Jan Wira Gotama Putra

Pengenalan Konsep Pembelajaran Mesin dan Deep Learning

Edisi 1.0

March 11, 2018

Untuk Tuhan, Bangsa, dan Almamater

Kata Pengantar

Buku ini ditujukan sebagai bahan penunjuang (atau pengantar) mata kuliah *machine learning* untuk mahasiswa di Indonesia, khususnya tingkat sarjana (tidak menutup kemungkinan digunakan untuk tingkat pascasarjana). Buku ini hanya merupakan komplemen, bukan sumber informasi utama. Beberapa *reviewers* merasa materi buku ini relatif cukup berat, karena itu ada baiknya membaca buku pengantar yang lebih "ringan" sebelum membaca buku ini.

Walaupun tidak sempurna, mudah-mudahan buku ini mampu memberi inspirasi. Anggap saja membaca buku ini seperti sedang membaca "*light novel*". Penulis ingin buku ini bisa menjadi *pointer*; i.e. dengan membaca buku ini, diharapkan kawan-kawan juga mengetahui harus belajar apa (lebih jauhnya) dalam bidang *machine learning*. Setelah membaca buku ini, pembaca diharapkan mampu membaca literatur *machine learning* yang dijelaskan secara matematis (kami memberi rekomendasi bacaan lanjutan).

Di Indonesia, penulis banyak mendengar baik dari teman, junior, senior, dll; suatu pernyataan "kuliah mengajari teori saja, praktiknya kurang, dan tidak relevan dengan industri". Menurut saya di satu sisi itu benar; tapi di sisi lain, karena permikiran macam itu terkadang kita tidak benar-benar mengerti permasalahan. Ketika mengalami kendala, kita buntu saat mencari solusi karena fondasi yang tidak kokoh. Banyak orang terburu-buru "menggunakan *tools*" karena lebih praktikal. Penulis ingin mengajak saudara/i untuk memahami konsep *machine learning* secara utuh sebelum memanfaatkan.

Buku ini menjelaskan algoritma *machine learning* dari sudut pandang "agak" matematis. Pembaca disarankan sudah memahami/mengambil setidaknya mata kuliah statistika, kalkulus, aljabar linier/geometri, pengenalan kecerdasan buatan, dan logika fuzzy. Penulis merasa banyak esensi yang hilang ketika materi *machine learning* hanya dijelaskan secara deskriptif karena itu buku ini ditulis dengan bahasa "agak" matematis. Saat membaca buku ini, disarankan membaca secara runtun. Gaya penulisan buku ini **santai/semiformal** agar lebih mudah dipahami, dengan notasi matematis dibuat seminimal mungkin, mudah-mudahan tanpa mengurangi esensi materi.

Buku ini ditulis menggunakan template monograph (LaTeX) dari Springer yang dimodifikasi. Dengan demikian, mungkin ada kesalahan pemenggalan kata (karena dipenggal berdasarkan jumlah karakter).

Petunjuk Penggunaan

Struktur penyajian buku ini dapat dijadikan acuan sebagai struktur kuliah *machine learning* untuk satu semester (bab 1 untuk sesi pertama, dst), sementara materi mungkin masih perlu ditambahkan diluar buku ini. Penulis sangat menyarankan untuk membahas soal latihan sebagai tambahan materi (bisa juga sebagai PR). Soal latihan ditujukan untuk mengarahkan apa yang harus dibaca/dipahami lebih lanjut. Agar dapat memahami materi per bab, bacalah keseluruhan isi bab secara utuh sebelum mempertanyakan isi materi.

Pembaca dipersilahkan menyebar buku ini untuk alasan **NON KOMERSIAL** (pendidikan), tetapi **dimohon kesadarannya untuk tidak menyalin /meniru isi buku ini**. Bila ingin memuat konten diktat ini pada media yang pembaca kelola, dimohon untuk mengontak pengarang terlebih dahulu. Tidak semua istilah bahasa asing diterjemahkan ke Bahasa Indonesia supaya makna sebenarnya tidak hilang (atau penulis tidak tahu versi Bahasa Indonesia yang baku).

Bab lebih awal memuat materi yang relatif lebih "mudah" dipahami dibanding bab berikutnya. Buku ini memberikan contoh dimulai dari contoh sederhana (beserta contoh data). Semakin menuju akhir buku, notasi yang digunakan akan semakin simbolik, beserta contoh yang lebih abstrak. Penulis menyarankan untuk membaca buku ini secara sekuensial.

Kutipan

Buku ini tergolong *self-published work*, tetapi sudah di-*review* oleh beberapa orang. Kami yakin para *reviewers* adalah orang yang berkompeten. Silahkan merujuk buku ini sesuai dengan paduan cara merujuk *self-published work* (apabila diperbolehkan untuk merujuk *self-published work* pada pekerjaan kamu).

Notasi Penting

Karakter *bold* kapital merepresentasikan matriks ($\mathbf{X}, \mathbf{Y}, \mathbf{Z}$). Dimensi matriks ditulis dengan notasi $N \times M$ dimana N merepresentasikan banyaknya baris dan M merepresentasikan banyaknya kolom. Elemen matriks direpresentasikan oleh $\mathbf{X}_{i,j}$, $\mathbf{X}_{[i,j]}$, atau $x_{i,j}$ untuk baris ke-i kolom ke-j (penggunaan akan menyesuaikan konteks pembahasan agar tidak ambigu). Karakter di-*bold* merepresentasikan vektor ($\mathbf{x}$). Elemen vektor ke-i direpresentasikan oleh x_i atau $\mathbf{x}_{[i]}$ tergantung konteks. Ketika penulis menyebutkan vektor, yang dimaksud adalah **vektor baris** (*row vector*, memiliki dimensi $1 \times N$, mengadopsi notasi Goldberg [1]). Perhatikan, literatur *machine learning* lainnya mungkin tidak menggunakan notasi *row vector* tetapi *column vector*. Kami harap pembaca mampu beradaptasi. Simbol " $\cdot$ " digunakan untuk melambangkan operator *dot-product*.

Kumpulan data (atau himpunan) direpresentasikan dengan karakter kapital (C, Z), dan anggotanya (*data point*, *data entry*) ke-i direpresentasikan dengan karakter c_i. Perhatikan, elemen vektor dan anggota himpunan bisa memiliki notasi yang sama (himpunan dapat direpresentasikan di komputer sebagai *array* jadi, penggunaan notasi vektor untuk himpunan pada konteks pembicaraan kita adalah tidak salah). Penulis akan menggunakan simbol $\mathbf{x}_{[i]}$ sebagai elemen vektor apabila ambigu. Fungsi dapat direpresentasikan dengan huruf kapital maupun non-kapital $f(\dots), E(\dots), G(\dots)$. Ciri fungsi adalah memiliki parameter! Pada suatu koleksi vektor (himpunan vektor) $\mathbf{D}$, vektor ke-i direpresentasikan dengan $\mathbf{d}_i$, dan elemen ke-j dari vektor ke-i direpresentaiskan dengan $\mathbf{d}_{i[j]}$, $\mathbf{D}_{i,j}$, atau $\mathbf{D}_{[i,j]}$ (karena sekumpulan vektor dapat disusun sebagai matriks).

Karakter non-kapital tanpa *bold* atau indeks (a, b, c, x, y, z) merepresentasikan *random variable* (statistik) atau variabel (matematik). Secara umum, saat *random variable* memiliki tertentu, dinotasikan dengan $x = X$ (nilai tertentu dinotasikan dengan huruf kapital), kecuali disebutkan secara khusus saat pembahasan. Probabilitas direpresentasikan dengan karakter kapital (P), dengan karakter non-kapital merepresentasikan probability density (p). Penulis yakin pembaca dapat menyesuaikan interpretasi simbol berdasarkan konteks pembahasan. Untuk menginterpretasikan notasi lain, selain yang diberikan pada paduan ini, mohon menyesuaikan dengan ceritera pembahasan.

Ucapan Terima Kasih
Penulis ingin mengucapkan terima kasih pada Bapak/Ibu/Saudara/i atas kontribusi pada pengembangan dan penulisan buku ini: Adhiguna Surya Kuncoro, Arief Yudha Satria, Candy Olivia Mawalim, Chairuni Aulia Nusapati, Genta Indra Winata, Hayyu Luthfi Hanifah, I Gede Mahendra Darmawiguna, dan Tifani Warnita.

Tokyo, Jepang

Jan Wira Gotama Putra
`https://wiragotama.github.io/`

Daftar Isi

Bagian I Pengetahuan Dasar

1 Pengenalan . . . 3
1.1 Kecerdasan Buatan . . . 3
1.2 Intelligent Agent . . . 5
1.3 Konsep Belajar . . . 7
1.4 Statistical Learning Theory . . . 8
1.5 Training, Development, Testing Set . . . 10
1.6 Supervised Learning . . . 11
1.7 Regresi . . . 14
1.8 Semi-supervised Learning . . . 14
1.9 Unsupervised Learning . . . 14
1.10 Proses Belajar . . . 16
1.11 Tips . . . 17
1.12 Contoh Aplikasi . . . 17
Soal Latihan . . . 18

2 Fondasi Matematis . . . 19
2.1 Probabilitas . . . 19
2.2 Probability Density Function . . . 21
2.3 Expectation dan Variance . . . 22
2.4 Bayesian Probability . . . 23
2.5 Gaussian Distribution . . . 24
2.6 Teori Keputusan . . . 26
2.7 Teori Informasi . . . 28
2.7.1 Entropy . . . 29
2.7.2 Relative Entropy dan Mutual Information . . . 30
2.8 Matriks . . . 31
2.9 Bacaan Lanjutan . . . 32
Soal Latihan . . . 32

3 Data Analytics . . . 33
3.1 Pengenalan Data Analytics . . . 33
3.2 Nilai Atribut dan Transformasi . . . 35
3.3 Ruang Konsep . . . 36
3.4 Linear Separability . . . 37
3.5 Seleksi Fitur . . . 38
3.6 Classification, Association, Clustering . . . 39
3.7 Mengukur Kinerja . . . 40
3.8 Evaluasi Model . . . 40
3.9 Kategori Jenis Algoritma . . . 42
3.10 Tahapan Analisis . . . 42
Soal Latihan . . . 42

Bagian II Algoritma Pembelajaran Mesin

4 Algoritma Dasar . . . 45
4.1 Naive Bayes . . . 45
4.2 K-means . . . 47
4.3 K-nearest-neighbor . . . 50
Soal Latihan . . . 50

5 Model Linear . . . 53
5.1 Curve Fitting dan Error Function . . . 53
5.2 Overfitting dan Underfitting . . . 55
5.3 Binary Classification . . . 57
5.4 Log-linear Binary Classification . . . 58
5.5 Multi-class Classification . . . 59
5.6 Transformasi . . . 60
5.7 Pembelajaran sebagai Permasalahan Optimisasi . . . 61
5.8 Regularization . . . 64
5.9 Bacaan Lanjutan . . . 65
Soal Latihan . . . 66

6 Pohon Keputusan . . . 67
6.1 Inductive Learning . . . 67
6.2 ID3 . . . 68
6.3 Isu pada ID3 . . . 72
6.4 Hubungan Decision Tree dan Model Linear . . . 72
Soal Latihan . . . 73

7 Hidden Markov Model . . . 75
7.1 Probabilistic Reasoning . . . 75
7.2 Generative Model . . . 78
7.3 Part-of-speech Tagging . . . 79

7.4 Hidden Markov Model Tagger . . . 82
7.5 Algoritma Viterbi . . . 84
7.6 Proses Training Hidden Markov Model . . . 86
Soal Latihan . . . 89

8 Clustering . . . 91
8.1 K-means, Pemilihan Centroid, Kemiripan Data . . . 92
8.2 Hierarchical Clustering . . . 93
8.3 Evaluasi . . . 94
Soal Latihan . . . 96

Bagian III Neural Networks

9 Artificial Neural Network . . . 99
9.1 Definisi . . . 99
9.2 Single Perceptron . . . 100
9.3 Permasalahan XOR . . . 102
9.4 Multilayer Perceptron . . . 104
9.5 Interpretability . . . 107
9.6 Binary Classification . . . 108
9.7 Multi-label Classification . . . 109
9.8 Deep Neural Network . . . 110
9.9 Tips . . . 112
9.10 Regularization and Dropout . . . 113
9.11 Vanishing and Exploding Gradients . . . 114
9.12 Rangkuman . . . 114
Soal Latihan . . . 115

10 Dimensionality Reduction dan Representation Learning . . . 117
10.1 Curse of Dimensionality . . . 117
10.2 Singular Value Decomposition . . . 119
10.3 Ide Dasar Autoencoder . . . 120
10.4 Representing Context: Word Embedding . . . 122
10.4.1 Vector Space Model . . . 123
10.4.2 Sequential, Time Series, dan Compositionality . . . 124
10.4.3 Distributed Word Representation . . . 125
10.4.4 Distributed Sentence Representation . . . 127
10.5 Tips . . . 130
Soal Latihan . . . 130

11 Arsitektur Neural Network . . . 131
11.1 Convolutional Neural Network . . . 131
11.1.1 Convolution . . . 133
11.1.2 Pooling . . . 134

11.1.3 Rangkuman 136
11.2 Recurrent Neural Network 136
11.3 Part-of-speech Tagging Revisited 141
11.4 Sequence to Sequence 144
11.4.1 Encoder 145
11.4.2 Decoder 146
11.4.3 Beam Search 146
11.4.4 Attention-based Mechanism 148
11.4.5 Variasi Arsitektur Sequence to Sequence 150
11.4.6 Rangkuman 151
11.5 Arsitektur Lainnya 153
Soal Latihan 153

Bagian IV Aplikasi dan Topik Tambahan

12 Penerapan Pembelajaran Mesin 157
12.1 Sistem Rekomendasi 158
12.1.1 Content-based Filtering 158
12.1.2 Collaborative Filtering 160
12.2 Peringkasan Dokumen 161
12.2.1 Pipelined Approach 163
12.2.2 Single-view Approach 163
12.3 Konklusi 164
12.4 Saran Buku Lanjutan 165
Soal Latihan 167

Referensi 169

Bagian I

Pengetahuan Dasar

1

Pengenalan

> "People worry that computers will get too smart and take over the world, but the real problem is that they're too stupid and they've already taken over the world."
>
> Pedro Domingos

Penulis yakin istilah *machine learning* atau *deep learning* sudah tidak asing di telinga pembaca. *Machine learning* dan *deep learning* adalah salah satu materi kuliah pada jurusan Teknik Informatika atau Ilmu Komputer. Selain mengenal kedua istilah tersebut di kuliah, pembaca mungkin mengenal istilah tersebut karena digunakan untuk pemasaran (*marketing*). Sebagai permulaan, *machine learning* dan *deep learning* bukanlah kedua hal yang sangat berbeda[1]. Perlu diingat, *deep learning* adalah bagian dari *machine learning*. *Machine learning* sudah diaplikasikan pada banyak hal, baik untuk klasifikasi gambar, mobil tanpa pengemudi, klasifikasi berita, dsb. Bab ini menjelaskan konsep paling dasar dan utama *machine learning*.

1.1 Kecerdasan Buatan

Pada bagian pembukaan (*kata pengantar*) telah dijelaskan bahwa kami menganggap kamu sudah memiliki pengetahuan dasar tentang *artificial intelligence* (kecerdasan buatan), kami akan memberikan sedikit ikhtisar apa hubungan kecerdasan buatan dan pembelajaran mesin. Saat pertama kali kamu mendengar istilah "kecerdasan buatan", mungkin kamu akan terpikir robot yang memiliki raga fisik. Tetapi, kecerdasan buatan tidak hanya terbatas pada sesuatu yang memiliki raga fisik. Raga fisik berguna untuk interaksi

[1] Walau istilah *deep learning* belakangan ini lebih populer.

yang ramah bagi manusia. Tidak mesti memiliki raga fisik, kecerdasan buatan sesungguhnya adalah program[2] yang memiliki bentuk matematis (instruksi); kita sebut sebagai **agen**. Berbeda dengan program biasa yang menghasilkan aksi berdasarkan instruksi, tujuan kecerdasan buatan adalah menciptakan program yang mampu mem-program (*output* program adalah sebuah program). Secara teori, program adalah ***automaton***[3] yang menjalankan suatu instruksi. Secara sederhana, program kecerdasan buatan sama halnya dengan program pada umumnya, yaitu menjalankan instruksi. Yang menjadikanya beda dengan program biasa adalah **kemampuan untuk belajar**[4].

Pada bidang keilmuan kecerdasan buatan, kita ingin menciptakan agen yang mampu melakukan pekerjaan yang membutuhkan kecerdasan manusia. Perhatikan, disini disebut kecerdasan manusia; hewan pun cerdas, tapi kecerdasan manusia dan hewan berbeda; yang kita ingin aproksimasi adalah kecerdasan manusia. Akan tetapi, kecerdasan manusia susah didefinisikan karena memiliki banyak aspek misalnya nalar (logika), kemampuan berbahasa, seni, dsb. Karena kecerdasan manusia memiliki banyak dimensi, kita dapat mencoba menyelesaikan masalah pada sub bidang lebih kecil (*divide and conquer*). Sampai saat ini pun, peneliti belum juga mengetahui secara pasti apa yang membuat manusia cerdas, apa itu sesungguhnya cerdas, dan bagaimana manusia dapat menjadi cerdas. Dengan demikian, keilmuan kecerdasan buatan adalah interdisiplin, memuat: psikologis, linguistik, ilmu komputer, biologi, dsb. Bila kamu bertanya apakah program deterministik dapat disebut *kecerdasan buatan*, jawabannya "iya", *to some extent* (sampai pada level tertentu) karena memenuhi dimensi *acting rationally* (dijelaskan pada subbab 1.2).

Permasalahan utama bidang kecerdasan buatan terdiri dari (dari klasik sampai lebih modern)[5], yaitu:

1. ***Planning***. Diberikan *start state* dan *goal state*, agen harus merencanakan sekuens aksi untuk merubah *start state* menjadi *goal state*. Contoh permasalahan *planning* adalah merencanakan rute perjalanan. Bisa jadi, saat merencanakan sekuens aksi, ada kendala (*constraints*) yang harus dioptimisasi.

2. ***Representasi pengetahuan***, yaitu merepresentasikan pengetahuan dalam bentuk formal. Dengan representasi formal tersebut, kita dapat melakukan inferensi dengan operasi logika berbentuk simbolik, misal logika preposisi, logika orde pertama (*first-order logic*), teori Fuzzy, *abductive reasoning*, ontologi, maupun jaringan semantik (*semantic web*) [3].

---

[2] Secara sederhana, program adalah kumpulan atau sekuens instruksi.
[3] Kami sarankan untuk membaca buku [2] untuk materi automata.
[4] Perlu diperhatikan, definisi ini adalah pandangan modern.
[5] Silahkan merujuk Association for the Advancement of Artificial Intelligence (AAAI).

3. ***Machine learning***, yaitu teknik untuk melakukan inferensi terhadap data dengan pendekatan matematis. Inti *machine learning* adalah untuk membuat model (matematis) yang merefleksikan pola-pola data (seiring kamu membaca buku ini, kamu akan lebih mengerti). Ini adalah bahasan utama buku ini.

4. ***Multi-agent system***, yaitu sistem yang memiliki banyak agen berinteraksi satu sama lain untuk menyelesaikan permasalahan. Agen satu mengerjakan suatu hal tertentu, kemudian bekerja bersama untuk menyelesaikan masalah yang lebih besar (tidak dapat diselesaikan sendiri).

5. Dan lain sebagainya, silahkan mengacu pada topik konferensi *Association for the Advancement of Artificial Intelligence* (AAAI) .

Perhatikan, sub keilmuan representasi pengetahuan dan *machine learning* sama-sama melakukan inferensi, tetapi pada representasi yang berbeda. Bila kamu ingat dengan mata pelajaran matematika SMA (logika preposisi), kamu sadar bahwa membuat sistem cerdas menggunakan representasi pengetahuan simbolik itu susah. Kita harus mendefinisikan *term*, aturan logika, dsb. Belum lagi kita harus mendefinisikan aturan-aturan secara manual. Representasi pengetahuan secara tradisional dianggap relatif kurang *scalable*, khususnya apabila kita bekerja dengan data yang besar. Sementara itu, *machine learning* berada pada daerah representasi data/ilmu/pengetahuan dalam bentuk matematis karena keilmuan *machine learning* diturunkan dari matematika dan statistika.

Pada masa sekarang, kita dianugrahi dengan data yang banyak (bahkan tidak terbatas), teknik *machine learning* menjadi intuitif untuk melakukan inferensi pada data yang besar. Hal ini yang menyebabkan *machine learning* menjadi populer karena konstruksi model inferensi dapat dilakukan secara otomatis. *Machine learning* ibarat sebuah "alat", sama seperti rumus matematika. Bagaimana cara menggunakannya tergantung pada domain permasalahan. Dengan demikian, kamu harus paham betul bahwa memahami teknik-teknik *machine learning* saja tidak cukup. Kamu juga harus mengetahui domain aplikasi yang bersesuaian karena pemanfaatan teknik-teknik *machine learning* dapat berbeda pada domain yang berbeda. Sedikit cerita, sub keilmuan *data science* mempelajari banyak domain, misalnya data pada domain sosial, ekonomi, bahasa, maupun visual. Seiring kamu membaca buku ini, kami harap kamu semakin mengerti hal ini.

1.2 Intelligent Agent

Agen cerdas memiliki empat kategori berdasarkan kombinasi dimensi cara inferensi (*reasoning*) dan tipe kelakuan (*behaviour*) [4, 5]. Kategori agen dapat dilihat pada Gambar. 1.1 dengan penjelasan sebagai berikut:

	Rationally	Humanly
Acting	acting rationally	acting humanly
Thinking	thinking rationally	thinking humanly

Gambar 1.1. Dimensi Kecerdasan.

1. ***Acting Humanly***. Pada dimensi ini, agen mampu bertingkah dan berinteraksi layaknya seperti manusia. Contoh terkenal untuk hal ini adalah *turing test*. Tujuan dari *turing test* adalah untuk mengevaluasi apakah suatu sistem mampu "menipu" manusia. Disediakan seorang juri, kemudian juri berinteraksi dengan sesuatu di balik layar. Sesuatu di balik layar ini bisa jadi manusia atau program. Program dianggap mampu bertingkah (berinteraksi) seperti layaknya manusia apabila juri tidak dapat membedakan ia sedang berkomunikasi dengan manusia atau program.

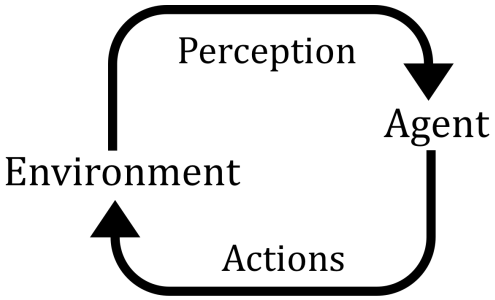
2. ***Acting Rationally***. Pada dimensi ini, agen mampu bertingkah dengan optimal. Tindakan optimal belum tentu menyerupai tindakan manusia, karena tindakan manusia belum tentu optimal. Misalnya, agen yang mampu memiliki rute terpendek dari suatu kota A ke kota B untuk mengoptimalkan penggunaan sumber daya. Sebagai manusia, bisa saja kita mencari jalan sesuka hati.

3. ***Thinking Humanly***. Pada dimensi ini, agen mampu berpikir seperti manusia dalam segi kognitif (e.g. mampu mengerti apa itu kesedihan atau kesenangan). Dapat dibilang, meniru bagaima proses berpikir di otak terjadi (pemodelan otak).

4. ***Thinking Rationally***. Pada dimensi ini, agen mampu berpikir secara rasional. Sederhananya sesuai dengan konsep logika matematika. *Thinking Humanly* lebih cenderung pada pemodelan kognitif secara umum, sementara dimensi *thinking rationally* cenderung pada pemodelan proses berpikir dengan prinsip optimisasi (apa yang harus dilakukan agar hasil optimal).

Perlu dicatat, "*acting*" berarti agen mampu melakukan aksi. Sementara "*thinking*" adalah pemodelan proses. Untuk mewujudkan interaksi manusia-komputer seperti manusia-manusia, tentunya kita ingin *intelligent agent* bisa mewujudkan dimensi *acting humanly* dan *thinking humanly*. Sayangnya, manusia tidak konsisten [6]. Sampai saat ini, konsep kecerdasan buatan adalah

meniru manusia; apabila manusia tidak konsisten, peneliti susah untuk memodelkan cara berpikir/tingkah laku manusia. Dengan hal itu, saat ini kita paling mungkin menciptakan agen yang mempunyai dimensi *acting rationally*.



Gambar 1.2. *Agent* vs *environment* [7].

Perhatikan Gambar. 1.2! Agen mengumpulkan informasi dari lingkungannya, kemudian memberikan respon berupa aksi. Lingkungan (*environment*) yang dimaksud bisa jadi macam-macam, misal: rumah, papan catur, agen lain, dsb. Kita ingin agen melakukan aksi yang benar. Tentu saja kita perlu mendefinisikan secara detail, teliti, tepat (*precise*), apa arti "aksi yang benar". Dengan demikian, lebih baik apabila kita mengukur kinerja agen, menggunakan ukuran kinerja (*performance measure*). Misalnya untuk robot pembersih rumah, *performance measure*-nya adalah seberapa persen debu yang dapat ia bersihkan. *Performance measure*, secara matematis dikenal sebagai fungsi utilitas (*utility function*), yaitu fungsi apa yang harus dimaksimalkan/diminimalkan oleh agen tersebut. Setiap tindakan yang dilakukan agen rasional harus mengoptimalkan nilai *performance measure* atau *utility function*. Pada buku ini, istilah *performance measure* dan *utility function* merujuk pada hal yang sama[6].

1.3 Konsep Belajar

Bayangkan kamu berada di suatu negara asing! Kamu tidak tahu norma yang ada di negara tersebut. Apa yang kamu lakukan agar bisa menjadi orang "normal" di negara tersebut? Tentunya kamu harus **belajar**! Kamu mengamati bagaimana orang bertingkah laku di negara tersebut dan perlahan-lahan mengerti norma yang berlaku. Belajar adalah usaha memperoleh kepandaian atau ilmu; berlatih; berubah tingkah laku atau tanggapan yang disebabkan

[6] Buku ini berfokus pada *performance measure* secara matematis, buku lain belum tentu berfokus pada hal yang sama.

oleh pengalaman[7]. Pembelajaran adalah proses, cara, perbuatan atau menjadikan orang atau makhluk hidup belajar[1]. Akan tetapi, pada *machine learning*, yang menjadi siswa bukanlah makhluk hidup, tapi mesin.

Definsi sebelumnya mungkin sedikit "abstrak", kita harus mengkonversi definisi tersebut sebagai definsi operasional (bentuk komputasi). Secara operasional, belajar adalah perubahan tingkah laku berdasarkan pengalaman (*event*/data) untuk menjadi lebih baik (mengoptimisasi parameter terhadap ***performance measure/utility function***).

1.4 Statistical Learning Theory

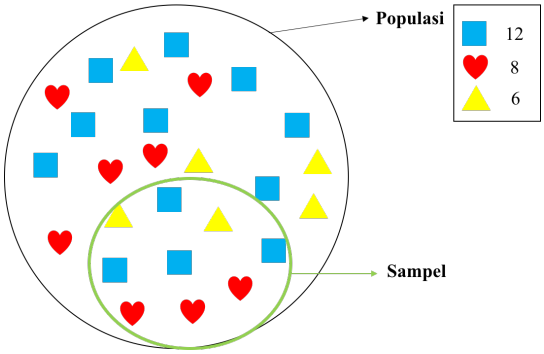
Pada masa sekarang ini data bertebaran sangat banyak dimana-mana. Pemrosesan data secara manual tentu adalah hal yang kurang bijaksana. Beberapa pemrosesan data yang dilakukan seperti kategorisasi (kategorisasi teks berita), peringkasan dokumen, ekstraksi informasi (mencari subjek, objek, dan relasi diatara keduanya pada teks), rekomendasi produk berdasarkan catatan transaksi, dll [7]. Tujuan *machine learning* minimal ada dua: **memprediksi masa depan** (*unobserved event*); dan/atau **memperoleh ilmu pengetahuan** (*knowledge discovery/discovering unknown structure*). Kedua hal ini berkaitan sangat erat. Sebagai contoh, manusia tahu bahwa cara menggunakan pensil dan pulpen sama, walaupun saat kita belum pernah menggunakan pulpen (penulis berasumsi kamu belajar menulis menggunakan pensil). Memprediksi masa depan berarti kita tahu bahwa pulpen adalah alat tulis. *Knowledge discovery* berarti kita tahu bahwa cara menggunakan pulpen dan pensil itu sama (walaupun belum pernah menggunakan pulpen sebelumnya).

Untuk mencapai tujuan tersebut, kita menggunakan data (sampel), kemudian membuat model untuk menggeneralisasi "aturan" atau "pola" data sehingga kita dapat menggunakannya untuk mendapatkan informasi/membuat keputusan [8, 9]. *Statistical learning theory* (yang diaplikasikan pada *machine learning*) adalah teknik untuk memprediksi masa depan dan/atau menyimpulkan/mendapatkan pengetahuan dari data **secara rasional dan nonparanormal**. Hal ini sesuai dengan konsep *intelligent agent*, yaitu bertingkah berdasarkan lingkungan. Dalam hal ini, yang bertindak sebagai lingkungan adalah data. *Performance measure*-nya adalah seberapa akurat prediksi agen tersebut atau seberapa mirip "pola" data yang ditemukan terhadap data asli. Disebut *statistical* karena basis pembelajarannya memanfaatkan banyak teori statistik untuk melakukan inferensi (misal memprediksi *unobserved event*)[8]

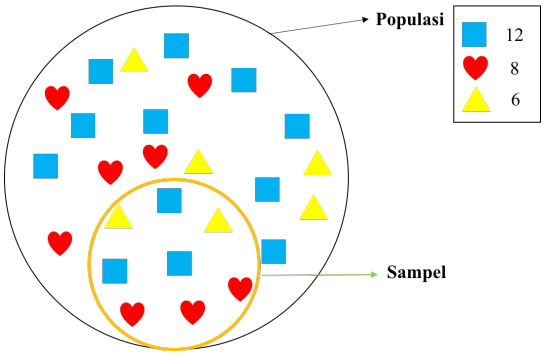
Perhatikan Gambar. 1.3 (permasalahan yang disederhanakan). Misalkan kamu diundang ke suatu pesta. Pada pesta tersebut ada 3 jenis kue yang disajikan. Kamu ingin mengetahui berapa rasio kue yang disajikan dibandingkan

[7] KBBI Web, Accessed on 10 October 2016

[8] Selain itu, *machine learning* juga banyak memanfaatkan teori aljabar linear.



Gambar 1.3. Ilustrasi makanan pesta 1.



Gambar 1.4. Ilustrasi makanan pesta 2.

masing-masing jenisnya (seluruh populasi). Tetapi, karena susah untuk menganalisis seluruh data atau keseluruhan data tidak tersedia, kamu mengambil beberapa sampel. Dari sampel tersebut, kamu mendapati bahwa ada 4 buah kue segi empat, 3 buah kue hati, dan 2 buah kue segitiga. Lalu kamu menyimpulkan (model) bahwa perbandingan kuenya adalah 4:3:2 (segiempat:hati:segitiga). Perbandingan tersebut hampir menyerupai kenyataan seluruh kue yaitu 4:2.67:2. Tentu saja kondisi ini terlalu ideal.

Perhatikan Gambar. 1.4, temanmu Ari datang juga ke pesta yang sama dan ingin melakukan hal yang sama (rasio kue). Kemudian ia mengambil beberapa sampel kue. Dari sampel tersebut ia mendapati bahwa ada 3 buah segiempat, 3 buah hati, dan 2 buah segitiga, sehingga perbandingannya adalah 3:3:2. Tentunya hal ini sangat melenceng dari populasi.

Dari dua sampel yang berbeda, kita menyimpulkan, menginferensi (*infer*) atau mengeneralisasi dengan berbeda. Kesimpulan yang kita buat berdasarkan sampel tersebut, kita anggap merefleksikan populasi, kemudian kita menganggap populasi memiliki aturan/pola seperti kesimpulan yang telah kita ciptakan [10]. Baik pada statistika maupun *statistical machine learning*, pemili-

han sampel (selanjutnya disebut ***training data***) adalah hal yang sangat penting. Apabila *training data* tidak mampu merepresentasikan populasi, maka model yang dihasilkan pembelajaran (*training*) tidak bagus. Untuk itu, biasanya terdapat juga ***development data*** dan ***test data***. Mesin dilatih menggunakan *training data*, kemudian diuji kinerjanya menggunakan *development data*[9] dan *test data*. Seiring dengan membaca buku ini, konsep *training data*, *development data*, dan *test data* akan menjadi lebih jelas.

Seperti halnya contoh sederhana ini, persoalan *machine learning* sesungguhnya menyerupai persoalan *statistical inference* [10]. Kita berusaha mencari tahu populasi dengan cara menyelidiki fitur (*features* atau sifat-sifat) yang dimiliki sampel. Kemudian, menginferensi aksi yang harus dilakukan terhadap *unobserved data* berdasarkan kecocokan fitur-fitur *unobserved data* dengan model/aturan yang sudah ada.

Dari sisi metode pembelajaran, algoritma *machine learning* dapat dikategorikan sebagai: *supervised learning* (subbab 1.6), *semi-supervised learning* (subbab 1.8), *unsupervised learning* (subbab 1.9), dan *reinforcement learning*. Masing-masing metode akan dibahas pada sub bab berikutnya (kecuali *reinforcement learning* diluar cakupan buku ini).

1.5 Training, Development, Testing Set

Terdapat dua istilah penting dalam pembangunan model *machine learning* yaitu: ***training*** dan ***testing***. *Training* adalah proses membangun model dan *testing* adalah proses menguji kinerja model pembelajaran. *Dataset* adalah kumpulan data (sampel dalam statistik). Sampel ini adalah data yang kita gunakan untuk membuat model maupun mengevaluasi model *machine learning*. Umumnya, *dataset* dibagi menjadi tiga jenis yang tidak beririsan (satu sampel pada himpunan tertentu tidak muncul pada himpunan lainnya):

1. ***Training set*** adalah himpunan data yang digunakan untuk melatih atau membangun model. Pada buku ini, istilah *training data(set)* mengacu pada *training set*.
2. ***Development set*** atau ***validation set*** adalah himpunan data yang digunakan untuk mengoptimisasi saat melatih model. Model dilatih menggunakan *training set* dan pada umumnya kinerja **saat latihan** diuji dengan *development set*. Hal ini berguna untuk generalisasi (agar model mampu mengenali pola secara generik). Pada buku ini, istilah *development/validation data(set)* mengacu pada hal yang sama.
3. ***Testing set*** adalah himpunan data yang digunakan untuk menguji model setelah **proses latihan selesai**. Pada buku ini, istilah *testing data(set)* atau *test set* mengacu pada *testing set*. Perlu kami tekankan, *testing set*

[9] Pada umumnya untuk optimisasi parameter (e.g., *stopping criterion*) saat proses *training*.

adalah *unseen data*. Artinya, model dan manusia tidak boleh melihat sampel ini saat proses latihan. Banyak orang yang tergoda untuk melihat *testing set* saat proses latihan walaupun itu adalah tingkah laku yang buruk karena menyebabkan *bias*.

Satu sampel pada himpunan data kita sebut sebagai ***data point*** atau instans (***instance***) yang merepresentasikan suatu kejadian statistik (*event*). Perlu diingat, *training*, *development*, dan *testing data* diambil (*sampled*) dari distribusi yang sama dan memiliki karakteristik yang sama (*independently and identically distributed*).

Pada umumnya, rasio pembagian *dataset* adalah (80% : 10% : 10%) atau (90% : 5% : 5%) (*training*:*development*:*testing*). *Development set* pada umumnya bisa tidak digunakan apabila *dataset* berukuran kecil (hanya dibagi menjadi *training* dan *testing set* saja). Dalam kasus ini, pembagian *dataset* menjadi *training* dan *testing* set pada umumnya memiliki rasio (90% : 10%), (80% : 20%), (70% : 30%), atau (50% : 50%). Pada kasus ini, kinerja saat *training* diuji menggunakan *training set*.

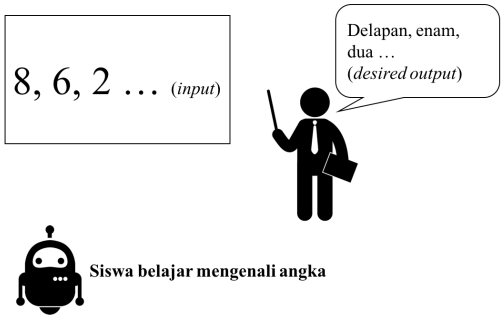
Saat tidak menggunakan *development set* (hanya ada *training* dan *testing set*), kita juga memiliki opsi untuk mengevaluasi model dengan metode K-*cross-validation*[10]. Artinya, kita membagi *dataset* menjadi K bagian. Kita menggunakan $K-1$ bagian untuk *training*, kemudian menguji kinerja model (*testing*) menggunakan satu bagian. Hal ini diulangi sebanyak K kali dimana sebuah bagian data digunakan sebagai *testing set* sebanyak sekali (bergilir). Hal ini akan dijelaskan lebih lanjut pada subbab 3.8.

1.6 Supervised Learning

Jika diterjemahkan secara literal, *supervised learning* adalah pembelajaran terarah/terawasi. Artinya, pada pembelajaran ini, ada guru yang mengajar (mengarahkan) dan siswa yang diajar. Kita disini berperan sebagai guru, kemudian mesin berperan sebagai siswa. Perhatikan Gambar. 1.5 sebagai ilustrasi! Pada Gambar. 1.5, seorang guru menuliskan angka di papan "8, 6, 2" sebagai contoh untuk siswanya, kemudian gurunya memberikan cara membaca yang benar untuk masing-masing angka. Contoh angka melambangkan ***input***, kemudian cara membaca melambangkan ***desired output***. Pasangan ***input-desired output*** ini disebut sebagai *instance* (untuk kasus supervised learning).

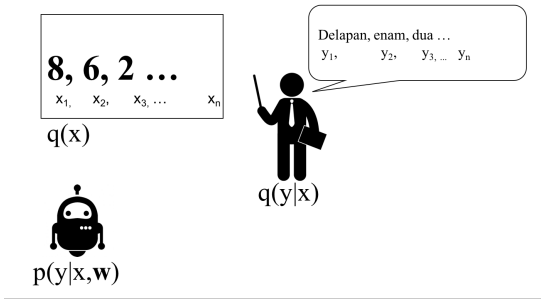
Perhatikan Gambar. 1.6 dan Gambar. 1.7, x adalah kejadian (*event – random variable*), untuk *event* tertentu dapat dinotasikan sebagai $\{x_1, x_2, x_3, ..., x_N\}$. x dapat berupa vektor, teks, gambar, dan lain sebagainya (perhatikan konteks pembahasan buku). Demi pembahasan yang cukup generik, pada bab ini kita membicarakan x yang merepresentasikan *event*, *data point*, atau *instance*. Seorang guru sudah mempunyai jawaban yang benar untuk masing-masing

[10] `https://www.openml.org/a/estimation-procedures/1`



Gambar 1.5. *Supervised Learning.*

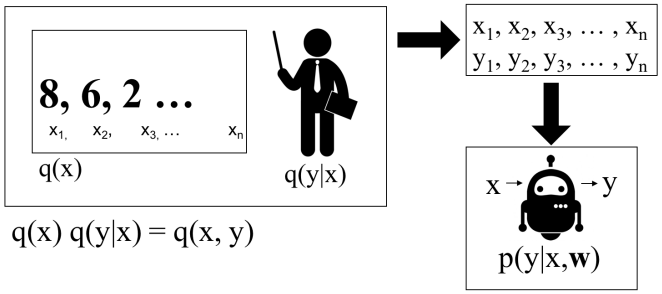
contoh dengan suatu fungsi distribusi probabilitas kondisional (***conditional probability density function***) $q(y \mid x)$ baca: *function q for y given x*, melambangkan hasil yang benar/diharapkan untuk suatu event. Siswa (mesin) **mempelajari tiap pasang pasangan *input-desired output (training data)*** dengan mengoptimalkan *conditional probability density function* $p(y \mid x, \mathbf{w})$, dimana y adalah target (*output*), x adalah input dan vektor $\mathbf{w}$ adalah ***learning parameters***. Proses belajar ini, yaitu mengoptimalkan $\mathbf{w}$ disebut sebagai training. Semakin kamu membaca buku ini, konsep ini akan menjadi semakin jelas.



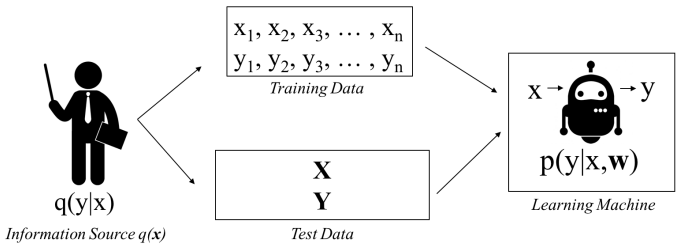
Gambar 1.6. *Supervised learning - mathematical explanation.*

Perhatikan Gambar. 1.8! $q(x)q(y \mid x) = q(x, y)$ memiliki panah ke *training data* dan *test data*, artinya model hasil *training* sangat bergantung pada **data dan guru**. Model yang dihasilkan *training* (hasil pembelajaran kemampuan siswa) untuk data yang sama bisa berbeda untuk guru yang berbeda[11].

[11] Penulis rasa hal ini sangat intuitif berhubung hal serupa terjadi pada manusia.

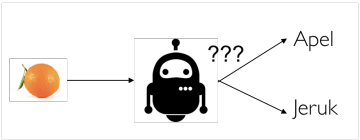


Gambar 1.7. *Supervised learning - mathematical explanation* 2.



Gambar 1.8. *Supervised learning framework.*

Tujuan *supervised learning*, secara umum untuk melakukan klasifikasi (*classification*). Misalkan mengklasifikasikan gambar buah (apa nama buah pada gambar), diilsutrasikan pada Gambar. 1.9. Apabila hanya ada dua kategori, disebut ***binary classification***. Sedangkan bila terdapat lebih dari dua kategori, disebut ***multi-label classification***. Ada tipe klasifikasi lain disebut *soft classification* yaitu klasifikasi menggunakan probabilitas (seperti pada *fuzzy logic*) misalkan suatu berita memuat 30% olah raga dan 70% politik.



Gambar 1.9. Ilustrasi klasifikasi buah.

$$p(y \mid x, \mathbf{w}) \tag{1.1}$$

Pemahaman *supervised learning* adalah mengingat persamaan 1.1. Ada tiga hal penting pada *supervised learning* yaitu *input*, *desired output*, dan *learning parameters*. Perlu ditekankan *learning parameters* berjumlah lebih dari satu, dan sering direpresentasikan dengan vektor (*bold*). Berdasarkan model yang dibuat, kita dapat melakukan klasifikasi (misal simbol yang ditulis di papan adalah angka berapa). Secara konseptual, klasifikasi didefinisikan sebagai persamaan 1.2 yaitu memilih label (kelas/kategori) paling optimal diberikan (*given*) suatu instans data tertentu.

$$y_i^{'} = \arg\max_{y_i \in Y} p(y_i \mid x_i, \mathbf{w}) \tag{1.2}$$

1.7 Regresi

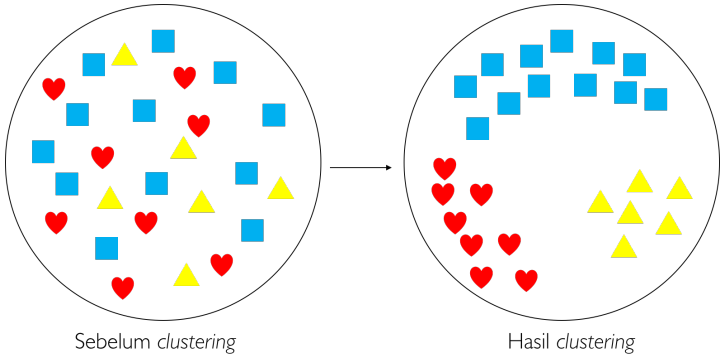
Pada persoalan regresi, kita ingin memprediksi *output* berupa bilangan kontinu. Misalnya pada regresi suatu fungsi polinomial, kita ingin mencari tahu fungsi $f(x)$ dari sampel data $\{x_1, \cdots, x_N\}$. Setelah itu, kita gunakan fungsi aproksimasi untuk mencari tahu nilai data baru x_{N+1}. Perbedaan regresi dan klasifikasi adalah pada tipe *output*. Untuk regresi, tipe *output* adalah nilai kontinu; sementara tipe *output* pada persoalan klasifikasi adalah suatu objek pada himpunan (i.e., memilih opsi pada himpunan jawaban).

1.8 Semi-supervised Learning

Semi-supervised learning mirip dengan *supervised learning*, bedanya pada proses pelabelan data. Pada *supervised learning*, ada "guru" yang harus membuat "kunci jawaban" *input-output*. Sedangkan pada *semi-supervised learning* tidak ada "kunci jawaban" eksplisit yang harus dibuat guru. Kunci jawaban ini dapat diperoleh secara otomatis (misal dari hasil *clustering*). Pada kategori pembelajaran ini, umumnya kita hanya memiliki sedikit data. Kita kemudian menciptakan data tambahan baik menggunakan *supervised* ataupun *unsupervised learning*, kemudian membuat model belajar dari data tambahan tersebut.

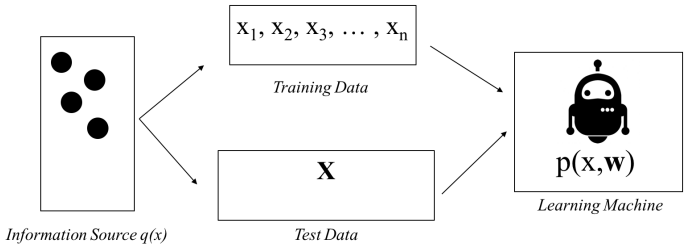
1.9 Unsupervised Learning

Jika pada *supervised learning* ada guru yang mengajar, maka pada *unsupervised learning* tidak ada guru yang mengajar. Contoh permasalahan unsupervised learning adalah *clustering*. Mengingat contoh kue sebelumnya, kita ingin mengelompokkan kue-kue yang sama, diilustrasikan oleh Gambar. 1.10.



Gambar 1.10. Ilustrasi *clustering*.

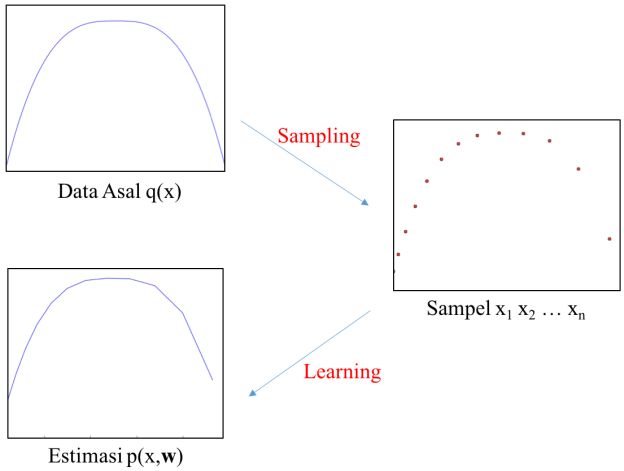
Yang kamu lakukan adalah membuat kelompok-kelompok berdasarkan karakteristik kue, misal kelompok kue biru, kelompok kue kuning, atau kelompok kue merah. Teknik-teknik mengelompokkan ini akan dibahas pada bab-bab berikutnya. Contoh algoritma *unsupervised learning* sederhana adalah *K-means* (bab 4).



Gambar 1.11. *Unsupervised learning framework.*

Perhatikan Gambar. 1.11 dan Gambar. 1.12! Berbeda dengan supervised learning yang memiliki *desired output*, pada *unsupervised learning* tidak ada *desired output* (jelas, tidak ada gurunya, tidak ada yang memberi contoh). Populasi asli mempunyai distribusi $q(x)$, kita ingin mengestimasi $q(x)$ tersebut dengan mengambil beberapa sampel, lalu melakukan *learning*. *Learning* dilakukan dengan mengoptimalkan $p(x \mid \mathbf{w})$ yang mengoptimasi parameter $\mathbf{w}$. Perbedaan antara estimasi dan fungsi asli disebut sebagai ***generalization loss*** (atau ***loss*** saja – dijelaskan pada bab 5). Kunci pemahaman *unsupervised learning* adalah mengingat persamaan 1.3, yaitu ada *input* dan parameter.

$$p(x \mid \mathbf{w}) \tag{1.3}$$



Gambar 1.12. *Generalization error of unsupervised learning.*

Perlu kami tekankan, *unsupervised learning* $\neq$ *clustering*! *Clustering* adalah salah satu bentuk *unsupervised learning*; yaitu salah satu hasil inferensi persamaan 1.3. Seiring membaca buku ini, kamu akan semakin paham.

1.10 Proses Belajar

Seperti yang sudah dijelaskan pada subbab sebelumnya, pada *supervised* maupun *unsupervised learning*, kita ingin mengestimasi sesuatu dengan teknik *machine learning*. Kinerja *learning machine* berubah-ubah sesuai dengan parameter **w** (parameter pembelajaran). Kinerja *learning machine* diukur oleh fungsi tujuan (*utility function/performance measure*), yaitu mengoptimalkan nilai fungsi tertentu; misalnya meminimalkan nilai *error*, atau meminimalkan *loss* (dijelaskan kemudian). Secara intuitif, *learning machine* sama seperti saat manusia belajar. Kita awalnya membuat banyak kesalahan, tetapi kita mengetahui/diberi tahu mana yang benar. Untuk itu kita menyesuaikan diri secara perlahan agar menjadi benar (iteratif). Inilah yang juga dilakukan *learning machine*, yaitu mengubah-ubah parameter **w** untuk mengoptimalkan suatu fungsi tujuan[12].

Secara bahasa lebih matematis, kami beri contoh *supervised learning*. Kita mempunyai distribusi klasifikasi asli $q(y \mid x)$. Dari distribusi tersebut, kita

[12] Saat membaca ini, kamu mungkin akan menganggap bahwa teknik *machine learning* adalah fungsi-parametrik. Sebenarnya, ada juga algoritma *machine learning* non-parametrik. Demi generalisasi pembahasan, kami menggunakan penjelasan secara parametrik karena algoritma parametrik yang lebih umum digunakan secara praktis.

diberikan beberapa sampel pasangan *input-output* $\{z_1, z_2, z_3, \ldots, z_n\}; z_i = (x_i, y_i)$. Kita membuat learning machine $p(y \mid x, \mathbf{w})$. Awalnya diberi (x_1, y_1), *learning machine* mengestimasi fungsi asli dengan mengoptimalkan parameter $\mathbf{w}$ sesuai dengan data yang ada. Seiring berjalannya waktu, ia diberikan data observasi lainnya, sehingga *learning machine* menyesuaikan dirinya (konvergen) terhadap observasi yang baru $(x_2, y_2), (x_3, y_3), \ldots$. Semakin lama, kita jadi makin percaya bahwa *learning machine* semakin optimal (mampu memprediksi fungsi aslinya).

1.11 Tips

Jujur, pengarang sendiri belum menguasai bidang ini secara penuh, tetapi berdasarkan pengalaman pribadi (dan membaca), dan beberapa rekan; ada beberapa materi wajib yang harus dipahami untuk mengerti bidang *machine learning*. Sederhananya, kamu harus menguasai banyak teori matematika dan probabilitas agar dapat mengerti *machine learning* sampai tulang dan jeroannya. Kami tidak menyebutkan bahwa mengerti *machine learning* secara intuitif (atau belajar dengan pendekatan deskriptif) itu buruk, tetapi untuk mengerti sampai dalam memang perlu mengerti matematika (menurut pengalaman kami). Disarankan untuk belajar materi berikut:

1. Matematika Diskrit dan Teori Bilangan
2. Aljabar Linier dan Geometri (vektor, matriks, skalar, dekomposisi, transformasi, tensor, dsb)
3. Kalkulus (diferensial dan integral)
4. Optimasi (*Lagrange multiplier, convex, Gradient Descent, Integer Linear Problem,* dsb)
5. Probabilitas dan Statistika (probabilitas, *probability densities*, *hypothesis testing*, *inter-rater agreement*, Bayesian, *statistical mechanics*)
6. Teori Fuzzy

Mungkin kamu sudah tahu, tetapi penulis ingin mengingatkan ada dua buku yang sangat terkenal ("kitab") sebagai materi belalajar *machine learning* dan *deep learning*:

1. Patten Recognition and Machine Learning, oleh Christopher M. Bishop [8]
2. Deep Learning, oleh Ian Goodfellow, Yoshua Bengio, dan Aaron Courville [11]

Apabila pembaca memiliki kesempatan, penulis sarankan untuk membaca kedua buku tersebut.

1.12 Contoh Aplikasi

Sebenarnya, aplikasi pemanfaatan *machine learning* sudah terasa dalam kehidupan sehari-hari. Contoh mudahnya adalah produk-produk Google, mis-

alnya google translate (machine translation, handwritten recognition, speech recognition, Alpha Go). Berikut adalah beberapa artikel menarik:

1. `techcrunch google AI beats go world champion`
2. `http://www-formal.stanford.edu/jmc/whatisai/node3.html`
3. `https://www.google.com/selfdrivingcar/`
4. `http://www.osnews.com/story/26838/Palm_I_m_ready_to_wallow_now/page2/`

Soal Latihan

1.1. Aplikasi

(a) Carilah contoh-contoh penerapan *machine learning* pada kehidupan sehari-hari selain yang telah disebutkan!
(b) Mengapa mereka menggunakan teknik *machine learning* untuk menyelesaikan permasalahan tersebut?
(c) Apakah tidak ada opsi teknik lainnya? Jelaskan bila ada!
(d) Apa kelebihan dan kekurangan teknik *machine learning* daripada teknik lainnya (yang kamu jelaskan pada soal (c))?

1.2. Kecerdasan

Jelaskan tahapan perkembangan kecerdasan manusia berdasarkan kategori usia! Dari hal ini, kamu akan mengerti kenapa beberapa peneliti membuat agen cerdas berdasarkan kategori usia tertentu.

2

Fondasi Matematis

> "He uses statistics as a drunken man uses lamp posts - for support rather than for illumination."
>
> Andrew Lang

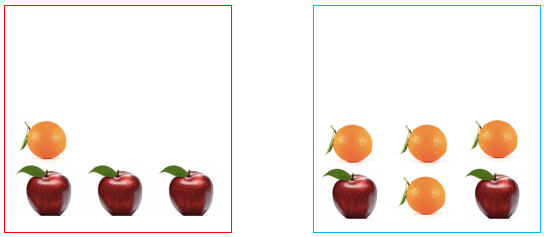
Mungkin saat pertama kali membaca bab ini, kamu merasa bab ini tidak masuk akal/kurang dibutuhkan. Seiring membaca buku ini, mungkin bab ini akan sering dikunjungi kembali. Bab ini hanyalah pengingat materi yang sudah kamu pernah pelajari saja (semacam *cheatsheet*). Kamu boleh melewati bab ini apabila sudah familiar dengan materi probabilitas, statistika, serta aljabar linier.

Bab ini memuat sekilas tentang probabilitas, statistika, dan operasi matriks. Tentunya untuk mengerti materi tersebut sebaiknya kamu mengambil kuliah khusus berkaitan karena kamu diharapkan sudah memiliki "cukup latar pengetahuan", bab ini sebenarnya hanyalah sekilas pengingat. Kami akan banyak memakai contoh-contoh dari buku Bishop [8] untuk bab ini (probabilitas).

2.1 Probabilitas

Di dunia ini, ada banyak hal yang tidak pasti (*uncertain*). Ssungguhnya, *machine learning* berurusan dengan ketidakpastian (*uncertainty*). Dengan hal itu, *machine learning* memiliki kaitan yang sangat erat dengan statistika. Probabilitas menyediakan *framework* untuk kuantifikasi dan manipulasi ketidakpastian [8]. Mari kita lihat contoh sederhana. Terdapat dua buah kotak berwarna merah dan berwarna biru. Pada kotak merah terdapat 3 apel dan 1 jeruk. Pada kotak biru, terdapat 2 apel dan 4 jeruk, kita ingin mengambil buah dari salah satu kotak tersebut. Ilustrasi persoalan dapat dilihat pada

Gambar. 2.1. Dalam hal ini, kotak adalah *random variable*. *Random variable* k (melambangkan kotak) dapat bernilai *merah* atau *biru*. Begitu pula dengan buah, dilambangkan dengan variabel b, dapat bernilai *apel* atau *jeruk*.



Gambar 2.1. Kotak Apel dan Jeruk.

Saat kita mengambil buah dari kotak biru, peluang untuk memilih *apel* bernilai 2/6, sedangkan peluang untuk memilih *jeruk* bernilai 4/6; kita tulis probabilitas ini sebagai $P(f = apel) = 2/6$; dan $P(f = jeruk) = 4/6$. Artinya, jika kita mengambil buah dari kotak biru, kemungkinan lebih banyak kejadian saat kita mendapat jeruk. Nilai suatu probabilitas harus lebih besar sama dengan nol sampai kurang dari atau sama dengan satu ($0 \leq P \leq 1$).

Lalu sekarang ada pertanyaan baru; pada suatu percobaan, berapakah probabilitas mengambil sebuah *apel* dari kotak biru **atau** sebuah *jeruk* dari kotak merah. Hal ini dituliskan sebagai $P((k = biru, b = apel)$ **atau** $(k = merah, b = jeruk))$. Nilai probabilitas tersebut dapat dihitung dengan

$$\begin{aligned} &P((k = biru, b = apel) \vee (k = merah, b = jeruk)) \\ &= P(k = biru, b = apel) + P(k = merah, b = jeruk) \end{aligned} \tag{2.1}$$

- $P(k = biru, b = apel)$ disebut *joint probability*, yaitu probabilitas kejadian yang dipengaruhi oleh beberapa variabel (kondisi untuk kedua variabel terpenuhi).
- $P(k = biru, b = apel) + P(k = merah, b = jeruk)$ disebut **aturan tambah**.

Penting untuk diingat bahwa hasil operasi apapun terhadap probabilitas (baik tambah, kurang, kali, atau bagi) haruslah lebih besar sama dengan nol sampai kurang dari atau sama dengan satu ($0 \leq P \leq 1$).

Misalkan terdapat percobaan lain, kali ini kamu mengambil 1 buah. Kamu ingin mengetahui berapakah probabilitas untuk mengambil buah *apel* kotak mana saja. Hal ini dihitung dengan

$$P(b = apel) = \sum_{i=1}^{I} P(k = k_i, b = apel) \tag{2.2}$$

Aturan tambah seperti ini disebut *marginal probability* karena hasilnya didapat dengan menjumlahkan probabilitas seluruh kemungkinan nilai pada variabel tertentu (buah) dengan mengontrol variabel lainnya (kotak).

Kemudian, kamu ingin melakukan percobaan lain. Kali ini kamu mengambil 2 buah sekaligus dari kedua kotak. Kamu ingin mengetahui berapakah probabilitas mengambil buah *apel* yang berasal dari kotak biru **dan** buah *jeruk* yang berasal dari kotak merah. Dalam kasus ini, kejadiannya adalah saling bebas, artinya mengambil buah dari kotak biru, pada saat yang bersamaan tidak akan mempengaruhi hasil pengambilan kotak merah. Apabila kedua *random variable* b dan f independen (tidak bergantung satu sama lain), maka $P(b = B, f = F) = P(B) \times P(F)$[1]. Permasalahan mengambil buah dapat dihitung dengan

$$\begin{aligned} &P((b = biru, f = apel) \wedge (b = merah, f = jeruk)) \\ &= P(b = biru, f = apel) \times P(b = merah, f = jeruk) \end{aligned} \tag{2.3}$$

Aturan ini disebut aturan kali.

Untuk *joint probability*, secara umum dapat ditulis sebagai $P(x, y)$. Apabila kedua variabel x dan y tidak saling bebas, maka keduanya disebut dependent. Artinya x dan y saling mempengaruhi. Apabila suatu variabel x dikondisikan (*conditioned*) oleh variabel lain (misal y). Maka probabilitas x adalah *conditional probability function*, ditulis $P(x \mid y)$. Artinya probabilitas x yang dikondisikan oleh y. $P(x \mid y)$ dapat dihitung dengan

$$P(x \mid y) = \frac{P(x, y)}{P(y)} \tag{2.4}$$

yaitu peluang kejadian x dan y muncul bersamaan dibagi dengan peluang kejadian y. Apabila x ternyata tidak dikondisikan oleh variabel y, maka $P(x \mid y) = P(x)$.

2.2 Probability Density Function

Probability density function dikenal juga dengan istilah distribusi, yaitu tentang persebaran nilai. Sebagai contoh, penulis menceritakan pelajaran di sekolah. Terdapat ujian mata pelajaran di kelas yang beranggotakan 10 siswa, diberikan pada Tabel. 2.1. Terdapat 3 orang anak mendapatkan nilai 50, 2 orang anak mendapatkan nilai 75 dan 80, 1 orang anak mendapatkan nilai 100, 1 orang anak mendapat nilai 40, serta 1 orang anak mendapatkan nilai 10.

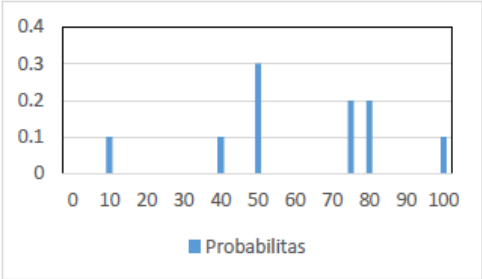
Guru ingin mengetahui persebaran (distribusi) nilai ujian untuk menentukan batas kelas nilai (misal nilai "A" adalah ≥ 85) jadi, ia mencari persebaran nilai siswa. Ia menghitung seberapa mungkin siswa tertentu mendapat

[1] Baca: probabilitas ketika kotak bernilai B dan buah bernilai F.

id	nilai
1	50
2	75
3	80
4	100
5	50
6	50
7	75
8	80
9	40
10	10

Tabel 2.1. Contoh daftar nilai siswa.

nilai tertentu, dapat dilihat pada Gambar. 2.2. Grafik itu disebut sebagai distribusi. Fungsi yang menghasilkan distribusi tersebut disebut *probability density function*. Apabila kita menjumlahkan probabilitas (probabilitas siswa mendapat nilai 0 - 100) nilainya adalah 1.



Gambar 2.2. Persebaran probabilitas nilai siswa.

Ini adalah contoh untuk data diskrit, tetapi sering kali kita berurusan dengan data kontinu. Untuk mengetahui nilai probabilitas dari himpunan *event*/kejadian, kita dapat mengintegralkan kurva distribusi kejadian pada interval tertentu. Ciri *probability density function*, nilai dibawah kurva pada interval $-\infty$ sampai ∞ adalah 1 ($0 \leq p \leq 1; \int_{-\infty}^{\infty} p = 1$).

2.3 Expectation dan Variance

Salah satu operasi paling penting dalam probabilitas adalah menemukan nilai rata-rata terbobot (*weighted average*) sebuah fungsi [8]. Hal ini disebut

menghitung ekspektasi (*expectation*). Untuk sebuah fungsi $f(x)$ dengan distribusi probabilitas *random variable* adalah $p(x)$, nilai expectation diberikan pada persamaan 2.5.

$$E(f) = \begin{cases} \sum_x p(x)f(x); & diskrit \\ \int p(x)f(x)dx; & kontinu \end{cases} \tag{2.5}$$

Dalam kasus nyata, misalkan diberikan N buah sampel, *random variable* x dan fungsi $f(x)$, dimana sampel tersebut diambil dengan distribusi tertentu yang kita tidak ketahui, maka fungsi untuk menghitung nilai *expectation* menjadi

$$E(f) \simeq \frac{1}{N}\sum_{i=1}^{N} f(x_i) \tag{2.6}$$

dimana x_i merepresentasikan data ke-i (*point*). Perhatikan, persamaan tersebut sama dengan persamaan untuk menghitung rata-rata (*mean* atau μ) seperti yang sudah Anda pelajari di SMA. Untuk mengetahui seberapa variasi nilai $f(x)$ di sekitar nilai rata-ratanya, kita menghitungnya mengunakan *variance*, disimbolkan dengan $var(f)$ atau σ^2.

$$\sigma^2 = var(f) = E(f(x) - E(f(x))^2) \tag{2.7}$$

Bila nilai *variance* tinggi, secara umum banyak variabel yang nilainya jauh dari nilai rata-rata. Interpretasi secara "geometris" mata, berarti distribusinya semakin "lebar" seperti pada Gambar. 2.3. Untuk fungsi dengan lebih dari satu variabel, kita menghitung *covariance*. *Covariance* adalah *variance* untuk kombinasi variabel.

2.4 Bayesian Probability

Dalam subbab sebelumnya, kita menghitung probabilitas dengan frekuensi kejadian yang dapat diulang. Pada pandangan Bayesian, kita ingin menguantifikasi ketidakpastian untuk kejadian yang mungkin sulit untuk diulang. Misalkan kita ingin tahu, seberapa peluang Mars dapat dihuni. Ini adalah sesuatu yang tidak dapat dihitung dengan frekuensi, maupun sebuah kejadian yang dapat diulangi (pergi ke mars, lihat berapa orang yang hidup). Akan tetapi, tentunya kita memiliki sebuah asumsi awal (*prior*). Dengan sebuah alat canggih baru, kita dapat mengumpulkan data baru tentang Mars. Dengan data tersebut, kita mengoreksi pendapat kita tentang Mars (*posterior*). Hal ini menyebabkan perubahan dalam pengambilan keputusan.

Pada keadaan ini, kita ingin mampu menguantifikasi ekspresi ketidakpastian; dan membuat revisi tentang ketidakpastian menggunakan bukti baru [8]. Dalam Bayesian, nilai probabilitas digunakan untuk merepresentasikan derajat kepercayaan/ketidakpastian.

$$P(x \mid y) = \frac{P(y \mid x)P(x)}{P(y)} \tag{2.8}$$

$P(x)$ disebut *prior*, yaitu pengetahuan/asumsi awal kita. Setelah kita mengobservasi fakta baru y (dapat berupa sekumpulan data atau satu *data point/event*), kita mengubah asumsi kita. $P(y \mid x)$ disebut ***likelihood function***. *Likelihood function* mendeskripsikan peluang data, untuk asumsi/ pengetahuan tentang x yang berubah-ubah (x sebagai parameter yang dapat diatur). Dengan *likelihood function* tersebut, kita mengoreksi pendapat akhir kita yang dapat digunakan untuk mengambil keputusan (*posterior*). Secara umum probabilitas Bayesian mengubah *prior* menjadi *posterior* akibat adanya kepercayaan baru (*likelihood*).

$$posterior \propto likelihood * prior \tag{2.9}$$

Teori ini hebat karena kita dapat mentransformasi $P(x \mid y)$ dimana x dependen terhadap y menjadi bentuk $P(y \mid x)$ yang mana y dependen terhadap x. Transformasi ini sangat berguna pada berbagai macam persoalan.

Pada umumnya, untuk mengestimasi *likelihood*, digunakan *maximum likelihood estimator*; yang berarti mengatur nilai x untuk memaksimalkan nilai $P(y \mid x)$. Dalam literatur *machine learning*, banyak menggunakan *negative log of likelihood function* [8]. Ingat kembali nilai probabilitas $0 \leq P \leq 1$. Kadangkala, nilai dibelakang koma (0.xxxx) sangatlah panjang, sehingga dapat terjadi $underflow$ pada komputer. Kita menggunakan nilai logaritma probabilitas untuk menghindari $underflow$. Nilai probabilitas $0 \leq P \leq 1$ membuat nilai logaritmanya negatif, secara monotonik menurun, maka memaksimalkan nilai *likelihood* ekuivalen dengan meminimalkan negatif logaritma probabilitas (contoh nyata akan diberikan pada subbab 2.5).

Perhatikan kembali persamaan 2.8, secara intuitif, *posterior* dipengaruhi *prior*, artinya bergantung pada sampel yang kita punya, karena *prior* didapatkan/disimpulkan berdasarkan sampel. Hal ini berlaku pada *machine learning*, kualitas model yang dihasilkan bergantung pada kualitas training data.

Pada umumnya, kita tidak mengetahui seluruh informasi tentang situasi tertentu dan tidak mengetahui seluruh informasi probabilitas. Sebagai contoh, probabilitas $P(x \mid y)$ dapat dihitung dengan $P(x, y)/P(x)$. Tetapi, kita tidak tahu seberapa banyak kejadian (x, y) pada saat bersamaan. Oleh sebab itu, kita bisa menggunakan teori bayes untuk menghitung probabilitas dengan informasi lain yang kita tahu.

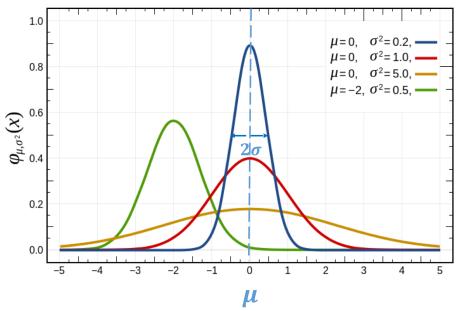
2.5 Gaussian Distribution

Distribusi adalah fenomena acak atau deskripsi matematis suatu *random variable*. Anda harusnya sudah mengetahui distribusi ini. Ini adalah distribusi yang sangat terkenal yaitu *bell curve*/distribusi normal. Distribusi normal

adalah bentuk khusus dari *Gaussian distribution*. Ada beberapa macam distribusi yang akan dibahas pada bab ini, yaitu: *Univariate Gaussian*, *Multivariate Gaussian*, dan *Gaussian Mixture Model*. Pertama kita bahas ***Univariate Gaussian*** terlebih dahulu.

Disebut *univariate* karena distribusinya bergantung pada **satu** input variabel, misalkan x. Distribusi univariate Gaussian dikarakteristikkan oleh variabel x, *mean* (μ) dan *variance* (σ^2) diberikan pada persamaan 2.10. μ dan σ^2 adalah rata-rata dan *variance* untuk kumpulan data. Karena nilai μ dan σ^2 bergantung pada x, maka kita dapat menyebutkan dengan aman bahwa *univariate gaussian* bergantung pada satu variabel saja yaitu x.

$$N(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{2.10}$$



Gambar 2.3. Univariate Gaussian [2].

Perhatikan Gambar. 2.3, x adalah absis dan nilai N untuk x tertentu (persamaan 2.10) adalah ordinat pada kurva ini. Bentuk distribusi berubah-ubah sesuai dengan nilai rata-rata (*mean*), serta *variance*. Semakin besar *variance*-nya, maka kurva distribusi semakin lebar (seperti yang dijelaskan sebelumnya). Untuk menggeser-geser kurva ke kiri maupun ke kanan, dapat dilakukan dengan menggeser nilai mean. Untuk mencari nilai pada suatu interval tertentu, cukup mengintegralkan fungsi pada interval tersebut. Nilai integral fungsi dari $-\infty$, hingga ∞ adalah satu.

Sekarang bayangkan kita diberikan N buah data hasil observasi. Diasumsikan observasi dihasilkan oleh distribusi univariate Gaussian dengan rata-rata μ dan *variance* σ^2. Setiap data diambil secara independen dari distribusi yang sama, disebut ***independent and identically distributed*** (iid). Kita tahu bahwa data yang independen, apabila dihitung probabilitasnya maka tersusun atas probabilitas masing-masing data, seperti pada persamaan 2.11.

[2] source: `wikimedia.org by Inductiveload`

$$p(x|\mu,\sigma^2) = \prod_{i=1}^{N} N(x_i|\mu,\sigma^2) \tag{2.11}$$

Kita ingin mencari tahu bagaimana distribusi yang sebenarnya. Untuk itu, kita mengoptimalkan fungsi *likelihood* agar *prior* berubah menjadi *posterior* (distribusi yang sebenarnya). Tetapi hal ini sulit dilakukan, bahkan sebaliknya kita memaksimalkan *log likelihood function* berdasarkan data yang kita miliki. Logaritma secara monotonik akan bertambah nilainya. Memaksimalkan fungsi logaritma sebanding dengan meminimalkan *error*, hal ini diberikan pada persamaan 2.12.

$$ln(p(x|\mu,\sigma^2)) = -\frac{1}{2\sigma^2}\sum_{i=1}^{N}(x_i-\mu)^2 - \frac{N}{2}ln(\sigma^2) - \frac{N}{2}ln(2\pi) \tag{2.12}$$

Untuk menyederhanakan pembahasan, solusi 2.12 diberikan langsung pada 2.13[3].

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i; \qquad \sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i-\mu)^2 \tag{2.13}$$

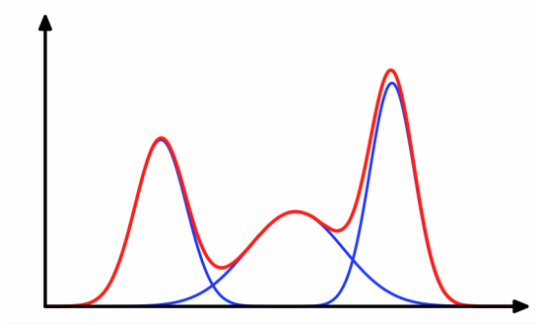
Dibanding langkah-langkah penurunannya, interpretasi berikut lebih penting! Arti persamaan 2.13 adalah kita dapat **mengestimasi distribusi populasi menggunakan sampel data yang kita miliki**. Mean distribusi populasi diestimasi dengan mean sampel. *Variance* distribusi populasi diestimasi dengan *variance* sampel. Inilah jantung *machine learning*! Masih ingat materi bab 1? Pada *machine learning*, kita mengestimasi sesuatu yang kita tidak ketahui dengan sampel data yang kita miliki. Dengan kata lain, kita menggunakan informasi probabilitas data yang kita ketahui untuk mengestimasi kejadian yang belum pernah kita temui sebelumnya. Proses estimasi akan dibahas lebih lanjut pada bab-bab lainnya.

Multivariate Gaussian adalah distribusi gaussian yang bergantung pada lebih dari satu variabel. Sedangkan *Gaussian Mixture Model* (GMM) adalah gabungan dari satu atau lebih distribusi Gaussian. Masing-masing distribusi Gaussian memiliki bobot yang berbeda di GMM. Konon katanya, GMM dapat memodelkan fungsi apapun [12]. Ilustrasinya diberikan pada Gambar. 2.4 yang tersusun dari 3 buah *Univariate gaussian*. Distribusi populasi berwarna merah, sedangkan GMM berwarna biru.

2.6 Teori Keputusan

Diberikan himpunan pasangan data *input-output* $(x_i, y_i); x = input, y = output/target$; walaupun tidak pasti, kita ingin mengestimasi hubungan antara *input* dan *output*. Untuk itu kita melakukan estimasi $p(y \mid x, \mathbf{w})$, dimana

[3] Kamu dapat mencoba menurunkannya sebagai latihan!
[4] `http://dirichletprocess.weebly.com/clustering.html`



Gambar 2.4. Gaussian Mixture Model [4].

w adalah *learning parameters*. Pada bab pertama, kamu telah mempelajari bahwa kita mampu melakukan hal ini dengan teknik *machine learning*. Lebih jauh lagi, kita juga harus mampu untuk membuat keputusan berbasiskan perkiraan nilai y, aspek ini disebut *decision theory* [8].

Dalam *machine learning* kita dapat membangun model dengan tujuan untuk meminimalkan *error* atau meminimalkan *loss*; konsep meminimalkan *error* dijelaskan pada materi *curve fitting* (bab 5). Ibaratnya untuk sebuah robot, kita ingin robot tersebut tidak melakukan tindakan yang salah. Tetapi, kadang kala meminimalkan *error* belum tentu membuat model menjadi "bagus". Kami ilustrasikan menggunakan contoh dari Bishop [8]. Misalkan kita diminta untuk membuat model klasifikasi kanker. Kita dapat mengklasifikasikan pasien menjadi dua kelas $\{kanker, normal\}$.

Apabila kita ingin meminimalkan *error* saja maka kita ingin mengklasifikasikan secara tepat orang yang kanker dianggap memiliki kanker dan yang tidak dianggap sebagai tidak. Akan tetapi, terdapat *tradeoff* yang berbeda saat salah klasifikasi. Apabila kita mengklasifikasikan orang yang normal sebagai kanker, konsekuensi yang mungkin adalah membuat pasien menjadi stres atau perlu melakukan pemeriksaan ulang. Tetapi bayangkan, apabila kita mengklasifikasikan orang kanker sebagai normal, konsekuensinya adalah penanganan medis yang salah. Kedua kasus ini memiliki beban yang berbeda. Secara formal, kasus ini disebut *loss*. Secara sederhana, kesalahan klasifikasi memiliki bobot berbeda untuk tiap kelasnya. Pada buku ini, kita anggap kesalahan klasifikasi memiliki penalti yang sama. Dengan demikian, *loss* dan *error* akan mengacu pada hal yang sama di bab-bab berikutnya. Demi istilah yang lebih generik, *loss* akan lebih sering digunakan.

Fungsi tujuan pembelajaran (secara umum untuk merepresentasikan *error* atau *loss*) dalam *utility function*. Sekali lagi kami tekankan, tujuan *machine learning* adalah memaksimalkan kinerja. Kinerja diukur berdasarkan *utility function*. ***Loss* adalah ukuran seberapa dekat/berbeda model yang dihasilkan dengan konsep asli**, sementara *error* adalah salah satu cara untuk mengukur *loss*.

Untuk mengukur nilai *loss*; dapat diekspresikan dengan *loss function*. Secara umum, ada dua macam *loss*, yaitu ***generalization loss/error*** dan ***training loss/error***. *Generalization loss/error* adalah ukuran sejauh mana algoritma mampu memprediksi *unobserved data* dengan tepat, karena kita hanya membangun model dengan data yang terbatas, tentunya bisa saja terdapat ketidakcocokan dengan data yang asli. Sedangkan *training loss/error* seperti namanya, ukuran *loss* saat *training*. Misalkan $q(x)$ adalah distribusi data asli. Menggunakan sampel data dengan distribusi $p(x)$, *generalization loss* dan *training loss* dihitung dengan persamaan 2.14 dan persamaan 2.15.

$$G(q,p) = \int q(x)log(p(x))dx \tag{2.14}$$

$$T(p) = \frac{1}{N}\sum_{i=1}^{N} log(p(x)) \tag{2.15}$$

Tentunya sekarang kamu bertanya-tanya. Kita tidak mengetahui bagaimana $q(x)$ aslinya, bagaimana cara menghitung *generalization loss*? Nah, untuk itulah ada teknik-teknik pendekatan distribusi populasi $q(x)$, misalnya **maximum likelihood method, maximum posterior method** dan *Bayesian method* (silahkan dieksplorasi).

Secara lebih filosofis, berkaitan dengan meminimalkan *loss*; tugas *machine learning* adalah untuk menemukan struktur tersembunyi (*discovering hidden structure*). Hal ini sangat erat kaitannya dengan *knowledge discovery* dan *data mining*. Bila Anda membuka forum di internet, kebanyakan akan membahas perihal *learning machine* yang memaksimalkan akurasi (meminimalkan *error*). Selain harus memaksimalkan akurasi (meminimalkan salah *assignment*), kita juga harus mampu membuat model yang cukup generik. Artinya tidak hanya memiliki kinerja tinggi pada *training data*, tapi juga mampu memiliki kinerja yang baik untuk *unseen data*. Hal ini dapat tercapai apabila model yang dihasilkan melakukan inferensi yang mirip dengan inferensi sebenarnya (konsep asli). Kami tekankan kembali, meminimalkan *loss* adalah hal yang lebih penting, dimana meminimalkan *error* dapat digunakan sebagai sebuah *proxy* untuk mengestimasi *loss*[5].

2.7 Teori Informasi

Kami tidak akan membahas bagian ini terlalu detail, jika kamu membaca buku, topik ini sendiri bisa mencapai satu buku [13]. Mudah-mudahan bab ini dapat memberikan gambaran (serius, ini sekedar gambaran!). *Information Theory*/Teori Informasi menjawab dua pertanyaan fundamental, pertama: bagaimana cara kompresi data terbaik (jawab: *entropy*); kedua: apakah cara

[5] Untuk banyak kasus.

transmisi komunikasi terbaik (jawab: channel capacity) [13]. Dalam *statistical learning theory*, fokus utama adalah menjawab pertanyaan pertama, yaitu bagaimana melakukan kompresi informasi. Contoh aplikasi *entropy* adalah *decision tree learning*.

Pada *machine learning*, kita ingin fitur pembelajaran yang digunakan mampu melambangkan *information source properties*. Artinya, kita ingin memilih fitur yang memuat informasi terbanyak (relatif terhadap *information source*). Karena hal tersebut, mengerti entropy menjadi penting. Ada sebuah strategi pemilihan fitur (*feature selection*) dengan membangun *decision tree*. Awalnya kita bentuk *training data* dengan semua kemungkinan fitur, kemudian mengambil beberapa fitur yang dekat dengan *root*. Hal tersebut dimaksudkan untuk mencari fitur yang memuat banyak informasi. Kemudian, fitur tersebut dapat dicoba pada algoritma learning lainnya. Detil akan dijelaskan pada bab yang memuat *decision tree*.

2.7.1 Entropy

Diberikan sebuah random variabel x, kita ingin mengetahui seberapa banyak informasi yang kita dapatkan ketika kita mengobservasi sebuah nilai spesifik x_i. Kuantitas informasi yang kita dapatkan bisa dipandang sebagai "*degree of surprise*" [8]. Misalkan kita mengetahui seorang teman A sering makan es krim. Suatu ketika kita diberitahu bahwa dia sedang makan es krim, tentu kita tidak heran lagi karena hal tersebut sudah lumrah. Tetapi, apabila kita diberitahu bahwa teman A tidak memakan es krim yang diberikan teman B (padahal kita tahu dia suka), maka akan ada efek "kaget". Kasus kedua memuat lebih banyak informasi karena suatu kejadian yang seharusnya tidak mungkin, terjadi. Hal ini dikuantifikasi dengan persamaan ***Shannon Entropy*** 2.16.

$$S(x) = -\sum_{i=1}^{N} p(x_i) log(p(x_i)) \tag{2.16}$$

Mari kita ambil contoh dari Bishop [8]. Misalkan sebuah *random variable* x memiliki 8 kemungkinan kejadian yang kemungkinannya sama (yaitu $\frac{1}{8}$). *Entropy* untuk kasus ini adalah (log dalam basis 2) diberikan oleh

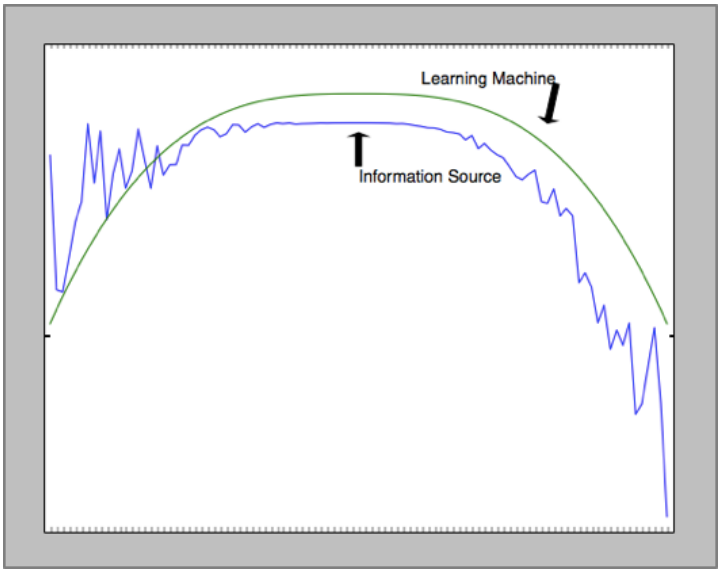
$$S = -8\frac{1}{8} log(\frac{1}{8}) = 3 \tag{2.17}$$

Sekarang mari kita ambil contoh dari [13]. Misalkan sebuah *random variable* x memiliki 8 kemungkinan kejadian $\{a, b, c, d, ..., h\}$ dengan peluang $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{64}, \frac{1}{64}, \frac{1}{64}, \frac{1}{64}$
Maka *entropy*-nya adalah 2. Dari contoh ini, kita tahu bahwa distribusi yang uniform memiliki *entropy* yang lebih besar dibanding distribusi yang tidak uniform. Seperti yang telah diceritakan sebelumnya, *event* yang memiliki "efek kaget" memiliki banyak informasi. Dari sisi *information transmission*, dapat

diinterpretasikan kita dapat mengirimkan data sebuah distribusi dengan jumlah bit lebih sedikit untuk distribusi yang uniform. Distribusi yang memberikan nilai *entropy* maksimal adalah distribusi Gaussian [8]. Nilai *entropy* bertambah seiring *variance* distribusi bertambah. Dari sisi fisika, Anda dapat mempelajari *entropy* pada *statistical mechanics* (*microstate*, *macrostate*).

2.7.2 Relative Entropy dan Mutual Information

Kami harap Anda masih ingat materi bab 1, karena materi bagian ini juga menyinggung kembali materi tersebut. Misalkan kita mempunyai data dengan *probability density function* $q(x)$. Sebuah *learning machine* mengaproksimasi data tersebut dengan *probability density function* $p(x)$. Ingat! *Machine learning* adalah pendekatan (*approximation*). Ketika kita melakukan aproksimasi, seringkali aproksimasi yang dilakukan tidaklah tepat seperti pada Gambar. 2.5.



Gambar 2.5. Information source vs learning machine.

Tentunya kita ingin tahu seberapa bagus aproksimasi kita, untuk mengukurnya terdapat sebuah perhitungan yang bernama **Kullback-Leibler Divergence** (KL-divergence). Secara konseptual, dirumuskan sebagai persamaan 2.18. Perlu diperhatikan $KL(q||p) \neq KL(p||q)$ (kecuali $p = q$).

$$KL(q||p) = -\int q(x) log \left(\frac{q(x)}{p(x)} \right) dx \tag{2.18}$$

Persamaan 2.18 dapat diminimalkan jika dan hanya jika $q(x) = p(x)$. Kita dapat menganggap KL-divergence sebagai ukuran seberapa jauh aproksimasi dan distribusi populasi. Akan tetapi, kita tidak mengetahui $q(x)$. Karena itu, kita harus mengaproksimasi KL-divergence. Misalkan kita diberikan *training data* $X = \{x_1, x_2, ...x_n\}$ yang kita asumsikan diambil (*drawn*) dari suatu distribusi $q(x)$. Lalu kita membuat *learning machine* $p(x \mid \mathbf{w})$. Ekspektasi terhadap $q(x)$ dapat diaproksimasi dengan menggunakan data sampel ini, sehingga menjadi persamaan 2.19 [8].

$$KL(q||p) \approx \frac{1}{N} \sum_{i=1}^{N} (-log(p(x_i \mid \mathbf{w}) + log(q(x_i))) \tag{2.19}$$

KL-divergence disebut juga sebagai *relative entropy*[6]. Dari sisi pemrosesan informasi, KL-divergence dapat diinterpretasikan sebagai berapa informasi tambahan rata-rata untuk mengirimkan data distribusi dengan menggunakan fungsi aproksimasi dibanding menggunakan distribusi sebenarnya, seberapa pengurangan ketidakyakinan terhadap *posterior* seiring diberikannya data observasi yang baru. Dengan kata lain, **seiring diberikan observasi yang baru, kita semakin yakin terhadap nilai posterior** (semakin banyak jumlah sampel yang kita miliki maka model lebih dapat dipercaya).

2.8 Matriks

Subbab ini adalah pengingat untuk operasi perjumlahan, pengurangan, perkalian, dan transpose matriks karena banyak digunakan di buku ini. Diberikan dua buah matriks $\mathbf{U}$ dan $\mathbf{V}$. $\mathbf{U}$ dan $\mathbf{V}$ dapat dijumlahkan jika dan hanya jika dimensi kedua matriks itu sama. Perjumlahan matriks dinotasikan dengan $\mathbf{U} + \mathbf{V} = \mathbf{C}$. Matriks $\mathbf{C}$ memiliki dimensi yang sama dengan $\mathbf{U}$ dan $\mathbf{V}$. Nilai elemen baris ke-i dan kolom ke-j ($\mathbf{C}_{i,j}$) dihitung sebagai penjumlahan nilai elemen matriks $\mathbf{U}$ dan $\mathbf{V}$ pada baris dan kolom yang bersesuaian, seperti diilustrasikan pada persamaan 2.20. Pengurangan dua buah matriks dilakukan serupa.

$$\mathbf{C}_{i,j} = \mathbf{U}_{i,j} + \mathbf{V}_{i,j} \tag{2.20}$$

Dua buah matriks $\mathbf{U}$ dan $\mathbf{V}$ dapat dikalikan jika $\mathbf{U}$ memiliki kolom sebanyak baris pada $\mathbf{V}$. Misalkan matriks $\mathbf{U}$ berdimensi $N \times M$ dan $\mathbf{V}$ berdimensi $M \times O$, maka kedua matriks tersebut dapat dikalikan dan menghasilkan matriks $\mathbf{C}$ dengan dimensi $N \times O$ (dimensi baris $\mathbf{U}$ dan kolom $\mathbf{V}$), dimana

[6] kamu dapat mencoba library entropy di scipy (python) untuk mendapat gambaran lebih detil
`https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.entropy.html`

tiap elemen pada matriks $\mathbf{C}$ dihitung dengan persamaan 2.21 (operasi antara vektor baris dan vektor kolom).

$$\mathbf{C}_{x,y} = \sum_{i=1}^{M} \mathbf{U}_{x,i} + \mathbf{V}_{i,y} \tag{2.21}$$

Selain perkalian antar dua buah matriks, sebuah matriks juga dapat dikalikan dengan skalar, dinotasikan dengan $a\mathbf{U}$. Hasil perkalian adalah sebuah matriks dengan dimensi yang sama dengan $\mathbf{U}$, dimana tiap elemen dikalikan dengan nilai skalar.

$$(a\mathbf{U})_{i,j} = a \times \mathbf{U}_{i,j} \tag{2.22}$$

Suatu matriks $\mathbf{U}$ berdimensi $N \times M$ apabila di transpose menghasilkan matriks $\mathbf{U}^T$ berdimensi $M \times N$, dimana elemen ke-i, j pada matriks $\mathbf{U}^T$ adalah elemen ke-j, i pada matriks $\mathbf{U}$, seperti diilustraiskan pada persamaan 2.21.

$$\mathbf{U}^T_{i,j} = \mathbf{U}_{j,i} \tag{2.23}$$

Ada satu istilah lagi yang perlu kamu ketahui yaitu **tensor**. Tensor adalah generalisasi untuk vektor (1 dimensi) dan matriks (2 dimensi) yang memiliki N dimensi. Tensor sering digunakan untuk notasi pada *artificial neural network*. Tetapi demi kemudahan pengertian, penulis menggunakan notasi matriks.

2.9 Bacaan Lanjutan

Untuk lebih mengerti, silahkan membaca buku *statistical mechanis* oleh Hitoshi Nishimori [14], buku probabilitas dan statistika oleh Walpole et al. [15] atau Brian Caffo [10], dan buku aljabar linear oleh Gilbert Strang [16].

Soal Latihan

2.1. KL-divergence
Cari tahu lebih lanjut apa itu Kullback-Leibler (KL) Divergence. Apa hubungan KL-divergence dengan *utility function*? Pada kasus apa saja kita dapat menggunakan KL-divergence sebagai *utility function*?

2.2. Utility Function
Selain *utility function* yang telah disebutkan, sebutkan dan jelaskan *utility function* lainnya!

2.3. Gaussian Mixture Model

(a) Sebutkan algoritma-algoritma *machine learning* yang (*in a sense*) bisa mengaproksimasi *Gaussian Mixture Model*!
(b) Apa yang begitu spesial pada GMM sehingga algoritma *machine learing* mencoba mengaproksimasi GMM?

3

Data Analytics

> "Hiding within those mounds of data is knowledge that could change the life of a patient, or change the world"
>
> Atul Butte

Bab ini memuat penjelasan tahapan-tahapan umum untuk analisis data, serta beberapa karakteristik tipe data. Materi pada bab ini dapat dianggap sebagai kerangka berpikir (*framework*) atau langkah kerja. Karena buku ini hanya bersifat pengantar, materi yang disampaikan mungkin kurang lengkap. Penulis menyarankan pembaca untuk membaca buku oleh Witten et al. [17] dan Jeff Leek [18].

3.1 Pengenalan Data Analytics

Secara umum, subbab ini adalah ringkasan dari buku Jeff Leek [18]. Untuk detailnya, kamu dapat membaca buku tersebut secara langsung. Penulis merekomendasikan buku tersebut karena ringkas dan mudah dipahami bahkan oleh pemula.

Kita tahu di dunia ini ada banyak masalah. Masalah adalah ketika tujuan yang diinginkan tidak tercapai (*current state* bukanlah *desired state*). Agar *current state* menjadi *desired state*, kita melakukan kegiatan yang disebut penyelesaian masalah (*problem solving*). Tiap bidang (domain) mendefinisikan permasalahan secara berbeda. Oleh karena itu, mengetahui teknik *machine learning* tanpa mengetahui domain aplikasi adalah sesuatu yang kurang baik (semacam buta). Kamu memiliki ilmu, tetapi tidak tahu ilmunya mau digunakan untuk apa. Contohnya, bidang keilmuan pemrosesan bahasa alami (*natural language processing*) menggunakan *machine learning* untuk mengklasifikasikan teks; bidang keilmuan pemrosesan suara menggunakan *machine*

learning untuk mentranskripsikan suara manusia menjadi teks. Tiap bidang merepresentasikan permasalahan ke dalam formulasi yang berbeda. Bisa jadi bentuk komputasi (representasi) pada bidang satu berbeda dengan bidang lainnya. Hal ini perlu kamu ingat karena interpretasi representasi sangat bergantung pada konteks permasalahan (domain). Buku ini adalah pengenalan tentang teknik yang bersifat umum.

Seperti yang sudah dijelaskan pada bab-bab sebelumnya. *Machine learning* adalah inferensi berdasarkan data. *Raw data* atau data mentah adalah sekumpulan fakta (*record*, *event*) yang kemungkinan besar tidak memberikan penjelasan apapun. Sama halnya dengan kebanyakan data di dunia nyata, *raw data* bersifat tidak rapih, misalnya mengandung *missing value* atau ada data yang tidak memiliki label padahal data lainnya memiliki label (ingat kembali materi bab 1). Agar mampu menganalisis *raw data* menggunakan teknik *machine learning*, pertama-tama kita harus merapikan data sesuai dengan format yang kita inginkan (*dataset*). Setelah itu, kita menggunakan teknik-teknik yang ada untuk menemukan pola-pola yang ada di data. Dalam komunitas peneliti basis data, dipercaya bahwa data memiliki sangat banyak relasi yang mungkin tidak bisa dihitung. Teknik *machine learning* hanya mampu mengeksplorasi sebagian relasi yang banyak itu. Lalu, kita analisis informasi yang kita dapatkan menjadi pengetahuan yang digunakan untuk memecahkan permasalahan atau membuat keputusan.

Setelah kita menganalisis data dan mendapatkan pengetahuan baru, pengetahuan yang kita temukan dari data pada umumnya dipresentasikan (konferensi, rapat, dsb). Hal umum yang dipaparkan saat presentasi, yaitu:

1. *Performance measure*. Seberapa "bagus" model/ metode yang kamu buat/ ajukan, dibandingkan menggunakan teknik-teknik yang sudah ada. *Performance measure* biasa disajikan dalam bentuk tabel. Perhatikan, mengatakan model/metode kamu "lebih bagus" adalah suatu hal subjektif, untuk itu gunakanlah metode kuantitatif, seperti *p-value* dalam statistik (*hypothesis testing*[1]) untuk mengatakan bahwa memang metode kamu lebih baik dengan peluang kejadian metode kamu memiliki kinerja lebih baik/buruk sebesar p.

2. Tren. Bagaimana pola-pola umum yang ada di data, sesuai dengan tujuan analisis (permasalahan). Biasa disajikan dalam bentuk teks, kurva, atau grafik.

3. *Outlier*. Sajikan data-data yang "jarang" atau tidak sesuai dengan tren yang ada. Apa beda sifat data *outlier* ini dibanding data pada tren? Kamu harus menganalisis hal ini untuk meningkatkan *performance measure* pada penelitian/analisis mendatang. Apakah *outlier* ini penting untuk diurus atau bisa dipandang sebelah mata tanpa membahayakan keputusan/sistem? Tidak jarang kamu mendapat inspirasi untuk meningkatkan

[1] `https://onlinecourses.science.psu.edu/statprogram/node/138`

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 3.1. Contoh dataset *play tennis* (UCI machine learning repository).

kinerja sistem setelah menganalisis *outlier*.

Langkah kerja yang dijelaskan ini adalah pekerjaan rutin *data scientist*. Penulis ingin menekankan sekali lagi, bahwa memahami *machine learning* saja tidak cukup, **kamu harus memahami domain permasalahan** agar mampu melakukan analisis dengan tepat. Terdapat banyak hal yang hanya mampu kamu pahami dari menganalisis data, apabila kamu mengerti domain aplikasi.

3.2 Nilai Atribut dan Transformasi

Perhatikan Tabel. 3.1 yang merupakan contoh *dataset* pada *machine learning*. **Dataset** adalah kumpulan data. Seorang anak ingin bermain tenis, tetapi keputusannya untuk bermain tenis (*play*) tergantung pada empat variabel {*outlook, temperature, humidity, windy*}. Keempat variabel ini disebut **fitur**. Setiap fitur memiliki **atribut** nilai dengan **tipe data** dan **range** tertentu. Keputusan untuk bermain (*play*) disebut sebagai label atau kelas (*class*). Pada bab 1 kamu telah diperkenalkan *supervised learning* dan *unsupervised learning*. Pada *supervised learning*, kita ingin mengklasifikasikan apakah seorang anak akan bermain atau tidak, diberikan fitur-fitur yang memuat kondisi observasi. Pada *unsupervised learning*, informasi kolom *play* tidak diberikan, kita harus mengelompokkan data tersebut sesuai dengan fitur-fiturnya (contoh lebih nyata diberikan pada bab 4).

Dari segi data statistik, terdapat beberapa tipe atribut [19]:

1. **Nominal**. Nilai atribut bertipe nominal tersusun atas simbol-simbol yang berbeda, yaitu suatu himpunan terbatas. Sebagai contoh, fitur *outlook*

pada Tabel. 3.1 memiliki tipe data **nominal** yaitu nilainya tersusun oleh himpunan {*sunny, overcast, rainy*}. Pada tipe nominal, tidak ada urutan ataupun jarak antar atribut. Tipe ini sering juga disebut **kategorial** atau **enumerasi**. Secara umum, tipe *output* pada *supervised learning* adalah data nominal.

2. **Ordinal**. Nilai ordinal memiliki urutan, sebagai contoh $4 > 2 > 1$. Tetapi jarak antar suatu tipe dan nilai lainnya tidak harus selalu sama, seperti $4-2 \neq 2-1$. Atribut ordinal kadang disebut sebagai **numerik** atau **kontinu**.

3. **Interval**. Tipe interval memiliki urutan dan *range* nilai yang sama. Sebagai contoh $1-5, 6-10, dst$. Kita dapat mentransformasikan/ mengkonversi nilai numerik menjadi nominal dengan cara merubahnya menjadi interval terlebih dahulu. Lalu, kita dapat memberikan nama (simbol) untuk masing-masing interval. Misalkan nilai numerik dengan *range* $1-100$ dibagi menjadi 5 kategori dengan masing-masing interval adalah $\{1-20, 21-40, \ldots, 81-100\}$. Setiap interval kita beri nama, misal interval $81-100$ diberi nama *nilai A*, interval $61-80$ diberi nama *nilai B*.

4. **Ratio**. Tipe *ratio* (rasio) didefinisikan sebagai perbandingan antara suatu nilai dengan nilai lainnya, misalkan massa jenis (fisika). Pada tipe *ratio* terdapat *absolute zero* (semacam *ground truth*) yang menjadi acuan, dan *absolute zero* ini memiliki makna tertentu.

Secara umum, data pada *machine learning* adalah nominal atau numerik (ordinal). Variabel yang kita prediksi yaitu *play* disebut **kelas/*class*/label**. Untuk setiap baris pada Tabel. 3.1, baris kumpulan nilai variabel non-kelas disebut **vektor fitur/*feature vector***. Contohnya pada Tabel. 3.1 $id = 4$, *feature vector*-nya adalah {*outlook=rainy, temperature=mild, humidity=high, windy=false*}. *Feature vector* adalah representasi dari suatu observasi/data. Pada *machine learning*, kita melakukan operasi terhadap data pada representasi *feature vector*-nya. Kami serahkan pada pembaca untuk mencari contoh dataset dengan tipe numerik sebagai pekerjaan rumah[2].

3.3 Ruang Konsep

Dengan data yang diberikan, kita ingin melakukan generalisasi aturan/ konsep yang sesuai dengan data. Hal ini disebut sebagai *inductive learning*. Cara paling sederhana untuk *inductive learning* adalah mengenumerasi seluruh kemungkinan kombinasi nilai sebagai *rule*, kemudian mengeleminasi *rule* yang tidak cocok dengan contoh. Metode ini disebut *list-then-eleminate*. Silahkan

[2] `https://www.cs.waikato.ac.nz/ml/weka/datasets.html`

baca buku Tom Mitchell [4] untuk penjelasakn lebih rinci. Kemungkinan kombinasi nilai ini disebut sebagai ruang konsep (***concept space***). Sebagai contoh pada Tabel. 3.1 himpunan nilai masing-masing atribut yaitu:

- $outlook = \{sunny, overcast, rainy\}$
- $temperature = \{hot, mild, cold\}$
- $humidity = \{high, normal\}$
- $windy = \{true, false\}$
- $play = \{yes, no\}$

sehingga terdapat $3 \times 3 \times 2 \times 2 \times 2 = 72$ kemungkinan kombinasi. Tentunya kita tidak mungkin mengenumerasi seluruh kemungkinan kombinasi nilai karena secara praktikal, atribut yang digunakan banyak. Terlebih lagi, apabila mengenumerasi kombinasi atribut bertipe numerik.

Ada algoritma lain yang mendaftar "seluruh kemungkinan kombinasi" bernama *candidate-elemination algorithm* yang lebih efisien dibanding *list-then-eliminate*. Akan tetapi, algoritma ini *computationally expensive* secara praktikal, dalam artian memiliki kompleksitas yang besar dan tidak bisa menyelesaikan permasalahan nyata. Kamu dapat membaca algoritma ini pada buku Tom Mitchell [4] juga.

Selain *inductive learning*, kita juga dapat melakukan *deductive learning* yaitu melakukan inferensi dari hal general menjadi lebih spesifik. Walau demikian, secara praktis yang dilakukan adalah *inductive learning*.

3.4 Linear Separability

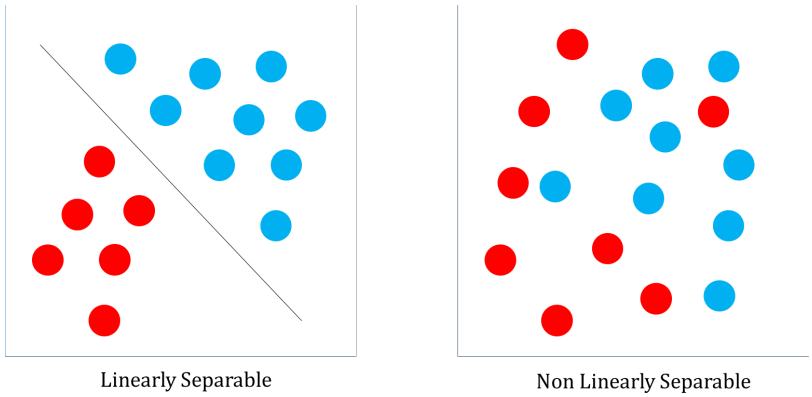
`id`	`humidity`	`windy`	`swim (class)`
1	high	high	yes
2	normal	normal	no

Tabel 3.2. Contoh dataset *linearly separable*.

Perhatikan Tabel. 3.2. Data pada tabel tersebut kita sebut ***linearly separable***. Sederhananya, untuk suatu nilai tertentu, fitur hanya berkorespondensi dengan kelas tertentu. Ambil contoh pada Tabel. 3.2, saat *humidity=high* maka *swim=yes*. Secara "geometris", bila kita proyeksikan *feature vector* ke suatu ruang dimensi, memisahkan kelas satu dan kelas lainnya dapat diperoleh dengan cara menciptakan garis linier (*linear line*) (secara lebih umum, menggunakan *hyperplane*[3]). Ilustrasi dapat dilihat pada Gambar. 3.1. Semantara itu pada Tabel. 3.1, bila kita hanya melihat fitur *humidity* saja, ketika *humidity=high* bisa jadi *play=yes* atau *play=no*. Kasus ini disebut ***non-linearly***

[3] `https://en.wikipedia.org/wiki/Hyperplane`

separable. Hidup kita tentu akan mudah apabila seluruh data bersifat *linearly separable*, sayangnya kebanyakan data yang ada bersifat *non-linearly separable*.



Gambar 3.1. *Linearly* vs *Non-Linearly Separable*.

Untuk memudahkan proses pada data *non-linearly separable*, kita pertamatama mentransformasikan data menjadi *linearly-separable*. Kita dapat menggunakan teknik transformasi data menggunakan *kernel function* seperti *radial basis function*[4]. Pada umumnya, *kernel function* mentransformasi data menjadi lebih tinggi (semacam menambah fitur). Misal dari data yang memiliki dua fitur, ditransformasi menjadi memiliki tiga fitur. Akan tetapi, hal ini tidak praktikal untuk banyak kasus (dijelaskan pada bab 9). Cara lainnya adalah memisahkan data menggunakan model non-linear, contoh: *artificial neural network*. Hal ini penting dipahami karena data yang bersifat *linearly separable* mudah dipisahkan satu sama lain sehingga mudah untuk melakukan *classification* atau *clustering*.

3.5 Seleksi Fitur

Pada subbab sebelumnya, telah dijelaskan bahwa kita dapat mentransformasi data *non-linearly separable* menjadi *linearly separable* dengan cara menambah dimensi data. Pada bab ini, penulis akan menjelaskan justru kebalikannya! Pada permasalahan praktis, kita seringkali menggunakan banyak fitur (*computationally expensive*). Kita ingin menyederhanakan fitur-fitur yang digunakan, misalkan dengan memilih subset fitur awal, atas dasar beberapa alasan [18, 4][5]:

[4] Silakan baca teknik transformasi lebih lanjut pada literatur lain.
[5] `https://en.wikipedia.org/wiki/Feature_selection`

1. Menyederhanakan data/model agar lebih mudah dianalisis.
2. Mengurangi waktu *training* (mengurangi kompleksitas).
3. Menghindari *curse of dimensionality*. Hal ini dijelaskan lebih lanjut pada bab 10.
4. Menghapus fitur yang tidak informatif.
5. Meningkatkan generalisasi dengan mengurangi *overfitting*.

Salah satu contoh cara seleksi fitur adalah menghapus atribut yang memiliki *variance* bernilai 0. Berdasarkan *information theory* atau *entropy*, fitur ini tidak memiliki nilai informasi yang tinggi. Dengan kata lain, atribut yang tidak dapat membedakan satu kelas dan lain bersifat tidak informatif. Kamu dapat membaca beberapa contoh algoritma seleksi fitur pada library sklearn[6].

3.6 Classification, Association, Clustering

Pada *supervsied learning*, kita memprediksi kelas berdasarkan *feature vector* yang merepresentasikan suatu instans (data/observasi). *Feature vector* bisa diibaratkan sebagai sifat-sifat atau keadaan yang diasosiasikan dengan kelas. Pada *supervised learning*, setiap *feature vector* berkorespondisi dengan kelas tertentu. Mencari kelas yang berkorespondensi terhadap suatu instans disebut **klasifikasi** (***classification***). Contoh klasifikasi adalah mengkategorikan gambar buah (e.g. apel, jeruk, dsb). Sementara itu, apabila kita ingin mencari hubungan antara satu atribut dan atribut lainnya, disebut **association**. Sebagai contoh pada Tabel. 3.1, apabila *outlook* = *sunny*, maka sebagian besar *humidity* = *high*. Di lain pihak, pada *unsupervised learning* tidak ada kelas yang berkorespondensi; kita mengelompokkan data dengan sifat-sifat yang mirip, disebut ***clustering***. Contoh *clustering* adalah pengelompokkan barang di supermarket. Perlu kamu catat bahwa *unsupervised learning* $\neq$ *clustering*. *Clustering* adalah salah satu *task* pada *unsupervised learning*.

Pada Tabel. 4.1, hanya ada dua kelas, klasifikasi data ini disebut ***binary classification***. Apabila kelas klasifikasi lebih dari dua, disebut ***multi-class classification/multi-label classification***. Mohon bedakan antara ***multi-class classification*** dan ***multi-level/hierarchical classification***. Pada *multi-level/hierarchical classification*, pertama-tama kita melakukan klasifikasi untuk suatu kelas generik, lalu dilanjutkan mengklasifikan data ke kelas yang lebih spesifik. Contoh *multi-level classification* adalah *kingdom* (biologi), pertama diklasifikasikan ke *kingdom animalia*, lalu lebih spesifiknya ke *phylum Vertebrata*, dst. *Multi-class/multi-label classification* hanya proses klasifikasi ke dalam banyak "kelas" tanpa tinjauan hirarkis. Klasifikasi yang telah disebutkan sebelumnya disebut juga sebagai *hard classification*, artinya apabila data diklasifikasikan ke kelas tertentu, maka tidak mungkin data berada di kelas lainnya (ya atau tidak). Selain *hard classification*, ada juga yang disebut sebagai *soft classification*, yaitu mengklasifikasikan data ke kelas-kelas

[6] `http://scikit-learn.org/stable/modules/feature_selection.html`

tertentu berdasarkan probabilitas (fuzzy). Misalkan data X memiliki 70% probabilitas sebagai kelas A dan 30% sebagai kelas B (sudah dijelaskan pada bab 1).

3.7 Mengukur Kinerja

Pada bab 1, sudah dijelaskan bahwa kita harus mengukur kinerja model dengan cara yang kuantitatif. Salah satu contoh *utility function* yaitu *squared error function* (dijelaskan kemudian pada bab 5). Selain *error function*, kamu juga dapat membandingkan kinerja dengan menggunakan fungsi lainnya seperti akurasi, presisi, *recall*, F1-measure, BLEU [20], ROUGE [21], *intra-cluster similarity*, dsb. Masing-masing *utility function* mengukur hal yang berbeda. Perlu kamu ketahui bahwa memilih ukuran kinerja tergantung pada domain permasalahan. Misalkan pada translasi otomatis, peneliti menggunakan ukuran BLEU; pada peringkasan dokumen, menggunakan ROUGE. Sementara itu, pada *information retrival*/sistem temu balik informasi menggunaan presisi, *recall*, F1-measure, atau *mean average precision* (MAP). Pada domain klasifikasi gambar, menggunakan akurasi. Masing-masing *utility function* dapat memiliki cara mencapai titik optimal yang berbeda. Kamu harus mengerti domain permasalahan untuk mengerti cara mencapai titik optimal. Sebagai pengantar, diktat ini tidak dapat membahas seluruh domain. Dengan demikian, kamu harus membaca lebih lanjut literatur spesifik domain, misal buku pemrosesan bahasa alami atau sistem temu balik informasi, dsb. Sebagai contoh, untuk permasalahan klasifikasi, akurasi sering digunakan. Akurasi didefinisikan pada persamaan 3.1

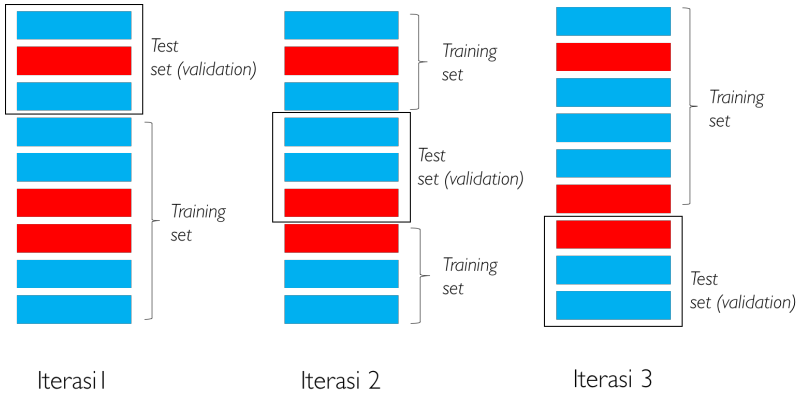
$$\text{akurasi} = \frac{\#\text{instans diklasifikasikan dengan benar}}{\text{banyaknya instans}} \tag{3.1}$$

3.8 Evaluasi Model

Ada beberapa hal yang perlu kamu catat tentang proses evaluasi suatu model pembelajaran mesin:

1. ***Data splitting***. Seperti yang sudah dijelaskan pada subbab 1.5, pada umumnya kita memiliki *training*, *validation*/*development*, dan *testing data*. Mesin dilatih menggunakan *training data*, saat proses *training*, *performance measure* diukur berdasarkan kemampuan mengenali/ mengeneralisasi *validation data*. Perlu diketahui, *performance measure* diukur menggunakan *validation data* untuk menghindari *overfitting* dan *underfitting*. Setelah selesai dilatih, maka model hasil pembelajaran dievaluasi dengan *testing data*. *Training*, *validation*, dan *testing* data tersusun oleh data yang independen satu sama lain (tidak beririsan) untuk memastikan model yang dihasilkan memiliki generalisasi cukup baik.

2. ***Overfitting dan Underfitting***. *Overfitting* adalah keadaan ketika model memiliki kinerja baik hanya untuk *training data/seen examples* tetapi tidak memiliki kinerja baik untuk *unseen examples*. *Underftting* adalah keadaan ketika model memiliki kinerja buruk baik untuk *training data* dan *unseen examples*. Hal ini akan dibahas lebih detil pada subbab 5.2.
3. ***Cross validation***. *Cross validation* adalah teknik untuk menganalisis apakah suatu model memiliki generalisasi yang baik (mampu memiliki kinerja yang baik pada *unseen examples*). Seperti yang kamu ketahui, kita dapat membagi data menjadi *training*, *validation*, dan *testing data*. Saat proses *training*, kita latih model dengan *training data* serta dievaluasi menggunakan *validation data*. Teknik *cross validation* bekerja dengan prinsip yang sama, yaitu membagi sampel asli menjadi beberapa subsampel dengan partisi sebanyak K (K-*fold*). Ilustrasi diberikan oleh Gambar. 3.2. Persegi panjang melambangkan suatu instans. Saat proses *training*, kita bagi data menjadi *training data* dan *test data* (i.e., *validation data*). Hal ini diulang sebanyak K kali. Kita evaluasi kemampuan generalisasi model dengan merata-ratakan kinerja pada tiap iterasi. Setelah itu, model dengan kinerja terbaik (pada iterasi teretentu) digunakan lebih lanjut untuk proses *testing* atau dipakai secara praktis. Perlu diperhatikan, setiap subsampel sebaiknya memiliki distribusi yang sama dengan sampel aslinya (keseluruhan sampel); i.e., pada contoh, proporsi warna biru dan merah adalah sama tiap partisi tiap iterasi. Konsep tersebut lebih dikenal dengan *stratified sampling*[7].



Gambar 3.2. Ilustrasi 3-*fold cross validation*.

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

3.9 Kategori Jenis Algoritma

Algoritma pembelajaran mesin dapat dibagi menjadi beberapa kategori. Dari sudut pandang apakah algoritma memiliki parameter yang harus dioptimasi, dapat dibagi menjadi:

1. Parametrik, contoh kelompok model linear, *artificial neural network*.
2. Non parametrik, contoh *Naive Bayes*, *decision tree* (ID3).

Dari sudut pandang lainnya, jenis algoritma dapat dibagi menjadi:

1. Model linear, contoh regresi linear, regresi logistik, *support vector machine*.
2. Model probabilistik, contoh *Naive Bayes*, *hidden markov model*.
3. Model non-linear, yaitu (*typically*) *artificial neural network*.

Selain kedua skema pengkategorian ini, terdapat skema pengkategorian lain (silahkan eksplorasi sendiri).

3.10 Tahapan Analisis

Bagian ini adalah ringkasan bab ini. Untuk menganalisis data, terdapat langkah yang perlu kamu perhatikan

1. Memutuskan tujuan analisis data (*defining goal*)
2. Mendapatkan data
3. Merapihkan data
4. Merepresentasikan data sebagai *feature vector*
5. Melakukan transformasi dan/atau *feature selection* (mengurasi dimensi *feature vector*)
6. Melatih model (*training*)
7. Melakukan *testing* dan analisis model baik secara kuantitatif dan kualitatif
8. Menyajikan data (presentasi)

Soal Latihan

3.1. Konversi atribut
Sebutkan dan jelaskan macam-macam cara untuk mengkonversi atribut! Sebagai contoh, numerik-nominal dan nominal-numerik.

3.2. Transformasi data
Sebutkan dan jelaskan macam-macam cara transformasi data (e.g. merubah *non-linearly separable* menjadi *linearly separable*)

3.3. Seleksi fitur
Bacalah algoritma seleksi fitur pada *library* sklearn. Jelaskan alasan (*rationale*) dibalik penggunaan tiap algoritma yang ada!

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-*

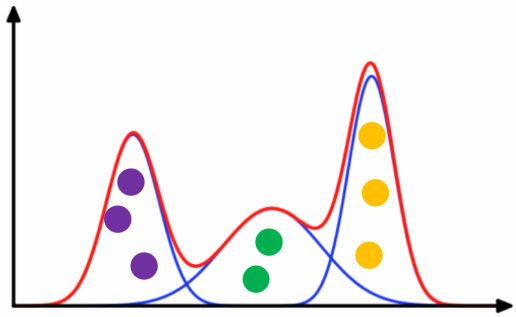
[1] Cara lain memilih akan dijelaskan pada bab 8.

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.

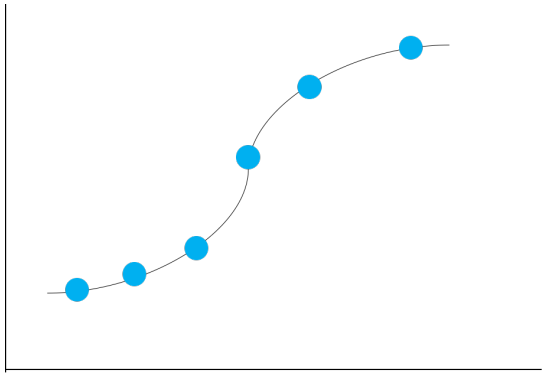
Error function paling mudah dijelaskan dalam permasalahan regresi. Masih ingat contoh bab sebelumnya tentang estimasi distribusi *Univariate Gaussian*? Ingat kembali konsep tersebut untuk mengerti bab ini. Diberikan (x, y) sebagai *random variable* berdimensi R^M dan R^N (keduanya berada pada Euclidean 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 $f(x)$!



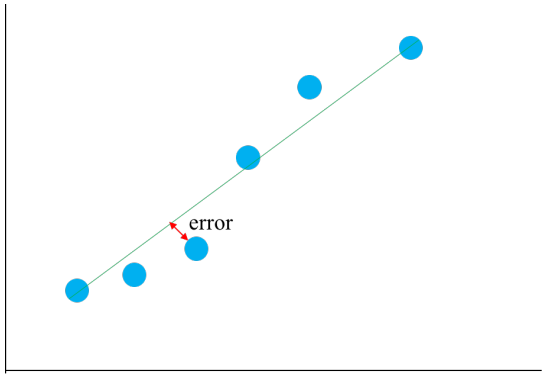
Gambar 5.1. Contoh fungsi Sigmoid.

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.

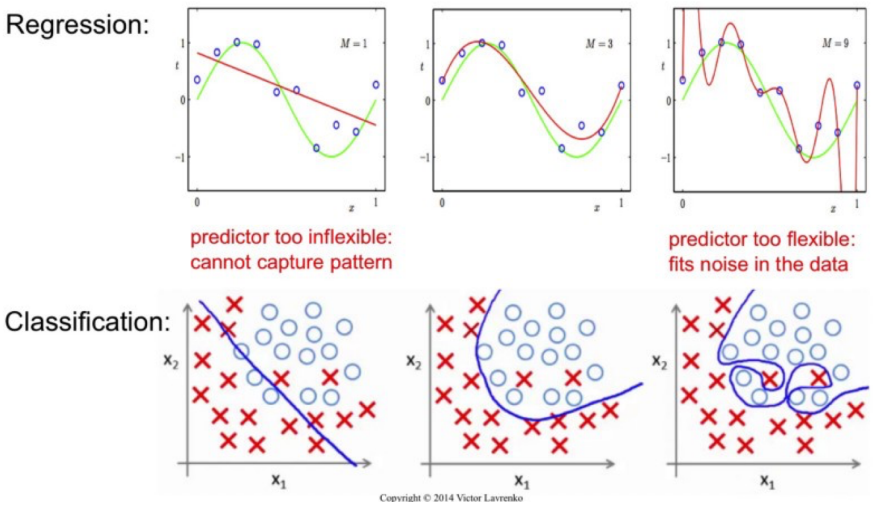
Ingat kembali bab 1, *learning machine* yang direpresentasikan dengan fungsi g bisa diatur kinerjanya dengan parameter *training* $\mathbf{w}$. *Square error* untuk *learning machine* dengan parameter *training* $\mathbf{w}$ diberikan oleh persamaan 5.3. (x_i, y_i) adalah pasangan *input-desired output*. 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).

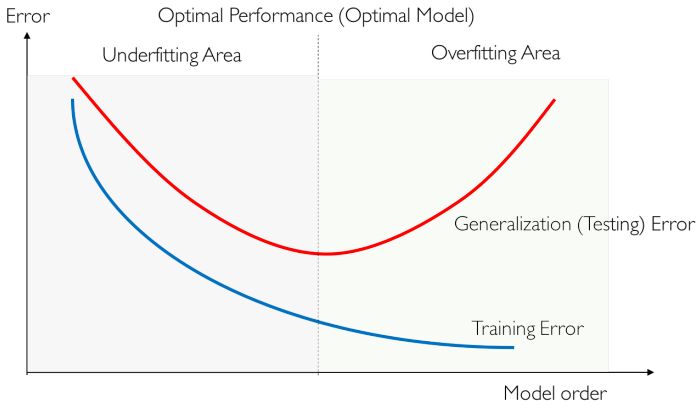


Gambar 5.3. *Underfitting* vs. *Overfitting*[3].

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`

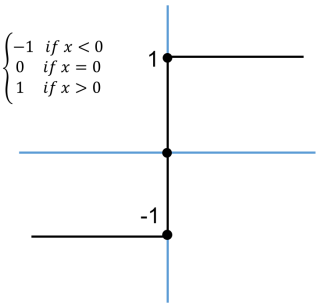
memastikan kesamaan karakteristik data (e.g., apakah *training* dan *testing* data benar diambil dari distribusi yang sama).



Gambar 5.4. *Selection Error.*

5.3 Binary Classification

$$\text{sgn}(x) = \begin{cases} -1 & if\ x < 0 \\ 0 & if\ x = 0 \\ 1 & if\ x > 0 \end{cases}$$



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

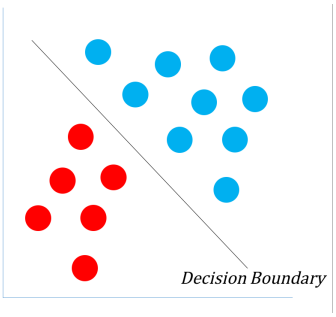
sebagai *feature vector* $\mathbf{x}$, dan terdapat *bias*[4] b. Klasifikasi dilakukan dengan melewatkan data pada fungsi yang memiliki parameter. Fungsi tersebut menghitung bobot setiap fitur pada vektor dengan mengalikannya dengan parameter (*dot product*). Persamaan 5.4 dapat ditulis kembali sebagai 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.

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

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).

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

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

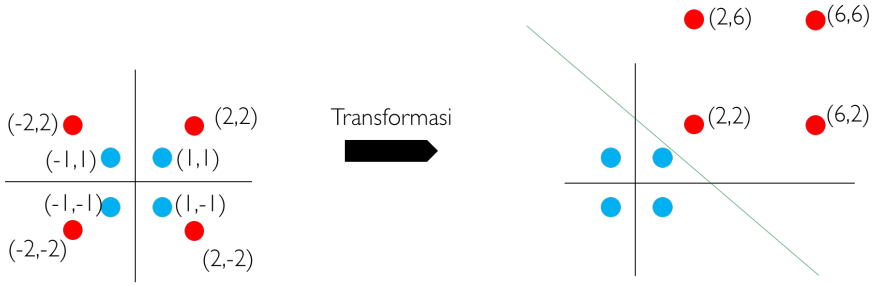
Seperti yang diceritakan pada subbab berikutnya, kita mungkin juga tertarik dengan probabilitas masing-masing kelas. Kita dapat menggunakan

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 θ adalah *learning parameter*[8], dan $\mathcal{L}$ adalah *loss function*. Perubahan parameter dapat menyebabkan perubahan *loss*. Karena itu, *loss function* memiliki θ sebagai parameternya.

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

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*.

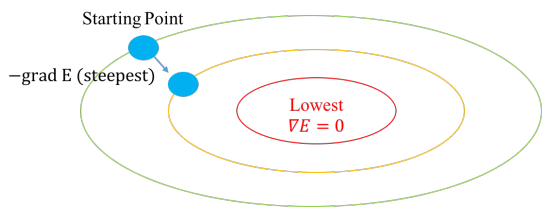
[8] Umumnya dapat berupa skalar, vektor, atau matriks. Demi istilah yang lebih generik, kita gunakan θ.

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

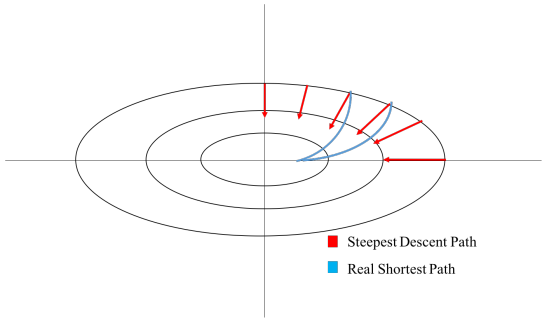
Tentunya seiring berjalannya waktu, kita mengubah-ubah parameter $\mathbf{w}$ agar kinerja model optimal. Nilai optimal diberikan oleh turunan $\mathbf{w}$ terhadap waktu, yang bernilai sama dengan $-grad\ E(\mathbf{w})$. Bentuk diskrit persamaan 5.16 diberikan pada persamaan 5.17

[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 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) [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).

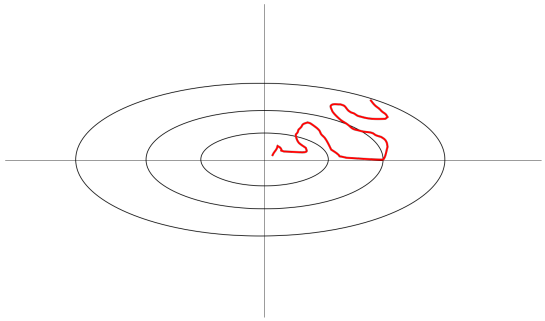


Gambar 5.9. *Stochastic Gradient Descent* 2.

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* (η). Apabila nilai *learning rate* (η) 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 (α) 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 $\mathbf{w}$ 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[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 λ 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*.

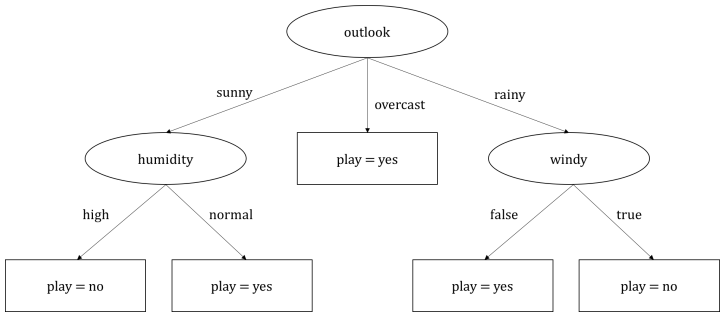
6.1 Inductive Learning

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*
- i*f 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.

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

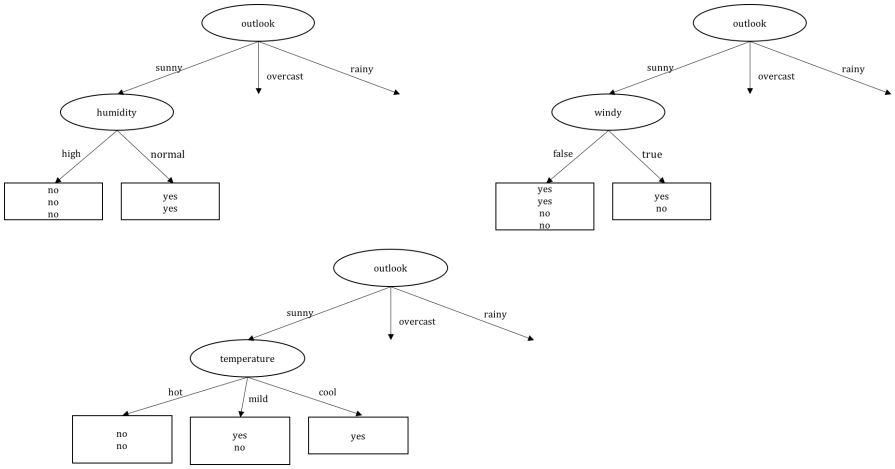
Dengan demikian, kita memilih atribut *outlook* sebagai *root*.

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.

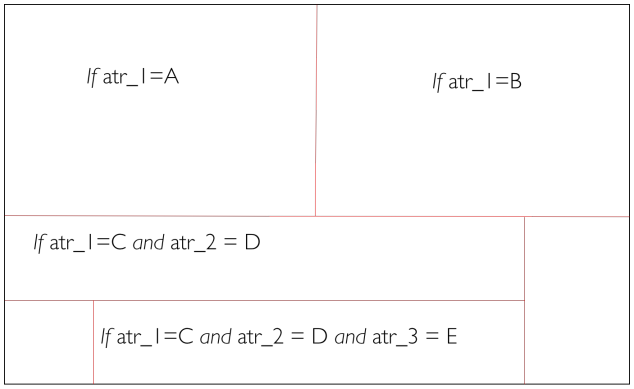
6.3 Isu pada ID3

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.

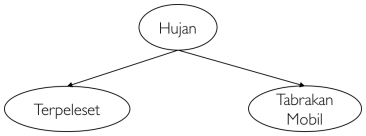
7.1 Probabilistic Reasoning

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.

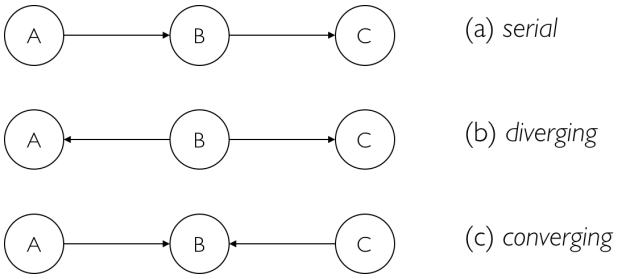
jadi, maka "terpeleset" juga pasti akan terjadi. Tetapi tidak sebaliknya, apabila kita "terpeleset", belum tentu "hujan" juga terjadi. Pada *probabilistic reasoning* kita mengkuantifikasi kepastian atau ketidakpastian itu. Apabila "hujan" terjadi, berapa besar kemungkinan "terpeleset" juga terjadi (dan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

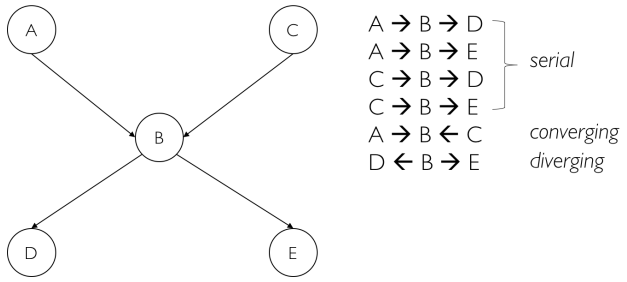
(b) *Diverging*. Bila kita mengetahui A maka kita bisa mengetahui sesuatu tentang C, dan sebaliknya. Tetapi apabila B diketahui, maka hubungan antara A dan C menjadi terputus. Dengan bahasa lebih matematis, A dan C independen kondisional ketika B diketahui.

(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.

$$P(A, B, C, D, E) = P(D \mid B)\ P(E \mid B)\ P(B \mid A, C)\ P(A)\ P(C) \tag{7.1}$$

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*.

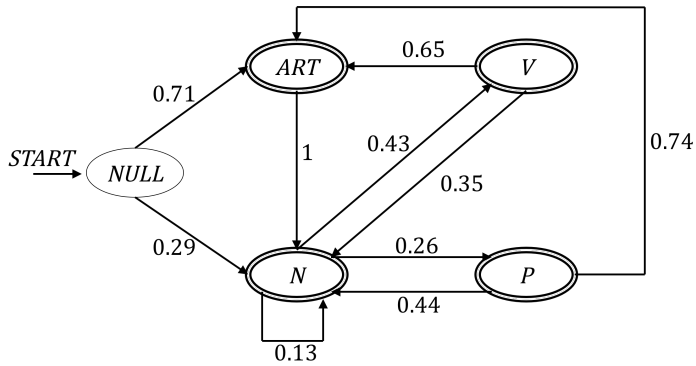
Ingat kembali persamaan 7.12 untuk POS tagging. $P(w_i|c_i)$ disebut *likelihood* dan $P(c_i|c_{i-1})$ disebut *prior*, *multiplication of probabilities* ($\prod$) melambangkan *markov chain*. ***Markov chain*** adalah kasus spesial *weighted automaton*[5] yang mana sekuens *input* menentukan *states* 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].

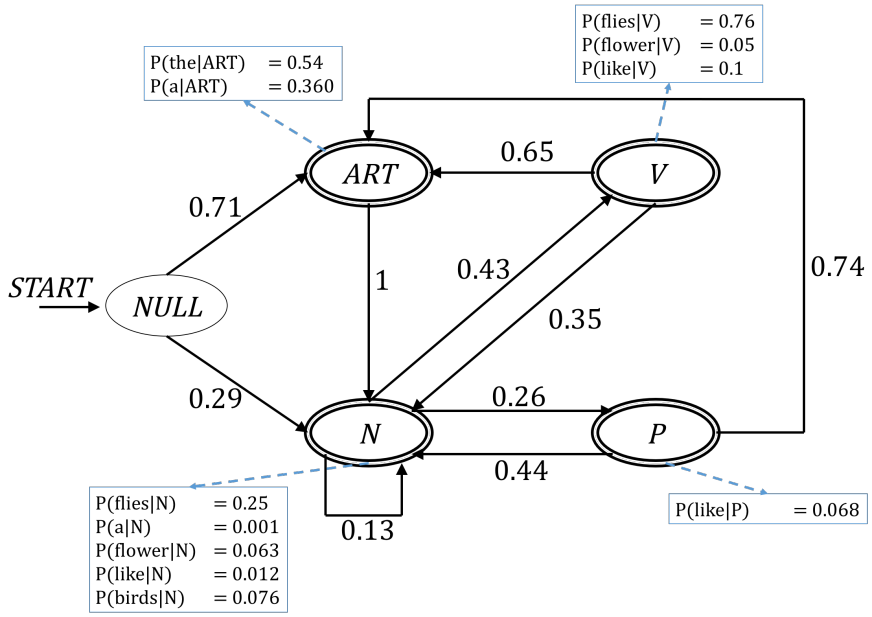
$P(the\|ART)$	0.54	$P(a\|ART)$	0.360
$P(flies\|N)$	0.025	$P(a\|N)$	0.001
$P(flies\|V)$	0.076	$P(flower\|N)$	0.063
$P(like\|V)$	0.1	$P(flower\|V)$	0.05
$P(like\|P)$	0.068	$P(birds\|N)$	0.076
$P(like\|N)$	0.012		

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

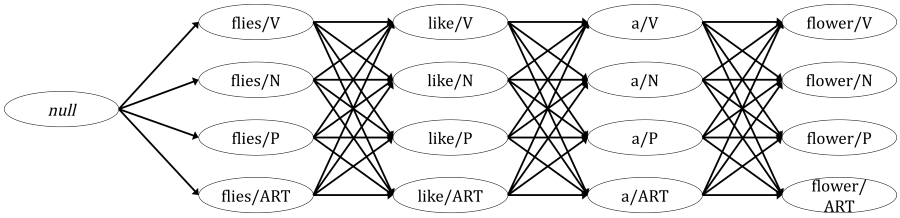
Diberikan kalimat "*flies like a flower*", untuk menghitung sekuens kelas kata untuk kalimat tersebut, kita menyelusuri automaton Gambar. 7.5. Hasil penelusuran memberikan kita kombinasi sekuens yang mungkin seperti pada Gambar. 7.6. Pekerjaan berikutnya adalah, dari seluruh kombinasi sekuens yang mungkin ($\prod_{i=1}^{N} P(w_i|c_i)P(c_i|c_{i-1})$), 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

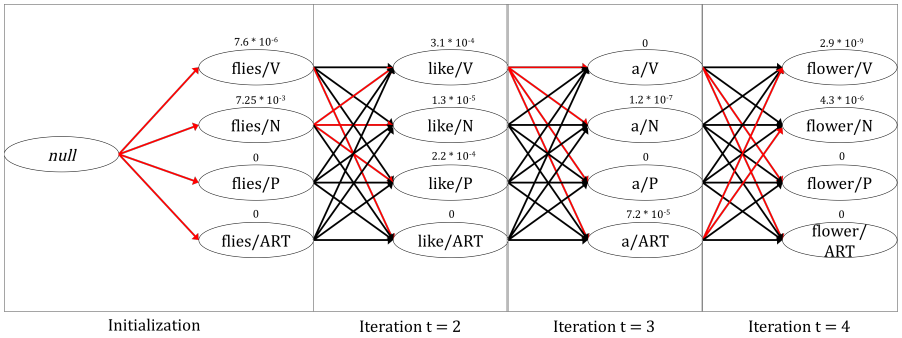
Sequence Identification Step
$c_T \leftarrow i$ that maximizes viterbi[i, T]
for $i = T - 1$ **to** 1 **do**
  $c_i \leftarrow$ backpointer[c_{i+1},$i+1$]

Gambar 7.7. Algoritma Viterbi [12, 32].

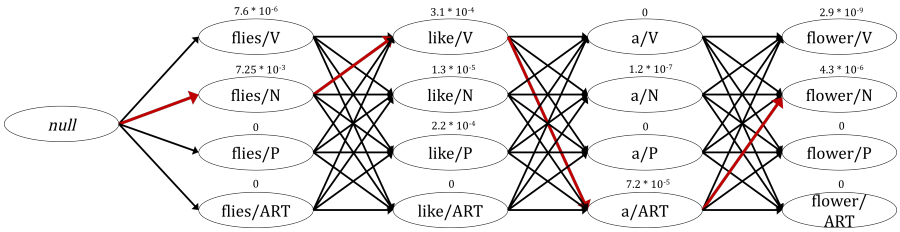
Perhatikan Gambar. 7.7 yang menunjukkan *pseudo-code* untuk algoritma Viterbi. Variabel c berarti kelas kata, a adalah *transition probability*, dan b adalah *lexical-generation probability*. Pertama-tama, algoritma tersebut membuat suatu matriks berukuran NxT dengan N adalah banyaknya *states* (tidak 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 γ dan ξ 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
    update $a_{ij}{}' = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \sum_{j=1}^{N} \xi_t(i,j)}$

    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.

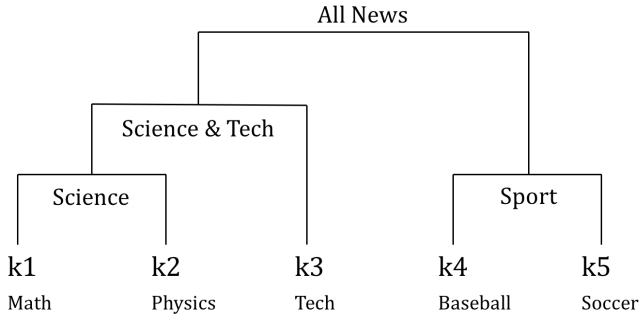
Sebagai contoh, algoritma *hierarchical clustering* menghasilkan struktur hirarkis seperti pada gambar 8.1 yang disebut **dendogram**. Dendogram melambangkan taksonomi, sebagai contoh taksonomi dokumen berita *sport*, dipecah menjadi *baseball* dan *soccer*.

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* **U** dan **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* **U** dan $\mathbf{c}^U$ adalah centroid untuk *cluster* **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!

Bagian III

Neural Networks

9

Artificial Neural Network

> "If you want to make information stick, it's best to learn it, go away from it for a while, come back to it later, leave it behind again, and once again return to it - to engage with it deeply across time. Our memories naturally degrade, but each time you return to a memory, you reactivate its neural network and help to lock it in."
>
> Joshua Foer

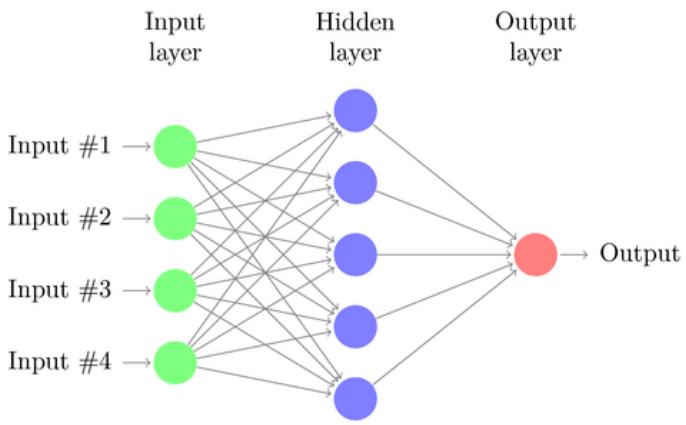
Bab ini membahas salah satu algoritma *machine learning* yang sedang populer belakangan ini, yaitu *artificial neural network*. Pembahasan dimulai dari hal-hal sederhana sampai yang lebih kompleks. Bab ini lebih berfokus pada penggunaan *artifical neural network* untuk *supervised learning*.

9.1 Definisi

Masih ingatkah Anda materi pada bab-bab sebelumnya? *Machine learning* sebenarnya meniru bagaimana proses manusia belajar. Pada bagian ini, peneliti ingin meniru proses belajar tersebut dengan mensimulasikan jaringan saraf biologis (*neural network*) [36, 37, 38, 39]. Kami yakin banyak yang sudah tidak asing dengan istilah ini, berhubung *deep learning* sedang populer dan banyak yang membicarakannya (dan digunakan sebagai trik pemasaran). *Artificial neural network* adalah salah satu algoritma *supervised learning* yang populer dan bisa juga digunakan untuk *semi-supervised* atau *unsupervised learning* [37, 39, 40, 41, 42]. Walaupun tujuan awalnya adalah untuk mensimulasikan jaringan saraf biologis, jaringan tiruan ini sebenenarnya simulasi

yang terlalu disederhanakan, artinya simulasi yang dilakukan tidak mampu menggambarkan kompleksitas jaringan biologis manusia[1].

Artificial Neural Network (selanjutnya disingkat ANN), menghasilkan model yang sulit dibaca dan dimengerti oleh manusia karena memiliki banyak layer (*multilayer perceptron*) dan sifat *non-linearity* (fungsi aktivasi). Pada bidang riset ini, ANN disebut agnostik (kita percaya, tetapi sulit membuktikan kenapa konfigurasi parameter yang dihasilkan *training* bisa benar). Secara matematis, ANN ibarat sebuah graf. ANN memiliki neuron/*node* (*vertex*), dan sinapsis (*edge*). Topologi ANN akan dibahas lebih detil subbab berikutnya. Sebagai gambaran, ANN berbentuk seperti Gambar. 9.1. Walaupun memiliki struktur seperti graf, operasi pada ANN paling baik dan mudah dijelaskan dalam notasi aljabar linear.



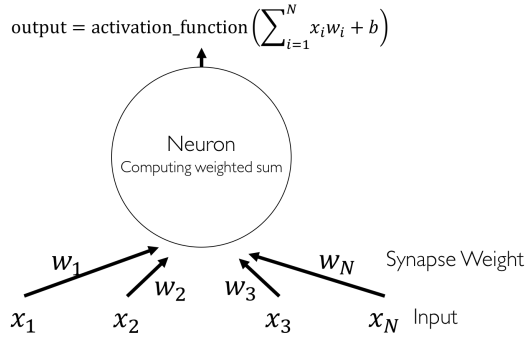
Gambar 9.1. *Multilayer Perceptron.*

9.2 Single Perceptron

Bentuk terkecil (minimal) sebuah ANN adalah *single perceptron* yang hanya terdiri dari sebuah neuron. Sebuah neuron diilustrasikan pada Gambar. 9.2. Secara matematis, terdapat *feature vector* $\mathbf{x}$ yang menjadi *input* bagi neuron tersebut. Neuron akan memproses *input* $\mathbf{x}$ melalui perhitungan jumlah perkalian antara nilai *input* dan *synapse weight*, yang dilewatkan pada **fungsi non-linear** [43, 44, 4]. Pada *training*, yang dioptimasi adalah nilai *synapse weight* (*learning parameter*). Selain itu, terdapat juga bias b sebagai kontrol tambahan (ingat materi *steepest gradient descent*). *Output* dari neuron

[1] `Quora: why is Geoffrey Hinton suspicious of backpropagation and wants AI to start over`

adalah hasil fungsi aktivasi dari perhitungan jumlah perkalian antara nilai *input* dan *synapse weight*. Ada beberapa macam fungsi aktivasi, misal ***step function***, ***sign function***, ***rectifier*** dan ***sigmoid function***. Untuk selanjutnya, pada bab ini, fungsi aktivasi yang dimaksud adalah jenis ***sigmoid function***. Silahkan eksplorasi sendiri untuk fungsi aktivasi lainnya. Salah satu bentuk tipe ***sigmoid function*** diberikan pada persamaan 9.1. Bila di-*plot* menjadi grafik, fungsi ini memberikan bentuk seperti huruf S.



Gambar 9.2. *Single Perceptron.*

$$\sigma(u) = \frac{1}{1 + e^{-u}} \tag{9.1}$$

Perhatikan kembali, Gambar. 9.2 sesungguhnya adalah operasi aljabar linear. Single perceptron dapat dituliskan kembali sebagai 9.2.

$$o = f(\mathbf{x} \cdot \mathbf{w} + b) \tag{9.2}$$

dimana o adalah *output* dan f adalah fungsi **non-linear yang dapat diturunkan secara matematis** (*differentiable non-linear function* – selanjutnya disebut "fungsi non-linear" saja.). Bentuk ini tidak lain dan tidak bukan adalah persamaan model linear yang ditransformasi dengan fungsi non-linear. Secara filosofis, ANN bekerja mirip dengan model linear, yaitu mencari *decision boudary*. Apabila beberapa model non-linear ini digabungkan, maka kemampuannya akan menjadi lebih hebat (subbab berikutnya).

Untuk melakukan pembelajaran *single perceptron*, *training* dilakukan menggunakan ***perceptron training rule***. Prosesnya sebagai berikut [4, 43, 44]:

1. Inisiasi nilai *synapse weights*, bisa *random* ataupun dengan aturan tertentu. Untuk heuristik aturan inisiasi, ada baiknya membaca buku referensi [1, 11].
2. Lewatkan input pada neuron, kemudian kita akan mendapatkan nilai *output*. Kegiatan ini disebut ***feedforward***.

3. Nilai *output* (*actual output*) tersebut dibandingkan dengan *desired output*.
4. Apabila nilai *output* sesuai dengan *desired output*, tidak perlu mengubah apa-apa.
5. Apabila nilai *output* tidak sesuai dengan *desired output*, hitung nilai *error* kemudian lakukan perubahan terhadap *learning parameter* (*synapse weight*).
6. Ulangi langkah-langkah ini sampai tidak ada perubahan nilai *error*, nilai *error* kurang dari sama dengan suatu *threshold* (biasanya mendekati 0), atau sudah mengulangi proses latihan sebanyak N kali (*threshold*).

Error function diberikan pada persamaan 9.3 dan perubahan *synapse weight* diberikan pada persamaan 9.4. y melambangkan *desired output*[2], $f(\mathbf{x}, \mathbf{w})$ melambangkan *actual output* untuk $\mathbf{x}$ sebagai input. η disebut sebagai *learning rate* [3].

$$E(\mathbf{w}) = (y - f(\mathbf{x}, \mathbf{w}))^2 \tag{9.3}$$

$$\Delta w_i = \eta(y - o)x_i \tag{9.4}$$

Hasil akhir pembelajaran adalah konfigurasi *synapse weight*. Saat klasifikasi, kita melewatkan *input* baru pada jaringan yang telah dibangun, kemudian tinggal mengambil hasilnya. Pada contoh kali ini, seolah-olah *single perceptron* hanya dapat digunakan untuk melakukan *binary classification* (hanya ada dua kelas, nilai 0 dan 1). Untuk *multi-label classification*, kita dapat menerapkan berbagai strategi. Metode paling umum adalah melewatkan *output neurons* pada fungsi softmax[4], sehingga mendapatkan nilai distribusi probabilitas untuk tiap kelas.

9.3 Permasalahan XOR

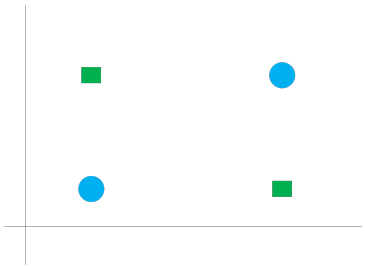
Sedikit sejarah, *perceptron* sebenarnya cukup populer sekitar tahun 1950-1960. Entah karena suatu sebab, *perceptron* menjadi tidak populer dan digantikan oleh model linear. Saat itu, belum ada algoritma yang bekerja dengan relatif bagus untuk melatih *perceptron* yang digabungkan (*multilayer perceptron*). Model linear mendapat popularitas hingga kira-kira dapat disebut sekitar tahun 1990'an atau awal 2000'an. Berkat penemuan *backpropagation* sekitar awal 1980[5], *multilayer perceptron* menjadi semakin populer. Perlu dicatat, komunitas riset bisa jadi seperti cerita ini.

Pada bab-bab sebelumnya, kamu telah mempelajari model linear dan model probabilistik. Kita ulangi kembali contoh data yang bersifat *non-linearly separable*, yaitu XOR yang operasinya didefinisikan sebagai:

[2] Pada contoh ini, kebetulan jumlah neuron output hanyalah satu.
[3] Pada umumnya, kita tidak menggunakan satu data, tetapi *batch-sized*.
[4] Sudah dibahas pada model linear.
[5] `http://people.idsia.ch/~juergen/who-invented-backpropagation.html`

- $\text{XOR}(0, 0) = 0$
- $\text{XOR}(1, 0) = 1$
- $\text{XOR}(0, 1) = 1$
- $\text{XOR}(1, 1) = 0$

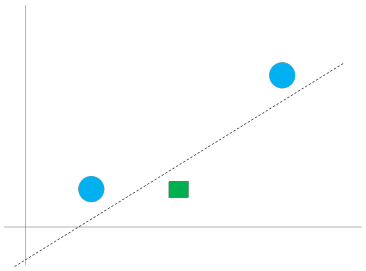
Ilustrasinya dapat dilihat pada Gambar. 9.3. Jelas sekali, XOR ini adalah fungsi yang tidak dapat diselesaikan secara langsung oleh model linear.



Gambar 9.3. Permasalahan XOR.

Seperti yang diceritakan pada bab model linear, solusi permasalahan ini adalah dengan melakukan transformasi data menjadi *linearly-separable*, misalnya menggunakan fungsi non-linear pada persamaan 9.5 dimana (x, y) adalah absis dan ordinat. Hasil transformasi menggunakan fungsi ini dapat dilihat pada Gambar. 9.4. Jelas sekali, data menjadi *linearly separable*.

$$\phi(x, y) = (x \times y, x + y) \tag{9.5}$$



Gambar 9.4. XOR ditransformasi.

Sudah dijelaskan pada bab model linear, permasalahan yang ada tidak sesederhana ini (kebetulan ditransformasikan menjadi data dengan dimensi yang sama). Pada permasalahan praktis, kita harus mentransformasi data menjadi dimensi lebih tinggi (dari 2D menjadi 3D). Berbeda dengan ide utama linear model/*kernel method* tersebut, **prinsip ANN adalah untuk melewatkan data pada fungsi non-linear** (*non-linearities*). Sekali lagi penulis

ingin tekankan, ANN secara filosofis adalah ***trainable non-linear mapping functions***. ANN mentransformasi data ke *space*/ruang konsep yang berbeda, lalu mencari *non-linear decision boundary* dengan *non-linear functions*.

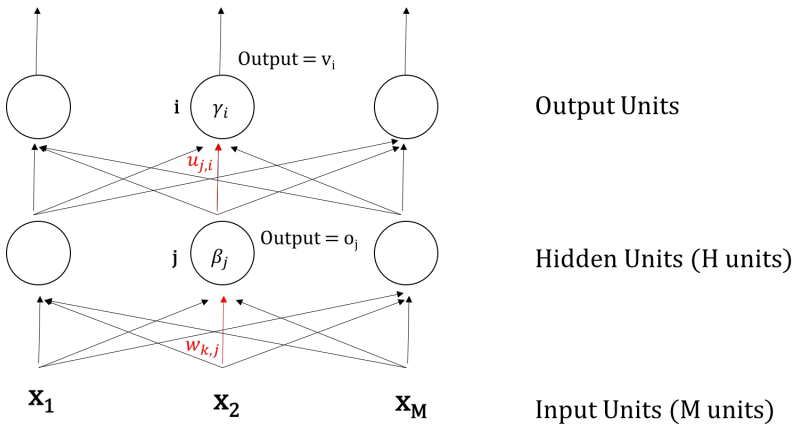
Perlu dicatat, kemampuan transformasi non-linear inilah yang membuat ANN menjadi hebat. ANN mungkin secara luas didefinisikan mencakup *perceptron* tetapi secara praktis, **ANN sebenarnya mengacu pada *multilayer perceptron* dan arsitektur lebih kompleks** (dijelaskan pada subbab berikutnya). Pada masa ini, hampir tidak ada lagi yang menggunakan *single perceptron*. Untuk bab-bab kedepan, ketika kami menyebut ANN maka yang diacu adalah *multilayer perceptron* dan arsitektur lebih kompleks (*single perceptron* di-*exclude*). Hal ini disebabkan oleh *single perceptron* tidak dapat mempelajari *XOR function* secara independen tanpa *feature engineering*, sementara *multilayer perceptron* bisa [45].

9.4 Multilayer Perceptron

Kamu sudah belajar *training* untuk *single perceptron*. Selanjutnya kita akan mempelajari *multilayer perceptron* (MLP) yang juga dikenal sebagai ***feed forward neural network***. Kami tekankan sekali lagi, **istilah "ANN" selanjutnya mengacu pada MLP dan arsitektur lebih kompleks**. Seperti ilustrasi pada Gambar. 9.5, *multilayer perceptron* secara literal memiliki beberapa *layers*. Pada *lecture note* ini, secara umum ada 3 *layers*: *input*, *hidden*, dan *output layer*. *Input layer* menerima *input* (tanpa melakukan operasi apapun), kemudian nilai *input* (tanpa dilewatkan ke fungsi aktivasi) diberikan ke *hidden units*. Pada *hidden units*, *input* diproses dan dilakukan perhitungan hasil fungsi aktivasi untuk tiap-tiap neuron, lalu hasilnya diberikan ke *layer* berikutnya. Hasil dari *input layer* akan diterima sebagai input bagi *hidden layer*. Begitupula seterusnya *hidden layer* akan mengirimkan hasilnya untuk *output layer*. Kegiatan ini dinamakan *feed forward* [37, 4]. Hal serupa berlaku untuk *artificial neural network* dengan lebih dari 3 *layers*. Parameter neuron dapat dioptimisasi menggunakan metode *gradient-based optimization* (dibahas pada subabb berikutnya, ingat kembali bab 5). Perlu diperhatikan, MLP adalah gabungan dari banyak fungsi non-linear. Seperti yang disampaikan pada subbab sebelumnya, gabungan banyak fungsi non-linear ini lebih hebat dibanding *single perceptron*.

$$o_j = \sigma\left(\sum_{k=1}^{M} x_k w_{k,j} + \beta_j\right) \tag{9.6}$$

$$v_i = \sigma\left(\sum_{j=1}^{H} o_j u_{j,i} + \gamma_i\right) = \sigma\left(\sum_{j=1}^{H} \sigma\left(\left(\sum_{k=1}^{M} x_k w_{k,j} + \beta_j\right) u_{j,i}\right) + \gamma_i\right) \tag{9.7}$$



Gambar 9.5. *Multilayer Perceptron* 2.

Perhatikan persamaan 9.6 dan 9.7 untuk menghitung *output* pada *layer* yang berbeda. u, w adalah *learning parameters*. β, γ melambangkan *noise* atau *bias*. M adalah banyaknya *hidden units* dan H adalah banyaknya *output units*. Persamaan 9.7 dapat disederhanakan penulisannya sebagai persamaan 9.8. Persamaan 9.8 terlihat relatif lebih "elegan". Seperti yang disebutkan pada subbab sebelumnya, ANN dapat direpresentasikan dengan notasi operasi aljabar.

$$\mathbf{v} = \sigma(\mathbf{o}\mathbf{U} + \gamma) = \sigma((\sigma(\mathbf{x}\mathbf{W} + \beta))\mathbf{U} + \gamma) \tag{9.8}$$

Untuk melatih MLP, algoritma yang umumnya digunakan adalah ***backpropagation*** [46]. Arti kata *backpropagation* sulit untuk diterjemahkan ke dalam bahasa Indonesia. Kita memperbaharui parameter (*synapse weights*) secara bertahap (dari *output* ke *input* layer, karena itu disebut *backpropagation*) berdasarkan *error/loss* (*output* dibandingkan dengan *desired output*). Intinya adalah mengkoreksi *synapse weight* dari *output layer* ke *hidden layer*, kemudian *error* tersebut dipropagasi ke layer sebelum-sebelumnya. Artinya, perubahan *synapse weight* pada suatu layer dipengaruhi oleh perubahan *synapse weight* pada layer setelahnya[6]. *Backpropagation* tidak lain dan tidak bukan adalah metode *gradient-based optimization* yang diterapkan pada ANN.

Pertama-tama diberikan pasangan *input* ($\mathbf{x}$) dan *desired output* ($\mathbf{y}$) sebagai *training data*. Untuk meminimalkan *loss*, algoritma *backpropagation* menggunakan prinsip *gradient descent* (ingat kembali materi bab model linear). Kamu akan memperlajari bagaimana cara menurunkan *backpropagation* menggunakan teknik *gradient descent*, yaitu menghitung *loss* ANN pada Gam-

[6] Kata "setelah" mengacu *layer* yang menuju *output layer*, "sebelum" mengacu layer yang lebih dekat dengan *input layer*.

bar. 9.5 **yang menggunakan fungsi aktivasi sigmoid**. Untuk fungsi aktivasi lainnya, pembaca dapat mencoba menurunkan persamaan sendiri!

Ingat kembali *chain rule* pada perkuliahan diferensial

$$f(g(x))' = f'(g(x))g'(x). \tag{9.9}$$

Ingat kembali *error*, untuk MLP diberikan oleh persamaan 9.3 (untuk satu data *point*), dimana I adalah banyaknya output neuron.

$$E(\mathbf{P}) = \frac{1}{I}\sum_{i=1}^{I}(y_i - v_i)^2 \tag{9.10}$$

Mari kita lakukan proses penurunan untuk melatih MLP. *Error/loss* diturunkan terhadap tiap *learning parameter*.

Diferensial $u_{j,i}$ diberikan oleh turunan *sigmoid function*

$$\begin{aligned}\frac{\delta E(\mathbf{P})}{\delta u_{j,i}} &= (y_i - v_i)\frac{\delta v_i}{\delta u_{j,i}} \\ &= (y_i - v_i)v_i(1 - v_i)o_j\end{aligned}$$

Diferensial $w_{k,j}$ diberikan oleh turunan *sigmoid function*

$$\begin{aligned}\frac{\delta E(\mathbf{P})}{\delta w_{k,j}} &= \sum_{i=1}^{H}(y_i - v_i)\frac{\delta v_i}{\delta w_{k,j}} \\ &= \sum_{i=1}^{H}(y_i - v_i)\frac{\delta v_i}{\delta o_j}\frac{\delta o_j}{\delta w_{k,j}} \\ &= \sum_{i=1}^{H}(y_i - v_i)[v_i(1 - v_i)u_{j,i}][o_j(1 - o_j)x_k]\end{aligned}$$

Perhatikan, diferensial $w_{k,j}$ memiliki $\sum$ sementara $u_{j,i}$ tidak ada. Hal ini disebabkan karena $u_{j,i}$ hanya berkorespondensi dengan satu *output* neuron. Sementara $w_{k,j}$ berkorespondensi dengan banyak *output* neuron. Dengan kata lain, nilai $w_{k,j}$ mempengaruhi hasil operasi yang terjadi pada banyak *output* neuron; atau banyak neuron mempropagasi *error* kembali ke $w_{k,j}$.

Metode penurunan serupa dapat juga digunakan untuk menentukan perubahan β dan γ. Jadi proses *backpropagation* untuk kasus Gambar. 9.5 dapat diberikan seperti pada Gambar. 9.6 dimana η adalah *learning rate*. Untuk *artificial neural network dengan* lebih dari 3 *layers*, kita pun bisa menurunkan persamaannya. Secara umum, proses melatih ANN (apapun variasi arsitekturnya) mengikuti *framework perceptron training rule* (subbab 9.2).

(2) Hidden to Output

$$v_i = \sigma\left(\sum_{j=1}^{H} o_j u_{j,i} + \gamma_i\right)$$

(3) Output to Hidden

$$\delta_i = (y_i - v_i)v_i(1 - v_i)$$
$$\Delta u_{j,i} = -\eta(t)\delta_i o_j$$
$$\Delta\gamma_i = -\eta(t)\delta_i$$

(1) Input to Hidden Layer

$$o_j = \sigma\left(\sum_{k=1}^{K} x_k w_{k,j} + \beta_j\right)$$

(4) Hidden to Input

$$\varphi_j = \sum_{i=1}^{H} \delta_i u_{j,i} o_j (1 - o_j)$$
$$\Delta w_{k,j} = -\eta(t)\varphi_j x_k$$
$$\Delta\beta_j = -\eta(t)\varphi_j$$

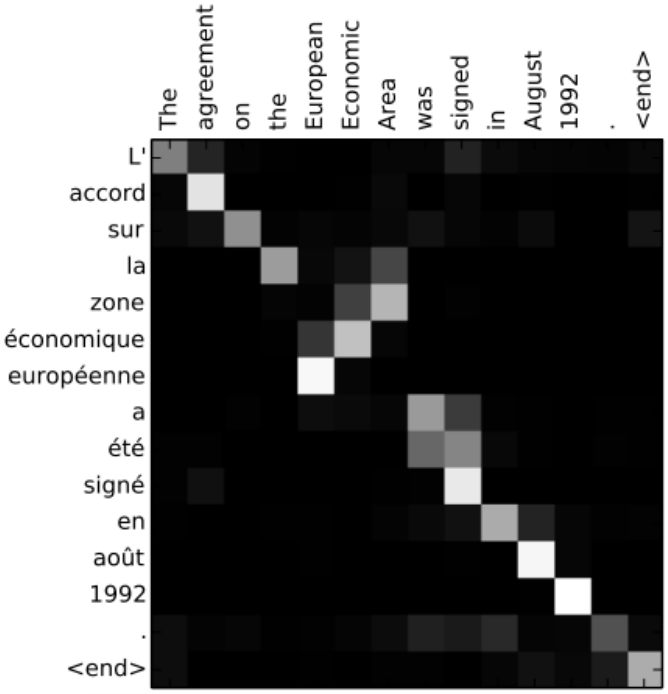
Gambar 9.6. Proses latihan MLP menggunakan *backpropagation*.

9.5 Interpretability

Interpretability ada dua macam yaitu *model interpretability* (i.e., apakah struktur model pembelajaran mesin dapat dipahami) dan *prediction interpretability* (i.e., bagaimana memahami dan memverifikasi cara *input* dipetakan menjadi *output*) [47]. Contoh teknik pembelajaran mesin yang mudah diinterpretasikan baik secara struktur dan prediksinya adalah *decision tree* (bab 6.2). Struktur *decision tree* berupa pohon keputusan mudah dimengerti oleh manusia dan prediksi (keputusan) dapat dilacak (*trace*). Seperti yang sudah dijelaskan pada bagian pengantar, ANN (MLP) biasanya dianggap sebagai metode *black box* atau susah untuk dinterpretasikan (terutama *model interpretability*-nya). Hal ini disebabkan oleh kedalaman (*depth*) yaitu memiliki beberapa *layer* dan *non-linearities*. Suatu unit pada *output layer* dipengaruhi oleh kombinasi (*arbitrary combination*) banyak parameter pada *layers* sebelumnya yang dilewatkan pada fungsi non-linear. Sulit untuk mengetahui bagaimana pengaruh bobot suatu unit pada suatu *layer* berpengaruh pada *output layer*, beserta bagaimana pengaruh kombinasi bobot. Berbeda dengan model linear, kita tahu bobot setiap *input* (dalam bentuk *feature vector*). Salah satu arah riset adalah mencari cara agar keputusan (karena struktur lebih susah, setidaknya beranjak dari keputusan terlebih dahulu) yang dihasilkan oleh ANN dapat dijelaskan [48], salah satu contoh nyata adalah *attention mechanism* [49, 50] (subbab 11.4.4) untuk *prediction interpretability*. Survey tentang *interpretability* dapat dibaca pada *paper* oleh Doshi-Velez dan Kim [51].

Cara paling umum untuk menjelaskan keputusan pada ANN adalah menggunakan *heat map*. Sederhananya, kita lewatkan suatu data $\mathbf{x}$ pada ANN, kemudian kita lakukan *feed-forward* sekali (misal dari *input* ke *hidden layer* dengan parameter $\mathbf{W}$). Kemudian, kita visualisasikan $\mathbf{x} \cdot \mathbf{W}$ (ilustrasi pada

Gambar. 9.7). Dengan ini, kita kurang lebih dapat mengetahui bagian *input* mana yang berpengaruh terhadap keputusan di *layer* berikutnya.



Gambar 9.7. Contoh *heat map* (*attention mechanism*) pada mesin translasi.

9.6 Binary Classification

Salah satu strategi untuk *binary classification* adalah dengan menyediakan hanya satu *output unit* di jaringan. Kelas pertama direpresentasikan dengan -1, kelas kedua direpresentasikan dengan nilai 1 (setelah diaktivasi). Hal ini dapat dicapai dengan fungsi non-linear seperti *sign*[7]. Apabila kita tertarik dengan probabilitas masuk ke dalam suatu kelas, kita dapat menggunakan fungsi seperti sigmoid[8] atau tanh[9].

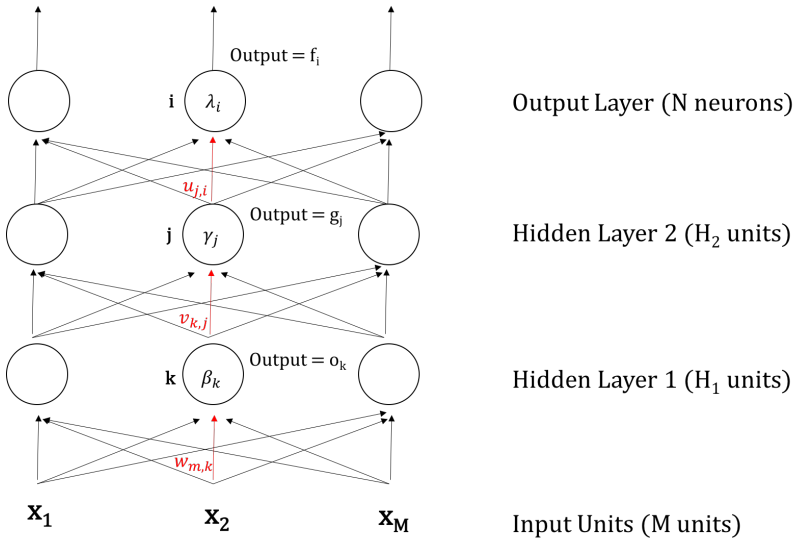
[7] `https://en.wikipedia.org/wiki/Sign_function`
[8] `https://en.wikipedia.org/wiki/Sigmoid_function`
[9] `https://en.wikipedia.org/wiki/Hyperbolic_function`

9.7 Multi-label Classification

Multilayer perceptron dapat memiliki *output unit* berjumlah lebih dari satu. Seumpama kita mempunyai empat kelas, dengan demikian kita dapat merepresentasikan keempat kelas tersebut empat *output units*. Kelas pertama direpresentasikan dengan unit pertama, kelas kedua dengan unit kedua, dst. Untuk C kelas, kita dapat merepresentasikannya dengan C *output units*. Kita dapat merepresentasikan data harus dimasukkan ke kelas mana menggunakan *sparse vector*, yaitu bernilai 0 atau 1. Elemen ke-i bernilai 1 apabila data masuk ke kelas c_i, sementara nilai elemen lainnya adalah 0 (ilurasi pada Gambar. 9.8).*Output* ANN dilewatkan pada suatu fungsi softmax yang melambangkan probabilitas *class-assignment*; i.e., kita ingin *output* agar semirip mungkin dengan *sparse vector-desired output*.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{array}{l} \text{Kelas pertama} \\ \text{Kelas kedua} \\ \text{Kelas ketiga} \\ \text{Kelas keempat} \end{array}$$

Gambar 9.8. Ilustrasi representasi *desired output* pada *Multi-label Classification*.



Gambar 9.9. *Deep Neural Network.*

9.8 Deep Neural Network

Deep Neural Network (DNN) adalah *artificial neural network* yang memiliki banyak *layer*. Pada umumnya, *deep neural network* memiliki lebih dari 3 *layers* (*input layer*, *N hidden layers*, *output layer*), dengan kata lain adalah MLP dengan lebih banyak *layer*. Karena ada relatif banyak layer, disebutlah *deep*. Proses pembelajaran pada DNN disebut sebagai *deep learning*[10] [9]. Jaringan *neural network* pada DNN disebut *deep network*.

Perhatikan Gambar. 9.9 yang memiliki 4 *layers*. Cara menghitung *final output* sama seperti MLP, diberikan pada persamaan 9.11 dimana β, γ, λ adalah *noise* atau *bias*.

$$f_i = \sigma\left(\sum_{j=1}^{H_2} u_{j,i}\sigma\left(\sum_{k=1}^{H_1} v_{k,j}\sigma\left(\sum_{m=1}^{M} x_m w_{m,k} + \beta_k\right) + \gamma_j\right) + \lambda_i\right) \tag{9.11}$$

Cara melatih *deep neural network*, salah satunya dapat menggunakan *backpropagation*. Seperti pada bagian sebelumnya, kita hanya perlu menurunkan rumusnya saja. **Penurunan diserahkan pada pembaca sebagai latihan**. Hasil proses penurunan dapat dilihat pada Gambar. 9.10.

(3) Hidden 2 to Output

$$f_i = \sigma\left(\sum_{j=1}^{H_2} g_j u_{j,i} + \lambda_i\right)$$

(4) Output to Hidden 2

$$\delta_i = (y_i - f_i) f_i (1 - f_i)$$
$$\Delta u_{j,i} = -\eta(t)\delta_i g_j$$
$$\Delta\lambda_i = -\eta(t)\delta_i$$

(2) Hidden 1 to Hidden 2

$$g_j = \sigma\left(\sum_{k=1}^{H_1} o_k v_{k,j} + \gamma_j\right)$$

(5) Hidden 2 to Hidden 1

$$\varphi_j = \sum_{i=1}^{H_2} \delta_i u_{j,i} g_j (1 - g_j)$$
$$\Delta v_{k,j} = -\eta(t)\varphi_j o_k$$
$$\Delta\gamma_j = -\eta(t)\varphi_j$$

(1) Input to Hidden Layer

$$o_k = \sigma\left(\sum_{m=1}^{M} x_m w_{m,k} + \beta_k\right)$$

(6) Hidden 1 to Input

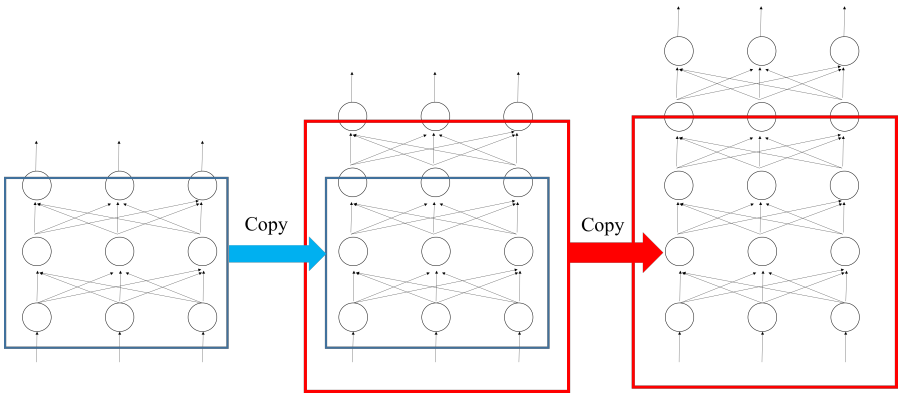
$$\mu_k = \sum_{j=1}^{H_1} \varphi_j v_{k,j} o_k (1 - o_k)$$
$$\Delta w_{m,k} = -\eta(t)\mu_k x_m$$
$$\Delta\beta_k = -\eta(t)\beta_k$$

Gambar 9.10. Proses latihan DNN menggunakan *backpropagation*.

[10] Hanya istilah keren saja, tak ada arti spesial!

Deep network terdiri dari banyak *layer* dan *synapse weight*, karenanya estimasi parameter susah dilakukan. Arti filosofisnya adalah susah/lama untuk menentukan relasi antara *input* dan *output*. Walaupun *deep learning* sepertinya kompleks, tetapi entah kenapa dapat bekerja dengan baik untuk permasalahan praktis [9]. *Deep learning* dapat menemukan relasi "tersembunyi" antara *input* dan *output*, yang tidak dapat diselesaikan menggunakan *multi-layer perceptron* (3 layers). Banyak orang percaya *deep neural network* lebih baik dibanding *neural network* yang lebar tapi sedikit *layer*, karena terjadi lebih banyak transformasi. Maksud lebih banyak transformasi adalah kemampuan untuk merubah *input* menjadi suatu representasi (tiap *hidden layer* dapat dianggap sebagai salah satu bentuk representasi *input*) dengan langkah *hierarchical*. Seperti contoh permasalahan XOR, permasalahan *non-linearly separable* pun dapat diselesaikan apabila kita dapat mentransformasi data (representasi data) ke dalam bentuk *linearly separable* pada ruang yang berbeda. Keuntungan utama *deep learning* adalah mampu merubah data dari *non-linearly separable* menjadi *linearly separable* melalui serangkaian transformasi (*hidden layers*).

Ada beberapa strategi untuk mempercepat pembelajaran menggunakan deep learning, misalnya: regularisasi, ***successive learning***, dan penggunaan **autoencoder** [9]. Sebagai contoh, saya akan menceritakan *successive learning*. Arti *successive learning* adalah jaringan yang dibangun secara bertahap. Misal kita latih ANN dengan 3 *layers*, kemudian kita lanjutkan 3 *layers* tersebut menjadi 4 *layers*, lalu kita latih lagi menjadi 5 *layers*, dst. Hal ini sesuai dengan [52], yaitu mulai dari hal kecil. Ilustrasinya dapat dilihat pada Gambar. 9.11. Menggunakan *deep learning* harus hati-hati karena pembelajaran cenderung *divergen* (artinya, *minimum square error* belum tentu semakin rendah seiring berjalannya waktu – *swing* relatif sering).



Gambar 9.11. Contoh *Succesive Learning*.

9.9 Tips

Pada contoh yang diberikan, *error* atau *loss* dihitung per tiap data point. Artinya begitu ada melewatkan suatu *input*, parameter langsung dioptimisasi sesuai dengan *loss*. Pada umumnya, hal ini tidak baik untuk dilakukan karena ANN menjadi tidak stabil. Metode yang lebih baik digunakan adalah teknik *minibatches*. Yaitu mengoptimisasi parameter untuk beberapa buah *inputs*. Jadi, update parameter dilakukan per *batch*. Data mana saja yang dimasukkan ke suatu *batch* dalam dipilih secara acak. Seperti yang mungkin kamu sadari secara intuitif, urutan data yang disajikan saat *training* mempengaruhi kinerja ANN. Pengacakan ini menjadi penting agar ANN mampu mengeneralisasi dengan baik. Kita dapat mengatur laju pembelajaran dengan menggunakan *learning rate*. Selain menggunakan *learning rate*, kita juga dapat menggunakan *momentum* (subbab 5.7).

Pada library/API *deep learning*, *learning rate* pada umumnya berubah-ubah sesuai dengan waktu. Selain itu, tidak ada nilai khusus (*rule-of-thumb*) untuk *learning rate* terbaik. Pada umumnya, kita inisiasi *learning rate* dengan nilai $\{0.001, 0.01, 0.1, 1\}$ [1]. Kemudian, kita mencari parameter terbaik dengan metode ***grid-search***[11], yaitu dengan mencoba-coba parameter secara *exhaustive* (*brute-force*) kemudian memilih parameter yang memberikan kinerja terbaik.

ANN sensitif terhadap inisialisasi parameter, dengan demikian banyak metode inisialisasi parameter misalkan, nilai *synapse weights* diambil dari distribusi binomial (silahkan eksplorasi lebih lanjut). Dengan hal ini, kinerja ANN dengan arsitektur yang sama dapat berbeda ketika dilatih ulang dari awal. Untuk menghindari *bias* inisialisasi parameter, biasanya ANN dilatih beberapa kali (umumnya 5, 10, atau 15 kali). Kinerja ANN yang dilaporkan adalah nilai kinerja rata-rata dan varians (*variance*). Untuk membandingkan dua arsitektur ANN pada suatu dataset, kita dapat menggunakan *two sample t-test unequal variance* (arsitektur X lebih baik dari arsitektur Y secara signifikan dengan nilai $p < V$).

Apabila kamu pikir dengan seksama, ANN sebenarnya melakukan transformasi non-linear terhadap *input* hingga menjadi *output*. Parameter diperbarahui agar transformasi non-linear *input* bisa menjadi semirip mungkin dengan *output* yang diharapkan. Dengan hal ini, istilah "ANN" memiliki asosiasi yang dekat dengan "transformasi non-linear". Kami ingin kamu mengingat, ANN (apapun variasi arsitekturnya) adalah **gabungan fungsi non-linear**, dengan demikian ia mampu mengaproksimasi fungsi non-linear (*decision boundary* dapat berupa fungsi non-linear).

Deep learning menjadi penting karena banyaknya transformasi (banyaknya *hidden layers*) lebih penting dibanding lebar jaringan. Seringkali (pada permasalahan praktis), kita membutuhkan banyak transformasi agar *input* bisa menjadi *output*. Setiap transformasi (*hidden layer*) merepresentasikan *input*

[11] `https://en.wikipedia.org/wiki/Hyperparameter_optimization`

menjadi suatu representasi. Dengan kata lain, *hidden layer* satu dan *hidden layer* lainnya mempelajari bentuk representasi atau karakteristik *input* yang berbeda.

Curriculum learning juga adalah tips yang layak disebutkan (*mention*) [53]. Penulis tidak mengerti detilnya, sehingga pembaca diharapkan membaca sendiri. Intinya adalah memutuskan apa yang harus ANN pelajari terlebih dahulu (mulai dari mempelajari hal mudah sebelum mempelajari hal yang susah).

9.10 Regularization and Dropout

Seperti yang sudah dijelaskan pada model linear. Kita ingin model mengeneralisasi dengan baik (kinerja baik pada *training data* dan *unseen examples*). Kita dapat menambahkan fungsi regularisasi untuk mengontrol kompleksitas ANN. Regularisasi pada ANN cukup *straightforward* seperti regularisasi pada model linear (subbab 5.8). Kami yakin pembaca bisa mengeksplorasi sendiri.

Selain itu, agar ANN tidak "bergantung" pada satu atau beberapa *synapse weights* saja, kita dapat menggunakan *dropout*. *Dropout* berarti me-*nol*-kan nilai *synapse weights* dengan nilai *rate* tertentu. Misalkan kita *nol*-kan nilai 30% *synapse weights* (*dropout rate*= 0.3) secara random. Hal ini dapat dicapai dengan teknik *masking*, yaitu mengalikan *synapse weights* dengan suatu *mask*.

Ingat kembali ANN secara umum, persamaan 9.12 dimana $\mathbf{W}$ adalah *synapse weights*, $\mathbf{x}$ adalah *input* (atau secara umum, dapat merepresentasikan *hidden state* pada suatu layer), b adalah *bias* dan f adalah fungsi aktivasi (non-linear). Kita buat suatu *mask* untuk *synapse weights* seperti pada persamaan 9.13, dimana $\mathbf{p}$ adalah vektor dan $p_i = [0, 1]$ merepresentasikan *synapse weight* diikutsertakan atau tidak. $r\%$ (*dropout rate*) elemen vektor $\mathbf{p}$ bernilai 0. Biasanya $\mathbf{p}$ diambil dari *bernoulli distribution* [1]. Kemudian, saat *feed forward*, kita ganti *synapse weights* menggunakan *mask* seperti pada persamaan 9.14. Saat menghitung *backpropagation*, turunan fungsi juga mengikutsertakan *mask* (gradient di-*mask*). Kami sarankan untuk membaca *paper* oleh Srivastava et al. [54] tentang *dropout* pada ANN. Contoh implementasi *dropout* dapat dilihat pada pranala berikut[12].

$$o = f(\mathbf{x} \cdot \mathbf{W} + b) \tag{9.12}$$

$$\mathbf{W}' = \mathbf{p} \cdot \mathbf{W} \tag{9.13}$$

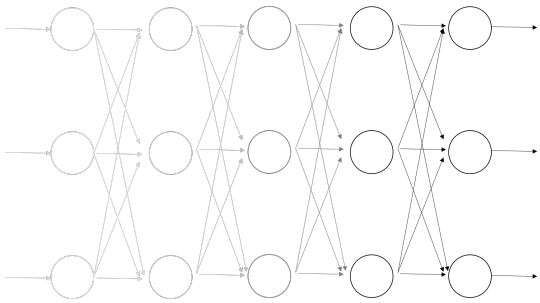
$$o = f(\mathbf{x} \cdot \mathbf{W}' + b) \tag{9.14}$$

Baik *regularization* maupun *dropout* sudah menjadi metode yang cukup "standar" dan diaplikasikan pada berbagai macam arsitektur (tidak terbatas pada MLP saja).

[12] `https://gist.github.com/yusugomori/cf7bce19b8e16d57488a`

9.11 Vanishing and Exploding Gradients

Pada beberapa kasus, nilai gradien ($\Delta\mathbf{W}$ - perubahan parameter) sangat kecil (mendekati 0 - *vanishing*) atau sangat besar (*explode*). *Vanishing gradient problem* umum terjadi untuk ANN yang sangat dalam (*deep*), yaitu memiliki banyak layer. Hal ini juga terjadi pada arsitektur khusus, seperti *recurrent neural network* saat diberikan input yang panjang [55]. Turunan suatu fungsi bernilai lebih kecil dari fungsi tersebut. Artinya nilai gradient pada *input layer* bernilai lebih kecil dari *output layer*. Apabila kita memiliki banyak *layer*, nilai gradient saat *backpropagation* mendekati 0 ketika diturunkan kembali dalam banyak proses. Ilustrasi *vanishing gradient* diberikan pada Gambar. 9.12 (analogikan dengan *heat map*). Saat melakukan *backpropagation*, nilai gradien menjadi mendekati 0 (warna semakin putih). Penanganan permasalahan ini masih merupakan topik riset tersendiri. Untuk saat ini, biasanya digunakan fungsi aktivasi *long short term memory* (LSTM) atau *gated recurrent unit* (GRU) untuk menanganinya. Selain nilai gradien, nilai *synapse weights* juga bisa sangat kecil atau sangat besar. Hal ini juga tidak baik!



Gambar 9.12. Ilustrasi *vanishing gradient problem*.

9.12 Rangkuman

Ada beberapa hal yang perlu kamu ingat, pertama-tama jaringan *neural network* terdiri atas:

1. *Input layer*
2. *Hidden layer(s)*
3. *Output layer*

Setiap *edge* yang menghubungkan suatu *node* dengan *node* lainnya disebut *synapse weight*. Pada saat melatih *neural network* kita mengestimasi nilai yang "bagus" untuk *synapse weights*.

Kedua, hal tersulit saat menggunakan *neural network* adalah menentukan topologi. Kamu bisa menggunakan berbagai macam variasi topologi *neural network* serta cara melatih untuk masing-masing topologi. Tetapi, suatu topologi tertentu lebih tepat untuk merepresentasikan permasalahan dibanding topologi lainnya. Menentukan tipe topologi yang tepat membutuhkan pengalaman.

Ketiga, proses *training* untuk *neural network* berlangsung lama. Secara umum, perubahan nilai *synapse weights* mengikuti tahapan (*stage*) berikut [9]:

1. *Earlier state*. Pada tahap ini, struktur global (kasar) diestimasi.
2. *Medium state*. Pada tahap ini, *learning* berubah dari tahapan global menjadi lokal (ingat *steepest gradient descent*).
3. *Last state*. Pada tahap ini, struktur detail sudah selesai diestimasi.

Neural network adalah salah satu *learning machine* yang dapat menemukan *hidden structure* atau pola data "implisit". Secara umum, *learning machine* tipe ini sering menjadi *overfitting/overtraining*, yaitu model memiliki kinerja sangat baik pada *training data*, tapi buruk pada *testing data/unseen example*. Oleh sebab itu, menggunakan *neural network* harus hati-hati.

Keempat, *neural network* dapat digunakan untuk *supervised*, *semi-supervised*, maupun *unsupervised learning*. Hal ini membuat *neural network* cukup populer belakangan ini karena fleksibilitas ini. Contoh penggunaan *neural network* untuk *unsupervised* learning akan dibahas pada bab 10. Semakin canggih komputer, maka semakin cepat melakukan perhitungan, dan semakin cepat melatih *neural network*. Hal ini adalah kemewahan yang tidak bisa dirasakan 20-30 tahun lalu.

Soal Latihan

9.1. Turunan

(a) Turunkanlah perubahan *noise/bias* untuk *training* pada MLP.
(b) Turunkanlah proses *training deep neural network* pada Gambar. 9.10 termasuk perubahan *noise/bias*.

9.2. Neural Network Training

(a) Sebutkan dan jelaskan cara lain untuk melatih *artificial neural network* (selain *backpropagation*) (bila ada)!
(b) Apa kelebihan dan kekurangan *backpropagation*?
(c) Tuliskan persamaan MLP dengan menggunakan momentum! (kemudian berikan juga *backpropagation*-nya)

9.3. Neural Network - Unsupervised Learning

Bagaimana cara menggunakan *artificial neural network* untuk *unsupervised learning*?

9.4. Regularization Technique

(a) Sebutkan dan jelaskan teknik *regularization* untuk *neural network*! (dalam bentuk formula)
(b) Mengapa kita perlu menggunakan teknik tersebut?

9.5. Softmax Function

(a) Apa itu *softmax function*?
(b) Bagaimana cara menggunakan *softmax function* pada *neural network*?
(c) Pada saat kapan kita menggunakan fungsi tersebut?
(d) Apa kelebihan fungsi tersebut dibanding fungsi lainnya?

9.6. Transformasi atribut

Pada bab 4, diberikan contoh klasifikasi dengan data dengan atribut nominal. Akan tetapi, secara alamiah *neural network* membutuhkan data dengan atribut numerik untuk klasifikasi. Jelaskan konversi/strategi penanganan atribut nominal pada *neural network*!

10

Dimensionality Reduction dan Representation Learning

> "The goal is to turn data into information, and information into insight."
>
> Carly Fiorina

Bab ini memuat materi yang relatif sulit (karena agak *high level*). Bab ini memuat materi autoencoder serta penerapannya pada pemrosesan bahasa alami (***natural language processing*** - NLP). Berhubung aplikasi yang diceritakan adalah aplikasi pada NLP, kami akan memberi sedikit materi (*background knowledge*) agar bisa mendapat gambaran tentang persoalan pada domain tersebut. Bagi yang tertarik belajar NLP, kami sarankan untuk membaca buku [56]. Teknik yang dibahas pada bab ini adalah ***representation learning*** untuk melakukan pengurangan dimensi pada *feature vector* (*dimensionality reduction*), teknik ini biasanya digolongkan sebagai *unsupervised learning*. Artinya, *representation learning* adalah mengubah suatu representasi menjadi bentuk representasi lain yang ekuvalen, tetapi berdimensi lebih rendah; sedemikian sehingga informasi yang terdapat pada representasi asli tidak hilang/terjaga. Ide dasar teknik ini bermula dari aljabar linear, yaitu dekomposisi matriks.

10.1 Curse of Dimensionality

Pada bab model linear, kamu telah mempelajari ide untuk mentransformasi data menjadi dimensi lebih tinggi agar data tersebut menjadi *linearly separable*. Pada bab ini, kamu mempelajari hal sebaliknya, yaitu mengurangi dimensi. *Curse of dimensionality* dapat dipahami secara mendalam apabila kamu membaca buku [57]. Untuk melakukan klasifikasi maupun *clustering*, kita membutuhkan fitur. Fitur tersebut haruslah dapat membedakan satu *instance* dan *instance* lainnya. Seringkali, untuk membedakan *instance* satu

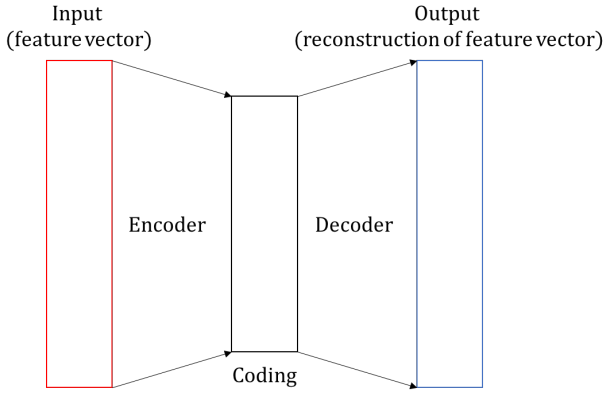
dan *instance* lainnya, kita membutuhkan *feature vector* yang berdimensi relatif "besar". Karena dimensi *feature vector* besar, kita butuh sumber daya komputasi yang besar juga. Untuk itu, terdapat metode-metode ***feature selection***[1] untuk memilih fitur-fitur yang dianggap "representatif" dibanding fitur lainnya. Sayangnya, bila kita menggunakan metode-metode *feature selection* ini, tidak jarang kita kelihangan informasi yang memuat karakteristik data. Dengan kata lain, ada karakteristik yang hilang saat menggunakan *feature selection*. Karena alasan tersebut, permasalahan ini disebut "*curse*".

Pertanyaan yang kita ingin jawab adalah apakah ada cara untuk merepresentasikan data ke dalam bentuk yang membutuhkan memori lebih sedikit tanpa adanya kehilangan informasi? Kita bisa menggunakan cara lain untuk mengurangi kompleksitas komputasi adalah dengan melakukan kompresi *feature vector*. ***Representation learning*** adalah metode untuk melakukan **kompresi** *feature vector*, umumnya (*typically*) menggunakan *neural network*[2]. Proses melakukan kompresi disebut ***encoding***, hasil *feature vector* dalam bentuk terkompres disebut ***coding***, proses mengembalikan hasil kompresi ke bentuk awal disebut (atau secara lebih umum, proses menginterpretasikan *coding*) ***decoding***. *Neural network* yang mampu melakukan hal ini disebut ***encoder*** [58, 59, 60, 61, 62]. Contoh *representation learning* paling sederhana kemungkinan besar adalah ***autoencoder*** yaitu *neural network* yang dapat merepresentasikan data kemudian merekonstruksinya kembali. Ilustrasi autoencoder dapat dilihat pada Gambar. 10.1. Karena tujuan *encoder* untuk kompresi, bentuk terkompresi haruslah memiliki dimensi lebih kecil dari dimensi *input*. *Neural network* mampu melakukan "kompresi" dengan baik karena ia mampu menemukan *hidden structure* dari data. Ukuran *utility function* atau *performance measure* untuk *autoencoder* adalah mengukur *loss*. Idealnya, *output* harus sama dengan *input*, yaitu *autoencoder* dengan tingkat *loss* 0%.

Contoh klasik lainnya adalah ***N-gram language modelling***, yaitu memprediksi kata y_t diberikan suatu konteks (*surrounding words*) misal kata sebelumnya y_{t-1} (bigram). Apabila kita mempunyai *vocabulary* sebesar $40,000$ berarti suatu bigram model membutuhkan *memory* sebesar $40,000^2$ (kombinatorial). Apabila kita ingin memprediksi kata diberikan *history* yang lebih panjang (misal dua kata sebelumnya - trigram) maka kita membutuhkan *memory* sebesar $40,000^3$. Artinya, *memory* yang dibutuhkan berlipat secara eksponensial. Tetapi, terdapat strategi menggunakan *neural network* dimana parameter yang dibutuhkan tidak berlipat secara eksponensial walau kita ingin memodelkan konteks yang lebih besar [63].

[1] `http://scikit-learn.org/stable/modules/feature_selection.html`

[2] Istilah *representation learning* pada umumnya mengacu dengan teknik menggunakan *neural network*.



Gambar 10.1. Contoh autoencoder sederhana.

10.2 Singular Value Decomposition

Sebelum masuk ke *autoencoder* secara matematis, penulis akan memberikan sedikit *overview* tentang dekomposisi matriks. Seperti yang sudah dijelaskan pada bab-bab sebelumnya, dataset dimana setiap instans direpresentasikan oleh *feature vector* dapat disusun menjadi matriks $\mathbf{D}$ berukuran $M \times N$, dimana M adalah banyaknya instans[3] dan N adalah dimensi fitur. Pada *machine learning*, dekomposisi atau reduksi dimensi sangat penting dilakukan terutama ketika dataset berupa *sparse matrix*. Matriks $\mathbf{D}$ dapat difaktorisasi menjadi tiga buah matriks, dimana operasi ini berkaitan dengan mencari eigenvectors, diilustrasikan pada persamaan 10.1.

$$\mathbf{D} = \mathbf{U}\ \mathbf{V}\ \mathbf{W} \tag{10.1}$$

dimana $\mathbf{U}$ berukuran $M \times L$, $\mathbf{V}$ berukuran $L \times L$, dan $\mathbf{W}$ berukuran $L \times N$. Perlu diperhatikan, matriks $\mathbf{V}$ adalah sebuah diagonal matriks (elemennya adalah nilai eigenvectors dari $\mathbf{D}$). Misalkan kita mempunyai sebuah matriks lain $\hat{\mathbf{V}}$ berukuran $K \times K$ ($K < L$), yaitu modifikasi matriks $\mathbf{V}$ dengan mengganti sejumlah elemen diagonalnya menjadi 0 (analogi seperti menghapus beberapa elemen yang dianggap kurang penting). Sebagai contoh, perhatikan ilustrasi berikut untuk $L = 3$ dan $K = 2$!

$$\mathbf{V} = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_L \end{bmatrix} \quad \hat{\mathbf{V}} = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_K & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Kita juga dapat me-nol-kan sejumlah baris dan kolom pada matriks $\mathbf{U}$ dan $\mathbf{W}$ menjadi $\hat{\mathbf{U}}$ ($M \times K$) dan $\hat{\mathbf{W}}$ ($K \times N$). Apabila kita mengalikan semuanya,

[3] Dapat dianalogikan dengan banyaknya training data.

kita akan mendapat matriks $\hat{\mathbf{D}}$ yang disebut *low rank approximation* dari matriks asli $\mathbf{D}$, seperti diilustrasikan pada persamaan 10.2.

$$\hat{\mathbf{D}} = \hat{\mathbf{U}}\ \hat{\mathbf{V}}\ \hat{\mathbf{W}} \tag{10.2}$$

Suatu baris dari matriks $\mathbf{E} = \hat{\mathbf{U}}\ \hat{\mathbf{V}}$ dianggap sebagai aproksimasi baris matriks $\mathbf{D}$ berdimensi tinggi [1]. Artinya, menghitung *dot-product* $\mathbf{E}_i \cdot \mathbf{E}_j = \hat{\mathbf{D}}_i \cdot \hat{\mathbf{D}}_j$. Artinya, operasi pada matriks aproksimasi (walaupun berdimensi lebih rendah), kurang lebih melambangkan operasi pada matriks asli. Konsep ini menjadi fundamental *autoencoder* yang akan dibahas pada bab berikutnya.

10.3 Ide Dasar Autoencoder

Seperti yang sudah dijelaskan *autoencoder* adalah *neural network* yang mampu merekonstruksi *input*. Ide dasar *autoencoder* tidak jauh dari konsep dekomposisi/*dimentionality reduction* menggunakan *singular value decomposition*. Diberikan dataset $\mathbf{D}$, kita ingin mensimulasikan pencarian matriks $\hat{\mathbf{D}}$ yang merupakan sebuah *low rank approximation* dari matriks asli. Arsitektur dasar *autoencoder* diberikan pada Gambar. 10.1. Kita memberi input matriks $\mathbf{D}$ pada *autoencoder*, kemudian ingin *autoencoder* tersebut menghasilkan matriks yang sama. Dengan kata lain, *desired output* sama dengan *input*. Apabila dihubungkan dengan pembahasan ANN pada bab sebelumnya, *error function* untuk melatih *autoencoder* diberikan pada persamaan 10.3, dimana $\mathbf{o}$ adalah output dari jaringan, θ adalah ANN (kumpulan *weight matrices*)[4], N adalah dimensi *output*, dan $\mathbf{d}_i$ adalah data ke-i (*feature vector* ke-i).

$$E(\theta) = \frac{1}{N} \sum_{i=j}^{N} \left(\mathbf{d}_{i[j]} - \mathbf{o}_{i[j]}\right)^2 \tag{10.3}$$

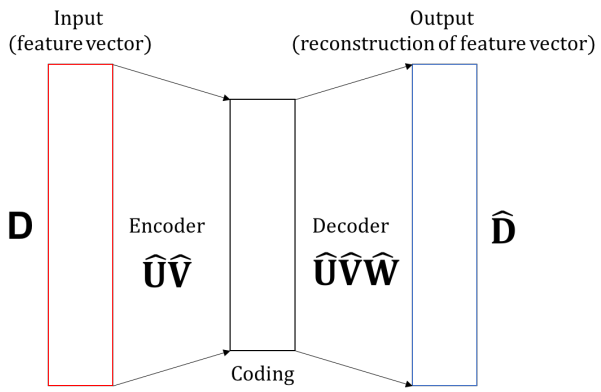
Persamaan 10.3 dapat kita tulis kembali sebagai persamaan 10.4, dimana f melambangkan fungsi aktivasi.

$$E(\theta) = \frac{1}{N} \sum_{j=1}^{N} \left(\mathbf{d}_{i[j]} - f(\mathbf{d}_i, \theta)_{[j]}\right)^2 \tag{10.4}$$

Seperti yang sudah dijelaskan sebelumnya, *desired output* sama dengan *input*. Tetapi seperti yang kamu ketahui, mencapai *loss* sebesar 0% adalah hal yang susah. Dengan demikian, kamu dapat memahami secara intuitif bahwa *autoencoder* melakukan aproksimasi terhadap data asli. Gambar. 10.2 mengilustrasikan hubungan antara *autoencoder* dan *singular value decomposition*[5]. Perhatikan, *hidden layer/coding* adalah $\mathbf{E} = \hat{\mathbf{U}}\ \hat{\mathbf{V}}$. Dengan kata lain, kita

[4] Pada banyak literatur, kumpulan *weight matrices* ANN sering dilambangkan dengan θ

[5] Hanya sebuah analogi.



Gambar 10.2. Hubungan autoencoder dan *singular value decomposition*.

dapat melakukan operasi *dot-product* pada *coding* untuk merepresentasikan *dot-product* pada data asli **D**. Ini adalah ide utama *autoencoder*, yaitu mengaproksimasi/mengkompresi data asli menjadi bentuk lebih kecil *coding*. Kemudian, operasi pada bentuk *coding* merepresentasikan operasi pada data sebenarnya.

Autoencoder terdiri dari *encoder* (sebuah *neural network*) dan *decoder* (sebuah *neural network*). *Encoder* merubah *input* ke dalam bentuk dimensi lebih kecil (dapat dianggap sebagai kompresi). *Decoder* berusaha merekonstruksi *coding* menjadi bentuk aslinya.

Sekarang kamu mungkin bertanya-tanya, bila *autoencoder* melakukan hal serupa seperti *singular value decomposition*, untuk apa kita menggunakan *autoencoder*? (mengapa tidak menggunakan aljabar saja?). Berbeda dengan teknik SVD, teknik *autoencoder* dapat juga mempelajari fitur non-linear[6]. Pada penggunaan praktis, *autoencoder* adalah *neural network* yang cukup kompleks (memiliki banyak *hidden layer*). Dengan demikian, kita dapat "mengetahui" berbagai macam representasi atau transformasi data. *Framework autoencoder* yang disampaikan sebelumnya adalah *framework* dasar. Pada kenyataannya, masih banyak ide lainnya yang bekerja dengan prinsip yang sama untuk mencari *coding* pada permasalahan khusus. *Output* dari *neural network* juga bisa tidak sama *input*-nya, tetapi tergantung permasalahan (kami akan memberikan contoh persoalan *word embedding*). Selain itu, *autoencoder* juga relatif fleksibel; dalam artian saat menambahkan data baru, kita hanya perlu memperbaharui parameter *autoencoder* saja. Kami sarankan untuk membaca *paper* [64, 65] perihal penjelasan lebih lengkap tentang perbedaan dan persamaan SVD dan *autoencoder* secara lebih matematis.

Apabila kamu hanya ingin mengerti konsep dasar *representation learning*, kamu dapat berhenti membaca sampai subbab ini. Secara sederhana

[6] Hal ini abstrak untuk dijelaskan karena membutuhkan pengalaman.

representation learning adalah teknik untuk mengkompresi *input* ke dalam dimensi lebih rendah tanpa (diharapkan) ada kehilangan informasi. Operasi vektor (dan lainnya) pada level *coding* merepresentasikan operasi pada bentuk aslinya. Untuk pembahasan *autoencoder* secara lebih matematis, kamu dapat membaca pranala ini[7]. Apabila kamu ingin mengetahui lebih jauh contoh penggunaan *representation learning* secara lebih praktis, silahkan lanjutkan membaca materi subbab berikutnya. Buku ini akan memberikan contoh penggunaan *representation learning* pada bidang *natural language processing* (NLP).

10.4 Representing Context: Word Embedding

Pada domain NLP, kita ingin komputer mampu mengerti bahasa selayaknya manusia mengerti bahasa. Misalkan komputer mampu mengetahui bahwa "meja" dan "kursi" memiliki hubungan yang erat. Hubungan seperti ini tidak dapat terlihat berdasarkan teks tertulis, tetapi kita dapat menyusun kamus hubungan kata seperti WordNet[8]. WordNet memuat ontologi kata seperti hipernim, antonim, sinonim. Akan tetapi, hal seperti ini tentu sangat melelahkan, seumpama ada kata baru, kita harus memikirkan bagaimana hubungan kata tersebut terhadap seluruh kamus yang sudah dibuat. Pembuatan kamus ini memerlukan kemampuan para ahli linguistik.

Oleh sebab itu, kita harus mencari cara lain untuk menemukan hubungan kata ini. Ide utama untuk menemukan hubungan antarkata adalah ***statistical semantics hypothesis*** yang menyebutkan pola penggunaan kata dapat digunakan untuk menemukan arti kata [66]. Contoh sederhana, kata yang muncul pada "**konteks**" yang sama cenderung memiliki makna yang sama. Perhatikan "konteks" dalam artian NLP adalah kata-kata sekitar (*surrounding words*)[9]; contohnya kalimat "budi menendang bola", "konteks" dari "bola" adalah "budi menendang". Kata "cabai" dan "permen" pada kedua kalimat "budi suka cabai" dan "budi suka permen" memiliki kaitan makna, dalam artian keduanya muncul pada konteks yang sama. Sebagai manusia, kita tahu ada keterkaitan antara "cabai" dan "permen" karena keduanya bisa dimakan.

Berdasarkan hipotesis tersebut, kita dapat mentransformasi kata menjadi sebuah bentuk matematis dimana kata direpresentasikan oleh pola penggunaannya [56]. Arti kata ***embedding*** adalah transformasi kata (beserta konteksnya) menjadi bentuk matematis (vektor). "Kedekatan hubungan makna" (***semantic relationship***) antarkata kita harapkan dapat tercermin pada operasi vektor. Salah satu metode sederhana untuk merepresentasikan kata sebagai vektor adalah ***Vector Space Model***. Konsep *embedding* dan *autoencoder*

[7] `https://jaan.io/what-is-variational-autoencoder-vae-tutorial/`

[8] `https://wordnet.princeton.edu/`

[9] Selain *surrounding words*, konteks dalam artian NLP dapat juga berupa kalimat, paragraph, atau dokumen.

	Dokumen 1	Dokumen 2	Dokumen 3	Dokumen 4	···
King	1	0	0	0	···
Queen	0	1	0	1	···
Prince	1	0	1	0	···
Princess	0	1	0	1	···
···					

Tabel 10.1. Contoh 1-of-V encoding.

sangatlah dekat, tapi kami ingin menakankan bahwa *embedding* adalah bentuk representasi konteks.

Semantic relationship dapat diartikan sebagai ***attributional*** atau ***relational similarity***. *Attributional similarity* berarti dua kata memiliki atribut/sifat yang sama, misalnya anjing dan serigala sama-sama berkaki empat, menggongong, serta mirip secara fisiologis. *Relational similarity* berarti derajat korespondensi, misalnya *anjing* : *menggongong* memiliki hubungan yang erat dengan *kucing* : *mengeong*.

10.4.1 Vector Space Model

Vector space model (VSM)[10] adalah bentuk *embedding* yang relatif sudah cukup lama tapi masih digunakan sampai saat ini. Pada pemodelan ini, kita membuat sebuah matriks dimana baris melambangkan kata, kolom melambangkan dokumen. Metode VSM ini selain mampu menangkap hubungan antarkata juga mampu menangkap hubungan antardokumen (*to some degree*). Asal muasalnya adalah *statistical semantics hypothesis*. Tiap sel pada matriks berisi nilai 1 atau 0. 1 apabila $kata_i$ muncul di $dokumen_i$ dan 0 apabila tidak. Model ini disebut ***1-of-V/1-hot encoding*** dimana V adalah ukuran kosa kata. Ilustrasi dapat dilihat pada Tabel. 10.1.

Akan tetapi, *1-of-V encoding* tidak menyediakan banyak informasi untuk kita. Dibanding sangat ekstrim saat mengisi sel dengan nilai 1 atau 0 saja, kita dapat mengisi sel dengan frekuensi kemunculan kata pada dokumen, disebut ***term frequency*** (TF). Apabila suatu kata muncul pada banyak dokumen, kata tersebut relatif tidak terlalu "penting" karena muncul dalam berbagai konteks dan tidak mampu membedakan hubungan dokumen satu dan dokumen lainnya (***inverse document frequency***/IDF). Formula IDF diberikan pada persamaan 10.5. Tingkat kepentingan kata berbanding terbalik dengan jumlah dokumen dimana kata tersebut dimuat. N adalah banyaknya dokumen, $\|d\epsilon D; t\epsilon d\|$ adalah banyaknya dokumen dimana kata t muncul.

$$IDF(t, D) = log\left(\frac{N}{|d\epsilon D; t\epsilon d|}\right) \tag{10.5}$$

[10] Mohon bedakan dengan VSM (*vector space model*) dan SVM (*support vector machine*)

Dengan menggunakan perhitungan TF-IDF yaitu $TF*IDF$ untuk mengisi sel pada matriks Tabel. 10.1, kita memiliki lebih banyak informasi. TF-IDF sampai sekarang menjadi *baseline* pada ***information retrieval***. Misalkan kita ingin menghitung kedekatan hubungan antar dua dokumen, kita hitung ***cosine distance*** antara kedua dokumen tersebut (vektor suatu dokumen disusun oleh kolom pada matriks). Apabila kita ingin menghitung kedekatan hubungan antar dua kata, kita hitung *cosine distance* antara kedua kata tersebut dimana vektor suatu kata merupakan baris pada matriks. Tetapi seperti intuisi yang mungkin kamu miliki, mengisi *entry* dengan nilai TF-IDF pun akan menghasilkan *sparse matrix*.

Statistical semantics hypothesis diturunkan lagi menjadi empat macam hipotesis [66]:

1. *Bag of words*
2. *Distributional hypothesis*
3. *Extended distributional hypothesis*
4. *Latent relation hypothesis*

Silakan pembaca mencari sumber tersendiri untuk mengerti keempat hipotesis tersebut atau membaca *paper* Turney dan Pantel [66].

10.4.2 Sequential, Time Series, dan Compositionality

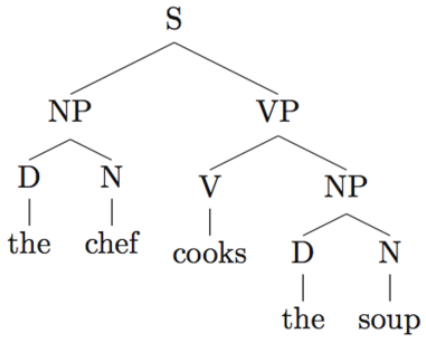
Bahasa manusia memiliki dua macam karakteristik yaitu adalah data berbentuk ***sequential data*** dan memenuhi sifat ***compositionality***. *Sequential data* adalah sifat data dimana suatu kemunculan $data_i$ dipengaruhi oleh data sebelumnya ($data_{i-1}, data_{i-2}, ...$). Perhatikan kedua kalimat berikut:

1. Budi melempar bola.
2. Budi melempar gedung bertingkat.

Pada kedua kalimat tersebut, kalimat pertama lebih masuk akal karena bagaimana mungkin seseorang bisa melempar "gedung bertingkat". Keputusan kita dalam memilih kata berikutnya dipengaruhi oleh kata-kata sebelumnya, dalam hal ini "Budi melempar" setelah itu yang lebih masuk akal adalah "bola". Contoh lain adalah data yang memiliki sifat ***time series*** yaitu gelombang laut, angin, dan cuaca. Kita ingin memprediksi data dengan rekaman masa lalu, tapi kita tidak mengetahui masa depan. Kita mampu memprediksi cuaca berdasarkan rekaman parameter cuaca pada hari-hari sebelumnya. Ada yang berpendapat beda *time series* dan *sequential* (sekuensial) adalah diketahuinya sekuens kedepan secara penuh atau tidak. Penulis tidak dapat menyebutkan *time series* dan sekuensial sama atau beda, silahkan pembaca menginterpretasikan secara bijaksana.

Data yang memenuhi sifat ***compositionality*** berarti memiliki struktur hirarkis. Struktur hirarkis ini menggambarkan bagaimana unit-unit lebih kecil berinteraksi sebagai satu kesatuan. Artinya, interpretasi/pemaknaan unit

yang lebih besar dipengaruhi oleh interpretasi/pemaknaan unit lebih kecil (subunit). Sebagai contoh, kalimat "saya tidak suka makan cabai hijau". Unit "cabai" dan "hijau" membentuk suatu frasa "cabai hijau". Mereka tidak bisa dihilangkan sebagai satu kesatuan makna. Kemudian interaksi ini naik lagi menjadi kegiatan "makan cabai hijau" dengan keterangan "tidak suka", bahwa ada seseorang yang "tidak suka makan cabai hijau" yaitu "saya". Pemecahan kalimat menjadi struktur hirarkis berdasarkan *syntactical role* disebut ***constituent parsing***, contoh lebih jelas pada Gambar. 10.3. *N* adalah *noun*, *D* adalah *determiner*, *NP* adalah *noun phrase*, *VP* adalah *verb phrase*, dan *S* adalah *sentence*. Selain bahasa manusia, gambar juga memiliki struktur hirarkis. Sebagai contoh, gambar rumah tersusun atas tembok, atap, jendela, dan pintu. Tembok, pintu, dan jendela membentuk bagian bawah rumah; lalu digabung dengan atap sehingga membentuk satu kesatuan rumah.



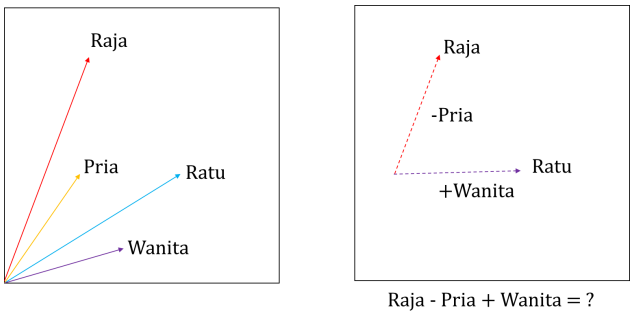
Gambar 10.3. Contoh Constituent Tree[11].

10.4.3 Distributed Word Representation

Seperti yang disebutkan pada bagian sebelumnya, kita ingin hubungan kata (yang diinferensi dari konteksnya) dapat direpresentasikan sebagai operasi vektor seperti pada ilustrasi Gambar. 10.4. Kata "raja" memiliki sifat-sifat yang dilambangkan oleh suatu vektor (misal 90% aspek loyalitas, 80% kebijaksanaan, 90% aspek kebangsaan, dst), begitu pula dengan kata "pria", "wanita", dan "ratu". Jika sifat-sifat yang dimiliki "raja" dihilangkan bagian sifat-sifat "pria"-nya, kemudian ditambahkan sifat-sifat "wanita" maka idealnya operasi ini menghasilkan vektor yang dekat kaitannya dengan "ratu". Dengan kata lain, raja yang tidak maskulin tetapi fenimin disebut ratu. Seperti yang disebutkan sebelumnya, ini adalah tujuan utama *embedding* yaitu merepresentasikan "makna" kata sebagai vektor sehingga kita dapat memanipulasi banyak hal berdasarkan operasi vektor. Hal ini mirip (tetapi tidak sama)

[11] source: Pinterest

dengan prinsip *singular value decomposition* dan *autoencoder* yang telah dijelaskan sebelumnya.

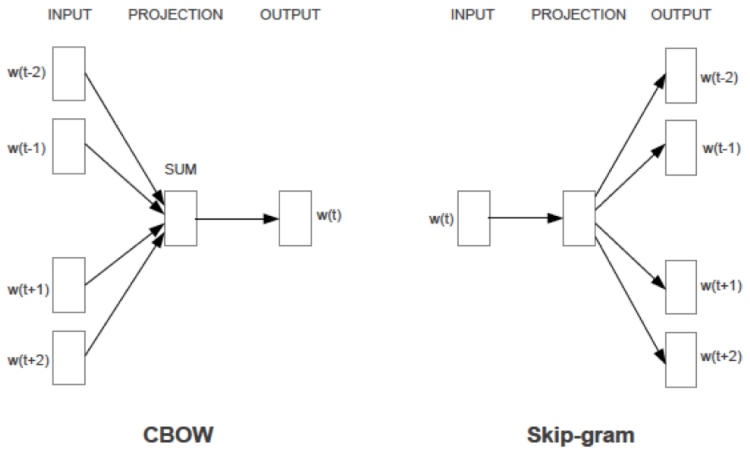


Gambar 10.4. Contoh Operasi Vektor Kata.

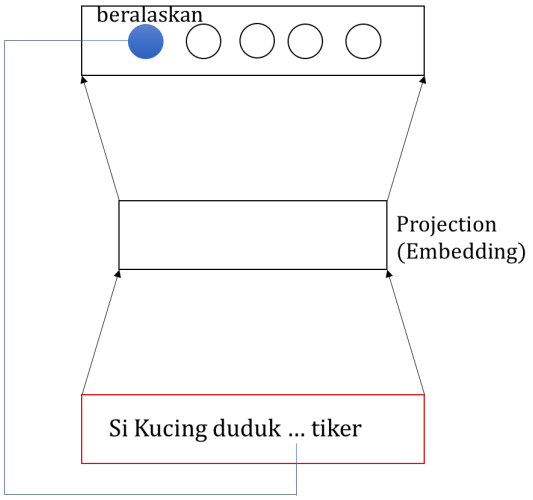
Selain *vector space model*, apakah ada cara lain yang mampu merepresentasikan kata dengan lebih baik? Salah satu kekurangan VSM adalah tidak memadukan sifat sekuensial pada konstruksi vektornya. Cara lebih baik ditemukan oleh [40, 41] dengan ekstensi pada [62]. Idenya adalah menggunakan teknik *representation learning* dan prinsip *statistical semantics hypothesis*. Metode ini lebih dikenal dengan sebutan *word2vec*. Tujuan *word2vec* masih sama, yaitu merepresentasikan kata sebagai vektor, sehingga kita dapat melakukan operasi matematis terhadap kata. *Encoder*-nya berbentuk ***Continous bag of words*(CBOW)** atau **Skip Gram**. Pada CBOW, kita memprediksi kata diberikan suatu "konteks". Pada arsitektur "Skip Gram" kita memprediksi konteks, diberikan suatu kata. Ilustrasi dapat dilihat pada Gambar. 10.5. Bagian *projection layer* pada Gambar. 10.5 adalah *coding* layer. Kami akan memberikan contoh CBOW secara lebih detil.

Perhatikan Gambar. 10.6. Diberikan sebuah konteks "si kucing duduk ... tiker". Kita harus menebak apa kata pada "..." tersebut. Dengan menggunakan teknik autoencoder, *output layer* adalah distribusi probabilitas $kata_i$ pada konteks tersebut. Kata yang menjadi jawaban adalah kata dengan probabilitas terbesar, misalkan pada kasus ini adalah "beralaskan". Dengan arsitektur ini, prinsip sekuensial atau *time series* dan *statistical semantics hypothesis* terpenuhi (*to a certain extent*). Teknik ini adalah salah satu contoh penggunaan *neural network* untuk *unsupervised learning*. Kita tidak perlu mengkorespondensikan kata dan *output* yang sesuai karena *input vektor* didapat dari statistik penggunaan kata. Agar lebih tahu kegunaan vektor kata, kamu dapat mencoba kode dengan bahasa pemrograman Python 2.7 yang disediakan penulis[12]. Buku ini telah menjelaskan ide konseptual *word embedding* pada level abstrak. Apabila kamu tertarik untuk memahami detilnya secara matem-

[12] `https://github.com/wiragotama/GloVe_Playground`



Gambar 10.5. CBOW vs Skip Gram [41].



Gambar 10.6. CBOW.

atis, kamu dapat membaca berbagai penelitian terkait[13]. Silahkan baca *paper* oleh Mikolov [40, 41] untuk detil implementasi *word embedding*.

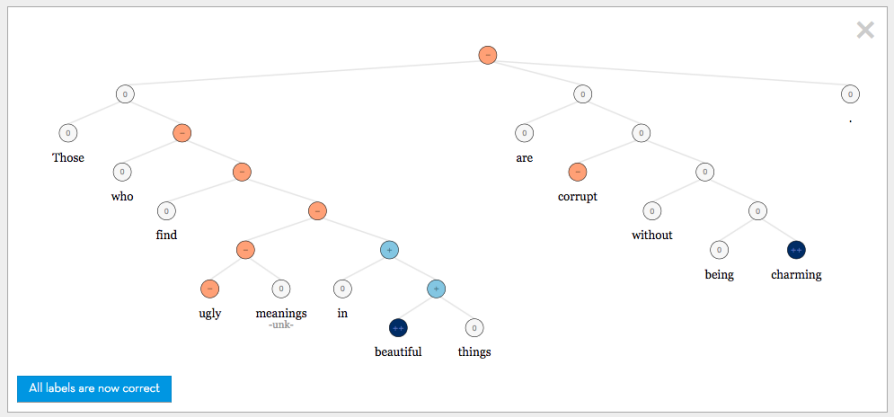
10.4.4 Distributed Sentence Representation

Kita sudah dapat merepresentasikan kata menjadi vektor, selanjutnya kita ingin mengonversi unit lebih besar (kalimat) menjadi vektor. Salah satu cara

[13] Beberapa orang berpendapat bahwa *evil is in the detail*.

paling mudah adalah menggunakan nilai rata-rata representasi *word embedding* untuk semua kata yang ada pada kalimat tersebut (*average of its individual word embeddings*). Cara ini sering digunakan pada bidang NLP dan cukup *powerful*, sebagai contoh pada *paper* oleh Putra dan Tokunaga [67]. Pada NLP, sering kali kalimat diubah terlebih dahulu menjadi vektor sebelum dilewatkan pada algoritma *machine learning*, misalnya untuk analisis sentimen (kalimat bersentimen positif atau negatif). Vektor ini yang nantinya menjadi *feature vector* bagi algoritma *machine learning*.

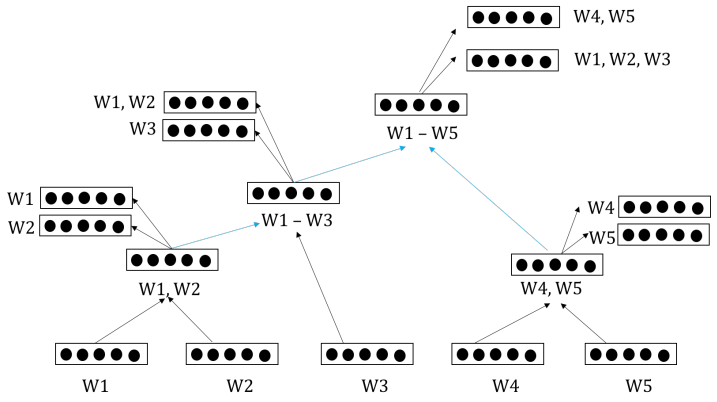
Kamu sudah tahu bagaimana cara mengonversi kata menjadi vektor, untuk mengonversi kalimat menjadi vektor cara sederhananya adalah merata-ratakan nilai vektor kata-kata pada kalimat tersebut. Tetapi dengan cara sederhana ini, sifat sekuensial dan *compositional* pada kalimat tidak terpenuhi. Sebagai contoh, kalimat "anjing menggigit Budi" dan "Budi menggigit anjing" akan direpresentasikan sebagai vektor yang sama karena terdiri dari kata-kata yang sama. Dengan demikian, representasi kalimat sederhana dengan merata-ratakan vektor kata-katanya juga tidaklah sensitif terhadap urutan[14]. Selain itu, rata-rata tidak sensitif terhadap *compositionality*. Misal frase "bukan sebuah pengalaman baik" tersusun atas frase "bukan" yang diikuti oleh "sebuah pengalaman baik". Rata-rata tidak mengetahui bahwa "bukan" adalah sebuah *modifier* untuk sebuah frase dibelakangnya. Sentimen dapat berubah bergantung pada komposisi kata-katanya (contoh pada Gambar. 10.7).



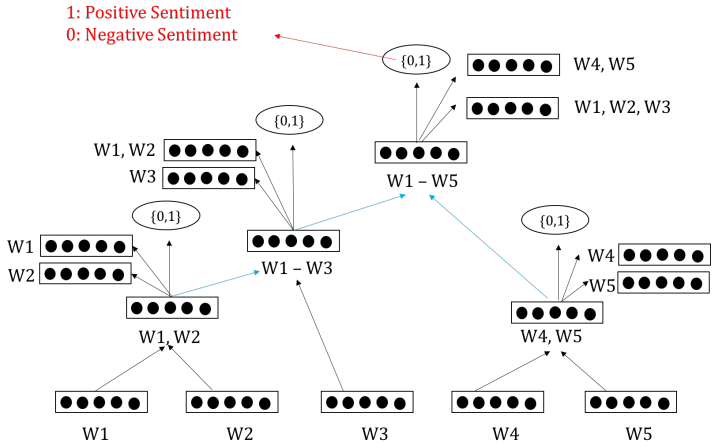
Gambar 10.7. Contoh analisis sentimen (Stanford)[15]

Cara lainnya adalah meng-*encode* kalimat sebagai *vektor* menggunakan *recursive autoencoder*. *Recursive* berarti suatu bagian adalah komposisi dari bagian lainnya. Penggunaan *recursive autoencoder* sangat rasional berhubung

[14] Karena ini *recurrent neural network* bagus untuk *language modelling*.
[15] `http://nlp.stanford.edu:8080/sentiment/rntnDemo.html`



Gambar 10.8. Contoh recursive autoencoder.



Gambar 10.9. Contoh recursive autoencoder dengan sentiment[59].

data memenuhi sifat *compositionality* yang direpresentasikan dengan baik oleh topologi *recursive neural network*. Selain itu, urutan susunan kata-kata juga tidak hilang. Untuk melatih *recursive autoencoder*, *output* dari suatu layer adalah rekonstruksi *input*, ilustrasi dapat dilihat pada Gambar. 10.8. Pada setiap langkah *recursive*, *hidden layer/coding layer* berusaha men-*decode* atau merekonstruksi kembali vektor *input*.

Lebih jauh, untuk sentimen analisis pada kata, kita dapat menambahkan output pada setiap *hidden layer*, yaitu sentimen unit gabungan, seperti pada Gambar. 10.9. Selain menggunakan *recursive autoencoder*, kamu juga dapat menggunakan *recurrent autoencoder*. Kami silahkan pada pembaca untuk memahami *recurrent autoencoder*. Prinsipnya mirip dengan *recursive autoencoder*.

Teknik yang disampaikan mampu mengonversi kalimat menjadi vektor, lalu bagaimana dengan paragraf, satu dokumen, atau satu frasa saja? Teknik umum untuk mengonversi teks menjadi vektor dapat dibaca pada [61] yang lebih dikenal dengan nama ***paragraph vector*** atau ***doc2vec***.

10.5 Tips

Bab ini menyampaikan penggunaan *neural network* untuk melakukan *kompresi data* (*representation learning*) dengan teknik *unsupervised learning*. Hal yang lebih penting untuk dipahami bahwa ilmu *machine learning* tidak berdiri sendiri. Walaupun kamu menguasai teknik *machine learning* tetapi tidak mengerti domain dimana teknik tersebut diaplikasikan, kamu tidak akan bisa membuat *learning machine* yang memuaskan. Contohnya, pemilihan fitur *machine learning* pada teks (NLP) berbeda dengan gambar (*visual processing*). Mengerti *machine learning* tidak semata-mata membuat kita bisa menyelesaikan semua macam permasalahan. Tanpa pengetahuan tentang domain aplikasi, kita bagaikan orang buta yang ingin menyetir sendiri!

Soal Latihan

10.1. Feature Selection
Sebutkan dan jelaskan berbagai macam teknik *feature selection*!

10.2. LSI dan LDA

(a) Jelaskanlah *matrix factorization* dan *principal component analysis*!
(b) Jelaskanlah Latent Semantic Indexing (LSI) dan Latent Dirichlet Allocation (LDA)!
(c) Apa persamaan dan perbedaan antara LSI, LDA, dan Autoencoder?

10.3. Variational Autoencoder
Jelaskan apa itu *Variational autoencoder*! Deskripsikan perbedaannya dengan *autoencoder* yang sudah dijelaskan pada bab ini?

11

Arsitektur Neural Network

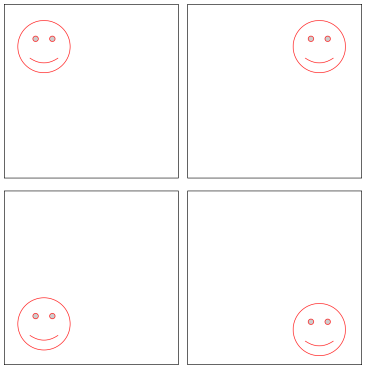
> As students cross the threshold from outside to insider, they also cross the threshold from superficial learning motivated by grades to deep learning motivated by engagement with questions. Their transformation entails an awakening–even, perhaps, a falling in love.
>
> John C. Bean

Seperti yang sudah dijelaskan pada bab 10, data memiliki karakteristik (dari segi *behaviour*) misal *sequential data*, *compositional data*, dsb. Terdapat arsitektur khusus *artificial neural network* (ANN) untuk menyelesaikan persoalan pada tipe data tertentu. Pada bab ini, kami akan memberikan beberapa contoh variasi arsitektur ANN yang cocok untuk tipe data tertentu. Penulis akan berusaha menjelaskan semaksimal mungkin ide-ide penting pada masing-masing arsitektur. Tujuan bab ini adalah memberikan pengetahuan konseptual (intuisi). Pembaca harus mengeksplorasi tutorial pemrograman untuk mampu mengimplementasikan arsitektur-arsitektur ini.

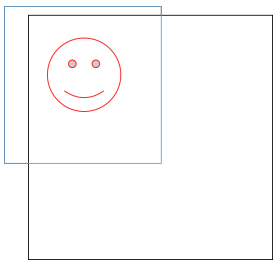
11.1 Convolutional Neural Network

Subbab ini akan memaparkan **ide utama** dari *convolutional neural network* (CNN) berdasarkan *paper* asli dari LeCun dan Bengio [68] (sekarang (2018) sudah ada banyak variasi). CNN memiliki banyak istilah dari bidang pemrosesan gambar (karena dicetuskan dari bidang tersebut), tetapi demi mempermudah pemahaman intuisi CNN, diktat ini akan menggunakan istilah yang lebih umum juga.

Sekarang, mari kita memasuki cerita CNN dari segi pemrosesan gambar. Objek bisa saja dterlatak pada berbagai macam posisi seperti diilustrasikan oleh Gambar. 11.1. Selain tantangan variasi posisi objek, masih ada juga tantangan lain seperti rotasi objek dan perbedaan ukuran objek (*scaling*). Kita ingin mengenali (memproses) objek pada gambar pada berbagai macam posisi yang mungkin (***translation invariance***). Salah satu cara yang mungkin adalah dengan membuat suatu mesin pembelajaran (ANN) untuk regional tertentu seperti pada Gambar. 11.2 (warna biru) kemudian meng-*copy* mesin pembelajaran untuk mampu mengenali objek pada regional-regional lainnya. Akan tetapi, kemungkinan besar ANN *copy* memiliki konfigurasi parameter yang sama dengan ANN awal. Hal tersebut disebabkan objek yang memiliki informasi prediktif (*predictive information*) yang berguna untuk menganalisisnya. Dengan kata lain, objek yang sama (*smile*) memiliki bentuk yang sama. ANN (MLP) bisa juga mempelajari prinsip *translation invariance*, tetapi memerlukan jauh lebih banyak parameter dibanding CNN (subbab berikutnya secara lebih matematis) yang memang dibuat dengan prinsip *translation invariance* (*built-in*).



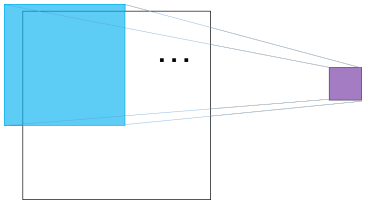
Gambar 11.1. Motivasi *convolutional neural network*.



Gambar 11.2. Motivasi *convolutional neural network*, solusi regional.

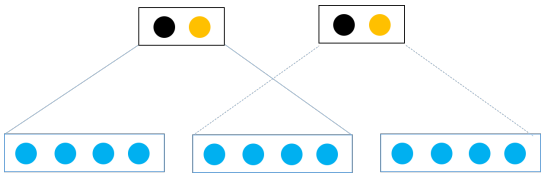
11.1.1 Convolution

Seperti yang sudah dijelaskan, motivasi CNN adalah untuk mampu mengenali aspek yang informatif pada regional tertentu (lokal). Dibanding meng-*copy* mesin pembelajaran beberapa kali untuk mengenali objek pada banyak regional, ide lebih baik adalah untuk menggunakan *sliding window*. Setiap operasi pada *window*[1] bertujuan untuk mencari aspek lokal yang paling informatif. Ilustrasi diberikan oleh Gambar. 11.3. Warna biru merepresentasikan satu *window*, kemudian kotak ungu merepresentasikan aspek lokal paling informatif (disebut *filter*) yang dikenali oleh *window*. Dengan kata lain, kita mentransformasi suatu *window* menjadi suatu nilai numerik (*filter*). Kita juga dapat mentransformasi suatu *window* (regional) menjadi d nilai numerik (d-*channels*, setiap elemen berkorespondensi pada suatu *filter*). *Window* ini kemudian digeser-geser sebanyak N kali, sehingga akhirnya kita mendapatkan vektor dengan panjang $d \times N$. Keseluruhan operasi ini disebut sebagai ***convolution***[2].



Gambar 11.3. *Sliding window.*

Agar kamu lebih mudah memahami prinsip ini, kami berikan contoh dalam bentuk 1-D pada Gambar. 11.4. Warna biru merepresentasikan *feature vector* (regional) untuk suatu *input* (e.g., regional pada suatu gambar, kata pada kalimat, dsb). Pada contoh ini, setiap 2 *input* ditransformasi menjadi vektor berdimensi 2 (2-*channels*); menghasilkan vektor berdimensi 4 (2 *window* $\times$ 2).

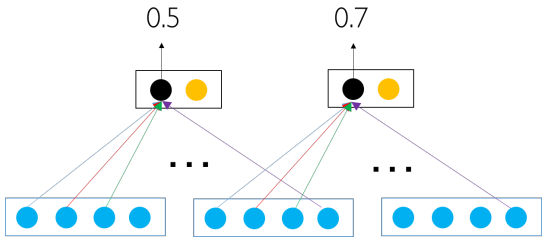


Gambar 11.4. *1D Convolution.*

[1] Dikenal juga sebagai *receptive field*.
[2] Istilah *convolution* yang diterangkan pada konteks *machine learning* memiliki arti yang berbeda pada bidang *signal processing*.

Pada contoh sebelumnya, kita menggunakan *window* selebar 2, satu *window* mencakup 2 data; i.e., $window_1 = (x_1, x_2)$, $window_2 = (x_2, x_3)$, $\cdots$. Untuk suatu *input* $\mathbf{x}$. Kita juga dapat mempergunakan *stride* sebesar s, yaitu seberapa banyak data yang digeser untuk *window* baru. Contoh yang diberikan memiliki *stride* sebesar satu. Apabila kita memiliki *stride*$= 2$, maka kita menggeser sebanyak 2 data setiap langkah; i.e., $window_1 = (x_1, x_2)$, $window_2 = (x_3, x_4)$, $\cdots$.

Selain *sliding window* dan *filter*, *convolutional layer* juga mengadopsi prinsip *weight sharing*. Artinya, *synapse weights* untuk suatu filter adalah sama walau *filter* tersebut dipergunakan untuk berbagai *window*. Sebagai ilustrasi, perhatikan Gambar. 11.5, warna yang sama pada *synapse weights* menunjukkan *synapse weights* bersangkutan memiliki nilai (*weight*) yang sama. Tidak hanya pada *filter* hitam, hal serupa juga terjadi pada *filter* berwarna oranye (i.e., *filter* berwarnya oranye juga memenuhi prinsip *weight sharing*). Walaupun memiliki konfigurasi bobot *synapse weights* yang sama, unit dapat menghasilkan *output* yang berbeda untuk *input* yang berbeda. Konsep *weight sharing* ini sesuai dengan cerita sebelumnya bahwa konfigurasi parameter untuk mengenali karakteristik informatif untuk satu objek bernilai sama walau pada lokasi yang berbeda. Dengan *weight sharing*, parameter *neural network* juga menjadi lebih sedikit dibanding menggunakan *multilayer perceptron* (*feed-forward neural network*).



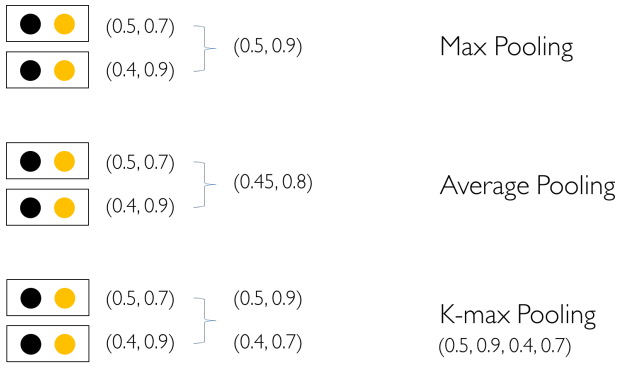
Gambar 11.5. Konsep *weight sharing*.

11.1.2 Pooling

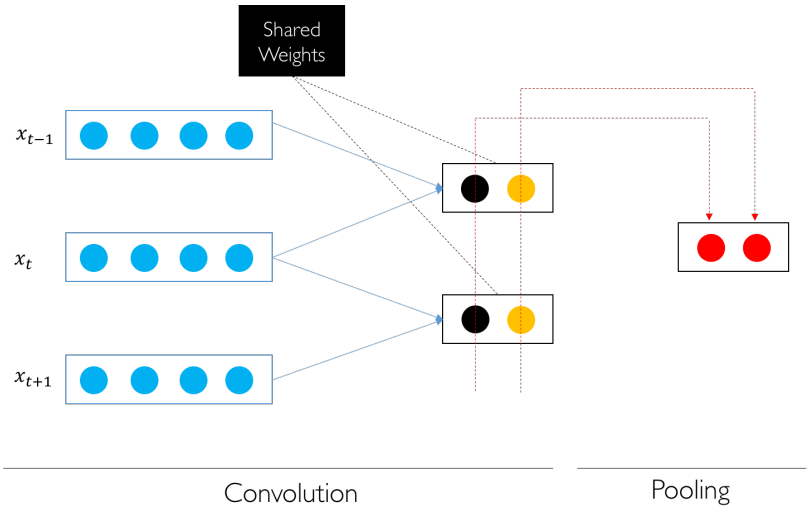
Pada tahap *convolution*, kita merubah setiap *k-sized window* menjadi satu vektor berdimensi d (yang dapat disusun menjadi matriks $\mathbf{D}$). Semua vektor yang dihasilkan pada tahap sebelumnya dikombinasikan (*pooled*) menjadi satu vektor $\mathbf{c}$. Ide utamanya adalah mengekstrak informasi paling informatif (semacam meringkas). Ada beberapa teknik *pooling*, diantaranya: *max pooling*, *average pooling*, dan *K-max pooling*[3]; diilustrasikan pada Gambar. 11.6. *Max pooling* mencari nilai maksimum untuk setiap dimensi vektor. *Average pooling*

[3] Kami ingin pembaca mengeksplorasi sendiri *dynamic pooling*.

mencari nilai rata-rata tiap dimensi. *K-max pooling* mencari K nilai terbesar untuk setiap dimensinya (kemudian hasilnya digabungkan). Gabungan operasi *convolution* dan *pooling* secara konseptual diilustrasikan pada Gambar. 11.7.

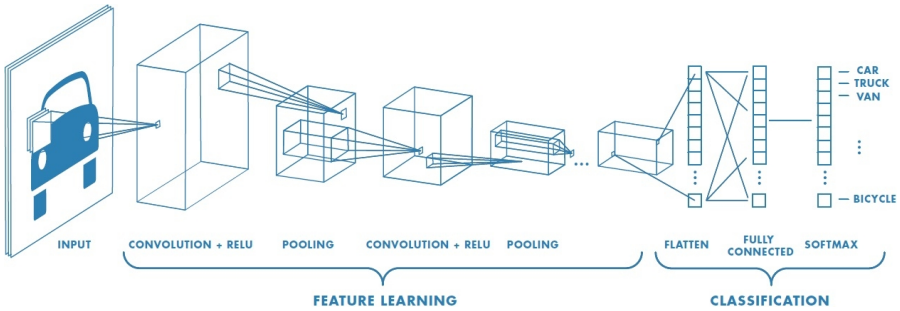


Gambar 11.6. Contoh *pooling*.



Gambar 11.7. *Convolution* dan *pooling*.

Setelah melewati berbagai operasi *convolution* dan *pooling*, kita akan memiliki satu vektor yang kemudian dilewatkan pada *multilayer perceptron* untuk melakukan sesuatu (tergantung permasalahan), misal klasifikasi gambar, klasifikasi sentimen, dsb (Ilustrasi pada Gambar. 11.8).



Gambar 11.8. *Convolutional Neural Network*[4].

11.1.3 Rangkuman

Kemampuan utama *convolutional neural network* (CNN) adalah arsitektur yang mampu mengenali informasi prediktif suatu objek (gambar, teks, potongan suara, dsb) walaupun objek tersebut dapat diposisikan dimana saja pada *input*. Kontribusi CNN adalah pada *convolution* dan *pooling* layer. *Convolution* bekerja dengan prinsip *sliding window* dan *weight sharing* (mengurangi kompleksitas perhitungan). *Pooling layer* berguna untuk merangkum informasi informatif yang dihasilkan oleh suatu *convolution* (mengurangi dimensi). Pada ujung akhir CNN, kita lewatkan satu vektor hasil beberapa operasi *convolution* dan *pooling* pada *multilayer perceptron* (*feed-forward neural network*), dikenal juga sebagai *fully connected layer*, untuk melakukan suatu pekerjaan (e.g., klasifikasi). Perhatikan, pada umumnya CNN tidak berdiri sendiri, dalam artian CNN biasanya digunakan (dikombinasikan) untuk arsitektur yang lebih besar.

11.2 Recurrent Neural Network

Ide dasar *recurrent neural network* (RNN) adalah membuat topologi jaringan yang mampu merepresentasikan data *sequential* (sekuensial) atau *time series* [69], misalkan data ramalan cuaca. Cuaca hari ini bergantung kurang lebih pada cuaca hari sebelumnya. Sebagai contoh apabila hari sebelumnya mendung, ada kemungkinan hari ini hujan[5]. Walau ada yang menganggap sifat data sekuensial dan *time series* berbeda, RNN berfokus sifat data dimana instans waktu sebelumnya $(t-1)$ mempengaruhi instans pada waktu berikutnya (t). Intinya, mampu mengingat *history*.

Secara lebih umum, diberikan sebuah sekuens data $\mathbf{x} = (x_1, \cdots, x_N)$. Data x_t (i.e., vektor, gambar, teks, suara) dipengaruhi oleh data sebelum-sebelumnya (*history*), ditulis sebagai $P(x_t \mid \{x_1, \cdots, x_{t-1}\})$. Kami harap

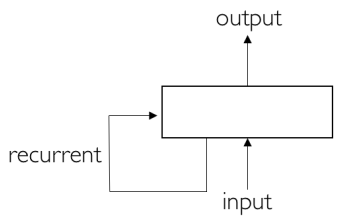
[4] `mathworks.com`

[5] Mohon bertanya pada ahli meteorologi untuk kebenaran contoh ini. Contoh ini semata-mata pengalaman pribadi penulis.

kamu ingat kembali materi *markov assumption* yang diberikan pada bab 7. Pada *markov assumption*, diasumsikan bahwa data x_t (data point) hanya dipengaruhi oleh **beberapa data sebelumnya saja** (analogi: *windowing*). Setidaknya, asumsi ini memiliki dua masalah:

1. Menentukan *window* terbaik. Bagaimana cara menentukan banyaknya data sebelumnya (secara optimal) yang mempengaruhi data sekarang.
2. Apabila kita menggunakan *markov assumption*, artinya kita mengganggap informasi yang dimuat oleh data lama dapat direpresentasikan oleh data lebih baru (x_t memuat informasi dari x_{t-J}; J adalah ukuran *window*). Penyederhanaan ini tidak jarang mengakibatkan informasi yang hilang.

RNN adalah salah satu bentuk arsitektur ANN untuk mengatasi masalah yang ada pada *markov assumption*. Ide utamanya adalah memorisasi[6], kita ingin mengingat **keseluruhan** sekuens (dibanding *markov assumption* yang mengingat sekuens secara terbatas), implikasinya adalah RNN mampu mengenali dependensi yang panjang (misal x_t ternyata dependen terhadap x_1). RNN paling sederhana diilustrasikan pada Gambar. 11.9. Ide utamanya adalah terdapat *pointer* ke dirinya sendiri.



Gambar 11.9. Bentuk konseptual paling sederhana recurrent NN.

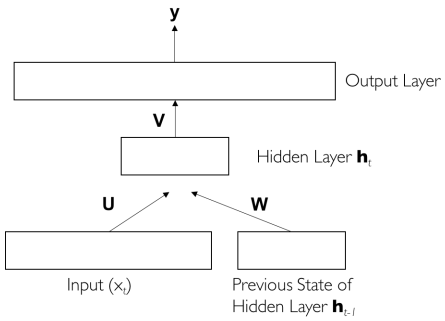
Ilustrasi Gambar. 11.9 mungkin sedikit susah dipahami karena berbentuk sangat konseptual. Bentuk lebih matematis diilustrasikan pada Gambar. 11.10 [69]. Perhitungan *hidden state* pada waktu ke-t bergantung pada *input* pada waktu ke-t (x_t) dan *hidden state* pada waktu sebelumnya (h_{t-1}).

Konsep ini sesuai dengan prinsip *recurrent* yaitu **mengingat** (memorisasi) kejadian sebelumnya. Kita dapat tulis kembali RNN sebagai persamaan 11.1.

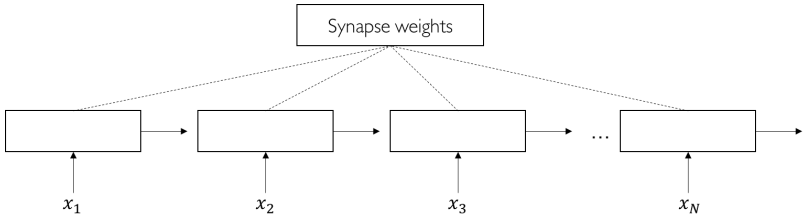
$$\mathbf{h}_t = f(x_t, \mathbf{h}_{t-1}, b) \tag{11.1}$$

dimana f adalah fungsi aktivasi (non-linear, dapat diturunkan). Demi menyederhanakan penjelasan, penulis tidak mengikutsertakan *bias* (b) pada fungsi-fungsi berikutnya. Kami berharap pembaca selalu mengingat bahwa *bias* adalah parameter yang diikutsertakan pada fungsi *artificial neural network*.

[6] Tidak merujuk hal yang sama dengan *dynamic programming*.



Gambar 11.10. Konsep Recurrent Neural Network.



Gambar 11.11. Konsep *feed forward* pada RNN.

Fungsi f dapat diganti dengan variasi *neural network*[7], misal menggunakan *long short-term memory network* (LSTM) [70]. Buku ini hanya akan menjelaskan konsep paling penting, silahkan eksplorasi sendiri variasi RNN.

Secara konseptual, persamaan 11.1 memiliki analogi dengan *full markov chain*. Artinya, *hidden state* pada saat ke-t bergantung pada semua *hidden state* dan *input* sebelumnya.

$$\begin{aligned}\mathbf{h}_t &= f(x_t, \mathbf{h}_{t-1}) \\ &= f(x_t, f(x_{t-1}, \mathbf{h}_{t-2})) \\ &= f(x_t, f(x_{t-1}, f(\{x_1, \cdots, x_{t-2}\}, \{\mathbf{h}_1, \cdots, \mathbf{h}_{t-3}\})))\end{aligned} \tag{11.2}$$

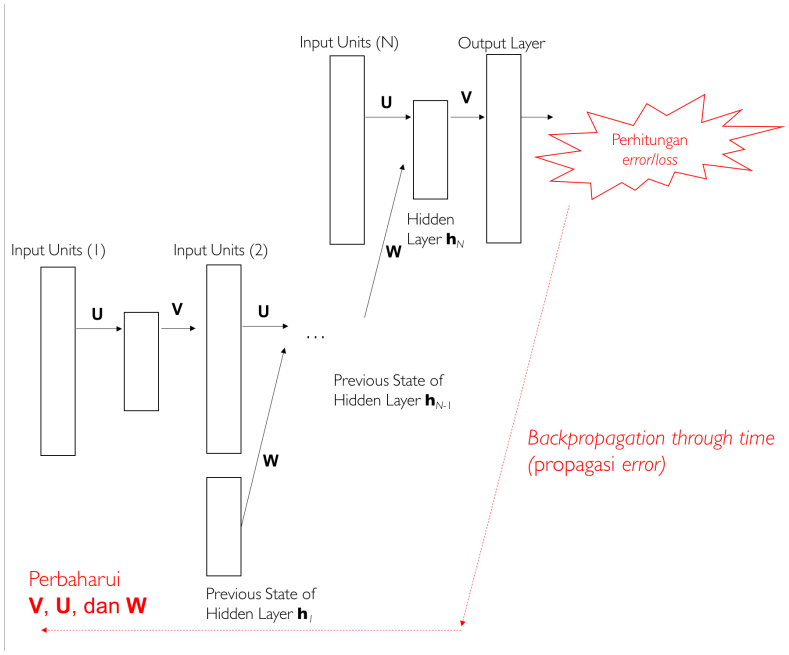
Training pada *recurrent neural network* dapat menggunakan metode *backpropagation*. Akan tetapi, metode tersebut kurang intuitif karena tidak mampu mengakomodasi *training* yang bersifat sekuensial *time series*. Untuk itu, terdapat metode lain bernama *backpropagation through time* [71].

Sebagai contoh kita diberikan sebuah sekuens $\mathbf{x}$ dengan panjang N sebagai input, dimana x_t melambangkan input ke-i (**data point** dapat berupa e.g., vektor, gambar, teks, atau apapun). Kita melakukan *feed forward* data tersebut ke RNN, diilustrasikan pada Gambar. 11.11. Perlu diingat, RNN

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

mengadopsi prinsip *parameter sharing* (serupa dengan *weight sharing* pada CNN) dimana neuron yang sama diulang-ulang saat process *feed forward*.

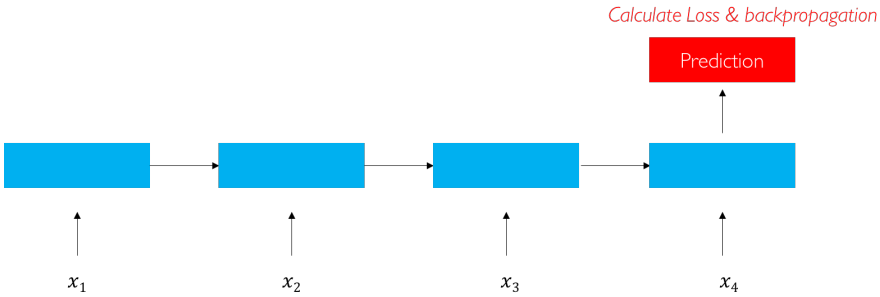
Kemudian kita memperbaharui parameter (*synapse weights*) berdasarkan propagasi *error* (*backpropagation*). Pada *backpropagation* biasa, kita perbaharui parameter sambil mempropagasi *error* dari *hidden state* ke *hidden state* sebelumnya. Tetapi, pada *backpropagation through time*, dilakukan *unfolding* pada *neural network*. Kita mengupdate parameter, saat kita sudah mencapai *hidden state* paling awal. Hal ini diilustrasikan pada Gambar. 11.12[8]. Gambar. 11.12 dapat disederhanakan menjadi bentuk lebih abstrak (konseptual) pada Gambar. 11.13.



Gambar 11.12. Konsep *Backpropagation Through Time* [39].

Kita mempropagasi *error* dengan adanya efek dari *next states of hidden layer*. *Synapse weights* diperbaharui secara *large update*. *Synapse weight* tidak diperbaharui per *layer*. Hal ini untuk merepresentasikan *neural network* yang mampu mengingat beberapa kejadian masa lampau dan keputusan saat ini dipengaruhi oleh keputusan pada masa lampau juga (ingatan). Untuk mengerti proses ini secara praktikal (dapat menuliskannya sebagai pro-

[8] Prinsip ini mirip dengan *weight sharing*.



Gambar 11.13. Konsep *Backpropagation Through Time* [1]. Persegi berwarna merah umumnya melambangkan *multi-layer perceptron*.

gram), penulis sarankan pembaca untuk melihat materi tentang ***computation graph***[9] dan disertasi PhD oleh Mikolov [39].

Walaupun secara konseptual RNN dapat mengingat seluruh kejadian sebelumnya, hal tersebut sulit untuk dilakukan secara praktikal untuk sekuens yang panjang. Hal ini lebih dikenal dengan *vanishing* atau *exploding gradient problem* [55, 72, 73]. Seperti yang sudah dijelaskan, ANN dan variasi arsitekturnya dilatih menggunakan teknik *stochastic gradient descent* (*gradient-based optimization*). Artinya, kita mengandalkan propagasi *error* berdasarkan turunan. Untuk sekuens *input* yang panjang, tidak jarang nilai *gradient* menjadi sangat kecil dekat dengan 0 (*vanishing*) atau sangat besar (*exploding*). Ketika pada satu *hidden state* tertentu, gradient pada saat itu mendekati 0, maka nilai yang sama akan dipropagasikan pada langkah berikutnya (menjadi lebih kecil lagi). Hal serupa terjadi untuk nilai gradient yang besar.

Berdasarkan pemaparan ini, RNN adalah teknik untuk merubah suatu sekuens *input*, dimana x_t merepresentasikan data ke-t (e.g., vektor, gambar, teks) menjadi sebuah *output* vektor $\mathbf{y}$. Vektor $\mathbf{y}$ dapat digunakan untuk permasalahan lebih lanjut (buku ini memberikan contoh *sequence to sequence* pada subbab 11.4). Bentuk konseptual ini dapat dituangkan pada persamaan 11.3. Biasanya, nilai y dilewatkan kembali ke sebuah *multi-layer perceptron* (MLP) dan fungsi softmax untuk melakukan klasifikasi akhir (*final output*) dalam bentuk probabilitas, seperti pada persamaan 11.4.

$$\mathbf{y} = \mathrm{RNN}(x_1, \cdots, x_N) \tag{11.3}$$

$$\text{final output} = \mathrm{softmax}(\mathrm{MLP}(\mathbf{y})) \tag{11.4}$$

Perhatikan, arsitektur yang penulis deskripsikan pada subbab ini adalah arsitektur paling dasar. Untuk arsitektur *state-of-the-art*, kamu dapat membaca *paper* yang berkaitan.

[9] `https://www.coursera.org/learn/neural-networks-deep-learning/lecture/4WdOY/computation-graph`

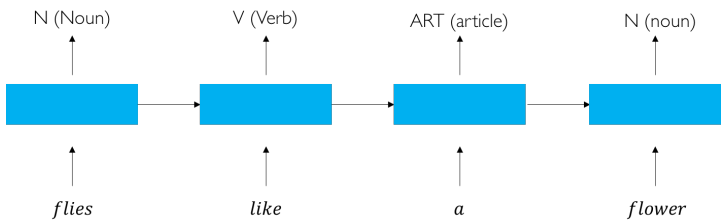
11.3 Part-of-speech Tagging Revisited

Pada bab sebelumnya, kamu telah mempelajari konsep dasar *recurrent neural network*. Selain digunakan untuk klasifikasi (i.e., *hidden state* terakhir digunakan sebagai *input* klasifikasi), RNN juga dapat digunakan untuk memprediksi sekuens seperti persoalan *part-of-speech tagging* (POS *tagging*) [74, 75, 76]. Kami harap kamu masih ingat materi bab 7 yang membahas apa itu persoalan POS *tagging*.

Diberikan sebuah sekuens kata $\mathbf{x} = \{x_1, \cdots, x_N\}$, kita ingin mencari sekuens *output* $\mathbf{y} = \{y_1, \cdots, y_N\}$ (*sequence prediction*); dimana y_i adalah kelas kata untuk x_i. Perhatikan, panjang *input* dan *output* adalah sama. Ingat kembali bahwa pada persoalan POS *tagging*, kita ingin memprediksi suatu kelas kata yang cocok y_i dari kumpulan kemungkinan kelas kata C ketika diberikan sebuah *history* seperti diilustrasikan oleh persamaan 11.5, dimana t_i melambangkan kandidat POS *tag* ke-i. Pada kasus ini, biasanya yang dicari tahu setiap langkah (*unfolding*) adalah probabilitas untuk memilih suatu kelas kata $t \in C$ sebagai kelas kata yang cocok untuk di-*assign* sebagai y_i.

Ilustrasi diberikan oleh Gambar. 11.14.

$$y_1, \cdots, y_N = \underset{t_1, \cdots, t_N; t_i \in C}{\arg\max}\, p(t_1, \cdots, t_N \mid x_1, \cdots, x_N) \tag{11.5}$$



Gambar 11.14. POS *tagging* menggunakan RNN.

Apabila kita melihat secara sederhana (*markov assumption*), hal ini tidak lain dan tidak bukan adalah melakukan klasifikasi untuk setiap *instance* pada sekuens *input* (persamaan 11.6). Pada setiap *time step*, kita ingin menghasilkan *output* yang bersesuaian.

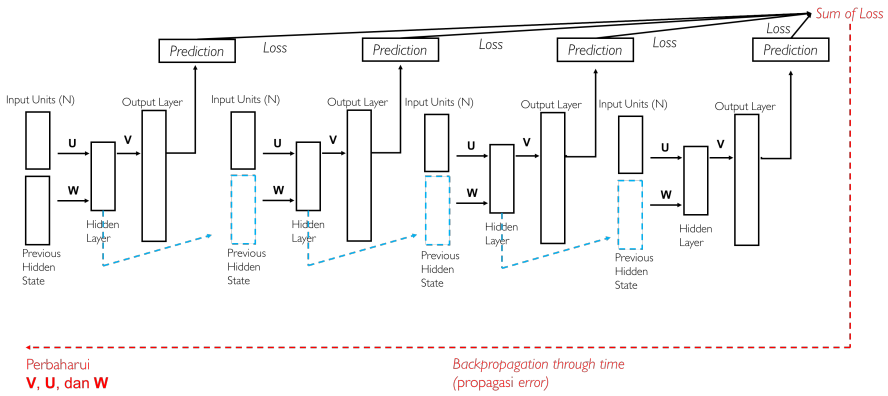
$$y_i = \underset{t_i \in C}{\arg\max}\, p(t_i | x_i) \tag{11.6}$$

Akan tetapi, seperti yang sudah dibahas sebelum sebelumnya, *markov assumption* memiliki kelemahan. Kelemahan utama adalah tidak menggunakan keseluruhan *history*. Persoalan ini cocok untuk diselesaikan oleh RNN karena kemampuannya untuk mengingat seluruh sekuens (berbeda dengan *hidden*

markov model (HMM) yang menggunakan *markov assumption*). Secara teoritis (dan juga praktis[10]), RNN lebih hebat dibanding HMM. Dengan ini, persoalan POS *tagging* (*full history*) diilustrasikan oleh persamaan 11.7.

$$y_i = \arg\max_{t_i \in C} p(t_i | x_1, \cdots, x_N) \tag{11.7}$$

Pada bab sebelumnya, kamu diberikan contoh persoalan RNN untuk satu *output*; i.e., diberikan sekuens *input*, *output*-nya hanyalah satu kelas yang mengkategorikan seluruh sekuens *input*. Untuk persoalan POS *tagging*, kita harus sedikit memodifikasi RNN untuk menghasilkan *output* bagi setiap elemen sekuens *input*. Hal ini dilakukan dengan cara melewatkan setiap *hidden layer* pada RNN pada suatu jaringan (anggap sebuah MLP + softmax). Kita lakukan prediksi kelas kata untuk setiap elemen sekuens *input*, kemudian menghitung *loss* untuk masing-masing elemen. Seluruh *loss* dijumlahkan untuk menghitung *backpropagation* pada RNN. Ilustrasi dapat dilihat pada Gambar. 11.15. Tidak hanya untuk persoalan POS *tagging*, arsitektur ini dapat juga digunakan pada persoalan *sequence prediction* lainnya seperti *named entity recognition*[11]. Gambar. 11.15 mungkin agak sulit untuk dilihat, kami beri bentuk lebih sederhananya (konseptual) pada Gambar. 11.16. Pada setiap langkah, kita menentukan POS *tag* yang sesuai dan menghitung *loss* yang kemudian digabungkan. *Backpropagation* dilakukan dengan mempertimbangkan keseluruhan (jumlah) *loss* masing-masing prediksi.

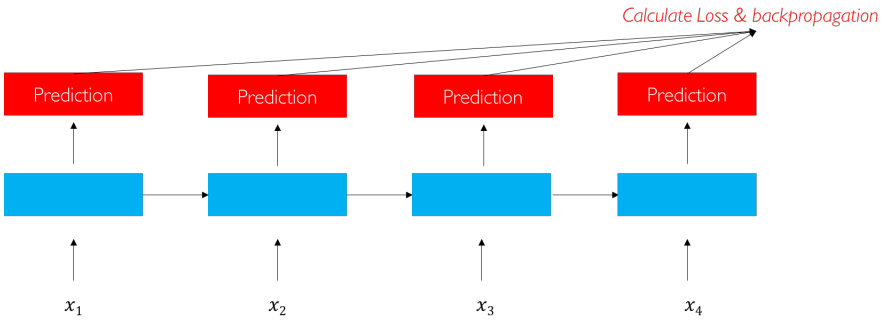


Gambar 11.15. *Sequence prediction* menggunakan RNN.

Berdasarkan arsitektur yang sudah dijelaskan sebelumnya, prediksi POS *tag* ke-i bersifat independen dari POS *tag* lainnya. Padahal, POS *tag* lain-

[10] Sejauh yang penulis ketahui. Tetapi hal ini bergantung juga pada variasi arsitektur.

[11] `https://en.wikipedia.org/wiki/Named-entity_recognition`



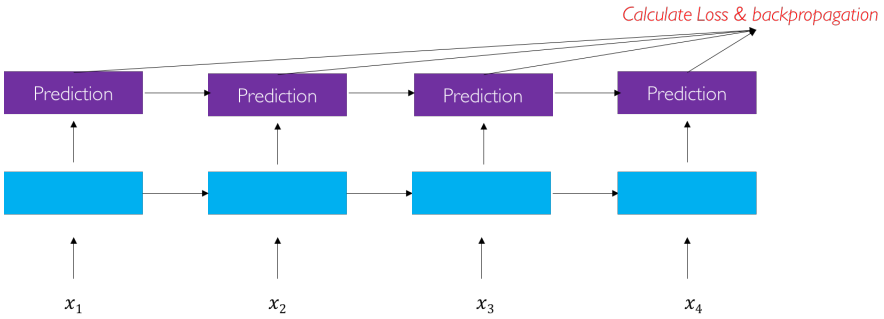
Gambar 11.16. *Sequence prediction* menggunakan RNN (disederhakan) [1]. Persegi berwarna merah umumnya melambangkan *multi-layer perceptron*.

nya memiliki pengaruh saat memutuskan POS *tag* ke-i (ingat kembali materi bab 7); sebagai persamaan 11.8.

$$y_i = \arg\max_{t_i \in C} p(t_i \mid y_1, \cdots, y_{i-1}, x_1, \cdots, x_i) \tag{11.8}$$

Salah satu strategi untuk menangani hal tersebut adalah dengan melewatkan POS *tag* pada sebuah RNN juga, seperti para persamaan 11.9 [1] (ilustrasi pada Gambar. 11.17). Untuk mencari keseluruhan sekuens terbaik, kita dapat menggunakan teknik *beam search* (detil penggunaan dijelaskan pada subbab berikutnya). RNN^{x} pada persamaan 11.9 juga lebih intuitif apabila diganti menggunakan *bidirectional RNN* (dijelaskan pada subbab berikutnya).

$$\begin{aligned} &p(t_i \mid y_1, \cdots, y_{i-1}, x_1, \cdots, x_i) = \\ &\text{softmax}(\text{MLP}([\text{RNN}^{\text{x}}(x_1, \cdots, x_i); \text{RNN}^{\text{t}}(t_1, \cdots, t_{i-1})])) \end{aligned} \tag{11.9}$$



Gambar 11.17. *Sequence prediction* menggunakan RNN (disederhakan). Persegi melambangkan RNN.

11.4 Sequence to Sequence

Pertama-tama, kami ingin mendeskripsikan kerangka ***conditioned generation***. Pada kerangka ini, kita ingin memprediksi sebuah kelas y_i berdasarkan kelas yang sudah di-hasilkan sebelumnya (*history* yaitu $y_1, \cdots, y_{i-1}$) dan sebuah *conditioning context* $\mathbf{c}$ (berupa vektor).

Arsitektur yang dibahas pada subbab ini adalah variasi RNN untuk permasalahan *sequence generation*[12]. Diberikan sekuens *input* $\mathbf{x} = (x_1, \cdots, x_N)$. Kita ingin mencari sekuens *output* $\mathbf{y} = (y_1, \cdots, y_M)$. Pada subbab sebelumnya, x_i berkorespondensi langsung dengan y_i; i.e., y_i adalah kelas kata (kategori) untuk x_i. Tetapi, pada permasalahan saat ini, x_i tidak langsung berkorespondensi dengan y_i. Setiap y_i dikondisikan oleh **seluruh** sekuens *input* $\mathbf{x}$ (*conditioning context* dan *history* $\{y_1, \cdots, y_{i-1}\}$. Dengan itu, M (panjang sekuens *output*) tidak mesti sama dengan N (panjang sekuens *input*). Permasalahan ini masuk ke dalam kerangka *conditioned generation* dimana keseluruhan *input* $\mathbf{x}$ dapat direpresentasikan menjadi sebuah vektor $\mathbf{c}$ (*coding*). Vektor c ini menjadi variabel pengkondisi untuk menghasilkan *output* $\mathbf{y}$.

Pasangan *input-output* dapat melambangkan teks bahasa X-teks bahasa Y (translasi), teks-ringkasan, kalimat-*POS tags*, dsb. Artinya ada sebuah *input* dan kita ingin menghasilkan (*generate/produce*) sebuah *output* yang cocok untuk *input* tersebut. Hal ini dapat dicapai dengan momodelkan pasangan *input-output* $P(\mathbf{y} \mid \mathbf{x})$. Umumnya, kita mengasumsikan ada kumpulan parameter θ yang mengontrol *conditional probability*, sehingga kita transformasi *conditional probability* menjadi $P(\mathbf{y} \mid \mathbf{x}, \theta)$. *Conditional probability* $P(\mathbf{y} \mid \mathbf{x}, \theta)$ dapat difaktorkan sebagai persamaan 11.10. Kami harap kamu mampu membedakan persamaan 11.10 dan persamaan 11.5 (dan 11.8) dengan jeli. Sedikit perbedaan pada formula menyebabkan makna yang berbeda.

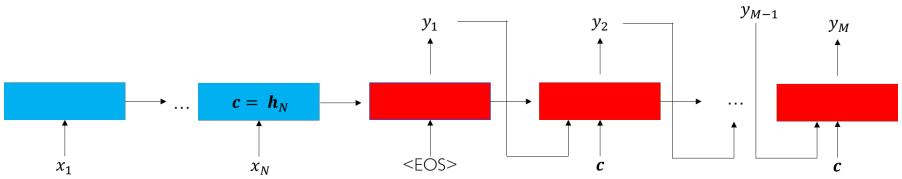
$$P(\mathbf{y} \mid \mathbf{x}, \theta) = \prod_{t=1}^{M} P(y_t \mid \{y_1, \ldots, y_{t-1}\}, \mathbf{x}, \theta), \tag{11.10}$$

Persamaan 11.10 dapat dimodelkan dengan ***encoder-decoder*** model yang terdiri dari dua buah RNN dimana satu RNN sebagai *encoder*, satu lagi sebagai *decoder*. Ilustrasi encoder-decoder dapat dilihat pada Gambar. 11.18. Gabungan RNN *encoder* dan RNN *decoder* ini disebut sebagai bentuk ***sequence to sequence***. Warna biru merepresentasikan *encoder* dan warna merah merepresentasikan *decoder*. "<EOS>" adalah suatu simbol spesial (*untuk praktikalitas*) yang menandakan bahwa sekuens *input* telah selesai dan saatnya berpindah ke *decoder*.

Sebuah *encoder* merepresentasikan sekuens *input* $\mathbf{x}$ menjadi satu vektor $\mathbf{c}$[13]. Kemudian, *decoder* men-*decode* representasi $\mathbf{c}$ untuk menghasilkan (*generate*) sebuah sekuens *output* $\mathbf{y}$. Perhatikan, arsitektur kali ini berbeda dengan arsitektur pada subbab 11.3. *Encoder-decoder* (*neural network*) bertin-

[12] Umumnya untuk bidang pemrosesan bahasa alami.

[13] Ingat kembali bab 10 untuk mengerti kenapa hal ini sangat diperlukan.



Gambar 11.18. Konsep *encoder-decoder* [73].

dak sebagai kumpulan parameter θ yang mengatur *conditional probability*. *Encoder-decoder* juga dilatih menggunakan prinsip *gradient-based optimization* untuk *tuning* parameter yang mengkondisikan *conditional probability* [73]. Dengan ini, persamaan 11.10 sudah didefinisikan sebagai *neural network* sebagai persamaan 11.11. "enc" dan "dec" adalah fungsi *encoder* dan *decoder*, yaitu sekumpulan transformasi non-linear.

$$y_t = \text{dec}(\{y_1, \cdots, y_{t-1}\}, \text{enc}(\mathbf{x}), \theta) \tag{11.11}$$

Begitu model dilatih, *encoder-decoder* akan mencari *output* $\mathbf{y}^*$ terbaik untuk suatu input $\mathbf{x}$, dillustrasikan pada persamaan 11.12. Masing-masing komponen *encoder-decoder* dibahas pada subbab-subbab berikutnya. Untuk abstraksi yang baik, penulis akan menggunakan notasi aljabar linear. Kami harap pembaca sudah familiar dengan representasi *neural network* menggunakan notasi aljabar linear seperti yang dibahas pada bab 9.

$$\mathbf{y}^* = \arg\max_{\mathbf{y}} p(\mathbf{y} \mid \mathbf{x}, \theta) \tag{11.12}$$

11.4.1 Encoder

Seperti yang sudah dijelaskan, *encoder* mengubah sekuens *input* $\mathbf{x}$ menjadi satu vektor $\mathbf{c}$. Tiap data point pada sekuens *input* x_t umumnya direpresentasikan sebagai *feature vector* $\mathbf{e}_t$. Dengan demikian, *encoder* dapat direpresentasikan dengan persamaan 11.13

$$\begin{aligned} \mathbf{h}_t &= f(\mathbf{h}_{t-1}, \mathbf{e}_t) \\ &= f(\mathbf{h}_{t-1}\mathbf{U} + \mathbf{e}_t\mathbf{W}) \end{aligned} \tag{11.13}$$

dimana f adalah fungsi aktivasi non-linear; $\mathbf{U}$ dan $\mathbf{W}$ adalah matriks bobot (*weight matrices*–merepresentasikan *synapse weights*).

Representasi *input* $\mathbf{c}$ dihitung dengan persamaan 11.14, yaitu sebagai *weighted sum* dari *hidden states* [49], dimana q adalah fungsi aktivasi non-linear. Secara lebih sederhana, kita boleh langsung menggunakan $\mathbf{h}_N$ sebagai $\mathbf{c}$ [73].

$$\mathbf{c} = q(\{\mathbf{h}_1, \cdots, \mathbf{h}_N\}) \tag{11.14}$$

Walaupun disebut sebagai representasi keseluruhan sekuens *input*, informasi-informasi awal pada *input* yang panjang dapat hilang. Artinya $\mathbf{c}$ lebih banyak memuat informasi *input* ujung-ujung akhir. Salah satu strategi yang dapat digunakan adalah dengan membalik (*reversing*) sekuens *input*. Sebagai contoh, *input* $\mathbf{x} = (x_1, \cdots, x_N)$ dibalik menjadi $(x_N, \cdots, x_1)$ agar bagian awal $(\cdots, x_2, x_1)$ lebih dekat dengan *decoder* [73]. Informasi yang berada dekat dengan *decoder* cenderung lebih diingat. Kami ingin pembaca mengingat bahwa teknik ini pun tidaklah sempurna.

11.4.2 Decoder

Seperti yang sudah dijelaskan sebelumnya, *encoder* memproduksi sebuah vektor $\mathbf{c}$ yang merepresentasikan sekuens *input*. *Decoder* menggunakan representasi ini untuk memproduksi (*generate*) sebuah sekuens *output* $\mathbf{y} = (y_1, \cdots, y_M)$, disebut sebagai proses ***decoding***. Mirip dengan *encoder*, kita menggunakan RNN untuk menghasilkan *output* seperti diilustrasikan pada persamaan 11.15.

$$\begin{aligned} \mathbf{h}'_t &= f(\mathbf{h}'_{t-1}, \mathbf{e}'_{t-1}, \mathbf{c}) \\ &= f(\mathbf{h}'_{t-1}\mathbf{H} + \mathbf{e}'_{t-1}\mathbf{E} + \mathbf{c}\mathbf{C}) \end{aligned} \tag{11.15}$$

dimana f merepresentasikan fungsi aktivasi non-linear; $\mathbf{H}$, $\mathbf{E}$, dan $\mathbf{C}$ merepresentasikan *weight matrices*. *Hidden state* $\mathbf{h}'_t$ melambangkan distribusi probabilitas suatu objek (e.g., POS tag, kata yang **berasal dari suatu himpunan**) untuk menjadi *output* y_t. Umumnya, y_t adalah dalam bentuk *feature-vector* $\mathbf{e}'_t$.

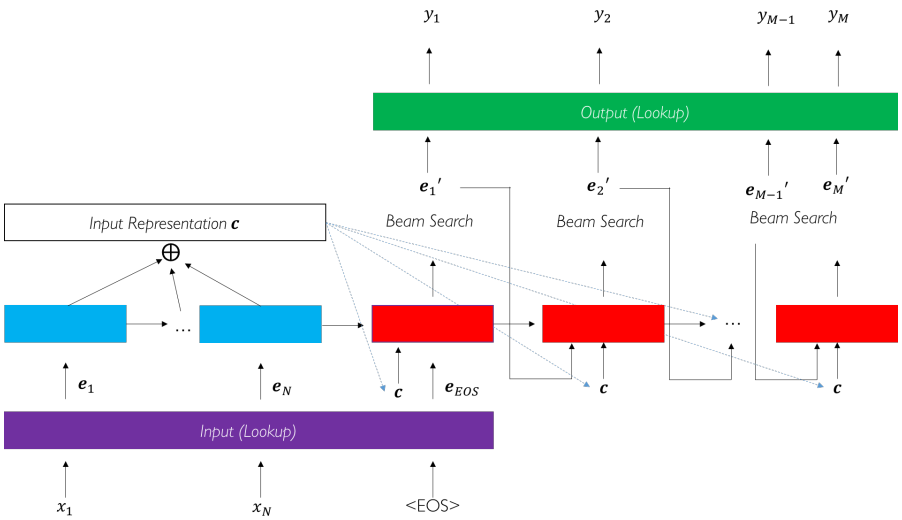
Dengan penjelasan ini, mungkin pembaca berpikir Gambar. 11.18 tidak lengkap. Kamu benar! Penulis sengaja memberikan gambar simplifikasi. Gambar lebih lengkap (dan lebih nyata) diilustrasikan pada Gambar. 11.19.

Kotak berwarna ungu dan hijau dapat disebut sebagai *lookup matrix* atau *lookup table*. Tugas mereka adalah mengubah *input* x_t menjadi bentuk *feature vector*-nya (e.g., *word embedding*) dan mengubah $\mathbf{e}'_t$ menjadi y_t. Komponen "*Beam Search*" dijelaskan pada subbab berikutnya.

11.4.3 Beam Search

Kita ingin mencari sekuens *output* yang memaksimalkan nilai probabilitas pada persamaan 11.12. Artinya, kita ingin mencari *output* terbaik. Pada suatu tahapan *decoding*, kita memiliki beberapa macam kandidat objek untuk dijadikan *output*. Kita ingin mencari sekuens objek sedemikian sehingga probabilitas akhir sekuens objek tersebut bernilai terbesar sebagai *output*. Hal ini dapat dilakukan dengan algoritma *Beam Search*[14].

[14] `https://en.wikipedia.org/wiki/Beam_search`



Gambar 11.19. Konsep *encoder-decoder* (*full*).

```
beamSearch(problemSet, ruleSet, memorySize)
        openMemory = new memory of size memorySize
        nodeList = problemSet.listOfNodes
        node = root or initial search node
        add node to OpenMemory;
        while(node is not a goal node)
                delete node from openMemory;
                expand node and obtain its children, evaluate those children;
                if a child node is pruned according to a rule in ruleSet, delete it;
                place remaining, non-pruned children into openMemory;
                if memory is full and has no room for new nodes, remove the worst
                        node, determined by ruleSet, in openMemory;
        node = the least costly node in openMemory;
```

Gambar 11.20. *Beam Search*[15].

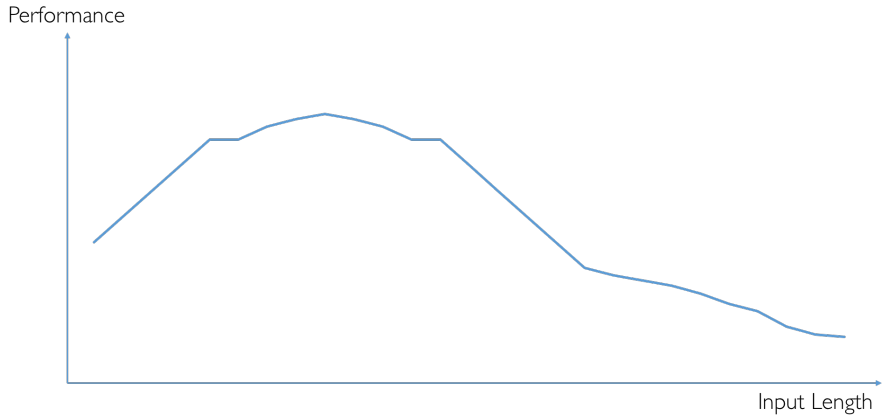
Secara sederhana, algoritma *Beam Search* mirip dengan algoritma Viterbi yang sudah dijelaskan pada bab 7, yaitu algoritma untuk mencari sekuens dengan probabilitas tertinggi. Perbedaannya terletak pada *heuristic*. Untuk menghemat memori komputer, algoritma *Beam Search* melakukan ekspansi terbatas. Artinya mencari hanya beberapa (B) kandidat objek sebagai sekuens berikutnya, dimana beberapa kandidat objek tersebut memiliki probabilitas $P(y_t \mid y_{t-1})$ terbesar. B disebut sebagai *beam-width*. Algoritma *Beam Search* bekerja dengan prinsip yang mirip dengan *best-first search* (*best-B search*) yang sudah kamu pelajari di kuliah algoritma atau pengenalan kecerdasan

[15] `https://en.wikibooks.org/wiki/Artificial_Intelligence/Search/Heuristic_search/Beam_search`

buatan[16]. Pseudo-code *Beam Search* diberikan pada Gambar. 11.20 (*direct quotation*).

11.4.4 Attention-based Mechanism

Seperti yang sudah dijelaskan sebelumnya, model *encoder-decoder* memiliki masalah saat diberikan sekuens yang panjang (*vanishing* atau *exploding gradient problem*). Kinerja model dibandingkan dengan panjang *input* kurang lebih dapat diilustrasikan pada Gambar. 11.21. Secara sederhana, kinerja model menurun seiring sekuens input bertambah panjang. Selain itu, representasi **c** yang dihasilkan *encoder* harus memuat informasi keseluruhan *input* walaupun sulit dilakukan. Ditambah lagi, *decoder* menggunakan representasinya **c** saja tanpa boleh melihat bagian-bagian khusus *input* saat *decoding*. Hal ini tidak sesuai dengan cara kerja manusia, misalnya pada kasus translasi bahasa. Ketika mentranslasi bahasa, manusia melihat bolak-balik bagian mana yang sudah ditranslasi dan bagian mana yang sekarang (difokuskan) untuk ditranslasi. Artinya, manusia berfokus pada suatu bagian *input* untuk menghasilkan suatu translasi.

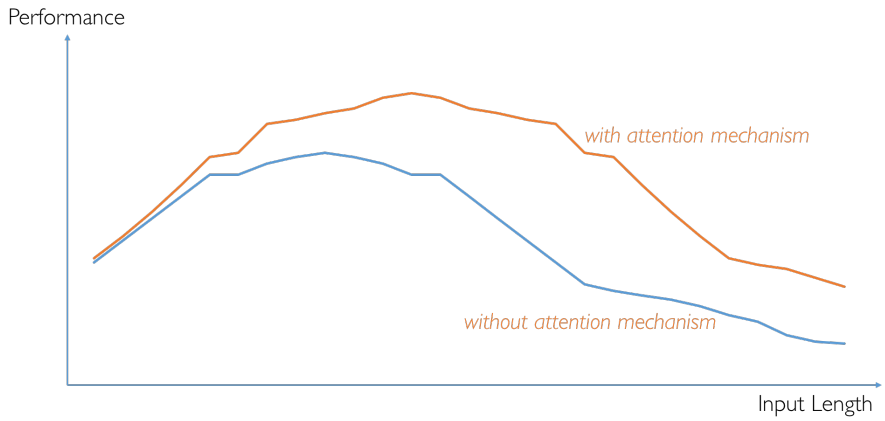


Gambar 11.21. Permasalahan *input* yang panjang.

Sudah dijelaskan sebelumnya bahwa representasi sekuens *input* **c** adalah sebuah *weighted sum*. **c** yang sama digunakan sebagai *input* bagi *decoder* untuk menentukan semua *output*. Akan tetapi, untuk suatu tahapan *decoding* (untuk *hidden state* $\mathbf{h}'_t$ tertentu), kita mungkin ingin model lebih berfokus pada bagian *input* tertentu daripada *weighted sum* yang sifatnya generik. Ide ini adalah hal yang mendasari ***attention mechanism*** [49, 50]. Ide ini sangat

[16] `https://www.youtube.com/watch?v=j1H3jAAGlEA&t=2131s`

berguna pada banyak aplikasi pemrosesan bahasa alami. *Attention mechanism* dapat dikatakan sebagai suatu *soft alignment* antara *input* dan *output*. Mekanisme ini dapat membantu mengatasi permasalahan *input* yang panjang, seperti diilustrasikan pada Gambar. 11.22.



Gambar 11.22. Menggunakan vs. tidak menggunakan *attention*.

Dengan menggunakan *attention mechanism*, kita dapat mentransformasi persamaan 11.15 pada *decoder* menjadi persamaan 11.16

$$\mathbf{h}'_t = f'(\mathbf{h}'_{t-1}, \mathbf{e}'_{t-1}, \mathbf{c}, \mathbf{k}_t) \tag{11.16}$$

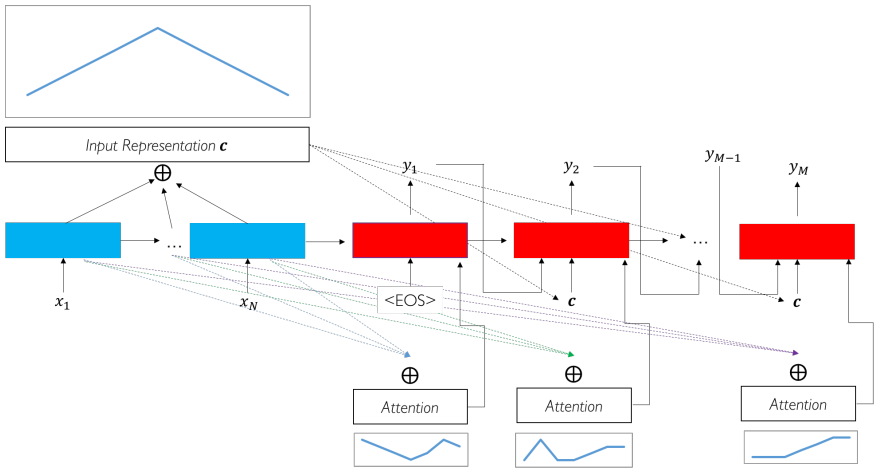
dimana $\mathbf{k}_t$ merepresentasikan seberapa (*how much*) *decoder* harus memfokuskan diri ke *hidden state* tertentu pada *encoder* untuk menghasilkan *output* saat ke-t. $\mathbf{k}_t$ dapat dihitung pada persamaan 11.17

$$\begin{aligned} \mathbf{k}_t &= \sum_{i=1}^{N} \alpha_{t,i} \mathbf{h}_i \\ \alpha_{t,i} &= \frac{\exp(\mathbf{h}_i \cdot \mathbf{h}'_{t-1})}{\sum_{z=1}^{N} \exp(\mathbf{h}_z \cdot \mathbf{h}'_{t-1})} \end{aligned} \tag{11.17}$$

dimana N merepresentasikan panjang *input*, $\mathbf{h}_i$ adalah *hidden state* pada *encoder* pada saat ke-i, $\mathbf{h}'_{t-1}$ adalah *hidden state* pada *decoder* saat ke $t-1$.

Sejatinya $\mathbf{k}_t$ adalah sebuah *weighted sum*. Berbeda dengan $\mathbf{c}$ yang bernilai sama untuk setiap tahapan *decoding*, *weight* atau bobot ($\alpha_{t,i}$) masing-masing *hidden state* pada *encoder* berbeda-beda untuk tahapan *decoding* yang berbeda. Perhatikan Gambar. 11.23 sebagai ilustrasi (lagi-lagi, bentuk *encoder-decoder* yang disederhanakan). Terdapat suatu bagian grafik yang

menunjukkan distribusi bobot pada bagian *input representation* dan *attention*. Distribusi bobot pada *weighted sum* **c** adalah pembobotan yang bersifat generik, yaitu berguna untuk keseluruhan (rata-rata) kasus. Masing-masing *attention* (semacam *layer* semu) memiliki distribusi bobot yang berbeda pada tiap tahapan *decoding*. Walaupun *attention mechanism* sekalipun tidak sempurna, ide ini adalah salah satu penemuan yang sangat penting.



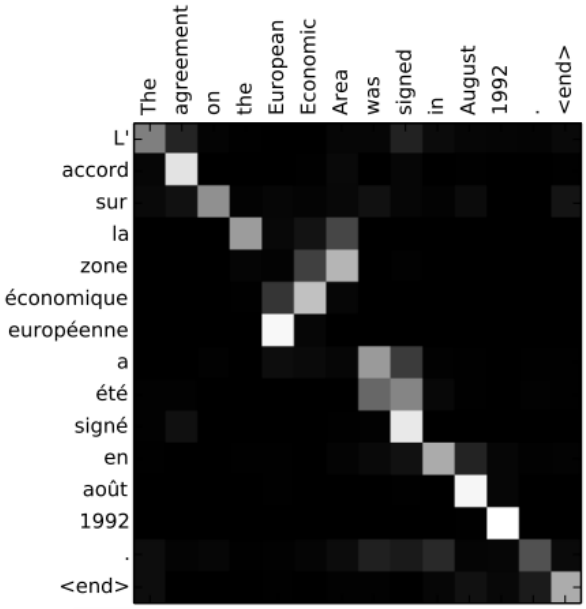
Gambar 11.23. *Encoder-decoder with attention.*

Seperti yang dijelaskan pada bab 9 bahwa *neural network* susah untuk dimengerti. *Attention mechanism* adalah salah satu cara untuk mengerti *neural network*. Contoh yang mungkin lebih mudah dipahami diberikan pada Gambar. 11.24 yang merupakan contoh kasus mesin translasi [49]. *Attention mechanism* mampu mengetahui *soft alignment*, yaitu kata mana yang harus difokuskan saat melakukan translasi bahasa (bagian *input* mana berbobot lebih tinggi). Dengan kata lain, *attention mechanism* memberi interpretasi kata pada *output* berkorespondensi dengan kata pada *input* yang mana. Sebagai informasi, menemukan cara untuk memahami (interpretasi) ANN adalah salah satu tren riset masa kini [48].

11.4.5 Variasi Arsitektur Sequence to Sequence

Selain RNN, kita juga dapat menggunakan *bidirectional* RNN (BiRNN) untuk mengikutsertakan pengaruh baik *hidden state* sebelum $(\mathbf{h}_1, \cdots, \mathbf{h}_{t-1})$ dan setelah $(\mathbf{h}_{t+1}, \cdots, \mathbf{h}_N)$ untuk menghitung *hidden state* sekarang $(\mathbf{h}_t)$ [77, 78, 79]. BiRNN menganggap $\mathbf{h}_t$ sebagai gabungan (*concatenation*) *forward hidden state* $\mathbf{h}_t^{\rightarrow}$ dan *backward hidden state* $\mathbf{h}_t^{\leftarrow}$, ditulis sebagai $\mathbf{h}_t = \mathbf{h}_t^{\rightarrow} + \mathbf{h}_t^{\leftarrow}$[17].

[17] Perhatikan! + disini dapat diartikan sebagai penjumlahan atau konkatenasi



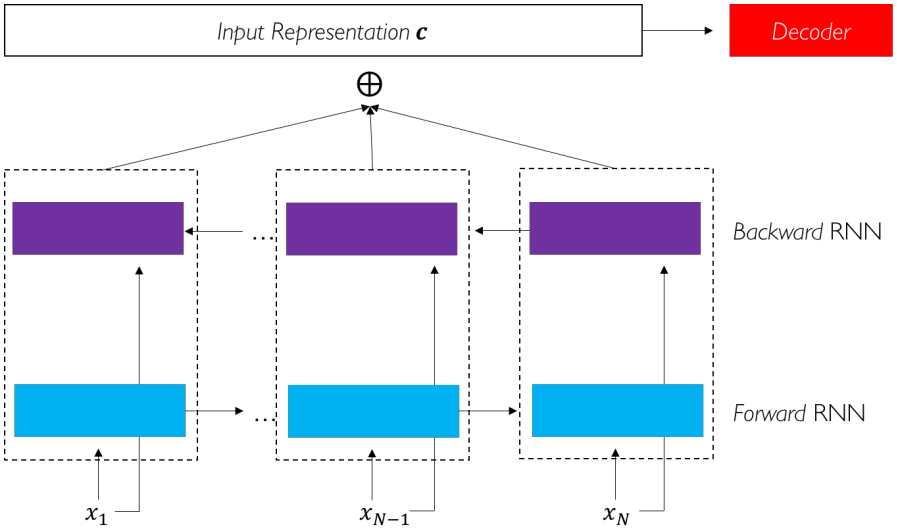
Gambar 11.24. *Attention mechanism* pada translasi bahasa [49]. Warna lebih terang merepresentasikan bobot (fokus/*attention*) lebih tinggi.

Forward hidden state dihitung seperti RNN biasa yang sudah dijelaskan pada subbab *encoder*, yaitu $\mathbf{h}_t^{\rightarrow} = f(\mathbf{h}_{t-1}^{\rightarrow}, \mathbf{e}_t)$.. *Backward hidden state* dihitung dengan arah terbalik $\mathbf{h}_t^{\leftarrow} = f(\mathbf{h}_{t+1}^{\leftarrow}, \mathbf{e}_t)$. Ilustrasi *encoder-decoder* yang menggunakan BiRNN dapat dilihat pada Gambar. 11.25.

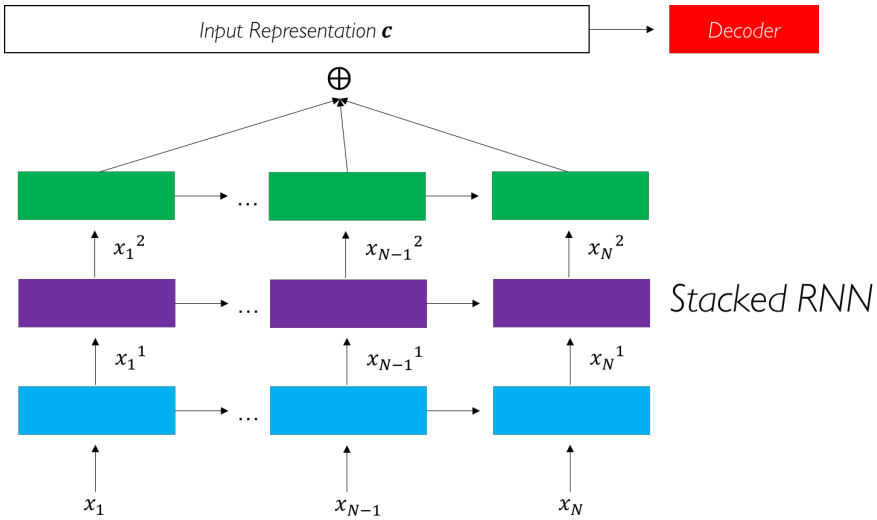
Selain variasi RNN menjadi BiRNN kita dapat menggunakan *stacked RNN* seperti pada Gambar. 11.26 dimana *output* pada RNN pertama bertindak sebagai *input* pada RNN kedua. *Hidden states* yang digunakan untuk menghasilkan representasi *encoding* adalah RNN pada tumpukan paling atas. Kita juga dapat menggunakan variasi *attention mechanism* seperti *neural checklist model* [80] atau *graph-based attention* [81]. Selain yang disebutkan, masih banyak variasi lain yang ada, silahkan eksplorasi lebih lanjut sendiri.

11.4.6 Rangkuman

Sequence to sequence adalah salah satu bentuk *conditioned generation*. Artinya, menggunakan RNN untuk menghasilkan (*generate*) suatu sekuens *output* yang dikondisikan oleh variabel tertentu. Diktat ini memberikan contoh bagaimana menghasilkan suatu sekuens *output* berdasarkan sekuens *input* (*conditioned on a sequence of input*). Selain *input* berupa sekuens, konsep ini juga dapat diaplikasikan pada bentuk lainnya. Misalnya, menghasilkan *caption* saat input yang diberikan adalah sebuah gambar [82]. Kita ubah *encoder* menjadi sebuah



Gambar 11.25. *Encoder-decoder* dengan BiRNN.



Gambar 11.26. *Encoder-decoder* dengan *stacked* RNN.

CNN (ingat kembali sub bab 11.1) dan *decoder* berupa RNN [82]. Gabungan CNN-RNN tersebut dilatih bersama menggunakan metode *backpropagation*.

Perhatikan, walaupun memiliki kemiripan dengan *hidden markov model*, *sequence to sequence* bukanlah *generative model*. Pada *generative model*, kita ingin memodelkan *joint probability* $p(x, y) = p(y \mid x)p(x)$ (walaupun secara tidak langsung, misal menggunakan teori Bayes). *Sequence to sequence* adalah *discriminative model* walaupun *output*-nya berupa sekuens, ia tidak memodelkan $p(x)$ (berbeda dengan *(*hidden markov model). Kita ingin memodelkan *conditional probability* $p(y \mid x)$ secara langsung, seperti *classifier* lainnya (e.g., *logistic regression*). Jadi yang dimodelkan antara *generative* dan *discriminative model* adalah dua hal yang berbeda.

11.5 Arsitektur Lainnya

Selain arsitektur yang sudah dipaparkan, masih banyak arsitektur lain baik bersifat generik (dapat digunakan untuk berbagai karakteristik data) maupun spesifik (cocok untuk data dengan karakteristik tertentu atau permasalahan tertentu) sebagai contoh, *Restricted Boltzman Machine*[18] dan *General Adversarial Network* (GaN)[19]. Saat buku ini ditulis, GaN dan *adversarial training* sedang populer.

Soal Latihan

11.1. POS *tagging* Pada subbab 11.3, disebutkan bahwa *bidirectional recurrent neural network* lebih cocok untuk persoalan POS *tagging*. Jelaskan mengapa! (hint: bab 7)

11.2. Eksplorasi Jelaskanlah pada teman-temanmu apa dan bagaimana prinsip kerja:

(a) *Restricted Boltzman Machine*
(b) *General Adversarial Network*

[18] `https://deeplearning4j.org/restrictedboltzmannmachine`
[19] `https://deeplearning4j.org/generative-adversarial-network`

Bagian IV

Aplikasi dan Topik Tambahan

12

Penerapan Pembelajaran Mesin

> "Leading is not the same as being the leader. Being the leader means you hold the highest rank, either by earning it, good fortune or navigating internal politics. Leading, however, means that others willingly follow you – not because they have to, not because they are paid to, but because they want to."
>
> Simon Sinek

Bab ini memuat contoh penggunaan *machine learning* untuk dua permasalahan praktis yaitu: (1) sistem rekomendasi dan (2) sistem peringkasan dokumen. Dua domain ini dipilih karena tidak asing (*familiar*) bagi penulis. Seperti yang sudah dideskripsikan pada bab-bab sebelumnya, penerapan *machine learning* pada suatu domain membutuhkan pengetahuan/keahlian pada domain tersebut. Bab ini tidak akan membahas domain secara detail, tetapi secara abstrak (bertujuan memberikan gambaran/pengenalan). Untuk mengerti domain yang dibahas secara mendalam, silakan membaca sumber lainnya. Bab ini akan memuat secara sangat singkat, apa guna *machine learning* dan pada contoh kasus seperti apa teknik *machine learning* diterapkan pada suatu domain permasalahan. Tujuan bab ini adalah untuk memberikan gambaran, bahwa mengetahui *machine learning* saja mungkin tidak cukup. Sekali lagi kami ingin menekankan, pembaca harus mengerti domain aplikasi.

12.1 Sistem Rekomendasi

Bagian sistem rekomendasi ditulis oleh **Candy Olivia Mawalim** (*Japan Advanced Institute of Science and Technology*). Kami ingin berterima kasih atas sumbangsih yang diberikan.

Salah satu penerapan pembelajaran mesin adalah sistem rekomendasi. Sistem rekomendasi dimotivasi oleh keinginan pemilik usaha untuk meningkatkan penjualan dengan cara mengerti pola pembelian pengguna. Aplikasi yang memanfaatkan sistem rekomendasi dapat kita temukan dalam kehidupan sehari-hari, misalnya *Youtube* yang memberikan rekomendasi video berdasarkan riwayat video yang telah kita lihat sebelumnya dan Amazon yang merekomendasikan produknya dengan cara menawarkan produk yang sering dibeli pengguna lain yang memiliki karakteristik yang "mirip" dengan kita. Ada dua komponen penting pada sistem rekomendasi yaitu: pengguna dan *item*. Pengguna adalah sang pengguna sistem, sementara *item* dapat berupa video, buku, dan lain sebagainya (produk/layanan yang ditawarkan sistem).

Secara umum, terdapat dua teknik untuk membangun sistem rekomendasi yaitu: (1) *content-based filtering* dan (2) *collaborative filtering*. Teknik pertama berfokus pada karakteristik pengguna secara spesifik. Teknik kedua berfokus pada selera terhadap suatu *item*. Dalam sistem rekomendasi, teknik *machine learning* dapat digunakan untuk memprediksi *item* yang disukai pengguna. Dua subbab berikutnya memuat aplikasi teknik *machine learning* untuk kedua teknik sistem rekomendasi(12.1.1 dan 12.1.2).

12.1.1 Content-based Filtering

Teknik membangun sistem rekomendasi berdasarkan *content-based filtering* memanfaatkan informasi mengenai profil seorang pengguna beserta uraian item yang sangat menarik bagi pengguna tersebut [83]. Profil pengguna diartikan sebagai karakteristik (atribut dan *behavior*) yang dimiliki pengguna. Atribut pengguna misalnya *gender*, kewarganegaraan, umur, dan lain-lain. Informasi mengenai *behavior* mencakup karakteristik *item* yang seorang pengguna sukai. Misalkan *item* adalah film, karakteristik film dapat ditentukan dari aktor yang terlibat dalam film, pembuat film, tahun pembuatan film, dan *genre* dari film (misalnya *action*, *horror*, *comedy*). Dengan karateristik ini, kita dapat menentukan kemiripan antar-film.

Kedua informasi ini dapat direpresentasikan sebagai vektor (ekuivalen dengan *feature vector*) agar mudah untuk dilakukan operasi aritmatika (dikenal dengan istilah ***user embedding*** dan ***item embedding***. Cara melakukan *embedding* untuk profil pengguna dan uraian item mirip dengan cara *word embedding* yang telah dijelaskan pada subbab 10.4.3).

Rekomendasi *item* yang diberikan pada pengguna adalah *item* yang paling mungkin disukai pengguna berdasarkan karakteristiknya. Agar mendapat gambaran lebih jelas, penulis mengajak pembaca untuk mengikuti tutorial

pembuatan sistem rekomendasi film sangat sederhana menggunakan dataset MovieLens [1] berukuran kecil (100K). Dataset ini memuat berbagai informasi sebagai berikut:

1. Informasi *rating* pengguna untuk masing-masing film.
2. Informasi film, misalkan berupa *genre* dan tanggal *release*.
3. Demografik pengguna, misal usia, *gender*, pekerjaan, dan lain lain.

Untuk menyederhanakan tutorial, kami hanya akan menjelaskan contoh penggunaan *genre* film untuk membuat sistem rekomendasi. Pertama-tama, kita bangun representasi *item embedding*, yaitu *genre* untuk masing-masing film (Tabel 12.1). Setiap baris pada tabel tersebut merepresentasikan *item embedding* untuk suatu film. Perhatikan! Setiap sel pada tabel diisi nilai "1" apabila film memiliki *genre* yang tertera pada kolom bersesuaian, "0" apabila tidak.

MovieId	Adventure	Animation	Children	Comedy
6	1	1	1	1
22	1	0	1	0
50	0	0	0	1

Tabel 12.1. Representasi *item embedding* untuk film berdasarkan *genre*.

Kedua, kita bangun representasi *user embedding*, yaitu apakah pengguna menyukai suatu film atau tidak (biner). Suka atau tidaknya pengguna terhadap suatu film dapat dilihat berdasarkan *rating* yang ia berikan. Sebagai contoh sederhana, kita anggap apabila pengguna menyukai suatu film apabila memberi nilai *rating* lebih dari atau sama dengan 4. Sebagai contoh, perhatikan Tabel 12.2. Apabila user menyukai suatu film , nilai "1" diisi pada kolom `Like or Not`, dan "0" apabila sebaliknya.

`UserId`	`MovieId`	`Like or Not`
1	6	0
1	22	0
1	50	1

Tabel 12.2. Representasi *user embedding* berdasarkan *rating* yang diberikan pengguna.

Berdasarkan *item embedding* dan *user embedding* yang kita punya, kita ganti kolom `MovieId` pada Tabel 12.2 menggunakan baris pada *item embedding*. Sekarang, kita memiliki dataset *behavior* pengguna dengan `UserId`

[1] `https://grouplens.org/datasets/movielens/`

= 1 (Tabel 12.3). Perhatikan! Tabel tersebut seperti dataset *machine learning* yang sudah kamu pelajari pada bab-bab sebelumnya. Diberikan *feature vector* dan kelas (`Like or Not`) yang berkorespondensi. Menggunakan data

`MovieId`	`Adventure`	`Animation`	`Children`	`Comedy`	`Like or Not`
6	1	1	1	1	0
22	1	0	1	0	0
50	0	0	0	1	1

Tabel 12.3. Dataset *behavior* pengguna dengan `UserId` = 1.

seperti pada Tabel 12.3, kita dapat menggunakan teknik *machine learning* untuk memprediksi apakah suatu pengguna akan menyukai film tertentu, berdasarkan *genre* yang dimuat oleh film tersebut.

Teknik *content-based filtering* memiliki keunggulan dimana kita tidak memerlukan banyak informasi tentang pengguna lain. Kita hanya memerlukan informasi uraian *item* dan informasi karakteristik suatu pengguna. Hal ini mengakibatkan rekomendasi yang diberikan sangat bergantung pada kepribadian pengguna. Apabila pengguna tidak konsisten, sistem rekomendasi juga bingung.

12.1.2 Collaborative Filtering

Teknik ini diperkenalkan oleh Paul Resnick dan Hal Varian pada 1997 [84]. Prinsip *collaborative filtering* adalah asumsi bahwa selera penggunaan terhadap suatu *item* cenderung sama dari waktu ke waktu [85]. Pada contoh kasus sederhana untuk sistem rekomendasi film, teknik ini memanfaatkan informasi *rating* dari banyak pengguna. Kita dapat merepresentasikan tingkah laku (*behaviour*) semua pengguna menggunakan matriks utilitas dimana baris merepresentasikan profil pengguna dan kolom merepresentasikan *item*. Sebagai contoh, perhatikanlah Tabel 12.4.

	$item_1$	$item_2$	...	$item_N$
$user_1$	2	3	...	4
$user_2$	5	3	...	1
...				
$user_3$	4	1	...	2

Tabel 12.4. Matriks Utilitas.

Ada dua metode varian *collaborative filtering* yaitu: (1) *neighborhood-based collaborative filtering* (*memory-based method*) dan (2) *model-based collabora-*

tive filtering. Method *neighborhood-based collaborative filtering* bekerja dengan fakta bahwa pengguna yang "mirip" memiliki pola yang "mirip" dalam memberikan *rating* untuk *item* [86] (pada Tabel 12.4, pengguna yang mirip memiliki baris yang mirip). Selain itu, *item* yang memiliki kemiripan, akan memiliki pola *rating* yang mirip (pada Tabel 12.4, *item* yang mirip memiliki kolom yang mirip). Dengan itu, kita dapat menggunakan perhitungan kemiripan vektor untuk menghitung pengguna mana yang mirip dengan suatu pengguna p. Saat memberikan rekomendasi film bagi pengguna p, kita tunjukkan film-film yang pernah ditonton pengguna yang mirip dengannya, atau kita tunjukkan film-film yang mirip dengan film-film yang pernah ditonton oleh pengguna p.

Untuk *model-based collaborative filtering*, prediksi dibangun dengan menggunakan teknik *machine learning* [86]. Untuk suatu sistem dengan data yang *sparse*, matriks utilitas seperti Tabel 12.4 tidaklah efisien secara memori. Kita dapat memanfaatkan teknik-tenik seperti *matrix factorization/principal component analysis* dan *autoencoder* untuk mengurangi ukuran matriks. Silakan membaca lebih lanjut materi-materi tersebut (ingat kembali materi bab 10).

12.2 Peringkasan Dokumen

Meringkas dokumen berarti mengerti keseluruhan isi teks/dokumen, kemudian mampu menyampaikan kembali **sebanyak/seakurat mungkin** maksud dokumen asli, ke dalam bentuk yang lebih singkat [87, 88, 49]. Suatu ringkasan harus lebih pendek dibanding dokumen asli. Dengan demikian, sulit untuk dikatakan bahwa suatu ringkasan dapat memuat keseluruhan isi dokumen. Karena itu, ringkasan hanya memuat **sebanyak/seakurat mungkin** maksud dokumen asli. Ada beberapa jenis peringkasan dokumen dilihat dari berbagai sudut pandang, misal:

1. Jumlah dokumen yang diringkas, meringkas satu dokumen (*single-document summarization*) [89] atau banyak dokumen menjadi satu ringkasan (*multi-document summarization*) [90].
2. Indikatif atau Informatif. Indikatif berarti menyediakan *pointer* ke bagian dokumen (misal membuat daftar isi). Informatif berarti meyampaikan sebanyak/seakurat mungkin maksud dokumen asli ke dalam bentuk lebih singkat. Pada masa sekarang, hampir seluruh riset peringkasan dokumen mengacu untuk menyediakan ringkasan yang informatif.
3. Domain. Hasil ringkasan bergantung pada domain dokumen, misalkan berita atau novel. Pada domain berita, hal yang penting untuk dimuat dalam ringkasan adalah 5W1H (*what, who, when, whom, where, how*) [91], sementara pada novel mungkin kita ingin tahu kejadian-kejadian yang ada (beserta urutannya).
4. Generik, *query-based*, atau *tailored*. Generik berarti menghasilkan ringkasan untuk umum, dalam artian tidak ada *target user* secara spesifik (contohnya adalah *review* buku) [87]. *Query-based* artinya menghasilkan ringkasan

dokumen berdasarkan *query* yang diberikan *user* (ringkasan adalah informasi yang dibutuhkan oleh *user* yang ditemukan di dokumen) [92]. *Tailored* berarti menyajikan informasi dengan tipe spesifik. Misalkan pada berita terdapat informasi 5W1H, kita hanya ingin mencari tahu informasi *how*.

5. Ekstraktif [93] atau abstraktif [49]. Ekstraktif berarti ringkasan hanya memuat unit informasi yang ada di dokumen asli. Analoginya seperti memilih potongan dari dokumen asli sebagai ringkasan (*copy*). Abstraktif berarti ringkasan dapat memuat unit informasi yang mungkin tidak ada di dokumen asli. Analoginya seperti mengerti dokumen kemudian menulis ulang (parafrase).

Hal terpenting untuk diingat adalah peringkasan dokumen (apa yang diringkas, ringkasan harus memuat apa, dsb) sangat bergantung pada **tujuan ringkasan** (siapa pembaca ringkasan, untuk apa ringkasan dihasilkan). Diktat ini akan membahas *framework* (kerangka kerja/berpikir) peringkasan dokumen dan bagaimana *machine learning* membantu peringkasan dokumen.

Secara umum, ada beberapa tahap pada proses peringkasan dokumen secara otomatis, terlepas dari tipe peringkasan dokumen yang dijabarkan [87, 94, 95, 88, 96, 91, 90, 89, 49, 97]:

1. *Representation* – Melakukan pemisahan dan representasi unit informasi; pada umumnya adalah kalimat, kata, atau frase. Dalam artian, kita menganggap dokumen tersusun atas sekuens kalimat, kata, atau frase. Kita potong-potong unit informasi pada dokumen lalu, unit informasi dapat direpresentasikan sebagai vektor untuk tiap unitnya [98]. Kita pun dapat menggali hubungan antara unit informasi (misal direpresentasikan sebagai graf) untuk kemudian mencari unit mana yang sentral [91, 99].
2. *Filtering* – Menyaring informasi. Unit manakah yang penting/tidak penting. Unit mana yang dibutuhkan/tidak dibutuhkan. Unit mana yang relevan/tidak relevan. Unit mana yang representatif/tidak representatif. Teknik menyaring informasi sangat bergantung pada representasi unit pada tahap sebelumnya (e.g., vektor atau graf). Secara singkat, tahap ini memilih unit informasi berdasarkan objektif.
3. *Generation* – Menghasilkan (*generate*) ringkasan. Bagian ini membutuhkan pengetahuan mengenai bidang pemrosesan bahasa alami (*natural language processing*) (NLP). Pada diktat ini, tidak akan dibahas.

Teknik *machine learning* berperan penting pada tahap 1 dan 2. Kami akan memberikan studi kasus pada subbab 12.2.1 agar lebih jelas. Sistem peringkasan dokumen yang membagi-bagi proses menjadi tahapan-tahapan tersebut disebut *pipelined approach*. Artinya kita melakukan proses diatas sebagai tahap-tahap berbeda (independen satu sama lain). Selain *pipelined approach*, kita juga dapat memandang peringkasan dokumen dengan *single-view approach* (subbab 12.2.2), artinya keseluruhan proses tersebut terjadi bersamaan.

12.2.1 Pipelined Approach

Subbab ini akan memuat cerita singkat tentang peringkasan *paper* [94, 97]. Berdasarkan teori *argumentative zoning* [94], *paper* terdiri dari zona-zona dengan tujuan komunikatif yang berbeda. Dengan bahasa yang lebih mudah, tiap kalimat (unit informasi) pada *paper* memiliki tujuan komunikasi yang berbeda. Misalnya ada kalimat yang menyatakan tujuan penelitian, latar belakang penelitian, atau hasil penelitian. Tujuan komunikasi kalimat disebut *rethorical categories* [100].

Ringkasan, dalam bentuk abstrak atau judul *paper* memiliki pola [101, 97]. Dalam artian, suatu ringkasan bisa jadi hanya memuat informasi dengan *rhetorical categories* tertentu (*tailored summary*). *Machine learning* dapat digunakan untuk mengklasifikasikan kalimat (unit informasi) ke kelas masing-masing [100, 97]. Pertama-tama setiap kalimat direpresentasikan menjadi *feature vector* (ingat kembali materi bab 3 dan bab 4). Sebagai contoh, *paper* [97] merepresentasikan kalimat pada *paper* sebagai *feature vector* berdasarkan fitur-fitur sebagai berikut:

1. Posisi. Lokasi kalimat dibagi dengan panjang teks (numerik).
2. Kata kunci (*lexicon*). Apakah kalimat memuat kata kunci yang eksklusif untuk *rhetorical category* tertentu (biner – ya atau tidak).
3. Bobot. Jumlah bobot TF-IDF (ingat bab 10) kata pada kalimat (numerik).
4. Kategori sebelumnya. *Rhetorical category* kalimat sebelumnya (nominal).

Setelah diubah menjadi *feature vector*, teknik *machine learning* digunakan untuk mengklasifikasikan kalimat menjadi kelas *rhetorical categories* yang sesuai. Dengan demikian, kita dapat menyaring kalimat berdasarkan tipe informasi mana yang kita inginkan, berdasarkan tujuan peringkasan. Selanjutnya, teknik NLP digunakan untuk memproses informasi yang disaring untuk menghasilkan ringkasan. Materi tersebut diluar bahasan diktat ini.

12.2.2 Single-view Approach

Pada *pipelined approach*, setiap tahapan peringkasan dokumen (*representation*, *filtering*, dan *generation*) dianggap sebagai tahap yang independen satu sama lain. Permasalahannya adalah kesalahan pada suatu tahap akan mempengaruhi tahap berikutnya. Opsi lain adalah dengan melakukan keseluruhan proses sebagai satu tahapan yang utuh (*end-to-end*). Metode ini memiliki kaitan yang erat dengan teknik *machine translation*, yang pada masa sekarang populer didekati dengan teknik *deep learning* (*sequence to sequence encoder–decoder*) [49, 73, 72, 81, 102, 103]. Dokumen tersusun atas sejumlah N sekuens unit informasi $\mathbf{u} = u_1, u_2, \ldots, u_N$. Sebuah ringkasan adalah M buah sekuens unit informasi $\mathbf{r} = r_1, r_2, \ldots, r_M$ dimana $M < N$. Tujuan peringkasan adalah untuk mencari $\mathbf{r}$ terbaik, sedemikian sehingga dapat memenuhi persamaan 12.1.

$$\arg\max_{\mathbf{r}} P(\mathbf{r} \mid \mathbf{u}) \tag{12.1}$$

Pada umumnya, terdapat suatu variabel tambahan θ yang mengendalikan (*govern*) probabilitas tersebut. Sehingga secara lebih tepat, persamaan 12.1 diubah menjadi persamaan 12.2.

$$\arg\max_{\mathbf{r}} P(\mathbf{r} \mid \mathbf{u}, \theta) \tag{12.2}$$

Perhatikan, persamaan 12.2 adalah bentuk yang dapat dimodelkan menggunakan *sequence to sequence encoder–decoder*. Selain peringkasan dokumen, banyak permasalahan pada bidang *natural language processing* memiliki bentuk yang "mirip", seperti: *part-of-speech tagging* [104], *named entity recognition* [105], mesin translasi [73], dan rekonstruksi paragraf [106].

12.3 Konklusi

Teknik *machine learning* sangatlah berguna untuk berbagai macam permasalahan. Tetapi perlu dipahami bahwa teknik yang ada belum mampu memodelkan proses berpikir manusia dengan benar. Proses berpikir manusia sangatlah kompleks, dan model yang ada sekarang ini adalah bentuk simplifikasi. Sebagai contoh, seorang bayi sekali melihat anjing mungkin akan mengetahui anjing lainnya walaupun tidak memiliki rupa yang sama persis. Sementara itu, model *machine learning* harus diberikan banyak sampel. Manusia memiliki otak yang luar biasa hebat karena mampu belajar dengan sedikit contoh.

Persoalan-persoalan yang diberikan pada buku ini secara umum mencakup *supervised* dan *unsupervised learning* saja. Terdapat satu pemodelan masalah penting lainnya yaitu *reinforcement learning* dimana kita ingin memaksimalkan hasil untuk sekuens aksi (sekuens keputusan). Setiap keputusan dapat diberikan bobot yang berbeda. Kemudian, agen memaksimalkan nilai untuk sekuens keputusan (*cummulative reward*)[2]. Pada *supervised learning*, kita hanya perlu membuat satu keputusan saja (menggolongkan data ke kelas mana). Pada konferensi-konferensi, para *master* banyak menyebutkan masa depan *machine learning* berada pada *unsupervised* atau *reinforcement learning*. Aplikasi mutakhir yang ada sekarang ini didominasi oleh *supervised learning*. Pada kenyataannya, sebagian besar (hampir semua) data yang ada di dunia ini tidak berlabel. Karena itu, representasi data secara *unsupervised* menjadi topik riset hangat (*representation learning*).

Buku ini telah menjelaskan berbagai macam teknik, tetapi perlu diketahui bahwa materi yang diberikan adalah simplifikasi agar mampu dipahami secara intuitif. Selain itu, buku ini di desain sebagai materi pengantar saja. Untuk memahami teknik *machine learning* secara lebih jauh dan sempurna, penulis menyarankan untuk membaca buku referensi [8, 11].

[2] `https://en.wikipedia.org/wiki/Reinforcement_learning`

Materi yang disajikan pada buku ini adalah persoalan-persoalan *machine learning* dari sudut pandang optimisasi. Yaitu mencari himpunan (*set of*) parameter agar model pembelajaran yang dibangun mampu memberi keputusan yang optimal. Kami menyarankan pembaca untuk mencari referensi lebih jauh tentang *machine learning* dari sudut pandang eksplorasi, yaitu mencari *concept space* dimana model dapat bekerja dengan "baik". Sebagai contoh, kita memiliki sebuah mesin (fisik) dengan berbagai macam konfigurasi (parameter). Apabila kita menjalankan mesin dengan konfigurasi parameter tertentu, mesin akan rusak. Tugas kita adalah mencari *parameters space* dimana mesin dapat berjalan. Sedangkan, pada persoalan optimisasi, kita mencari satu konfigurasi terbaik.

Machine learning adalah bidang yang sangat luas (lebih luas dari apa yang diceritakan pada buku ini) dan berkembang pesat. Penulis menyarankan pembaca untuk membaca makalah dari konferensi[3] *top-tier* untuk mengetahui perkembangan terkini. Sebagai contoh (diurutkan berdasarkan abjad)[4]:

- AAAI. AAAI Conference on Artificial Intelligence
- ACL. Annual Meeting of Association for Computational Linguistics
- CVPR. IEEE Conference on Computer Vision and Pattern Recognition
- EMNLP. Empirical Methods in Natural Language Processing
- ICCV. IEEE International Conference on Computer Vision
- ICLR. International Conference on Learning Representation
- ICML. International Conference on Machine Learning
- IJCAI. International Conference on Artificial Intelligence
- INTERSPEECH. Conference of the International Speech Association
- NIPS. Neural Information Processing System
- SIGIR. ACM Special Interest Group in Information Retrieval
- SIGKDD. ACM Special Interest Group in Knowledge Discovery and Data Mining

12.4 Saran Buku Lanjutan

Penulis ingin memberikan beberapa rekomendasi bacaan pembelajaran mesin selanjutnya (teoritis). Buku-buku berikut (diurutkan secara subjektif berdasarkan kualitas konten dan kemudahan dimengerti):

1. *Deep Learning* oleh Ian Goodfellow et al. [11]. Banyak yang menggganggap buku ini sebagai "kitab" *deep learning* modern. Buku ini juga men-

[3] Kamu lebih mudah mengetahui informasi terkini dari konferensi dibanding jurnal karena proses *review* yang lebih cepat. Pada bidang keilmuan tertentu, konferensi lebih penting dibanding jurnal.

[4] Konferensi penting untuk bidang pemrosesan bahasa alami terdaftar pada `http://aclweb.org/anthology/` atau `https://aclanthology.coli.uni-saarland.de/`

cakup materi matematika dasar (e.g., aljabar linier) yang dibutuhkan untuk mengerti *deep learning*.

2. *Neural Network Methods in Natural Language Processing* oleh Goldberg [1]. Buku ini ditujukan sebagai bahan transisi bagi peneliti bidang pemrosesan bahasa alami untuk menggunakan metode *neural network*. Apabila kamu tidak tertarik dengan pemrosesan bahasa alami, kamu bisa membaca bab 1-5 pada buku ini sebagai pengenalan *neural network*.
3. *Pattern Recognition and Machine Learning* oleh Christopher M. Bishop [8]. Menurut penulis, banyak yang tahu buku ini karena dianggap sebagai "kitab". Penjelasan buku ini sangat matematis dan relatif berat untuk dimengerti. Tetapi, kamu dapat menjadi *master* apabila memahami seluruh materi pada buku ini. Algoritma *machine learning* yang disajikan juga relatif lebih "klasik" dibanding rekomendasi pertama.
4. *Machine Learning* oleh Tom Mitchel [4]. Buku ini memuat materi *machine learning* yang cukup "klasik". Buku ini kurang lebih cocok sebagai pengenalan, tapi relatif kurang dalam.

Sedikit tambahan pesan sponsor karena penulis berlatar belakang dari bidang pemrosesan bahasa alami, penulis menyarankan membaca buku-buku (teoritis) berikut sebagai dasar pengetahuan pemrosesan bahasa alami (diurutkan dari konten paling dasar):

1. *Foundations of Statistical Natural Language Processing* oleh Christopher D. Mannning dan Hinrich Schutze [56]. Buku ini dapat dideskripsikan dengan satu kata, **TOP**. Buku ini memuat materi pemrosesan bahasa alami dengan sudut pandang matematis, dapat dianggap sebagai *framework* berpikir yang modern.
2. *Speech and Language Processing* oleh Daniel Jurafsky dan James H. Martin [12]. Buku ini jauh lebih tebal dari buku pertama dan memuat materi yang lebih luas. Penulis merekomendasikan buku ini untuk mengetahui permasalahan-permasalahan pemrosesan bahasa alami.
3. *An introduction to Information Retrieval* oleh Manning et al. [57]. Walaupun berjududul *information retrieval*, penulis pertama kali mengenal konsep *embedding* dari buku ini. Buku ini ditulis dengan apik.
4. *Neural Network Methods in Natural Language Processing* oleh Goldberg [1]. Apabila kamu telah membaca buku-buku yang disebutkan sebelumnya, kamu dapat membaca buku ini untuk transisi ke metode *neural network*.

Pada saat menulis buku ini, penulis berharap bisa menulis pada level diantara buku Tom Mitchel [4] dan Bishop [8] yaitu cukup matematis, lumayan mudah dipahami, dan lumayan dalam. Mudah-mudahan pembaca merasa tujuan ini tercapai dan menikmati membaca buku ini. Sebagai kalimat penutup, terimakasih sudah membaca sampai tahap ini.

Soal Latihan

12.1. Cold Start Problem

(a) Jelaskan apa itu *cold start problem* pada sistem rekomendasi, serta bagaimana cara menangani permasalahan tersebut!
(b) Pada teknik sistem rekomendasi manakah (*content-based filtering* atau *collaborative filtering*) *cold start problem* mungkin muncul? Mengapa?

12.2. Eksplorasi Sistem Rekomendasi

Bangunlah suatu sistem rekomendasi dengan teknik *collaborative filtering* pada dataset MovieLens dengan memanfaatkan library *recommenderlab* [5]!

12.3. Peringkasan Dokumen

(a) Presentasikanlah di kelasmu, *paper* sistem peringkasan dokumen otomatis oleh Kupiec et al. (1995) [98] [6] yang menggunakan *pipelined approach*!
(b) Presentasikanlah di kelasmu, *paper* sistem peringkasan dokumen otomatis oleh Cheng and Lapata [103] [7] yang menggunakan *single-view approach*!
(c) Jelaskan perbedaan, kelebihan, dan kelemahan pendekatan-pendekatan sistem peringkasan dokumen!

[5] `https://cran.r-project.org/web/packages/recommenderlab/index.html`
[6] `https://dl.acm.org/citation.cfm?id=215333`
[7] `http://www.aclweb.org/anthology/P16-1046`

Referensi

1. Yoav Goldberg. *Neural Network Methods in Natural Language Processing.* Morgan & Claypool Publishers, 2017.
2. Peter Linz. *An Introduction to Formal Language and Automata.* Jones and Bartlett Publishers, Inc., USA, 2006.
3. Ronald Brachman and Hector Levesque. *Knowledge Representation and Reasoning.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.
4. Tom Mitchell. *Machine Learning.* McGraw-Hill, 1997.
5. Stuart Russel and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 1995.
6. Jonathan Gratch and Stacell Marsella. Computationally modelling human emotion. *Communications of the ACM*, 57(12):71–99, 2014.
7. Masayu Leylia Khodra and Dessi Puji Lestari. Odd semester lecture on machine learning. Lecture of Institute Teknologi Bandung, 2015.
8. Christopher M. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.
9. Sumio Watanabe and Hidetoshi Nishimori. Fall lecture note on statistical learning theory. Lecture note for Tokyo Institute of Technology, 2016.
10. Brian Caffo. *Statistical Inference for Data Science.* Lean Publishing, 2015.
11. Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016.
12. Daniel Jurafsky and James H. Martin. *Speech and Language Processing Second Edition.* Prentice Hall, 2009.
13. Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory.* Wiley, 1991.
14. Hidetoshi Nishimori. *Statistical Physics of Spin Glasses and Information Processing: An Introduction.* Clarendon Press, 2001.
15. Sharon L. Myres Ronald E. Walpole, Raymond H. Myers and Keying Ya. *Probability and Statistics for Engineers and Scientists.* Prentice Hall, 2012.
16. Gilbert Strang. *Linear algebra and its applications.* Thomson, Brooks/Cole, Belmont, CA, 2006.
17. Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data Minin gg: Practical Machine Learning Tools and Techniques.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2011.
18. Jeff Leek. *The Elements of Data Analytic Style.* Leanpub, 2015.

19. Takao Terano and Tsuyoshi Murata. Spring lecture on machine learning. Lecture of Tokyo Institute of Technology, 2017.
20. Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 311–318, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.
21. Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Proc. ACL workshop on Text Summarization Branches Out*, page 10, 2004.
22. Irina Rish. An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46. IBM New York, 2001.
23. J. A. Hartigan and M. A. Wong. A k-means clustering algorithm. *JSTOR: Applied Statistics*, 28(1):100–108, 1979.
24. T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theor.*, 13(1):21–27, September 2006.
25. Marti A. Hearst. Support vector machines. *IEEE Intelligent Systems*, 13(4):18–28, July 1998.
26. John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization, 2010.
27. J. R. Quilan. *Discovering rules by induction from large collections of examples.* Edinburgh University Press, 1979.
28. J.R. Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106, March 1986.
29. C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.
30. Takao Terano and Tsuyoshi Murata. Spring lecture on machine learning. Lecture of Tokyo Institute of Technology, 2017.
31. L. R. Rabiner and B. H. Juang. An introduction to hidden markov models. *IEEE ASSp Magazine*, 1986.
32. James Allen. *Natural Language Understanding.* Benjamin-Cummings Publishing Co., Inc., 1995.
33. George Karypis Michael Steinbach and Vipin Kumar. A comparison of document clustering techniques. In *KDD Workshop on Text Mining*, pages 525 – 526, 2000.
34. Omer I. E. Mohamed Fathi H. Saad and Rafa E. Al-Qutaish. Comparison of hierarchical agglomerative algorithms for clustering medical documents. *International Journal of Software Engineering and Applications (IJSEA)*, 3(3), 2012.
35. Rajeev Rastogi Sudipto Guha and Kyuseok Shim. Cure: An efficient clustering algorithm for large databases. In *Proceedings of ACMSIGMOD International Conference on Management of Data*, pages 73 – 84, 1998.
36. Jack D. Cowan. Neural networks: The early days. In *Proceedings of Advances in Neural Information Processing Systems 2*, 1989.
37. Amir Atiya. *Learning Algorithms for Neural Network.* PhD thesis, California Institute of Technology, 1994.
38. Al. Cripps. Using artificial neural nets to predict academic performance. In *Proceedings of the 1996 ACM Symposium on Applied Computing*, pages 33–37, 1996.

39. Thomas Mikolov. *Statistical Language Models Based on Neural Networks*. PhD thesis, Brno University of Technology, 2012.
40. Kai Chen Greg Corrado Thomas Mikolov, Ilya Sutskever and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of CoRR*, 2013.
41. Gred Corrado Thomas Mikolov, Kai Chen and Jeffrey Dean. Efficient estimation of word representations in vector space. In *Proceedings of CoRR*, 2013.
42. Kai Yu. Large-scale deep learning at baidu. In *Proceedings of the 22nd ACM International Conference on Information and Knowledge Management*, pages 2211–2212, 2013.
43. M. A. Aizerman, E. A. Braverman, and L. Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning. In *Automation and Remote Control,*, number 25, pages 821–837, 1964.
44. F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.
45. Marvin L. Minsky and Seymour A. Papert. *Perceptrons: Expanded Edition*. MIT Press, Cambridge, MA, USA, 1988.
46. D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter Learning Internal Representations by Error Propagation, pages 318–362. MIT Press, Cambridge, MA, USA, 1986.
47. Tao Lei, Regina Barzilay, and Tommi Jaakkola. Rationalizing neural predictions. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 107–117, Austin, Texas, November 2016. Association for Computational Linguistics.
48. David Alvarez-Melis and Tommi Jaakkola. A causal framework for explaining the predictions of black-box sequence-to-sequence models. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 412–421, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.
49. Dzmitry Bahdanau, KyungHyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *Proceedings of the International Conference on Learning and Representation (ICLR)*, 2015.
50. Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal, September 2015. Association for Computational Linguistics.
51. Finale Doshi-Velez and Been Kim. A roadmap for a rigorous science of interpretability. In *ArXiv e-prints*.
52. Jeffrey L. Elman. Learning and development in neural networks: The importance of starting small. *Journal of Cognition*, (48):71–99, 1993.
53. Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 41–48, New York, NY, USA, 2009. ACM.
54. Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014.

55. Dzmitry Bahdanau Kyunghyun Cho, Bart van Merrienboer and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, Doha, Qatar, October 2014. Association for Computational Linguistics.
56. Christopher D. Manning and Hinrich Schutze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.
57. Prabhakar Raghavan Christopher D. Manning and Hinrich Schutze. *An Introduction to Information Retrieval*. Cambridge UP, 2009.
58. Andrew Y. Ng Richard Socher, Cliff Chiung-Yu Lin and Christopher D. Manning. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th International Conference on Machine Learning*, 2011.
59. Jean Y. Wu Jason Chuang Richard Socher, Alex Perelygin and Christopher D. Manning. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the Emperical Methods in Natural Language Processing*, 2013.
60. Erhc H. Huang Andrew Y. Ng Richard Socher, Jeffrey Pennington and Christoper D. Manning. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the Emperical Methods in Natural Language Processing*, 2011.
61. Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31$_{st}$ International Conference on Machine Learning*, 2014.
62. Richard Socher Jeffrey Pennington and Christopher D. Manning. Glove: Global vectors for word representation. In *Proceedings of the Emperical Methods in Natural Language Processing*, pages 1532 – 1543, 2014.
63. Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, March 2003.
64. Omer Levy, Yoav Goldberg, and Ido Dagan. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225, 2015.
65. Guoqiang Zhong, Li-Na Wang, Xiao Ling, and Junyu Dong. An overview on data representation learning: From traditional feature learning to recent deep learning. *The Journal of Finance and Data Science*, 2(4):265 – 278, 2016.
66. Peter D. Turney and Patrick Pantel. From frequency to meaning: Vector space models of semantics. *Journal of Artificial Intelligence Research*, (37):141–188, 2010.
67. Jan Wira Gotama Putra and Takenobu Tokunaga. Evaluating text coherence based on semantic similarity graph. In *Proceedings of TextGraphs-11: the Workshop on Graph-based Methods for Natural Language Processing*, pages 76–85, Vancouver, Canada, August 2017. Association for Computational Linguistics.
68. Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time-series. In M. A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*. MIT Press, 1995.
69. Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

70. Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.
71. Paul J. Werbos. Backpropagation through time: what does it do and how to do it. In *Proceedings of IEEE*, volume 78, pages 1550–1560, 1990.
72. Caglar Gulcehre Dzmitry Bahdanau Fethi Bougares HolgerSchwenk Kyunghyun Cho, Bart van Merrienboer and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics.
73. Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems*, NIPS'14, pages 3104–3112, Cambridge, MA, USA, 2014. MIT Press.
74. Yan Shao, Christian Hardmeier, Jörg Tiedemann, and Joakim Nivre. Character-based joint segmentation and pos tagging for chinese using bidirectional rnn-crf. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 173–183, Taipei, Taiwan, November 2017. Asian Federation of Natural Language Processing.
75. Tobias Horsmann and Torsten Zesch. Do lstms really work so well for pos tagging? – a replication study. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 727–736, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.
76. Barbara Plank, Anders Søgaard, and Yoav Goldberg. Multilingual part-of-speech tagging with bidirectional long short-term memory models and auxiliary loss. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 2: Short Papers*, 2016.
77. Ryohei Sasano Hiroya Takamura Yuta Kikuchi, Graham Neubig and Manabu Okumura. Controlling output length in neural encoder-decoders. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1328–1338, Austin, Texas, November 2016. Association for Computational Linguistics.
78. Ramesh Nallapati, Bowen Zhou, Cícero Nogueira dos Santos, and aglar Gülehre and Bing Xiang. Abstractive text summarization using sequence-to-sequence rnns and beyond. In *CoNLL*, 2016.
79. Yan-Kai Lin Cun-Chao Tu Yu Zhao Zhi-Yuan Liu Ayana, Shi-Qi Shen and Mao-Song Sun. Recent advances on neural headline generation. *Journal of Computer Science and Technology*, 32(4):768–784, Jul 2017.
80. Chloé Kiddon, Luke Zettlemoyer, and Yejin Choi. Globally coherent text generation with neural checklist models. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 329–339, Austin, Texas, November 2016. Association for Computational Linguistics.
81. Xiaojun Wan Jiwei Tan and Jianguo Xiao. Abstractive document summarization with a graph-based attentional neural model. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1171–1181, Vancouver, Canada, July 2017. Association for Computational Linguistics.

82. Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3156–3164, 2015.
83. Daniel Billsus and Michael J. Pazzani. The adaptive web. chapter Adaptive News Access, pages 550–570. Springer-Verlag, Berlin, Heidelberg, 2007.
84. Paul Resnick and Hal R. Varian. Recommender systems. *Commun. ACM*, 40(3):56–58, March 1997.
85. Daniar Asanov. Algorithms and methods in recommender systems. Berlin Institute of Technology, 2011.
86. Charu C. Aggrawal. *Recommender Systems: The Textbook*. Springer International Publishing Switzerland, 2016.
87. Eduard Hovy and Chin-Yew Lin. Automated text summarization and the summarist system. In *Proceedings of a Workshop on Held at Baltimore, Maryland: October 13-15, 1998*, TIPSTER '98, pages 197–214, Stroudsburg, PA, USA, 1998. Association for Computational Linguistics.
88. Liang Zhou and Eduard Hovy. Template-filtered headline summarization. In *In the Proceedings of the ACL workshop, Text Summarization Branches Out*, pages 56–60, 2004.
89. Amin Mantrach Carlos A. Colmenares, Marina Litvak and Fabrizio Silvestri. Heads: Headline generation as sequence prediction using an abstract feature-rich space. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 133–142, Denver, Colorado, May–June 2015. Association for Computational Linguistics.
90. Daniele Pighin Enrique Alfonseca and Guillermo Garrido. Heady: News headline abstraction through event pattern clustering. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1243–1253, Sofia, Bulgaria, August 2013. Association for Computational Linguistics.
91. Pierre-Etienne Genest and Guy Lapalme. Framework for abstractive summarization using text-to-text generation. In *Proceedings of the Workshop on Monolingual Text-To-Text Generation*, pages 64–73, Portland, Oregon, June 2011. Association for Computational Linguistics.
92. Shufeng Xiong and Donghong Ji. Query-focused multi-document summarization using hypergraph-based ranking. *Inf. Process. Manage.*, 52(4):670–681, July 2016.
93. David Zajic, Bonnie J. Dorr, and Richard Schwartz. Bbn/umd at duc-2004: Topiary. In *Proceedings of the North Americal Chapter of the Association for Computational Linguistics Workshop on Document Understanding*, pages 112–119, 2004.
94. Simone Teufel and Marc Moens. Argumentative classification of extracted sentences as a first step towards flexible abstracting. In *Advances in automatic Text Summarization*, pages 155–171. MIT Press, 1999.
95. Bonnie J. Dorr, David Zajic, and Richard Schwartz. Hedge trimmer: A parse-and-trim approach to headline generation. In Dragomir Radev and Simone Teufel, editors, *Proceedings of the HLT-NAACL 03 Text Summarization Workshop*, pages 1–8, 2003.
96. Jurij Leskovec, Natasa Milic-Frayling, and Marko Grobelnik. Extracting summary sentences based on the document semantic graph. *Microsoft Research*, 2005.

97. Jan Wira Gotama Putra. Rhetorical sentence classification for automatic title generation in scientific article. *TELKOMNIKA*, 15(2):656–664, 2017.
98. Jan Pedersen Julian Kupiec and Francine Chen. A trainable document summarizer. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '95, pages 68–73, New York, NY, USA, 1995. ACM.
99. Hans-Martin Ramsl Daraksha Parveen and Michael Strube. Topical coherence for graph-based extractive summarization. In *Conference on Empirical Methods in Natural Language Processing*, pages 1949–1954. The Association for Computational Linguistics, 2015.
100. Simone Teufel and Marc Moens. Summarizing scientific articles: Experiments with relevance and rhetorical status. *Comput. Linguist.*, 28(4):409–445, December 2002.
101. Diarmuid Ó Séaghdha and Simone Teufel. Unsupervised learning of rhetorical structure with un-topic models. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 2–13, Dublin, Ireland, August 2014. Dublin City University and Association for Computational Linguistics.
102. Vibhu O. Mittal Michele Banko and Michael J. Witbrock. Headline generation based on statistical translation. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, ACL '00, pages 318–325, Stroudsburg, PA, USA, 2000. Association for Computational Linguistics.
103. Jianpeng Cheng and Mirella Lapata. Neural summarization by extracting sentences and words. *CoRR*, abs/1603.07252, 2016.
104. Christopher D. Manning. Part-of-speech tagging from 97% to 100%: Is it time for some linguistics? In *Proceedings of the 12th International Conference on Computational Linguistics and Intelligent Text Processing - Volume Part I*, CICLing'11, pages 171–189, Berlin, Heidelberg, 2011. Springer-Verlag.
105. Lev Ratinov and Dan Roth. Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, CoNLL '09, pages 147–155, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.
106. Jiwei Li and Dan Jurafsky. Neural net models of open-domain discourse coherence. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 198–209, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

Biografi Penulis

Jan **Wira** Gotama Putra adalah mahasiswa pascasarjana di Computational Linguistics/Natural Language Processing group (Tokunaga-Fujii-lab), Artificial Intelligence Major, Department of Computer Science, Tokyo Institute of Technology. Sebelumnya, penulis mendapatkan gelar sarjana di jurusan Teknik Informatika, Institut Teknologi Bandung. Penulis memiliki pengalaman menulis makalah ilmiah pada bidang pemrosesan bahasa alami (termasuk yang menggunakan teknik pembelajaran mesin) dan menerapkan teknik pembelajaran mesin untuk perusahaan IT. Penulis pernah mendapat *best paper award* untuk publikasi ilmiahnya, menjuarai *data mining hackathon*, serta memiliki prestasi akademik lainnya. Penulis masih terus mempelajari teknik *machine learning*, buku ini adalah catatan yang ingin ia bagikan.

`https://wiragotama.github.io/`