimal. Perhatikan, "arg min" dapat juga diganti dengan "arg max" tergantung **optimisasi** apa yang ingin dilakukan.

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

Sekarang, mari kita hubungkan dengan contoh yang sudah diberikan pada subbab sebelumnya. Kita coba melakukan estimasi *minimum squared error*, dengan mencari nilai *learning parameters* $\mathbf{w}$ yang meminimalkan nilai *error* pada model linear (persamaan 5.18)[6]. Parameter model pembelajaran mesin biasanya diinisialisasi secara acak atau menggunakan distribusi tertentu. Terdapat beberapa cara untuk memimalkan *squared error*. Yang penulis akan bahas adalah *stochastic gradient descent method*[7]. Selanjutnya apda buku ini, istilah *gradient descent*, *gradient-based method* dan *stochastic gradient descent* mengacu pada hal yang sama.

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

Bayangkan kamu sedang berada di puncak pegunungan. Kamu ingin mencari titik terendah pegunungan tersebut. Kamu tidak dapat melihat keseluruhan pegunungan, jadi yang kamu lakukan adalah mencari titik terendah (lokal) sejauh mata memandang, kemudian menuju titik tersebut dan menganggapnya sebagai titik terendah (global). Layaknya asumsi sebelumnya, kamu juga turun menuju titik terendah dengan cara melalui jalanan dengan kemiringan paling tajam, dengan anggapan bisa lebih cepat menuju ke titik terendah [9]. Sebagai ilustrasi, perhatikan Gambar 5.10!



Gambar 5.10. *Stochastic Gradient Descent*

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

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

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

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

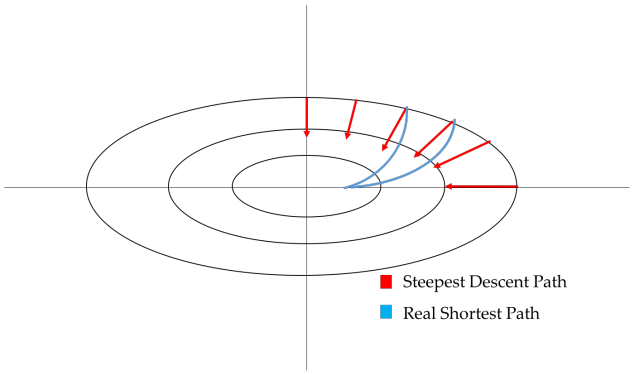
[7] Konsep *Hill Climbing* dapat digunakan untuk memaksimalkan *utility function*. Konsep tersebut sangat mirip dengan *gradient descent* `https://en.wikipedia.org/wiki/Hill_climbing`.

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

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

dimana η disebut ***learning rate*** dan $\mathbf{w}(t)$ adalah nilai $\mathbf{w}$ saat waktu/iterasi t. *Learning rate* digunakan untuk mengatur seberapa pengaruh keterjalan terhadap pembelajaran. Silahkan mencari sumber tambahan lagi agar dapat mengerti *learning rate* secara lebih dalam/matematis. Pada implementasi, η juga sering diubah-ubah nilainya sepanjang waktu. Semakin kita sudah dekat dengan tujuan (titik *loss* terendah), kita mengurangi nilai η, ibaratnya seperti mengerem kalau sudah dekat dengan tujuan [27].

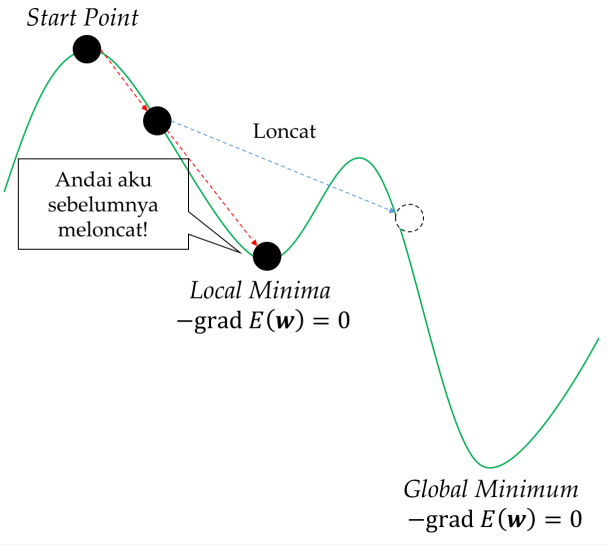
Walaupun kamu berharap bisa menuju titik terendah dengan menelusuri jalan terdekat dengan kemiringan paling tajam, tapi kenyataanya hal tersebut bisa jadi bukanlah jalan tercepat, seperti yang diilustrasikan pada Gambar 5.11. Warna merah melambangkan jalan yang dilalui *gradient descent*, sementara warna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



Gambar 5.11. *Stochastic Gradient Descent* 2

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

Dalam *local point of view*, *steepest gradient descent* adalah cara tercepat menuju titik terendah, tetapi tidak pada *global point of view*. Kita dapat



Gambar 5.12. *Stuck at local minima*

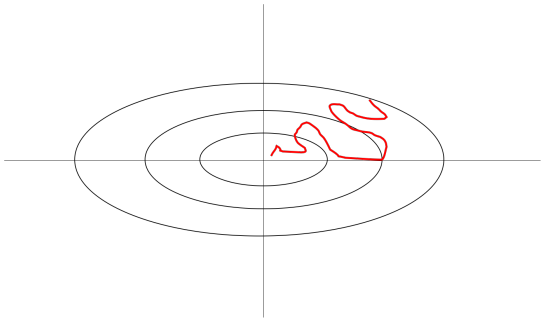
macet/berhenti saat sudah mencapai *local minima*, yaitu nilai minimum pada suatu daerah lokal saja. Sebagai ilustrasi, perhatikan Gambar 5.12. Kamu berada di puncak pegunungan kemudian turun bertahap. Kemudian, kamu sampai di suatu daerah landai ($-\text{grad } E(\mathbf{w}) = 0$). Kamu pikir daerah landai tersebut adalah titik terendah, tetapi, kamu ternyata salah. Untuk menghindari hal tersebut, kita menggunakan *learning rate* (η). Apabila nilai *learning rate* (η) pada persamaan 5.21 relatif kecil, maka dinamika perubahan parameter $\mathbf{w}$ juga kecil. Tetapi, bila nilainya besar, maka jalanan menuju titik terendah akan bergoyang-goyang (*swing*), seperti pada Gambar 5.13. Goyangan tersebut ibarat "meloncat-loncat" pada ilustrasi Gambar 5.12.

Untuk mengontrol *learning parameter* $\mathbf{w}$ sehingga memberikan nilai $E(\mathbf{w})$ terendah, persamaan *steepest gradient descent* dapat ditambahkan dengan momentum (α) pada persamaan 5.23. Alfa adalah momentum karena dikalikan dengan hasil perbedaan descent pada tahap sebelumnya. Alfa adalah parameter kontrol tambahan untuk mengendalikan *swing* yang sudah dibahas sebelumnya.

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

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

Apabila gradien bernilai 0, artinya model sudah berada pada titik *local/global optimum*. Kondisi ini disebut sebagai **konvergen** (*converging*). Se-



Gambar 5.13. *Swing*

mentara model yang tidak menuju titik optimal, malah menuju ke kondisi yang semakin tidak optimum, disebut **divergen** (*diverging*).

5.7 Batasan Model Linear

Model linear, walaupun mudah dimengerti, memiliki beberapa batasan. Ada dua batasan paling kentara [17]: (1) *additive assumption* dan (2) *linear assumption*.

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

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

Dengan persamaan 5.24, apabila x_1 berubah, maka kontribusi x_2 terhadap *output* juga akan berubah (dan sebaliknya). Akan tetapi, seringkali interaksi antar-fitur tidaklah sesederhana ini. Misal, semua fitur berinteraksi satu sama lain secara non-linear.

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

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

Asumsi yang sebelumnya dijelaskan pada pemodelan polinomial, dapat diekstensi menjadi *generalized additive model* (GAM) untuk mengatasi masalah *linear assumption*, seperti diilustrasikan pada persamaan 5.26 [17]. Artinya, kita melewatkan setiap fitur x_i pada suatu fungsi g_i, sehingga delta x_i tidak mengakibatkan perubahan yang konstan terhadap *output*. Ekstensi ini dapat memodelkan hubungan non-linear antara fitur dan *output*.

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

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

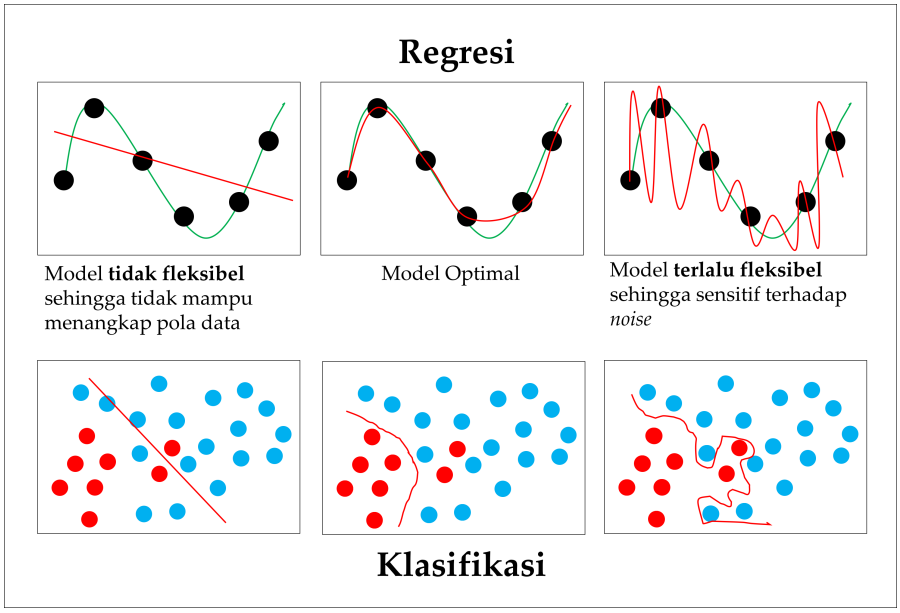
5.8 Overfitting dan Underfitting

Tujuan *machine learning* adalah membuat model yang mampu memprediksi data yang belum pernah dilihat (*unseen instances*) dengan tepat; disebut sebagai generalisasi (*generalization*). Seperti yang sudah dijelaskan pada bab pertama, kita dapat membagi dataset menjadi *training*, *development*, dan *testing* dataset. Ketiga dataset ini berasal dari populasi yang sama dan dihasilkan oleh distribusi yang sama (*identically and independently distributed*). Dalam artian, ketiga jenis dataset mampu melambangkan (merepresentasikan) karakteristik yang sama[8]. Dengan demikian, kita ingin *loss* atau *error* pada *training*, *development*, dan *testing* bernilai kurang lebih bernilai sama (i.e., kinerja yang sama untuk data dengan karakteristik yang sama). Akan tetapi, ***underfitting*** dan ***overfitting*** mungkin terjadi.

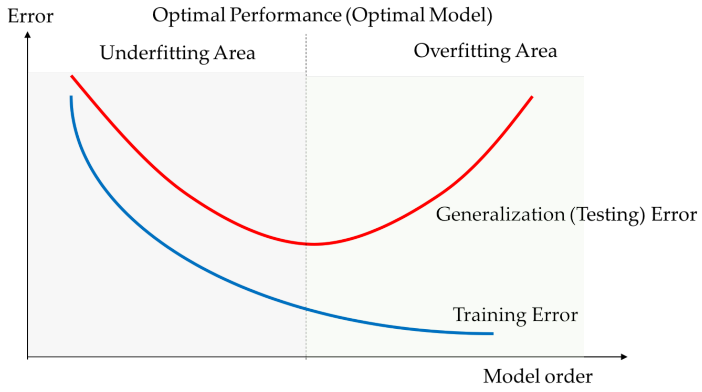
Underfitting adalah keadaan ketika kinerja model bernilai buruk baik pada *training* atau *development* maupun *testing data*. *Overfitting* adalah keadaan ketika kinerja model bernilai baik untuk *training* tetapi buruk pada *unseen data*. Hal ini diilustrasikan pada Gambar 5.14. *Underfitting* terjadi akibat model yang terlalu tidak fleksibel, yaitu memiliki kemampuan yang rendah untuk mengestimasi variasi fungsi. Sedangkan, *overfitting* terjadi ketika model terlalu fleksibel, yaitu memiliki kemampuan yang terlalu tinggi untuk mengestimasi banyak fungsi atau terlalu mencocokkan diri terhadap *training* data. Perhatikan kembali Gambar 5.14, dataset asli diambil (*sampled*) dari fungsi polinomial orde-3. Model *underfitting* hanya mampu mengestimasi dalam orde-1 (kemampuan terlalu rendah), sedangkan model *overfitting* mampu mengestimasi sampai orde-9 (kemampuan terlalu tinggi).

Apabila kita gambarkan grafik kinerja terhadap konfigurasi model (*model order*), fenomena *underfitting* dan *overfitting* dapat diilustrasikan seperti Gambar 5.15. Model yang ideal adalah model yang memiliki kinerja yang

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



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



Gambar 5.15. *Selection Error*

baik pada *training*, *development*, dan *testing* data. Artinya, kita ingin perbedaan kinerja model pada berbagai dataset bernilai sekecil mungkin. Untuk menghindari *overfitting* atau *underfitting*, kita dapat menambahkan fungsi ***noise/bias*** (selanjutnya disebut *noise/bias* saja) dan regularisasi (subbab 5.9). Hal yang paling perlu pembaca pahami adalah untuk jangan merasa senang ketika model *machine learning* yang kamu buat memiliki kinerja baik pada *training data*. Kamu harus mengecek pada *development* dan *testing* data, serta memastikan kesamaan karakteristik data, e.g., apakah *training* dan *testing*

data benar diambil dari distribusi yang sama. Selain itu, kamu juga harus memastikan apakah data yang digunakan mampu merepresentasikan kompleksitas pada permasalahan asli/dunia nyata. Sering kali, dataset yang digunakan pada banyak eksperimen adalah *toy dataset*, semacam simplifikasi permasalahan dengan jumlah instans yang relatif sedikit. Kamu harus hati-hati terhadap *overclaiming*, i.e., menjustifikasi model dengan performa baik pada *toy dataset* sebagai model yang baik secara umum.

5.9 Regularization

Gradient-based method mengubah-ubah parameter model $\mathbf{w}$ sehingga *loss/error* dapat diminimalisir. Perhatikan kembali Gambar 5.2, ibaratnya agar fungsi aproksimasi kita menjadi sama persis dengan fungsi asli pada Gambar 5.1. Perhatikan, karena nilai $\mathbf{w}$ berubah-ubah seiring waktu, bisa jadi urutan *training data* menjadi penting[10]. Pada umumnya, kita menggunakan *batch method* agar kinerja model tidak bias terhadap urutan data. Artinya, menghitung *loss* untuk beberapa data sekaligus. Hal ini akan kamu lebih mengerti setelah mambaca bab 11.

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

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

Pilihan umum untuk fungsi *regularization* pada umumnya adalah L_2 dan L_1 norm. L_2 *regularization* menggunakan jumlah kuadrat dari *Euclidean norm*[11] seluruh parameter, seperti pada persamaan 5.28. Sedangkan, L_1 *regularization* menggunakan jumlah dari nilai *absolute-value norm* seluruh parameter, diberikan pada persamaan 5.29.

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

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

Perlu kamu perhatikan, metode *gradient descent* memiliki syarat bahwa fungsi yang digunakan haruslah dapat diturunkan (*differentiable*). L_2 dapat

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

diturunkan pada seluruh poin, sementara L_1 tidak dapat diturunkan pada semua poin[12]. Kita dapat menggunakan teknik *subgradient* untuk menyelesaikan permasalahan ini. Kami serahkan pembaca untuk mengeksplorasi teknik tersebut sendiri. Implikasi penggunakan regularisasi terhadap kompleksitas model dibahas lebih lanjut pada bab 9.

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

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

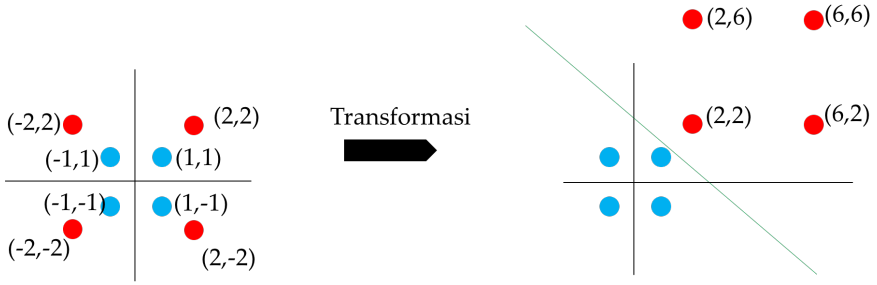
5.10 Transformasi Data

Seperti yang sudah dijelaskan sebelumnya, alangkah baik apabila semua data memiliki hubungan secara linear atau bersifat *linearly separable*. Kenyataannya, kebanyakan data bersifat *non-linearly separable*. Kita dapat mentransformasi data yang bersifat *non-linearly separable* menjadi *linearly-separable* sebelum menggunakan model linear untuk mengklasifikasikan data. Sebagai contoh, perhatikan Gambar 5.16. Pada gambar bagian kiri, terdapat empat titik yang *non-linearly separable*. Titik-titik itu ditransformasi sehingga menjadi gambar bagian kanan. Fungsi transformasi yang digunakan diberikan pada persamaan 5.30.

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

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

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



Gambar 5.16. Contoh transformasi [7]

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

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

Untuk penjelasan masing-masing *kernel*, silahkan membaca literatur lain lebih lanjut. Model linear yang memanfaatkan fungsi-fungsi *kernel* ini adalah *support vector machine* (SVM) [28, 29]. Perhatikan, algorithma SVM sebenarnya sangatlah penting. Akan tetapi, perlu kami informasikan bahwa buku ini tidak memuat materi SVM secara detil. Dengan demikian, kami harap pembaca dapat mencari referensi lain tentang SVM. Metode transformasi pun tidaklah sempurna, seperti yang akan dijelaskan pada bab 11, karena memang data secara alamiah memiliki sifat yang non-linear. Dengan demikian, lebih baik apabila kita langsung saja memodelkan permasalahan dengan fungsi non-linear.

5.11 Bacaan Lanjutan

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

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

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

Soal Latihan

5.1. Hill Climbing
Baca dan jelaskanlah konsep Hill Climbing!

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

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

5.4. Sum of Squared Errors
Salah satu cara untuk mencari nilai parameter pada regresi adalah menggunakan teknik *sum of squared errors*. Jelaskanlah bagaimana cara kerja metode tersebut! (Hint: baca buku oleh James et al. [17])

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