Jan Wira Gotama Putra

Pengenalan Konsep Pembelajaran Mesin

Dijelaskan dengan banyak contoh dan gambar

Edisi 0.9

October 15, 2017

Untuk Tuhan, Bangsa, dan Almamater

Preface

Diktat ini ditujukan sebagai bahan penunjuang (atau pengantar) mata kuliah *machine learning* untuk mahasiswa tingkat sarjana di Indonesia. Diktat ini hanya merupakan komplemen, bukan sumber informasi utama. Beberapa *reviewers* merasa materi diktat ini relatif cukup berat, karena itu ada baiknya membaca buku pengantar yang lebih "ringan" sebelum membaca diktat ini. Mudah-mudahan penulis dapat menyempurnakan gaya penyampaian pada diktat ini di versi-versi berikutnya (termasuk *citation* yang lebih tepat). Sebagian besar konsumsi mahasiswa adalah buku dengan bahasa asing/terjemahan. Terkadang, mahasiswa menganggap belajar menggunakan bahasa asing sebagai hal yang cukup sulit. Di lain pihak, *dengan segala hormat* buku terjemahan terkadang kurang pas karena maksud pengarang belum tentu bisa diterjemahkan secara sempurna ke bahasa lainnya. Untuk membantu mengatasi masalah tersebut, penulis menulis diktat ini dari nol.

Diktat ini adalah **ringkasan catatan kuliah penulis** yang mudah-mudahan dapat memberikan gambaran apa itu *machine learning*. Anggap saja membaca diktat ini seperti sedang membaca "*light novel*". Dengan membaca diktat [1] ini, diharapkan kawan-kawan juga mengetahui harus belajar apa (lebih jauhnya) dalam bidang *machine learning*.

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. Saya ingin mengajak saudara/i untuk memahami konsep *machine learning* secara utuh sebelum memanfaatkan.

Diktat ini menjelaskan algoritma *machine learning* dari sudut pandang "agak" matematis. Pembaca disarankan sudah memahami/mengambil seti-

[1] untuk selanjutnya, istilah *lecture note* mengacu pada "diktat" (digunakan secara bergantian)

daknya mata kuliah statistika, kalkulus, aljabar linier/geometri, pengenalan kecerdasan buatan, dan logika fuzzy. Saat membaca buku ini, disarankan membaca secara runtun. Gaya penulisan diktat ini **santai/semiformal**, dengan notasi matematis yang minimal, mudah-mudahan tanpa mengurangi esensi materi. Diktat ini lebih difokuskan pada konsep dan teori dasar, kami juga memberikan beberapa contoh sederhana agar pembaca mendapatkan bayangan seperti apa aplikasi sebuah konsep. Tentunya, penulis dapat menulis dengan cara penulisan deskriptif. Akan tetapi, penulis merasa banyak esensi yang hilang ketika materi *machine learning* hanya dijelaskan secara deskriptif.

Diktat ini dibuat menggunakan template monograph (LaTex) dari Springer. Formatting diatur secara otomatis oleh template. Dengan hal itu, mungkin terdapat beberapa kesalahan pemotongan kata di ujung kanan baris, contoh "menga-proksimasi" dapat ditulis sebagai "men-gaproksimasi" karena template memotong berdasarkan karakter.

Petunjuk Penggunaan
Struktur penyajian diktat 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 diktat ini. Penulis sangat menyarankan untuk membahas latihan soal sebagai tambahan materi (bisa juga sebagai PR), Satu sesi perkuliahan (90 sampai 120 menit) dapat menggunakan satu sampai dua bab materi pada diktat ini. Agar dapat memahami materi per bab, bacalah keseluruhan isi bab secara utuh sebelum mempertanyakan isi materi.

Pembaca dipersilahkan menyebar/mencetak diktat ini untuk alasan **NON KOMERSIAL**, tetapi dimohon untuk tidak menyalin/meniru isi diktat. 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). Diktat ini memuat soal latihan, tujuan latihan tersebut adalah untuk mengarahkan apa yang harus dibaca/dipahami lebih lanjut. Silahkan kontak penulis melalui surel untuk kolaborasi.

Kutipan
Penulis tidak menyarankan untuk mengutip diktat ini. Kutiplah *paper*/buku yang digunakan sebagai referensi pada diktat ini. Berikan kredit bagi mereka yang sepantasnya mendapatkan kredit atas ide mereka.

Ucapan Terima Kasih
Penulis ingin mengucapkan terima kasih pada Bapak/Ibu/Saudara/i:

1. Bapak I Gede Mahendra Darmawiguna. Karena diskusi bersama beliau, penulis memutuskan untuk menulis diktat ini. Terimakasih juga atas *review* pada versi awal yang memuat materi diktat paling utama.

2. Candy Olivia Mawalim. Terima kasih atas sumbangsih yang diberikan pada penulisan diktat ini baik untuk materi (bab 11.1) dan *review*-nya.
3. Hayyu Luthfi Hanifah. Terima kasih karena telah menjadi *reviewer* setia untuk setiap versi. Luar biasa untuk mata elang Hayyu!
4. Chairuni Aulia Nusapati. Terima kasih atas *review* dan berkenan mendengarkan keluh kesah penulis. Penulis bersyukur atas motivasi yang diberikan.
5. Arief Yudha Satria. Terima kasih karena mau membaca diktat ini dengan detail dan memberikan *review*. Moga-moga ada kesempatan kita bertemu lagi.
6. Tifani Warnita. Terima kasih atas *review* dan *stay tune*. Penulis bersyukur bahwa ternyata ada yang mau membaca diktat penulis sehingga penulisan diktat tetap dilanjutkan.
7. Yuni Susanti, Mochammad Dikra Prasetya + Jonas Stephen Mulyono duo kombi, group *holy land citizens*, dan group *Mr. Brown for keeping my "sanity" in check*.
8. Joshua Bezaleel Abednego dan Fabiola Maria. Terima kasih atas motivasi dan cerita inspiratifnya.
9. Tak lupa guru-guru dan dosen-dosen penulis yang tidak dapat disebutkan satu per satu. Karena atas kuliah beliau-beliau, penulis mampu menulis diktat ini.

Tokyo, Jepang

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

Contents

Part 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

Bab ini adalah bab paling penting pada *lecture note* ini karena memuat ide utama tentang *machine learning*.

1.1 Kecerdasan Buatan

Pada bagian pembukaan (*preface*) telah dijelaskan bahwa kami berharap kamu sudah mendapatkan 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 yang ramah bagi manusia. Kecerdasan buatan sesungguhnya adalah program komputer, yaitu bentuk matematis penuh dengan formula; kita sebut sebagai **agen**. Secara teori, program adalah ***automaton***; dimana kamu tahu *non-deterministic automata* dapat ditransformasi menjadi *deterministic automata*.

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*.

Permasalahan utama bidang kecerdasan buatan terdiri dari (dari klasik sampai lebih modern):

1. ***Planning***. Misal merencanakan rute perjalanan.

2. ***Constraint satisfaction problem***. Menyelesaikan permasalahan dengan kendala variabel-variabel tertentu. Analogi dengan *integer linear programming*.

3. ***Representasi pengetahuan***, yaitu melakukan inferensi dengan operasi logika berbentuk simbolik, misal logika preposisi, logika orde pertama (*first-order logic*), teori Fuzzy, *abductive reasoning*, ontologi, jaringan semantik (*semantic web*), dsb.

4. ***Machine learning***, yaitu melakukan inferensi dengan pendekatan matematis.

5. ***Multi-agent system***, misalnya *negotiation agent*.

6. Dan lain sebagainya.

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. Intinya, representasi pengetahuan memerlukan pekerjaan manual. 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* lebih intuitif untuk melakukan inferensi pada data yang besar. Hal ini yang menyebabkan *machine learning* menjadi populer.

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 tahu domain aplikasi 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, visual, dsb. Seiring kamu membaca diktat ini, kami harap kamu semakin mengerti hal ini.

1.2 Intelligent Agent

Sebuah agen cerdas (*intelligent agent*) didefinisikan memiliki empat macam dimensi sebagai berikut [1]:

1. ***Acting Humanly***. Pada dimensi ini, agen mampu bertingkah dan berinteraksi layaknya seperti manusia (baca: *turing test*).

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 memiliki rute lain yang memiliki pemandangan indah.

3. ***Thinking Humanly***. Pada dimensi ini, agen mampu berpikir seperti manusia dalam segi kognitif.

4. ***Thinking Rationally***. Pada dimensi ini, agen mampu berpikir secara rasional. Sederhananya sesuai dengan konsep logika matematika.

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 [2]. 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 hanya mempu mengoptimalkan agen yang mempunyai dimensi *acting rationally* dan *thinking rationally*.

Perhatikan Fig. 1.1! 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 *performance measure*. Misalnya untuk robot pembersih rumah, *performance measure*-nya adalah seberapa persen debu yang dapat ia bersihkan. *Performance measure*, secara matematis didefinisikan sebagai *utility function*, yaitu fungsi

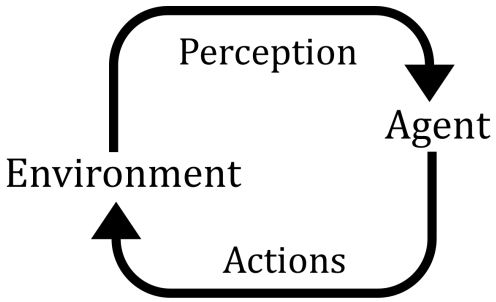

Fig. 1.1. Agent vs environment [3]

apa yang harus dimaksimalkan/diminimalkan oleh agen tersebut. Setiap tindakan yang dilakukan agen rasional, harus mengoptimalkan nilai *performance measure* atau *utility function*.

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 oleh pengalaman[1]. 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 (meningkatkan ***performance measure/ utility function***).

1.4 Konsep Statistical Machine Learning

Pada masa sekarang ini data bertebaran sangat banyak dimana-mana. Pemrosesan data secara manual tentu adalah hal yang kurang bijaksana. Beberapa pemrosesan yang dilakukan, misal kategorisasi (kategorisasi teks berita), peringkasan dokumen, ekstraksi informasi (mencari 5W+1H pada teks berita), rekomendasi produk berdasarkan catatan transaksi, dll [3]. Tujuan *machine learning* minimal ada dua: **memprediksi masa depan** (*unobserved event*);

[1] KBBI Web, Accessed on 10 October 2016

dan/ atau **memperoleh ilmu pengetahuan** (*knowledge discovery/ discovering unknown structure*). 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 [4, 5]. Disebut *statistical* karena basis pembelajarannya memanfaatkan data, juga menggunakan banyak teori statistik untuk melakukan inferensi (misal memprediksi *unobserved event*). Jadi, *statistical machine learning* adalah cara untuk memprediksi masa depan dan/atau menyimpulkan/mendapatkan pengetahuan dari data **secara rasional dan non-paranormal**. Hal ini sesuai dengan konsep *intelligent agent*, yaitu bertingkah berdasarkan lingkungan. Dalam hal ini, lingkungannya adalah data. *Performance measure*-nya adalah seberapa akurat prediksi agen tersebut, atau seberapa mirip "pola" data yang ditemukan terhadap data asli.

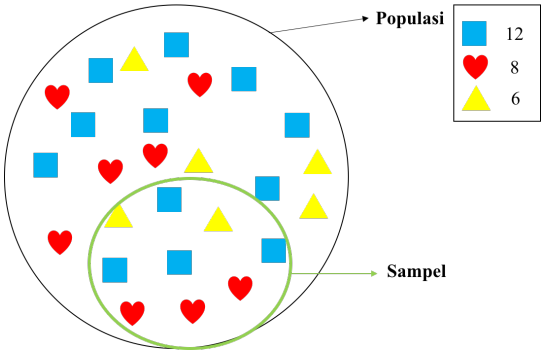

Fig. 1.2. Ilustrasi makanan pesta 1

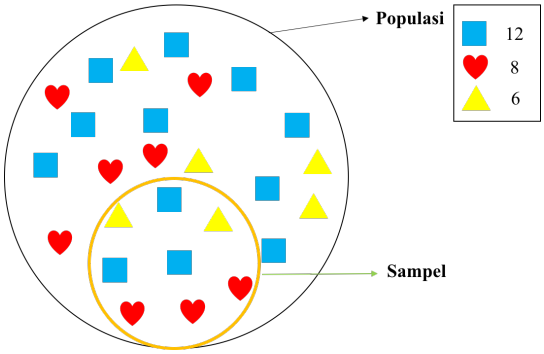

Fig. 1.3. Ilustrasi makanan pesta 2

Perhatikan Fig. 1.2 (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 masing-masing jenisnya (seluruh populasi). Tapi, 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 Fig. 1.3, 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 kedua contoh ini, kita menyimpulkan, menginferensi (*infer*) atau mengeneralisasi sampel. Kesimpulan yang kita buat berdasarkan sampel tersebut, kita anggap merefleksikan populasi, kemudian kita menganggap populasi memiliki aturan/pola seperti kesimpulan yang telah kita ciptakan [6]. Baik pada statistika maupun *statistical machine learning*, pemilihan 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 ***test data*** sebagai penyeimbang. Mesin dilatih menggunakan *training data*, kemudian diuji kinerjanya menggunakan *test data*. Seiring dengan membaca buku ini, konsep *training data* dan *test data* akan menjadi lebih jelas.

Seperti halnya contoh sederhana ini, persoalan *machine learning* sesungguhnya menyerupai persoalan *statistical inference* [6]. Kita berusaha mencari tahu populasi dengan cara menyelidiki fitur (*features* atau sifat-sifat) yang dimiliki sampel. Kemudian, menginferensi *unobserved data* berdasarkan kecocokan *features* dengan model/aturan yang sudah ada.

1.5 Supervised Learning

Jika diterjemahkan secara literal, *supervised learning* adalah pembelajaran terarah. Artinya, pada pembelajaran ini, ada guru yang mengajar (mengarahkan) dan siswa yang diajar. Kita disini berperan sebagai guru, kemudian mesin berperan sebagai siswa. Perhatikan Fig. 1.4 sebagai ilustrasi! Pada Fig. 1.4 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 *training data* (untuk kasus supervised learning).

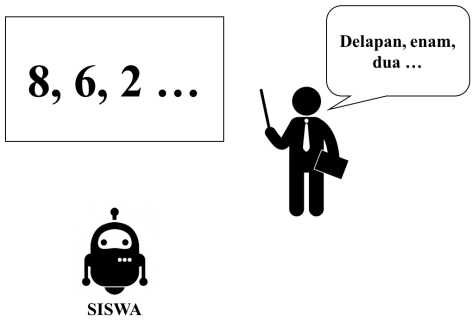

Fig. 1.4. Supervised Learning

Perhatikan Fig. 1.5 dan Fig. 1.6, x adalah *event* (*random variable*), untuk *event* tertentu dapat dinotasikan sebagai $\{x_1, x_2, x_3, ..., x_n\}$ Seorang guru sudah mempunyai jawaban yang benar untuk masing-masing contoh dengan suatu fungsi distribusi probabilitas kondisional (***conditional probability density function***) $q(y|x)$ 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* $p(y|x, \overrightarrow{\mathbf{w}})$[2], dimana y adalah target (*output*), x adalah input dan vektor $\overrightarrow{\mathbf{w}}$ adalah ***learning parameter***. Proses belajar ini, yaitu mengoptimalkan $\overrightarrow{\mathbf{w}}$ disebut sebagai training. Semakin kamu membaca *lecture note* ini, konsep ini akan menjadi semakin jelas.

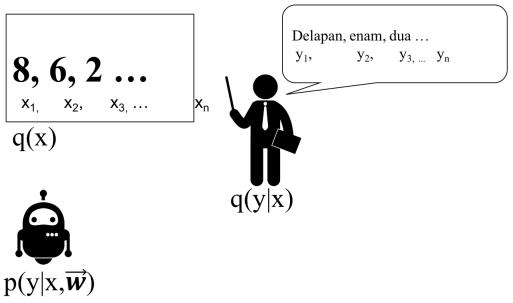

Fig. 1.5. Supervised learning - mathematical explanation

Perhatikan Fig. 1.7! $q(x)q(y|x) = q(x, y)$ memiliki panah ke *training data* dan *test data*, artinya model hasil *training* sangat bergantung pada **data dan**

[2] x_i dan y_i dapat diganti dengan vektor

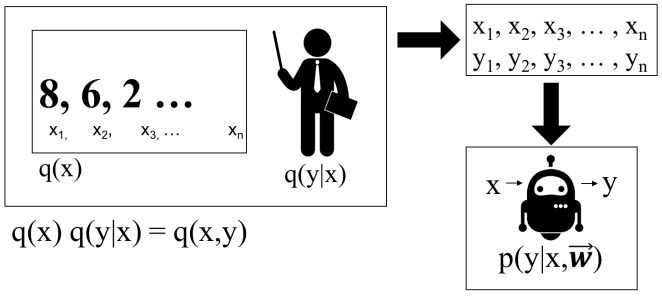

Fig. 1.6. Supervised learning - mathematical explanation 2

guru. Model yang dihasilkan *training* (hasil pembelajaran kemampuan siswa) untuk data yang sama bisa berbeda untuk guru yang berbeda.

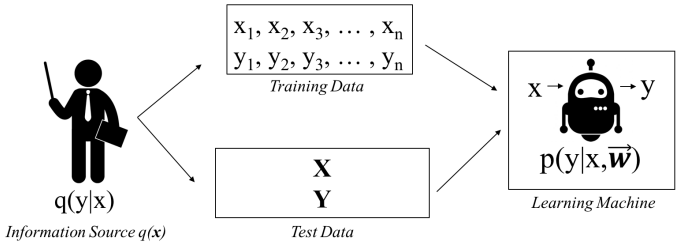

Fig. 1.7. Supervised learning framework

Selain *supervised learning*, masih ada metode pembelajaran lainnya yaitu *unsupervised learning*, *semi-supervised learning*, dan *reinforcement learning*. Tetapi *lecture note* ini berfokus pada pembahasan *supervised*, *semi-supervised*, dan *unsupervised learning*.

Tujuan *supervised learning*, secara umum untuk melakukan klasifikasi (*classification*). Misalkan mengklasifikasikan teks berita menjadi salah satu kategori olah raga, politik, nasional, regional, hiburan,atau teknologi. Apabila hanya ada dua kategori, disebut ***binary classification***. Sedangkan bila terdapat lebih dari dua kategori, disebut ***multi-label classification***. Ada tipe klasifikasi lain (*soft*), seperti pada fuzzy logic, misalkan suatu berita memuat 30% olah raga dan 70% politik. Diktat ini tidak akan terlalu berfokus pada *soft classification*.

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

Pemahaman *supervised learning* adalah mengingat persamaan 1.1. Ada tiga hal penting pada *supervised learning* yaitu *input*, *desired output*, dan *learning parameters*. 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.

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

1.6 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 dari sumber lainnya (misal dari hasil *clustering*). Kami akan memberi contoh *semi-supervised learning* berdasarkan ***autoencoder*** (bab 10).

1.7 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*. Misalnya kamu membuka sebuah toko serba ada, agar pelanggan lebih mudah belanja, kamu mengelompokkan barang-barang, tetapi definisi kelompoknya belum ada. Yang kamu lakukan adalah membuat kelompok-kelompok berdasarkan karakteristik barang-barang. Teknik-teknik mengelompokkan ini akan dibahas pada bab-bab berikutnya. Contoh algoritma *unsupervised learning* sederhana adalah *K-nearest-neighbor*.

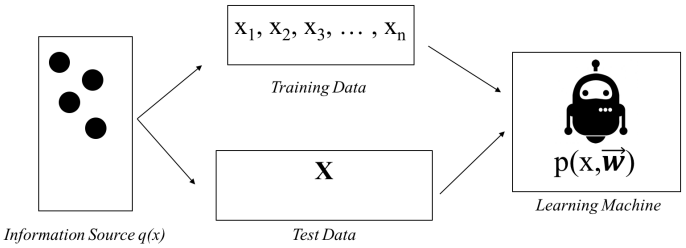

Fig. 1.8. Unsupervised learning framework

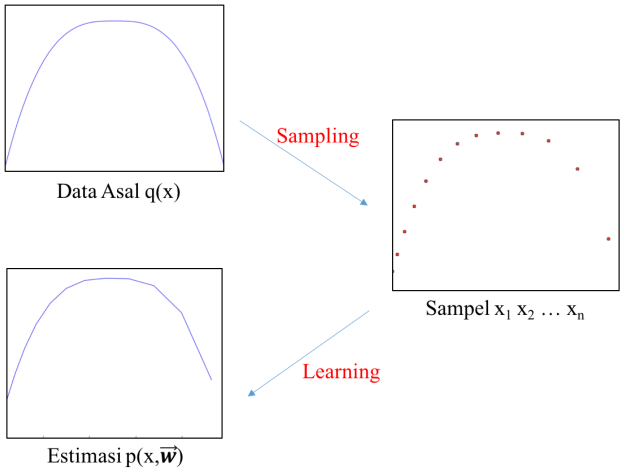

Fig. 1.9. Generalization error of unsupervised learning

Perhatikan Fig. 1.8 dan Fig. 1.9! 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 $p(x|\vec{\mathbf{w}})$ yang mengoptimasi parameter $\vec{\mathbf{w}}$. Perbedaan antara estimasi dan fungsi asli disebut sebagai ***generalization loss***.

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

Kunci pemahaman *unsupervised learning* adalah mengingat persamaan 1.3. Ada dua hal penting yaitu: *input* dan *learning parameters*.

1.8 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 $\vec{\mathbf{w}}$ (parameter belajar). 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 $\overrightarrow{\mathbf{w}}$ untuk mengoptimalkan suatu fungsi tujuan.

Secara bahasa lebih matematis, kami beri contoh *supervised learning*. Kita mempunyai distribusi klasifikasi asli $q(y|x)$. Dari distribusi tersebut, kita diberikan beberapa sampel pasangan *input-output* $\{z_1, z_2, z_3, \ldots, z_n\}; z = (x, y)$. Kita membuat learning machine $p(y|x, \overrightarrow{\mathbf{w}})$. Awalnya diberi x_1, *learning machine* mengestimasi fungsi asli dengan mengoptimalkan parameter $\overrightarrow{\mathbf{w}}$ sesuai dengan data yang ada. Seiring berjalannya waktu, ia diberikan data observasi lainnya, sehingga *learning machine* menyesuaikan dirinya terhadap observasi yang baru $(x_2, x_3, \ldots)$. Semakin lama, kita jadi makin percaya bahwa *learning machine* semakin optimal (mampu memprediksi fungsi aslinya).

1.9 Tipe Permasalahan di Dunia

Ada dua tipe permasalahan *(debatable)*, yaitu klasifikasi dan konstruksi. Permasalahan klasifikasi adalah mengategorikan sesuatu sesuai kelompok yang telah ditentukan sebelumnya. Contohnya klasifikasi buku di perpustakaan. Perpustakaan sudah menentukan kelompok-kelompok buku, misalnya teknik, sains, dan seni. Saat ada buku baru, perpustakaan nantinya menaruh buku pada tempat dengan kelompok bersesuaian. Contoh lainnya adalah klasifikasi makhluk hidup berdasarkan *kingdom*.

Jenis permasalahan kedua adalah konstruksi. Permasalahan konstruksi misalnya kita memiliki banyak buku di rumah, agar bukunya rapi kita ingin mengelompokkan buku-buku tersebut sesuai dengan kecocokan satu sama lain (*clustering*). Contoh lainnya adalah bagaimana cara menyusun kelompok/susunan/struktur *Kingdom* makhkluk hidup (ontologi). Menurut pendapat saya, regresi dapat dikategorikan sebagai permasalahan konstruksi, karena harus membangun (menebak) suatu fungsi yang mampu memprediksi output.

1.10 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 matematikanya (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

1.11 Contoh Aplikasi

Sebenarnya, aplikasi pemanfaatan *machine learning* sudah terasa dalam kehidupan sehari-hari. Contoh mudahnya adalah produk-produk Google, misalnya google translate (machine translation, handwritten recognition, speech recognition, Alpha Go). Berikut adalah beberapa artikel menarik:

1. https://techcrunch.com/2016/03/15/google-ai-beats-go-world-champion-again-to-complete-historic-4-1-series-victory/
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?
(d) Apa kelebihan dan kekurangan teknik *machine learning* daripada teknik lainnya?

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, Anda merasa bab ini tidak masuk akal/kurang dibutuhkan. Seiring membaca buku ini, mungkin bab ini akan sering dikunjungi kembali. Bab ini hanyalah pengantar saja, tentunya untuk mengerti probabilitas, sebaiknya Anda mengambil kuliah khusus tentang materi itu. Karena Anda diharapkan sudah memiliki "cukup latar pengetahuan", bab ini sebenarnya hanyalah sekilas pengingat. Kami akan banyak memakai contoh-contoh dari buku Bishop [4] untuk bab ini.

2.1 Probabilitas

Kita tahu bahwa banyak hal yang tidak pasti (*uncertain*), sebetulnya pada *machine learning*, kita juga berurusan dengan ketidakpastian (*uncertainty*). Dengan hal itu, *machine learning* memiliki kaitan yang sangat erat dengan statistika. Probabilitas menyediakan *framework* untuk kuantifikasi dan manipulasi ketidakpastian [4]. 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, dalam hal ini, kotak adalah *random variable*. *Random variable* b (melambangkan kotak) dapat bernilai *merah* atau *biru*. Begitu pula dengan buah, dilambangkan dengan variabel f, dapat bernilai *apel* atau *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 haruslah berada diantara $[0, 1]$.

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 $P(b = biru, f = apel)$ **atau** $P(b = merah, f = jeruk)$. Nilai probabilitas tersebut dapat dihitung dengan

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

- $P(b = biru, f = apel)$ disebut *joint probability*, yaitu probabilitas ketika suatu kejadian yang dipengaruhi beberapa variabel.
- $P(b = biru, f = apel) + P(b = merah, f = jeruk)$ disebut **aturan tambah**.

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

$$P(f = apel) = \sum_{k=1}^{K} P(b = b_k, f = apel) \tag{2.2}$$

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

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 x dan y independen (tidak bergantung satu sama lain), maka $P(x = X, y = Y) = P(X) * P(Y)$. Permasalahan mengambil buah dapat dihitung dengan

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

Aturan ini disebut aturan kali. Untuk *joint probability*, secara umum dapat ditulis sebagai $P(x = X, y = Y)$. Sebaliknya, apabila X tidak saling bebas Y, 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 $P(x|y)$. Artinya probabilitas x yang dikondisikan oleh y. Apabila x ternyata tidak dikondisikan oleh variabel y, maka $P(x|y) = P(x)$. Contoh kasus ini adalah gempa bumi, tidak dikondisikan oleh kegiatan menabung. $P(x|y)$ dapat dihitung dengan

$$P(x|y) = P(x, y)/P(y) \tag{2.4}$$

yaitu peluang kejadian x dan y muncul bersamaan dibagi dengan peluang kejadian y.

2.2 Probability Density Function

Kali ini tentang pelajaran di sekolah. Terdapat ujian mata pelajaran di kelas yang beranggotakan N siswa. Guru ingin mengetahui persebaran (distribusi) nilai ujian untuk menentukan batas kelas nilai (misal nilai "A" adalah ≥ 85), jadi ia membuat grafik nilai ujian untuk tiap-tiap siswa. Sebut saja variabel nilai siswa adalah x. Sumbu horizontal menandakan nomor urut siswa.

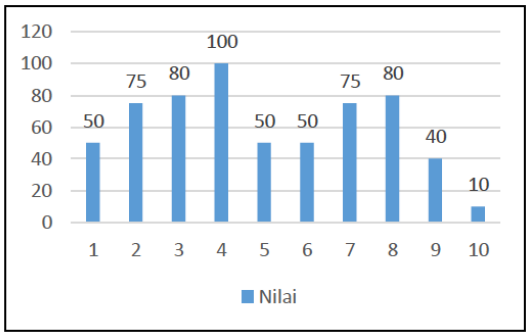

Fig. 2.1. Nilai siswa

Perhatikan Fig. 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. Persebaran probabilitas nilai dapat dilihat pada Fig. 2.2.

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. Nilai dibawah kurva pada interval $-\infty$ sampai ∞ adalah 1.

2.3 Expectation dan Variance

Salah satu operasi paling penting dalam probabilitas adalah menemukan nilai rata-rata terbobot (*weighted average*) sebuah fungsi [4]. Hal ini disebut

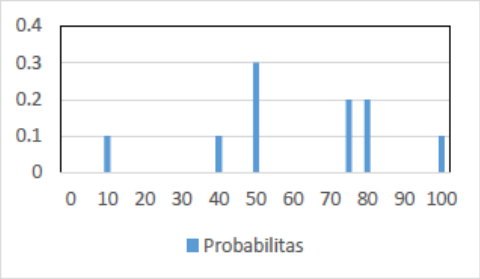

Fig. 2.2. Persebaran probabilitas nilai siswa

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

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

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.

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

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 Fig. 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 menguan-

tifikasi ketidakpastian. 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 [4]. Dalam Bayesian, nilai numerik digunakan untuk merepresentasikan derajat kepercayaan/ketidakpastian.

$$P(a|B) = \frac{P(B|a)P(a)}{P(B)} \tag{2.8}$$

$P(a)$ disebut *prior*, yaitu pengetahuan/asumsi awal kita. Setelah kita mengobservasi fakta baru B, kita mengubah asumsi kita. $P(B|a)$ disebut ***likelihood function***. *Likelihood function* mendeskripsikan peluang data, untuk asumsi/pengetahuan tentang a yang berubah-ubah (a 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*).

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

Pada umumnya, untuk mengestimasi *likelihood*, digunakan *maximum likelihood estimator*; yang berarti mengatur nilai a untuk memaksimalkan nilai $P(B|a)$. Dalam literatur *machine learning*, banyak menggunakan *negative log of likelihood function* [4]; karena nilai logaritma negatif, secara monotonik menurun, maka memaksimalkan nilai *likelihood* ekuivalen dengan meminimalkan negatifnya (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. 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 $P(X|Y)$ 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

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** variabel, misalkan x. Distribusi sebenarnya adalah fenomena random atau deskripsi matematis suatu *random variable*.

Distribusi univariate Gaussian dikarakteristikkan oleh *mean* (μ) dan *variance* (σ^2) diberikan pada persamaan 2.10

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

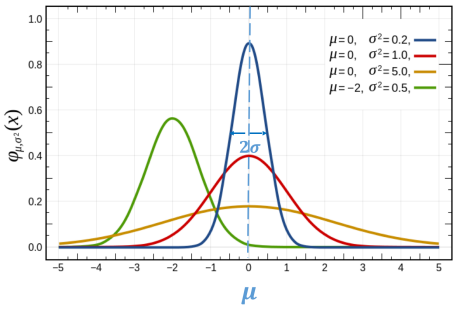

Fig. 2.3. Univariate Gaussian [1]

Perhatikan Fig. 2.3, nilai N (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*. Kita tahu bahwa data yang independen, apabila dihitung probabilitasnya maka tersusun atas probabilitas masing-masing data. Hal ini diberikan pada persamaan 2.11.

[1] source: wikimedia.org

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

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.

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

solusi 2.12 diberikan pada 2.13.

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

Perhatikan baik-baik interpretasi berikut! Artinya 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 [7]. Ilustrasinya diberikan pada Fig. 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 $p(y|x, \overrightarrow{\mathbf{w}})$. Pada bab

[2] http://dirichletprocess.weebly.com/clustering.html

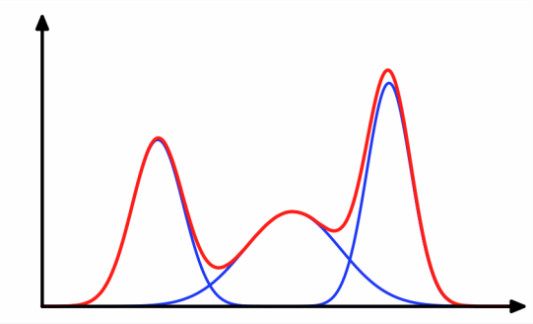

Fig. 2.4. Gaussian Mixture Model [2]

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 adalah *decision theory* [4].

Dalam *machine learning* kita dapat membangun model untuk dua tujuan: meminimalkan *error* atau meminimalkan *loss*; konsep meminimalkan *error* dijelaskan pada materi *curve fitting*. 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 [4]. Misalkan kita diminta untuk membuat model klasifikasi kanker. Kita dapat mengklasifikasikan pasien menjadi dua kelas $\{kanker, normal\}$.

Apabila kita ingin meminimalkan *error*, 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*. 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*.

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 $p(x)$. Maka *generalization loss* dan *training loss* dihitung dengan persamaan 2.14 dan persamaan 2.15.

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

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

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*).

2.7 Teori Informasi

Kami tidak akan membahas bagian ini terlalu detail, jika kamu membaca buku, topik ini sendiri bisa mencapai satu buku [8]. 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 transmisi komunikasi terbaik (jawab: channel capacity) [8]. 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 algorithma 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*" [4]. 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 [4]. 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 [8]. 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. Dari sisi *information transmission*, dapat diinterpretasikan kita dapat mengirimkan data sebuah distribusi dengan jumlah bit lebih sedikit. Distribusi yang memberikan nilai *entropy* maksimal adalah distribusi Gaussian [4]. 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 Fig. 2.5.

Tentunya kita ingin tahu seberapa bagus aproksimasi kita, untuk mengukurnya terdapat sebuah perhitungan yang bernama **Kullback-Leibler**

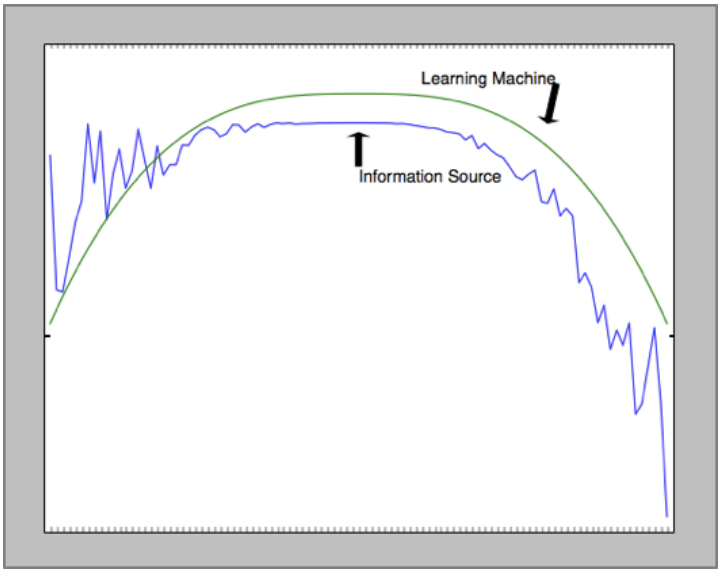

Fig. 2.5. Information source vs learning machine

Divergence (KL-divergence). Secara konseptual, dirumuskan sebagai persamaan 2.18.

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

Persamaan tersebut 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 dapat mengaproksimasi KL-divergence. Misalkan kita diberikan *training data* $\{x_1, x_2, ...x_n\}$ yang kita asumsikan diambil dari $q(x)$. Lalu kita membuat *learning machine* $p(x|\vec{\mathbf{w}})$. Ekspektasi terhadap $q(x)$ dapat diaproksimasi dengan menggunakan data sampel ini, sehingga menjadi persamaan 2.19 [4].

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

KL-divergence disebut juga sebagai *relative entropy*. 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. Konsep *mutual information* dipandang sebagai pengurangan ketidakyakinan terhadap *posterior*, seiring diberikannya data observasi yang baru. Dengan kata lain,

seiring diberikan observasi yang baru, kita semakin yakin terhadap nilai posterior.

2.8 Bacaan Lanjutan

Untuk lebih mengerti, silahkan membaca [9, 10, 6].

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

3

Utility Function

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

Bab ini akan membahas tentang *curve fitting problem* (regresi), sebagai contoh pembelajaran yang sederhana karena cukup mudah dipahami idenya. Bab ini juga membahas *error function* dan *steepest gradient descent*.

3.1 Curve Fitting dan Error Function

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), *which is subject to a simultaneous 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 3.1 [5].

$$g(x) = \int y \; q(y|x)dy \tag{3.1}$$

persamaan 3.1 dibaca sebagai "expectation of y, with the distribution of q". Secara statistik, regresi dapat disebut sebagai expectation untuk y berdasarkan/ dengan *input* x. Regresi adalah pendekatan belum tentu 100% tepat sasaran.

Sebagai ilustrasi *curve fitting problem*, kamu diberikan fungsi $f(x)$ seperti pada Fig. 3.1. 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 $f(x)$!

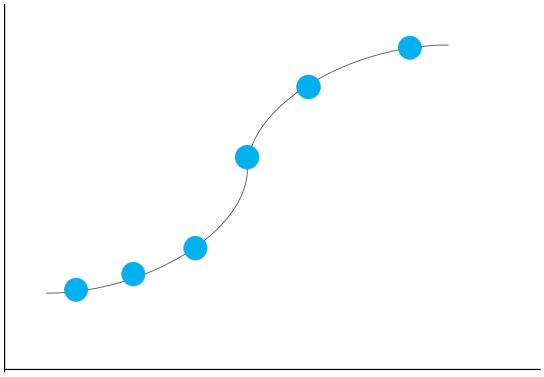

Fig. 3.1. Contoh fungsi Sigmoid

Anggap dengan metode regresi, kamu berhasil melakukan pendekatan dan menghasilkan fungsi seperti Fig. 3.2 (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*.

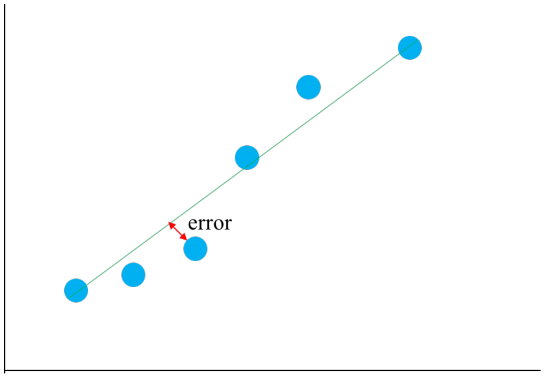

Fig. 3.2. Pendekatan fungsi sigmoid

Salah satu cara menghitung *error* fungsi $g(x)$ adalah menggunakan ***squared error function*** dengan bentuk konseptual pada persamaan 3.2. Estimasi terhadap persamaan tersebut disajikan dalam bentuk diskrit pada persamaan 3.3. x_i, y_i adalah *training data*. Nilai *squared error* dapat menjadi tolak ukur untuk membandingkan kinerja suatu *learning machine*. Secara umum, bila nilainya tinggi, maka kinerja dianggap relatif buruk; sebaliknya bila rendah, kinerja dianggap relatif baik. Hal ini sesuai dengan konsep *intelligent agent* [1].

$$E(g) = \int \int |y - g(x)|^2 \, q(x, y) \, dx \, dy \tag{3.2}$$

$$E(\overrightarrow{\mathbf{w}}) = \sum_{i=1}^{N} \left\| y_i - g(x_i, \overrightarrow{\mathbf{w}}) \right\|^2 \tag{3.3}$$

Ingat kembali bab 1, *learning machine* yang direpresentasikan dengan fungsi g bisa diatur kinerjanya dengan parameter training $\overrightarrow{\mathbf{w}}$. *Square error* untuk *learning machine* dengan parameter training $\overrightarrow{\mathbf{w}}$ diberikan oleh persamaan 3.3. (x_i, y_i) adalah pasangan *input-desired output*. Selain untuk menghitung ***square error*** pada *training data*, persamaan 3.3 juga dapat digunakan untuk menghitung *square error* pada testing data. Tujuan dari regresi/*machine learning* secara umum adalah untuk meminimalkan nilai *error function*. Dalam hal ini *utility function* adalah *error function*. Akan tetapi, fenomena ***overfitting*** dapat terjadi apabila nilai *error function* saat *training* kecil, tetapi besar saat *testing*. Artinya, *learning machine* terlalu menyesuaikan diri terhadap *training data*. Untuk menghindari *overfitting*, kadang ditambahkan fungsi ***noise/bias*** (selanjutnya disebut *noise/bias* saja).

3.2 Steepest Gradient Descent

Salah satu tujuan dari pembelajaran adalah untuk meminimalkan *error*, sehingga kinerja *learning machine* diukur oleh *square error*. Dengan kata lain, *utility function* adalah **meminimalkan *square error***. Secara matematis, yang kita lakukan adalah mengestimasi *minimum square error*, dengan mencari nilai *learning parameter* $\overrightarrow{\mathbf{w}}$ yang meminimalkan nilai *error*. Terdapat beberapa cara untuk memimalkan *square error* seperti *steepest gradient descent*, *stochastic gradient descent*, dsb. Pada *lecture note* ini, hanya *steepest gradient descent* yang dibahas.

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 [5]. Sebagai ilustrasi, perhatikan Fig. 3.3!

Jalanan dengan kemiringan paling tajam adalah $-grad\ E(\overrightarrow{\mathbf{w}})$. Dengan definisi $grad\ E(\overrightarrow{\mathbf{w}})$ diberikan pada persamaan 3.4 dan persamaan 3.5.

$$grad\ E(\overrightarrow{\mathbf{w}}) = \left(\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, ..., \frac{\partial E}{\partial w_N} \right) \tag{3.4}$$

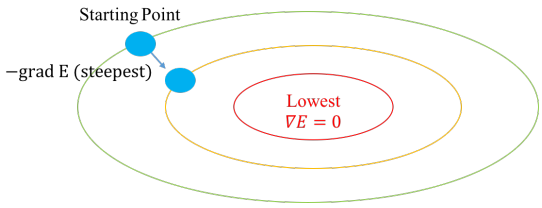

Fig. 3.3. Steepest Gradient Descent

$$\frac{d\overrightarrow{\mathbf{w}}}{dt} = -grad\ E(\overrightarrow{\mathbf{w}}); t = time \tag{3.5}$$

Ingat kembali materi diferensial. Gradien adalah turunan (diferensial) fungsi. Untuk mencari turunan paling terjal, sama halnya mencari nilai −gradient terbesar. Dengan demikian, menghitung $-grad\ E(\overrightarrow{\mathbf{w}})$ terbesar sama dengan jalanan turun paling terjal.

Tentunya seiring berjalannya waktu, kita mengubah-ubah parameter $\overrightarrow{\mathbf{w}}$ agar kinerja model optimal. Nilai optimal diberikan oleh turunan $\overrightarrow{\mathbf{w}}$ terhadap waktu, yang bernilai sama dengan $-gradE(\overrightarrow{\mathbf{w}})$. Bentuk diskrit persamaan 3.5 diberikan pada persamaan 3.6.

$$\overrightarrow{\mathbf{w}}(t+1) - \overrightarrow{\mathbf{w}}(t) = -\eta\ grad\ E(\overrightarrow{\mathbf{w}}(t)) \tag{3.6}$$

η disebut ***learning rate***. *Learning rate* digunakan untuk mengatur seberapa pengaruh keterjalan terhadap pembelajaran. Silahkan mencari sumber tambahan lagi agar dapat mengerti *learning rate* secara lebih dalam.

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 Fig. 3.4.

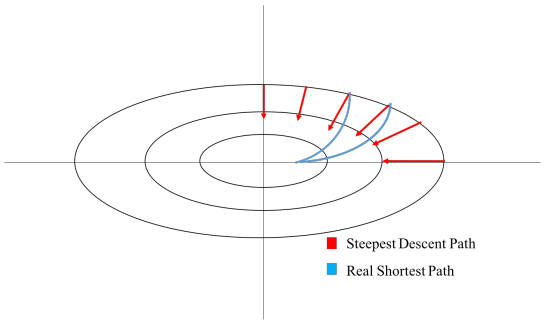

Fig. 3.4. Steepest Gradient Descent 2

Pandangan kita yang terbatas layaknya kita tidak bisa melihat keseluruhan pengunungan secara keseluruhan, kita juga tidak bisa melihat keseluruhan nilai *error* untuk semua parameter $\overrightarrow{\mathbf{w}}$. 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 3.6 relatif kecil, maka dinamika perubahan parameter $\overrightarrow{\mathbf{w}}$ juga kecil. Tetapi, bila nilainya besar, maka jalanan menuju titik terendah akan bergoyang-goyang (*swing*), seperti pada Fig. 3.5.

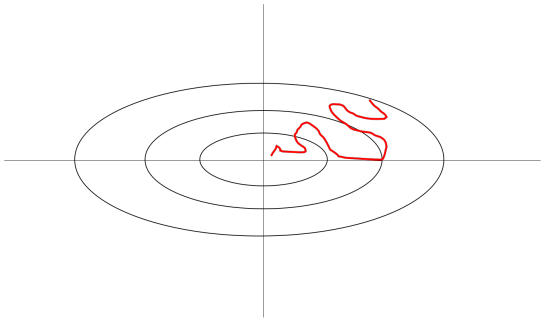

Fig. 3.5. Swing

Untuk kontrol tambahan proses mengestimasi *learning parameter* $\overrightarrow{\mathbf{w}}$ sehingga memberikan nilai $E(\overrightarrow{\mathbf{w}})$ terendah, persamaan *steepest gradient descent* dapat ditambahkan dengan momentum (α) pada persamaan 3.7. Alfa adalah momentum karena dikalikan dengan hasil descent pada tahap sebelumnya. Alfa adalah parameter kontrol tambahan untuk mengendalikan *swing* yang sudah dibahas sebelumnya.

$$\overrightarrow{\mathbf{w}}(t+1) - \overrightarrow{\mathbf{w}}(t) = -\eta \; gradE(\overrightarrow{\mathbf{w}}(t)) + \alpha(\overrightarrow{\mathbf{w}}(t+1) - \overrightarrow{\mathbf{w}}(t)) \tag{3.7}$$

3.3 Bacaan Lanjutan

Karena penggunaan *learning rate* dan momentum paling kentara di *Neural Network*, silahkan baca artikel menarik berikut untuk tambahan lanjutan.

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

Soal Latihan

3.1. Hill Climbing
Baca dan jelaskanlah konsep Hill Climbing!

3.2. Global Minimal/Maximal
Selain *learning rate*, sebutkan metode lain untuk menghindari *learning machine* buntu pada *local minimal/maximal*! (agar bisa menuju *global minimal/maximal*). Kata kunci: *regularization*.

3.3. Gradient Descent
Baca dan jelaskanlah variasi konsep *gradient descent* selain *steepest gradient descent*, misalnya *stochastic gradient descent*!

4

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.

4.1 Pengenalan Data Analytics

Secara umum, subbab ini adalah ringkasan dari [11]. Untuk detilnya, kamu dapat membaca buku tersebut secara langsung. Kami merekomendasikan buku tersebut karena ringkas dan mudah dipahami bahkan oleh pemula.

Kita tahu di dunia ini ada banyak masalah. Masalah adalah keadaan ketidaktercapaian tujuan yang diinginkan (*current state* bukanlah *desired state*). Penyelesaian masalah (*problem solving*) adalah kegiatan menjembatani kedua hal ini, agar keadaan saat ini bisa berubah menjadi keadaan yang diinginkan. Definisi permasalahan berbeda-beda tergantung domain atau bidang. 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, dsb. Strategi suatu bidang untuk merepresentasikan masalah menjadi bentuk komputasi berbeda dengan bidang lain, hal ini yang perlu kamu ingat karena representasi hanyalah simbol. Interpretasi simbol sangat bergantung pada konteks pengetahuan.

Raw data atau data mentah adalah sekumpulan (*record*) fakta 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). Agar mampu menganalisisnya, pertama-tama kita harus merapikan data sesuai dengan format yang kita inginkan. Setelah itu, kita menggunakan teknik-teknik yang kita sudah pelajari 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 *data mining* (menggunakan *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 tahap analisis selesai dilakukan, 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. 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*) untuk mengatakan bahwa memang metode kamu lebih baik, dengan peluang kejadian motode 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?

Secara umum, tahapan-tahapan yang dijelaskan sebelumnya adalah pekerjaan rutin *data scientist*. Kami ini tekankan sekali lagi, bahwa memahami *machine learning* saja tidak cukup, **kamu harus memahami domain permasalahan** agar mampu melakukan analisis dengan tepat. Terdapat banyak yang hanya mampu kamu pahami dari data, apabila kamu mengerti domain aplikasi.

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

Table 4.1. Contoh dataset *play tennis* (UCI machine learning repository)

4.2 Nilai Atribut dan Transformasi

Perhatikan Table. 4.1 yang merupakan contoh dataset pada *machine learning*. **Dataset** adalah kumpulan data. Seorang anak ingin bermain tenis, tetapi keputusannya untuk bermain tenis tergantung pada empat variabel {*outlook, temperature, humidity, windy*}. Keempat variabel ini disebut **fitur**. Setiap fitur memiliki **atribut** nilai dengan **tipe data** dan **range** tertentu.

Dari segi data statistik, terdapat beberapa tipe atribut [12]:

1. **Nominal**. Nilai atribut bertipe nominal adalah simbol-simbol yang berbeda, yaitu suatu himpunan terbatas. Sebagai contoh, fitur *outlook* pada Table. 4.1 memiliki tipe data **nominal** yaitu nilainya tersusun oleh himpunan {*sunny, overcast, rainy*}. Pada tipe nominal, tidak ada urutan ataupun jarak antaratribut. Tipe ini sering juga disebut **kategorial** atau **enumerasi**.

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

$1 - 20$ diberi nama $intv1$.

4. **Ratio**. Tipe *ratio* (rasio) didefinisikan sebagai perbandingan antara suatu nilai dengan nilai lainnya, misalkan massa jenis. 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***. Untuk setiap baris pada Table. 4.1, baris kumpulan nilai variabel non-kelas disebut **vektor fitur/*feature vector***. Contohnya pada Table. 4.1 $id = 4$, *feature vector*-nya adalah {*outlook=rainy, temperature=mild, humidity=high, windy=false*}.

4.3 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*. Kemungkinan kombinasi nilai ini disebut sebagai ruang konsep (***concept space***). Sebagai contoh pada Table. 4.1 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, bila menggunakan atribut bertipe numerik.

Algoritma lain yang bernama *candidate-elemination algorithm* 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 [13].

Selain *inductive learning*, kita juga dapat melakukan *deductive learning* yaitu melakukan inferensi dari hal general menjadi lebih spesifik. *Machine learning* adalah solusi untuk melakukan induksi/deduksi data dengan secara cepat, tepat, dan berkualitas.

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

Table 4.2. Contoh dataset *linearly separable*

4.4 Linear Separability

Perhatikan Table. 4.2. Data pada tabel tersebut kita sebut ***linearly separable***. Untuk suatu nilai variabel tertentu, ia hanya berkorespondensi dengan kelas tertentu. Ambil contoh pada Table. 4.2, saat *humidity=high* maka *swim=yes*. Semantara itu pada Table. 4.1, ketika *humidity=high* bisa jadi *play=yes* atau *play=no*. Kasus kedua disebut ***non linearly separable***. 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*) atau bidang datar. Ilustrasi dapat dilihat pada Fig. 4.1.

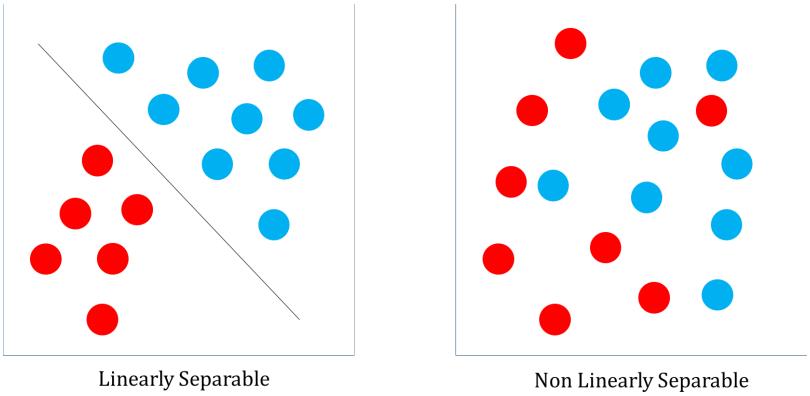

Fig. 4.1. Linearly vs Non Linearly Separable

Untuk memisahkan data *non-linearly separable*, kita pertama-tama mentransformasikan data menjadi *linearly-separable*. Kita dapat menggunakan teknik transformasi data menggunakan *kernel function* seperti *radial basis function*. Silakan baca teknik transformasi lebih lanjut pada literatur lain. Hal ini penting dipahami karena data yang bersifat *linearly separable* mudah dipisahkan satu sama lain sehingga mudah untuk melakukan *classification* atau *clustering*. Dengan demikian, wajar ada algoritma untuk melakukan transformasi data.

4.5 Seleksi Fitur

Pada permasalahan praktis, kita menggunakan banyak fitur. Kita ingin menyederhanakan fitur-fitur yang digunakan, misalkan dengan memilih subset fitur awal, atas dasar empat alasan[1]:

1. Menyederhanakan data/model agar lebih mudah dianlisis.
2. Mengurangi waktu *training* (mengurangi kompleksitas).
3. Menghindari *curse of dimensionality*. Hal ini dijelaskan lebih lanjut pada bab 10.
4. Meningkatkan generalisasi dengan mengurangi *overfitting*, secara formal dengan mengurangi *variance*.

Salah satu contoh 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[2].

4.6 Classification, Association, Clustering

Pada *supervsied learning*, kita memprediksi kelas berdasarkan *feature vector* yang merepresentasikan suatu instans. *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***). Sementara itu, apabila kita ingin mencari hubungan antara satu atribut dan atribut lainnya, disebut **association**. Sebagai contoh pada Table. 4.1, 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.*** Perlu kamu catat bahwa *unsupervised learning* $\neq$ *clustering. Clustering* adalah salah satu *task* pada *unsupervised learning.*

Pada Table. 5.1, 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 animali*a, lalu lebih spesifiknya ke *phylum Vertebrata*, dst. *Multi-class/multi-label classification* hanya proses klasifikasi ke dalam banyak "kelas" tanpa tinjauan hirarkis.

[1] https://en.wikipedia.org/wiki/Feature_selection
[2] http://scikit-learn.org/stable/modules/feature_selection.html

4.7 Mengukur Kinerja

Kamu sudah tahu salah satu contoh *utility function* yaitu *squared error function* pada bab 3. Selain *error function*, kamu juga dapat membandingkan kinerja dengan menggunakan fungsi lainnya seperti akurasi, presisi, *recall*, F1-measure, BLEU, ROUGE, *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). Tiap-tiap *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 4.1

$$\text{akurasi} = \frac{\#\text{instans diklasifikasikan dengan benar}}{\text{banyaknya instans}} \tag{4.1}$$

4.8 Metode Evaluasi

Pada umumnya, kita memiliki *training*, *validation*, dan *testing data*. Mesin dilatih menggunakan *training data*, saat proses *training*, *performance measure* diukur berdasarkan kemampuan mengenali/mengeneralisasi *validation data*. Perlu diketahui, *performance measure* dikur menggunakan *validation data* untuk menghindari *overfitting*. 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. Untuk mengetahui bagaimana membentuk komposisi *training*, *validation*, dan *testing data*; kamu dapat membaca materi ***cross-validation***[3]

4.9 Tahapan Analisis

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

1. Memutuskan tujuan analisis data (*defining goal*)

[3] https://en.wikipedia.org/wiki/Cross-validation_(statistics)

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

Soal Latihan

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

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

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

Part II

Algoritma Pembelajaran Mesin

5

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, kami akan memberi contoh algoritma yang lebih mudah yaitu **Naive Bayes**, **K-means**, dan **K-nearest-neighbor**. Bab ini akan memuat contoh sederhana *supervised* dan *unsupervised learning*. Sebelum masuk ke materi algoritma, kami akan membahas tentang data terlebih dahulu. Mudah-mudahan bab ini memberikan kamu gambaran aplikasi *machine learning* sederhana.

5.1 Naive Bayes

Naive Bayes adalah algoritma *supervised learning* yang sangat sederhana. Idenya mirip dengan probabilitas bayesian pada bab 2. Secara formal, persamaan *Naive Bayes* untuk klasifikasi diberikan pada persamaan 5.1 dimana c_i adalah suatu nilai kelas, C adalah kelas (*variabel*), t adalah fitur dan F adalah banyaknya fitur. Kita memprediksi kelas berdasarkan probabilitas kemunculan nilai fitur pada kelas tersebut. Hal ini didefinisikan pada persamaan 5.1 dan 5.2.

$$likelihood(c_i) = \arg\max_{c_i \epsilon C} P(c_i) \prod_{f=1}^{F} P(t_f|c_i) \tag{5.1}$$

$$P(c_i) = \frac{likelihood(c_i)}{\sum_{c_j \epsilon C} likelihood(c_j)} \tag{5.2}$$

Mari kita bangun model Naive Bayes untuk Table. 5.1. Dengan hal ini, kita sebut Table. 5.1 sebagai ***training data***. Untuk menghitung probabilitas, pertama-tama kita hitung terlebih dahulu frekuensi nilai atribut seperti

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

Table 5.1. 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						

Table 5.2. 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						

Table 5.3. Probabilitas setiap nilai atribut

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

Table 5.4. Contoh testing data *play tennis* [3]

pada Table. 5.2, setelah itu kita bangun model probabilitasnya seperti pada Table. 5.3.

Untuk menguji kebenaran model yang telah kita bangun, kita menggunakan ***testing data***, diberikan pada Table. 5.4. *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. Menghitung probabilitas penting pada kasus *likelihood* dengan nilai yang tidak berbeda jauh. Misalkan *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.

5.2 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**, kita harus mengikuti langkah-langkah 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

Table 5.5. 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

Table 5.6. Assignment K-means Langkah 1

1. Tentukan jumlah kelompok yang kita inginkan.
2. Inisiasi **centroid** untuk setiap kelompok (pada bab ini, secara acak).
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 Table. 5.5, 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. 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, centroid untuk kelompok pertama adalah id_2 sementara untuk kelompok kedua adalah id_4. Kita hitung kembali *assignment* anggota kelompok yang ilustrasinya dapat dilihat pada Table. 5.7. 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	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

Table 5.7. Assignment K-Means Langkah 2

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 Fig. 5.1! Suatu *cluster* atau kelas, seolah olah "dibungkus" oleh suatu distribusi gaussian. Distribusi seluruh dataset dapat diaproksimasi dengan *gaussian mixture model*.

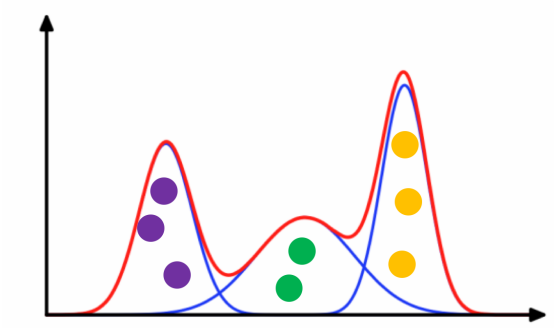

Fig. 5.1. Ilustrasi Hubungan Clustering, Kelas, dan Gaussian

Ingat kembali bahwa data memiliki suatu pola (dalam statistik disebut distribusi), kemudian pada bab 2 telah disebutkan bahwa gaussian mixture model 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.

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

Table 5.8. Perbedaan data baru vs data orang kaya

5.3 K-nearest-neighbor

Ide **K-nearest-neighbor** (KNN) adalah mengelompokkan data ke kelompok yang memiliki sifat termirip dengannya. Hal ini sangat mirip dengan **K-means**. Bila K-means digunakan untuk *clustering*, KNN digunakan untuk klasifikasi. Algoritma klasifikasi ini disebut juga algoritma malas. Pada subbab 5.2, 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 Table. 5.8. *feature vector* termirip dimiliki oleh data dengan id_1, id_5, id_6 seperti diilustrasikan pada Table. 5.8. Kita dapat menggunakan strategi untuk mengurusi permasalahan ini, misalnya memberikan prioritas memasukkan ke kelompok yang anggotanya lebih banyak menjadi *nearest neighbor*. Dengan strategi tersebut, kita mengklasifikasikan data baru ke kelompok pertama.

Soal Latihan

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

5.2. K-means, KNN, GMM, EM
Buktikan bahwa K-means, K-nearest-neighbor, *Gaussian Mixture Model*, dan Expectation Maximization Algorithm memiliki hubungan! (apa kesamaan mereka).

5.3. 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*.

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 [14, 15] yang terinspirasi oleh teori informasi [16]. Paper-paper tersebut sudah cukup lama, 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 penyakit A. Pada jaman dahulu, orang membuat agen cerdas berdasarkan program yang berisi transkripsi aturan-aturan eksplisit untuk membuat keputusan, berdasarkan pengetahuan dari ahli. 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 4.3 yaitu mendaftarkan seluruh kemungkinan aturan yang ada, kemudian menghapus yang kurang cocok. Algoritma lebih baik adalah dengan membangun *decision tree*.

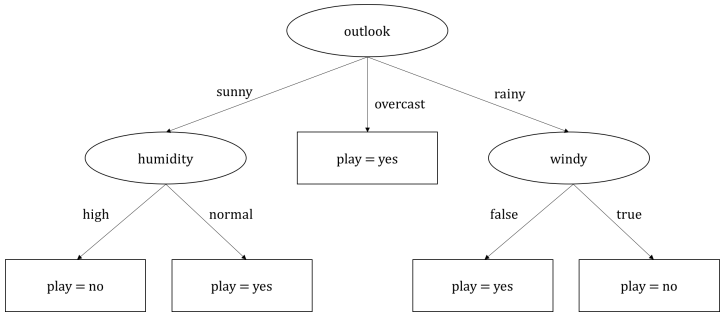

Fig. 6.1. Final Decision Tree

6.2 ID3

Seperti yang dijelaskan pada subbab sebelumnya, *decision tree* adalah varian dari *inductive learning*. ID3 adalah salah satu algoritma varian *decision tree* [15]. *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* [15]. 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.

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

Table 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 pada percabangan di cabang tertentu.

Sebagai contoh, perhatikanlah Fig. 6.1 yang merupakan hasil ID3 untuk Table. 6.1. Bentuk elips merepresentasikan nama atribut, sementara *edge* (panah) merepresentasikan nilai atribut. Bentuk segi empat merepresentasikan klasifikasi kelas (*leaf*). Pohon keputusan pada Fig. 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 [3].

Sebelum masuk ke perumusan *information gain*, kami akan memperkenalkan **entropy** terlebih dahulu. Entropy 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** sebai 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. *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)

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

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.

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

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

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

  Perhatikan $log\ 0$ pada umumnya tidak terdefinisi, tapi kita anggap $0\ log\ 0$ sebagai 0.
- $outlook = rainy$
  Ada tiga instans dengan $play = yes$ dan dua instans dengan $play = no$ saat $outlook = rainy$, dengan demikian kita hitung *Info*-nya.

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

Kita hitung *informaiton gain* untuk atribut *outlook* sebagai

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

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 Fig. 6.2.

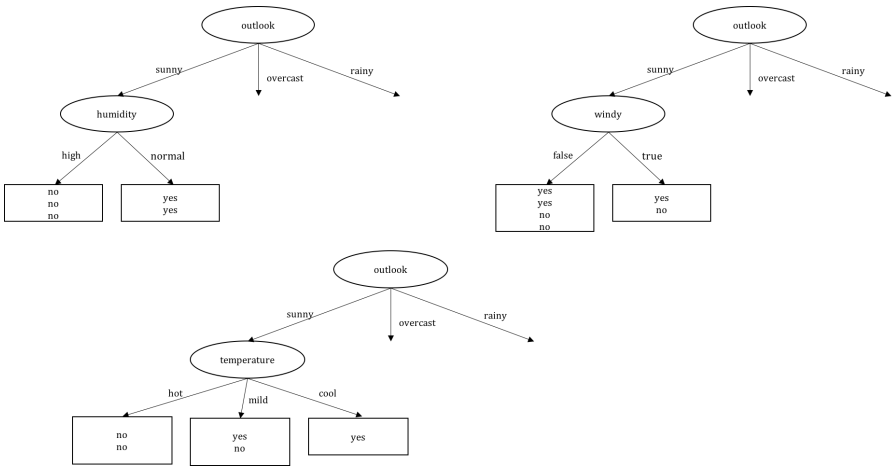

Fig. 6.2. Percabangan

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*.

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

Proses ini dilanjutkan sampai kita tidak bisa mencabang lagi.

6.3 Isu pada ID3

Pada algoritma *decision tree* secara umum, terdapat beberapa isu diantara lain [3, 17]:

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*

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 penyumpurnaan!

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 Table. 6.1 hingga membentuk *decision tree* akhir seperti pada Fig. 6.1!

7

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 algorithma *machine learning* yang sedang popular belakangan ini, yaitu *artificial neural network*. Pembahasan akan dimulai dari hal-hal sederhana sampai yang lebih kompleks. Bab ini juga mencakup variasi *neural network* seperti ***deep neural network*, *recurrent neural network*, dan *recursive neural network***. Bab ini akan lebih berfokus pada penggunaan *artifical neural network* untuk *supervised learning*.

7.1 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*) [18, 19, 20, 21]. 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* [19, 21, 22, 23, 24], contoh penerapannya dijelaskan pada bab 10.

Artificial Neural Network (selanjutnya disingkat ANN), menghasilkan model yang sulit dibaca dan dimengerti oleh manusia. ANN menggunakan relatif banyak parameter sehingga kita sulit untuk mengetahui apa saja yang terjadi saat proses pembelajaran. Pada bidang riset ini, ANN disebut agnostik (kita percaya, tetapi sulit membuktikan kenapa bisa benar). Secara matematis, ANN ibarat sebuah graf. ANN memiliki neuron/*node* (*vertex*), dan sinapsis (*edge*). Topologi ANN akan dibahas lebih detil sub bab berikutnya. Sebagai gambaran, ANN berbentuk seperti Fig. 7.1.

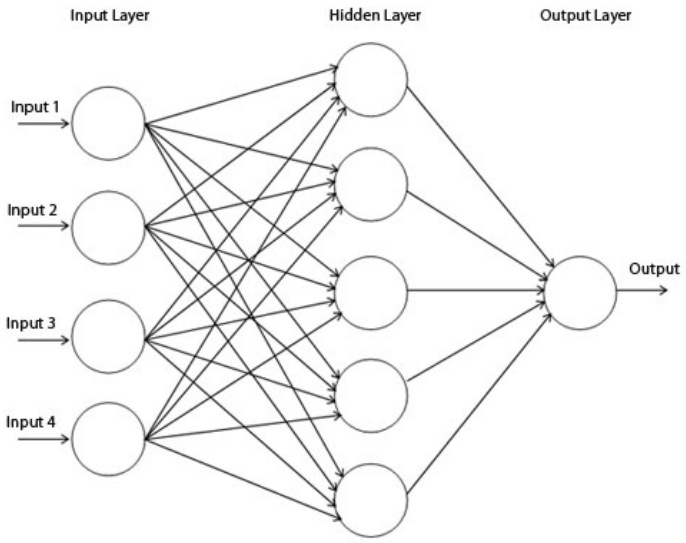

Fig. 7.1. Multilayer Perceptron

7.2 Single Perceptron

Bentuk terkecil (minimal) sebuah ANN adalah *single perceptron* yang hanya terdiri dari sebuah neuron. Sebuah neuron diilustrasikan pada Fig. 7.2. Secara matematis, terdapat *feature vector* $\overrightarrow{\mathbf{x}}$ yang menjadi *input* bagi neuron tersebut. Neuron akan memproses *input* $\overrightarrow{\mathbf{x}}$ melalui perhitungan jumlah perkalian antara nilai *input* dan *synapse weight*. Pada *training*, yang dioptimasi adalah nilai *synapse weight* (*learning parameter*). Selain itu, terdapat juga bias θ sebagai kontrol tambahan (ingat materi *steepest gradient descent*). *Output* dari neuron adalah hasil fungsi aktivasi dari perhitungan jumlah perkalian antara nilai *input* dan *synapse weight*. Ada beberapa macam fungsi aktivasi, misal ***step function***, ***sign function***, 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 7.1. Bila di-*plot* menjadi grafik, fungsi ini memberikan bentuk seperti huruf S.

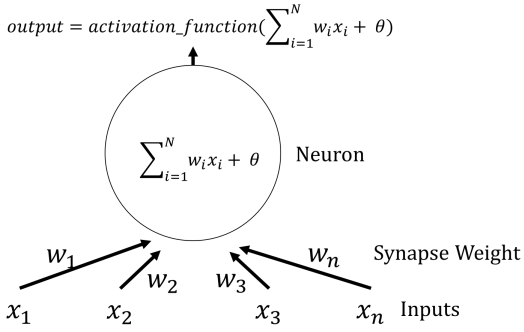

Fig. 7.2. Single Perceptron

$$\sigma(u) = \frac{1}{1 + e^{-u}} \tag{7.1}$$

Untuk melakukan pembelajaran *single perceptron*, *training* dilakukan berdasarkan ***perceptron training rule***. Prosesnya sebagai berikut [13]:

1. Inisiasi nilai *synapse weights*, bisa *random* ataupun dengan aturan tertentu.
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 maka lakukan perubahan terhadap *learning parameter* (*synapse weight*).
6. Ulangi langkah-langkah ini sampai tidak ada perubahan nilai *error*, atau nilai *error* adalah 0.

Error function diberikan pada persamaan 7.2 dan perubahan *synapse weight* diberikan pada persamaan 7.3. y melambangkan *desired output*, o melambangkan *actual output*. η disebut sebagai *learning rate*. Pada kasus *single perceptron* nilai K adalah 1.

$$E(\overrightarrow{\mathbf{w}}) = \frac{1}{2}\sum_{k=1}^{K}(y_k - o_k)^2; K = \#output\ neurons \tag{7.2}$$

$$\Delta w_i = \eta(y - o)x_i \tag{7.3}$$

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. Salah satu strategi sederhana adalah membagi-bagi kelas menjadi range nilai. Seumpama kita mempunyai lima kelas. Kelas pertama direpresentasikan dengan nilai *output* $0.0 - 0.2$, kelas kedua $0.2 - 0.4$, dst.

7.3 Multilayer Perceptron

Kamu sudah belajar *training* untuk *single perceptron*. Selanjutnya kita akan mempelajari *multilayer perceptron* (MLP). Seperti ilustrasi pada Fig. 7.3, *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*, 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 dinamakan *feed forward* [19, 13]. Hal serupa berlaku untuk *artificial neural network* dengan lebih dari 3 *layers*.

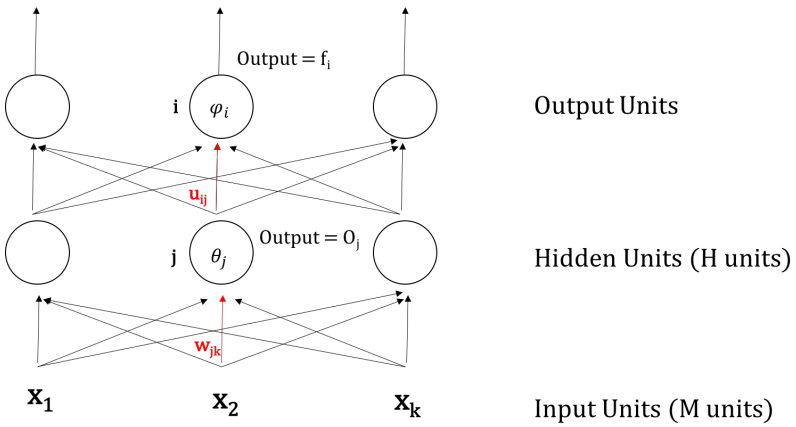

Fig. 7.3. Multilayer Perceptron 2

$$o_j = \sigma(\sum_{k=1}^{M} w_{jk}x_k + \theta_j) \tag{7.4}$$

$$f_i = \sigma(\sum_{j=1}^{H} u_{ij}o_j + \gamma_i) = \sigma(\sum_{j=1}^{H} u_{ij}\sigma(\sum_{k=1}^{M} w_{jk}x_k + \theta_j) + \gamma_i) \tag{7.5}$$

Perhatikan persamaan 7.4 dan 7.5 untuk menghitung *output* pada *layer* yang berbeda. u, w, θ, γ adalah *learning parameters*. θ, γ melambangkan *noise* atau *bias*. M adalah banyaknya *hidden units* dan H adalah banyaknya *output units*.

Untuk melatih MLP, algoritma yang umumnya digunakan adalah ***backpropagation***. Arti kata *backpropagation* sulit untuk diterjemahkan ke dalam bahasa Indonesia. Idenya adalah, dari *output layer* bisa ada *error* dibandingkan *desired output*; dari *error* tersebut, kita perbaharui parameter (*synapse weights*). Intinya adalah mengkoreksi *synapse weight* dari *output layer* ke *hidden layer*, kemudian *error* tersebut dipropagasi ke layer berikut-berikutnya. Artinya, perubahan *synapse weight* pada suatu layer dipengaruhi oleh perubahan *synapse weight* pada layer setelahnya.

Ingat kembali materi *gradient descent*. Untuk meminimalkan *error*, kita menggunakan prinsip *gradient descent*. Kita akan memperlajari bagaimana cara menurunkan *backpropagation* menggunakan *gradient descent* yaitu menghitung $-grad((y_i - f_i(\overrightarrow{\mathbf{x}}, \overrightarrow{\mathbf{w}}))^2)$; untuk semua *output neurons*.

Ingat kembali *chain rule* pada perkuliahan diferensial $f(g(x))' = f'(g(x))g'(x)$. Ingat kembali *error*, untuk MLP diberikan oleh persamaan 7.2.

$$E(\overrightarrow{\mathbf{w}}) = \frac{1}{2}\sum_{i=1}^{I}(y_i - f_i)^2 \tag{7.6}$$

Mari kita lakukan proses penurunan untuk melatih MLP.
Diferensial u_{ij} diberikan oleh turunan *sigmoid function*

$$\begin{aligned}\frac{\delta E(\overrightarrow{\mathbf{w}})}{\delta u_{ij}} &= (y_i - f_i)\frac{\delta f_i}{\delta u_{ij}} \\ &= (y_i - f_i)f_i(1 - f_i)o_j\end{aligned}$$

Diferensial w_{jk} diberikan oleh turunan *sigmoid function*

$$\begin{aligned}\frac{\delta E(\overrightarrow{\mathbf{w}})}{\delta w_{jk}} &= \sum_{i=1}^{H}(y_i - f_i)\frac{\delta f_i}{\delta w_{jk}} \\ &= \sum_{i=1}^{H}(y_i - f_i)\frac{\delta f_i}{\delta o_j}\frac{\delta o_j}{\delta w_{jk}} \\ &= \sum_{i=1}^{H}(y_i - f_i)[f_i(1 - f_i)u_{ij}][o_j(1 - o_j)x_k]\end{aligned}$$

Metode penurunan serupa dapat juga digunakan untuk menentukan perubahan θ dan γ. Jadi proses *backpropagation* untuk kasus Fig. 7.3 dapat diberikan seperti pada Fig. 7.4. Untuk *artificial neural network dengan* lebih dari 3 *layers*, kita pun bisa menurunkan persamaannya.

(2) Hidden to Output

$$f_i = \sigma\left(\sum_{j=1}^{H} u_{ij} o_j + \varphi_i\right)$$

(3) Output to Hidden

$$\delta_i = (y_i - f_i) f_i (1 - f_i)$$
$$\Delta u_{ij} = -\eta(t)\delta_i o_j$$
$$\Delta\varphi_i = -\eta(t)\delta_i$$

(1) Input to Hidden Layer

$$o_j = \sigma\left(\sum_{k=1}^{K} w_{jk} x_k + \theta_j\right)$$

(4) Hidden to Input

$$\gamma_j = \sum_{i=1}^{H} \delta_i u_{ij} o_j (1 - o_j)$$
$$\Delta w_{jk} = -\eta(t)\gamma_j x_k$$
$$\Delta\theta_j = -\eta(t)\gamma_j$$

Fig. 7.4. Proses latihan MLP menggunakan *backpropagation*

7.4 Binary Classification

Salah satu strategi untuk *binary classification* adalah dengan menyediakan hanya satu *output unit* di jaringan. Kelas pertama direpresentasikan dengan 0, kelas kedua direpresentasikan dengan nilai 1 (setelah diaktivasi).

7.5 Multi-label Classification

Multilayer perceptron dapat memiliki *output unit* berjumlah lebih dari satu. Oleh karena itu, terdapat keuntungan saat melakukan *multi-label classification*. Seumpama kita mempunyai empat kelas, dengan demikian kita dapat merepresentasikan keempat kelas tersebut dengan ${}^2log(4)$ *output units* = 2. Kelas pertama direpresentasikan dengan *output layer* memberikan hasil 00, kelas kedua 01, kelas ketiga 10, dan kelas keempat 11. Untuk C kelas, kita dapat merepresentasikannya dengan ${}^2log(C)$ *output units*. Selain untuk merepresentasikan *multi-label*, *output units* juga dapat merepresentasikan distribusi [22, 23].

7.6 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*, *L hidden layers*, *output layer*). Proses pembelajaran pada DNN disebut sebagai *deep learning* [5]. Jaringan *neural network* pada DNN disebut *deep network*.

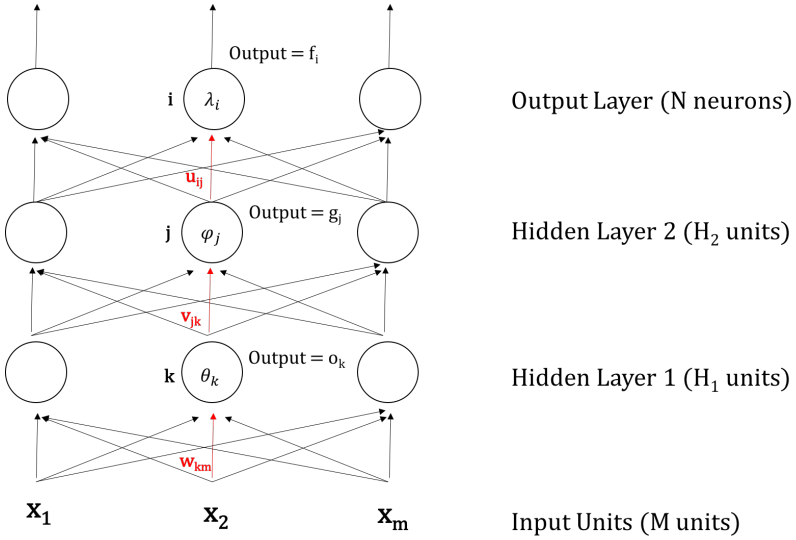

Fig. 7.5. Deep Neural Network

Perhatikan Fig. 7.5 yang memiliki 4 *layers*. Cara menghitung *final output* sama seperti MLP, diberikan pada persamaan 7.7 dimana θ, φ, λ adalah *noise* atau *bias*.

$$f_i = \sigma(\sum_{j=1}^{H_2} u_{ij}\sigma(\sum_{k=1}^{H_1} v_{jk}\sigma(\sum_{m=1}^{M} w_{km}x_m + \theta_k) + \varphi_j) + \lambda_i) \tag{7.7}$$

Cara melatih *deep neural network*, salah satunya dapat menggunakan *back-propagation*. Seperti pada bagian sebelumnya, kita hanya perlu menurunkan rumusnya saja. **Penurunan diserahkan pada pembaca sebagai latihan**. Hasil proses penurunan dapat dilihat pada Fig. 7.6.

Karena *deep network* terdiri dari banyak *layer* dan *synapse weight*, 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 [5]. *Deep learning* dapat menemukan relasi "tersembunyi"

(3) Hidden 2 to Output

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

(4) Output to Hidden 2

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

(2) Hidden 1 to Hidden 2

$$g_j = \sigma\left(\sum_{k=1}^{H_1} v_{jk} o_k + \varphi_j\right)$$

(5) Hidden 2 to Hidden 1

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

(1) Input to Hidden Layer

$$o_k = \sigma\left(\sum_{m=1}^{M} w_{km} x_m + \theta_k\right)$$

(6) Hidden 1 to Input

$$\beta_k = \sum_{j=1}^{H_1} \gamma_j v_{jk} o_k (1 - o_k)$$
$$\Delta w_{km} = -\eta(t)\beta_k x_m$$
$$\Delta\theta_k = -\eta(t)\beta_k$$

Fig. 7.6. Proses latihan DNN menggunakan *backpropagation*

antara *input* dan *output*, yang tidak dapat diselesaikan menggunakan *multi-layer perceptron* (3 layers).

Ada beberapa strategi untuk mempercepat pembelajaran menggunakan deep learning, misalnya: **Lasso** atau **Ridge**, ***successive learning***, dan penggunaan **autoencoder** [5]. 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 [25], yaitu mulai dari hal kecil. Ilustrasinya dapat dilihat pada Fig. 7.7.

Menggunakan *deep learning* harus hati-hati, karena pembelajaran cenderung *divergen*. Artinya *minimum square error* belum tentu semakin rendah seiring berjalannya waktu (*swing* relatif sering).

7.7 Recurrent Neural Network

Ada banyak variasi topologi pada ANN, salah satunya adalah ***recurrent neural network***. Idenya adalah membuat topologi jaringan yang mampu merepresentasikan *sequence* atau *compositionality*. Kita menggunakan hasil pembelajaran pada tahap sebelumnya $(t-1)$ untuk tahap sekarang (t) [21]. Ilustrasi *recurrent neural network* dapat lihat pada Fig. 7.8. Perhitungan fungsi aktivasi pada *hidden layer* dipengaruhi oleh *state hidden layer* pada iterasi

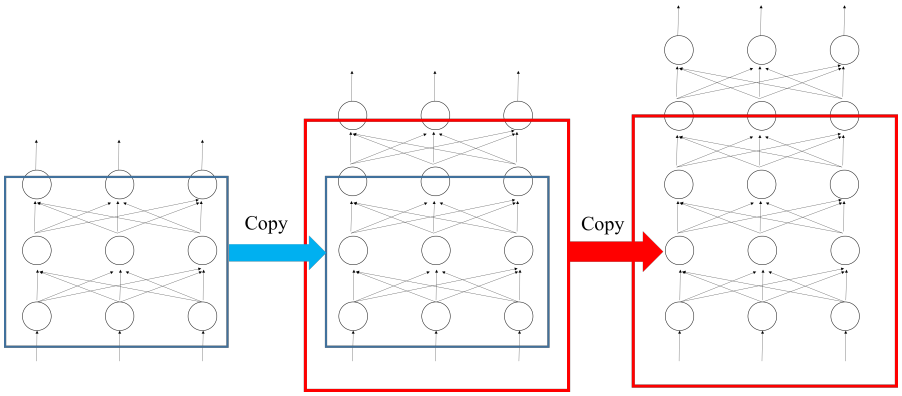

Fig. 7.7. Contoh Succesive Learning

sebelumnya. Hal ini sesuai dengan konsep *recurrent* yaitu "mengingat" kejadian sebelumnya. Secara konseptual, dapat direpresentasikan dengan persamaan 7.8, RNN ini memiliki analogi dengan *full markov chain*.

$$output = f(x, H_{t-1}) = f(x, f(x_{t-1}, H_{t-2})) = f(x, f(x_{t-1}, f(x_{t-2}, \dots))) \tag{7.8}$$

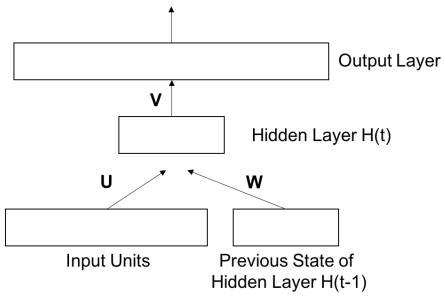

Fig. 7.8. Konsep Recurrent Neural Network

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* [26]. Awalnya kita melakukan *feedforward* seperti biasa, kemudian ketika melakukan propagasi *error*, dilakukan *unfolding* pada *neural network*. Beberapa *state* terdahulu *hidden layer* diingat kembali saat melakukan *backpropagation*. Ilustrasi dapat dilihat pada Fig. 7.9. Kita mempropagasi *error* dengan adanya efek dari *previous states of hidden layer*. *Synapse weight* 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). Bentuk topologi yang lebih besar dapat dilihat pada Fig. 7.10.

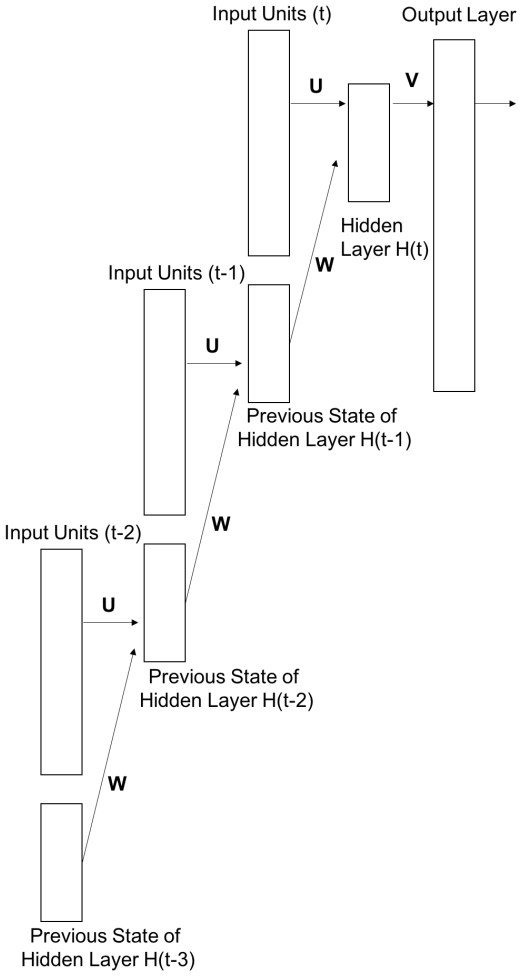

Fig. 7.9. Konsep Backpropagation Through Time [21]

7.8 Recursive Neural Network

Seperti namanya, **recursive neural network** memiliki struktur rekursif (contoh: struktur data pohon). Ilustrasi dapat dilihat pada Fig. 7.11. Bila

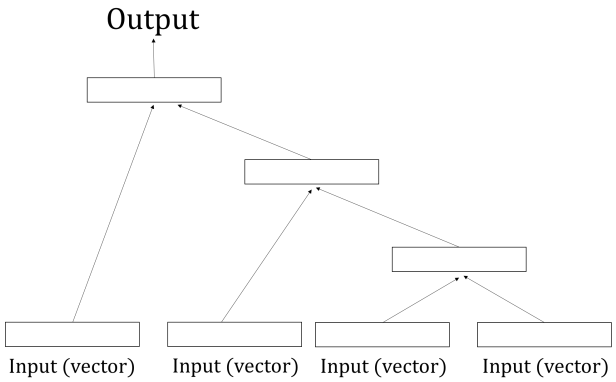

Fig. 7.10. Recurrent Neural Network

kamu perhatikan baik-baik, topologi ini berbeda dengan *recurrent neural network*. Pada *recurrent neural network* bentuk topologinya melambangkan *sequence*, sedangkan pada *recursive neural network* melambangkan *compositionality*. Kedua jargon ini akan dibahas pada bab 10. Yang perlu diingat adalah, struktur *recursive neural network* melambangkan hirarki.

7.9 Rangkuman

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

1. *Input layer*
2. *Hidden layer*
3. *Output layer*

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

Kedua, hal tersulit saat menggunakan *neural network* adalah menentukan topologi. Kamu bisa menggunakan berbacai 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 berikut [5]:

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 *semi-supervised* dan *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.

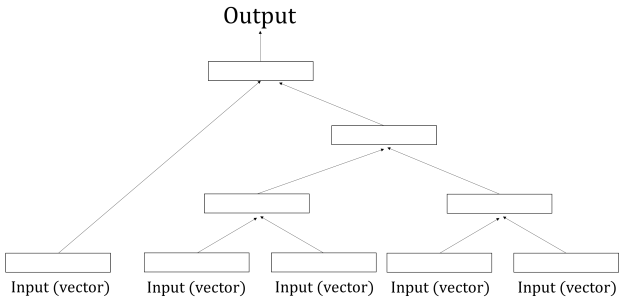

Fig. 7.11. Recursive Neural Network

Soal Latihan

7.1. Turunan
(a) Turunkanlah perubahan *noise/bias* untuk *training* pada MLP.
(b) Turunkanlah proses *training deep neural network* pada Fig. 7.6 termasuk perubahan *noise/bias*.

7.2. Lasso and Ridge
(a) Apa itu metode Lasso dan Ridge?
(b) Bagaimana cara mengimplementasikan metode tersebut pada *neural network*?
(c) Apa kelebihan dan kekurangan mereka?

7.3. Arsitektur Neural Network
Sebutkan dan jelaskan arsitektur *artificial neural network* dan cara melatih

mereka! Minimal kamu harus menjelaskan *convolutional neural network* dan *long-short term memory network*.

7.4. Recursive Neural Network
Bagaimana cara melakukan *training* untuk *recursive neural network*?

7.5. Neural Network Training
(a) Sebutkan dan jelaskan cara lain untuk melatih *artificial neural network* (selain *backpropagation*)!
(b) Apa kelebihan dan kekurangan mereka dibanding *backpropagation*?

7.6. Neural Network - Unsupervised Learning
Bagaimana cara menggunakan *artificial neural network* untuk *unsupervised learning*?

7.7. Regularization Technique
(a) Sebutkan dan jelaskan teknik *regularization* untuk *neural network*!
(b) Mengapa kita perlu menggunakan teknik tersebut?

7.8. 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?

7.9. Transformasi atribut
Pada bab 5, 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*!

8

Hidden Markov Model

> "Probability is expectation founded upon partial knowledge. A perfect acquaintance with all the circumstances affecting the occurrence of an event would change expectation into certainty, and leave neither room nor demand for a theory of probabilities."
>
> George Boole

Hidden Markov Model (HMM) adalah algoritma yang relatif sudah cukup "lama". Tetapi algoritma ini penting untuk diketahui karena merupakan *baseline* untuk *automatic speech recognizer* (ASR) dan *part-of-speech* (POS) *tagging*. Bab ini akan membahas HMM serta penggunaannya pada POS *tagging* karena penulis lebih familiar dengan POS *tagging*. HMM adalah kasus spesial ***Bayesian Inference*** [7, 27]. Untuk mengerti *Bayesian Inference*, ada baiknya kamu membaca materi ***graphical model*** pada buku [4].

8.1 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 [1] seperti pada Table. 8.1.

[1] 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

Table 8.1. Penn Treebank POS Tag

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 8.1 [7, 27].

$$c_1^{'}, c_2^{'}, \ldots, c_N^{'} = \underset{c_1, c_2, \ldots, c_N; c_i \epsilon C}{\arg\max} P(c_1, c_2, \ldots, c_N | w_1, w_2, \ldots, w_N) \tag{8.1}$$

Dimana C adalah daftar kelas kata. Akan tetapi, menghitung persamaan 8.1 sangatlah sulit karena dibutuhkan data yang sangat banyak. Teori Bayes untuk melakukan aproksimasi permasalahan ini. Ingat kembali teori Bayes seperti pada persamaan 8.2.

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)} \tag{8.2}$$

Dengan menggunakan teori Bayes, kita dapat mentransformasi persamaan 8.1 menjadi persamaan 8.3.

$$c_1, c_2, \ldots, c_N = \underset{c_1, c_2, \ldots, c_N; c_i \epsilon C}{\arg\max} \frac{P(w_1, w_2, \ldots, w_N | c_1, c_2, \ldots, c_N) P(c_1, c_2, \ldots, c_N)}{P(w_1, w_2, \ldots, w_N)} \tag{8.3}$$

Untuk suatu sekuens input, $P(w_1, w_2, \ldots, w_N)$ akan selalu sama untuk suatu sekuens sehingga dapat diabaikan. Oleh karena itu, persamaan 8.3 dapat disederhanakan menjadi 8.4.

$$c_1, c_2, \ldots, c_N = \underset{c_1, c_2, \ldots, c_N; c_i \epsilon C}{\arg\max} P(w_1, w_2, \ldots, w_N | c_1, c_2, \ldots, c_N) P(c_1, c_2, \ldots, c_N) \tag{8.4}$$

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 8.4 masih dapat disederhanakan kembali menggunakan **Markov Assumption**, yaitu dengan membuat asumsi saling lepas pada sekuens (*independence assumption*). Terdapat dua asumsi, pertama, kategori suatu kata hanya bergantung pada dirinya sendiri tanpa memperhitungkan kelas kata disekitarnya seperti pada persamaan 8.5. Asumsi kedua adalah suatu kemunculan kategori kata hanya bergantung pada kelas kata sebelumnya seperti pada persamaan 8.6.

$$P(w_1, w_2, \ldots, w_N | c_1, c_2, \ldots, c_N) = \prod_{i=1}^{N} P(w_i | c_i) \tag{8.5}$$

$$P(c_1, c_2, \ldots, c_N) = \prod_{i=1}^{N} P(c_i | c_{i-1}) \tag{8.6}$$

Dengan demikian, persamaan 8.4 disederhanakan kembali menjadi persamaan 8.7. Persamaan ini disebut ***bigram assumption*** atau ***first-order markov chain***.

$$c_1, c_2, \ldots, c_N = \underset{c_1,c_2,\ldots,c_N;c_i \epsilon C}{\arg\max} \prod_{i=1}^{N} P(w_i|c_i)P(c_i|c_{i-1}) \tag{8.7}$$

Kita dapat membuat ekstensi persamaan 8.7 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, baru-baru ini peneliti dari Google berhasil memecahkan permasalahan yang mirip (*sequence-to-sequence*) menggunakan *artificial neural network* [28]. Anda dapat membaca paper tersebut, tapi 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.

$$\begin{aligned} P(noun, verb, noun) = {} & P(budi|noun)P(noun|null)P(menendang|verb) \\ & P(verb|noun)P(bola|noun)P(noun|verb) \end{aligned} \tag{8.8}$$

8.2 Hidden Markov Model Tagger

Pada subbab sebelumnya, POS *tagging* telah didefinisikan secara matematis. Kita tahu apa yang ingin kita hitung. Subbab ini adalah formalisasi *hidden markov model tagger*.

Ingat kembali persamaan 8.7 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* yang mana sekuens input menentukan *states* yang akan dilewati oleh automaton. Total bobot *outgoing edges* untuk masing-masing *state* pada automaton haruslah 1 apabila dijumlahkan.

Sebagai contoh, perhatikan Table. 8.2. *ART* adalah *article*, *N* adalah *noun*, *V* adalah *verb* dan *P* adalah *preposition*. Mereka adalah contoh kelas kata yang disederhanakan demi membuat contoh yang mudah. Tabel ini, ketika dikonversi menjadi *weighted automaton* akan menjadi Fig. 8.1.

Table. 8.2 dan Fig. 8.1 telah merepresentasikan probabilitas *prior*, sekarang kita ingin model yang kita punya juga mencakup ***lexical generation probabilities***, yaitu *likelihood* pada persamaan 8.7.

Seumpama kita mempunyai *lexical-generation probabilities* seperti pada Table. 8.3. Setiap *state* pada automaton, dapat menghasilkan/meng-*output*-kan suatu kata (*word*) dengan probabilitas pada Table. 8.3. Kita kembangkan lagi Fig. 8.1 dengan tambahan informasi *lexical generation probabilities* menjadi Fig. 8.2. Automaton ini disebut **hidden markov model** (HMM). Kata *hidden* berarti, untuk setiap kata pada sekuens, kita tidak mengetahui kata

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

Table 8.2. Probabilitas Bigram [27]

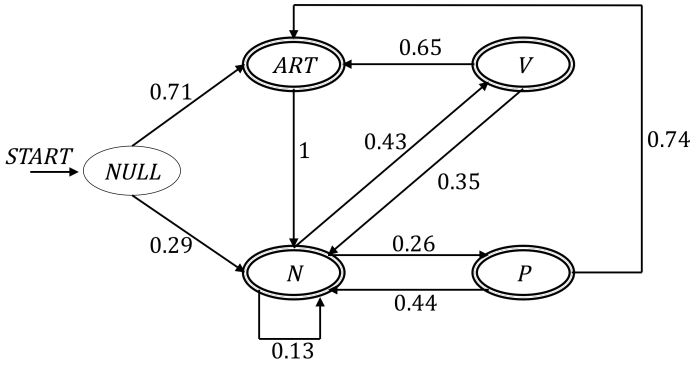

Fig. 8.1. Weighted Automaton [27]

$P(the\|ART)$	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		

Table 8.3. Lexical-generation Probabilities [27]

tersebut dihasilkan oleh *state* mana. Misalkan, kata *flies* dapat dihasilkan oleh *state* N atau V [27].

Diberikan kalimat "*flikes like a flower*", untuk menghitung sekuens kelas kata untuk kalimat tersebut, kita menyelulusuri automaton Fig. 8.2. Hasil penelusuran memberikan kita kombinasi sekuens yang mungkin seperti pada Fig. 8.3. 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).

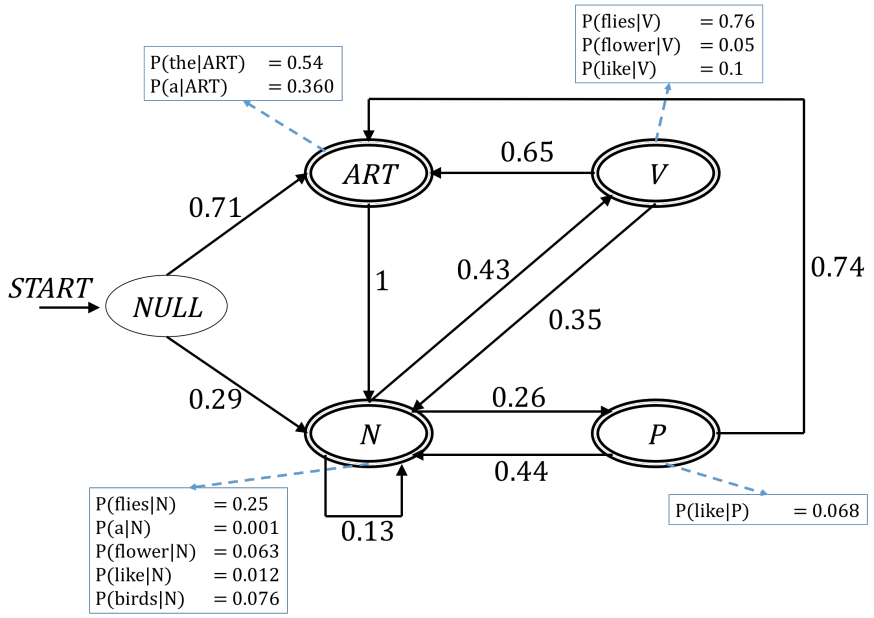

Fig. 8.2. Complete Hidden Markov Model

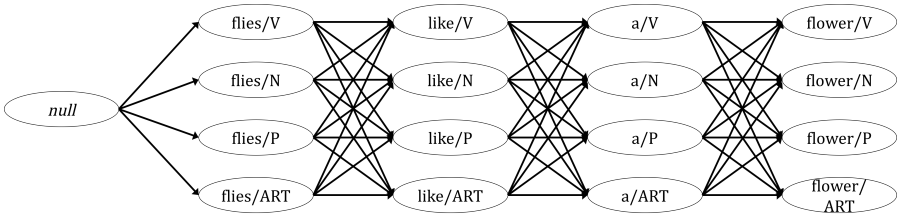

Fig. 8.3. Sekuens yang mungkin

Secara formal, ***hidden markov model tagger*** didefinisikan oleh beberapa komponen [7]:

1. $Q = \{q_1, q_2, \ldots, q_N\}$ yaitu himpunan ***states***.
2. $A = a_{00}, a_{11}, a_{22}, \ldots, a_{NN}$ yaitu **transition probability matrix** dari suatu *state* i menuju *state* j; dimana $\sum_{j=0}^{N} a_{ij} = 1$.
3. $O = o_1, o_2, \ldots, o_N$ yaitu sekuens observasi kata.

4. $B = b_i(o_w)$ yaitu sekuens dari *observation likelihood*, atau disebut dengan *emission probabilities*, merepresentasikan sebuah observasi kata o_w dihasilkan oleh suatu *state-i*.
5. q_0, Q_F yaitu kumpulan *state* spesial yang terdiri dari **start state** dan **final state(s)**.

8.3 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* [2]. Ide utama algoritma Viterbi adalah mengingat sekuens terbaik untuk kategori pada masing-masing posisi kata. Apabila kita telah sampai pada kata terakhir, kita lakukan *backtrace* untuk mendapatkan sekuens terbaik.

---

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] $\leftarrow a_{0,s} * b_s(o_1)$
    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*$[j, t-1] * a_{s,j} * b_s(o_t)$
        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},$i+1$]

Fig. 8.4. Algoritma Viterbi [7, 27]

[2] https://en.wikipedia.org/wiki/Edit_distance

Perhatikan Fig. 8.4 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. Fig. 8.5 adalah ilustrasi algoritma Viterbi untuk kalimat *input* (*observed sequence*) "*flies like a flower*" dengan *lexical generation probability* pada Table. 8.3 dan *transition probabilities* pada Table. 8.2 (bigram) [27]. Panah berwarna merah melambangkan *backpointer*. Setelah *iteration step* selesai, kita lakukan *backtrace* dan mendapat hasil seperti pada Fig. 8.6. Dengan itu, kita mendapatkan sekuens "flies/N like/V a/ART flower/N".

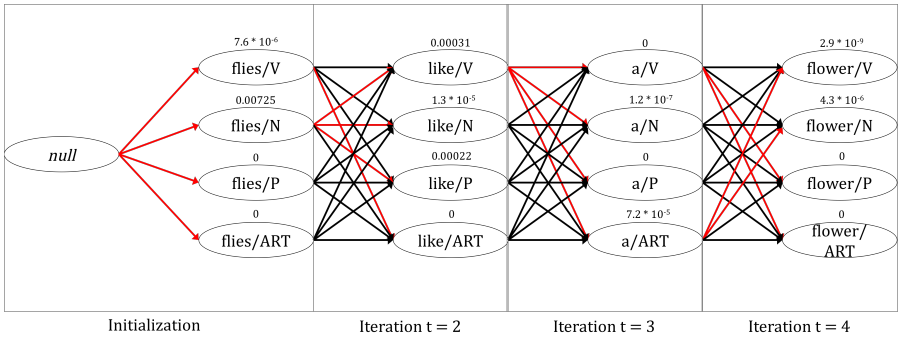

Fig. 8.5. Ilustrasi Algoritma Viterbi per Iterasi

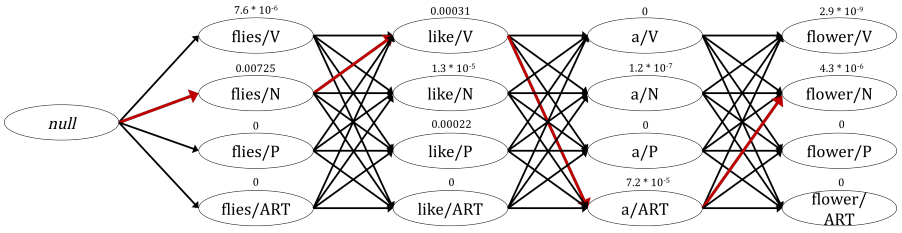

Fig. 8.6. Viterbi Backtrace

8.4 Proses Training Hidden Markov Model

Hidden Markov Model (HMM) adalah salah satu varian *supervised learning*, diberikan sekuens *input* dan *output* yang bersesuaian sebagai *training data*.

Pada kasus POS *tagging*, yaitu kalimat dan kelas kata untuk masing-masing kata/*token* pada kalimat. Saat melatih HMM, kita ingin mengestimasi matriks A dan B yaitu *transition probabilities* dan *emission probabilities/lexical-generation probabilities*. Kita melatih HMM dengan menggunakan **Algoritma Foward-Backward** (**Baum-Welch Algorithm**).

Cara paling sederhana untuk menghitung *emission probabilities* atau *transition probabilities* adalah dengan menghitung kemunculan. Sebagai contoh, *emission probability* suatu kata untuk setiap kelas kata diberikan pada persamaan 8.9 (bentuk *softmax function*).

$$P(w|c_i) = \frac{count(w, c_i)}{\sum_{j=1}^{N} count(w\, c_j)} \tag{8.9}$$

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 di *string*) pada semua kemungkinan sekuens, diberikan input $w_1, w_2, \ldots, w_N$. Kami ambil contoh, kata *flies* sebagai *noun* pada kalimat "*The flies like flowers*", dihitung sebagai penjumlahan seluruh sekuens yang berakhir dengan *flies* sebagai *noun*. Probabilitas $P(flies/N \mid The\ flies) = \frac{P(flies/N\ \&\ The\ flies)}{P(The\ flies)}$.

Agar lebih *precise*, secara formal, kita definisikan terlebih dahulu ***forward probability*** sebagai 8.10.

$$\alpha_i(t) = P(w_t/c_i \mid w_1, w_2, \ldots, w_N) \tag{8.10}$$

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 Fig. 8.7, dengan c_i adalah kelas kata ke-i dan w_i adalah kata ke-i. a adalah *transition probability* dan b adalah *emission probability*.

Initialization Step
for $i = 1$ **to** N **do**
  $\alpha_i(t) \leftarrow b_i(o_1) * a_{0,i}$

Comparing the Forward Probabilities
for $t = 2$ **to** T **do**
  for $i = 1$ **to** N **do**
    $\alpha_i(t) \leftarrow \sum_{j=1}^{N}(a_{ji} * \alpha_j(t-1)) * b_i(o_t)$

Fig. 8.7. Algoritma forward [27]

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 Fig. 8.8, dengan a adalah *transition probability* dan b adalah *emission probability*.

Initialization Step
for $i = 1$ **to** N **do**
  $\beta_i(T) \leftarrow P(c_T = c_i)$

Comparing the Backward Probabilities
for $t = T - 1$ **to** t **do**
  for $i = 1$ **to** N **do**
    $\beta_i(t) \leftarrow \sum_{j=1}^{N}(a_{ji} * \beta_i(t+1)) * b_j(o_{j+1})$

Fig. 8.8. Algoritma backward [27]

Gabungan *forward* dan *backward probability* dapat digunakan untuk mengestimasi $\gamma_j(t)$ yaitu probabilitas berada pada *state* c_j pada waktu ke-t dengan persamaan 8.11.

$$\gamma_j(t) = \frac{\alpha_i(t) * \beta_i(t)}{\sum_{j=1}^{N} \alpha_j(t) * \beta_j(t)} \tag{8.11}$$

Kita mengestimasi probabilitas keberadaan pada *state* tertentu, berdasarkan pengaruh probabilitas keseluruhan sekuens.

Dengan menggunakan *forward probability* dan *backward probability* sekaligus, kita definisikan $\xi_t(i, j)$ yaitu probabilitas berada di *state-i* pada waktu ke t dan *state-j* pada waktu ke-$(t + 1)$ dengan persamaan 8.12.

$$\xi_t(i, j) = \frac{\alpha_i(t) * a_{ij} * b_j(o_{t+1}) * \beta_j(t + 1))}{\alpha_N(T)} \tag{8.12}$$

Dengan a_{ij} adalah *transition probability* dan $b_j(o_{t+1})$ adalah *emission probability* (ingat kembali definisi formal HMM pada subbab 8.2). Pada setiap iterasi, kita ingin memperbaharui kembali parameter HMM yaitu matriks A dan B. Kita hitung kembali *transition probability*, diberikan oleh persamaan 8.13.

$$a_{ij}{}' = \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \sum_{j=1}^{N} \xi_t(i, j)} \tag{8.13}$$

Kita juga menghitung kembali *emission probability*, diberikan oleh persamaan 8.14.

$$b_j(o_k)' = \frac{\sum_{t=1, o_k = w_k}^{T} \gamma_j(t)}{\sum_{t=1}^{T} \gamma_j(t)} \tag{8.14}$$

Initialize A and B
Iterate until convergence
E-step

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

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

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

Fig. 8.9. Algoritma Forward-Backward (EM) [7]

$\sum_{t=1,o_k=w_k}^{T}$ 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** [7]. Pada E-step, kita mengestimasi probabilitas berada di suatu *state* c_j menggunakan $\gamma_j(t)$ dan mengestimasi transisi $\xi_t(i,j)$ berdasarkan matriks A dan B yang sudah diberikan pada tahap iterasi *training* (*epoch*) sebelumnya. Pada M-step, kita menggunakan γ dan ξ untuk mengestimasi kembali matriks A dan B. Hal ini dijelaskan secara formal pada Fig. 8.9. Menggunakan algoritma ini, walaupun HMM menggunakan *independence assumption*, kita dapat mengikutsertakan pengaruh probabilitas keseluruhan sekuens untuk perhitungan probabilitas keberadaan kita pada *state* di waktu tertentu, serta *emission probability*. Metode ini dapat dianggap sebagai suatu cara optimalisasi (ingat kembali materi *gradient descent* pada bab 3). Kita sudah berada pada titik *local optimal* apabila tidak ada perubahan parameter. Selain itu, dari sisi statistik, teknik ini dapat dianggap sebagai suatu metode *smoothing* [3].

Soal Latihan

8.1. Data Numerik

Pada bab ini, diberikan contoh aplikasi Hidden Markov Model (HMM) untuk POS *tagging*, dimana data kata adalah data nominal. Berikan strategi penggunaan HMM untuk data numerik! Misal, pada *automatic speech recognizer*.

[3] https://en.wikipedia.org/wiki/Smoothing

8.2. Ekstensi Algoritma Viterbi
Buatlah ekstensi algoritma Viterbi untuk asumsi trigram!

8.3. Maximum Entropy Markov Model
(a) Jelaskan konsep *maximum entropy*!
(b) Jelaskan *maximum entropy markov model*!

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

9

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 5, 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 [29], walaupun relatif lama, tetapi paper tersebut memberikan penjelasan yang mudah dimengerti tentang *clustering*. Selain itu, kamu juga dapat membaca paper [30].

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 [31]: (1) *partitional*, yaitu menentukan partisi sejumlah K dan (2) *hierarchical*, yaitu mengelompokan data berdasarkan struktur taksonomi. Contoh algoritma *partitional* adalah **K-means** pada subbab 9.1, sementara contoh algoritma *hierarchical* adalah **agglomerative clustering** pada subbab 9.2.

9.1 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.
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 5, 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*.

Diberikan sekumpulan data $\{d_1, d_2, \cdots, d_n\}$ pada suatu *cluster* dalam bentuk *feature vector*, maka centroid c untuk *cluster* itu dihitung dengan

$$c = \frac{1}{n} \sum_{i=1}^{n} d_i \tag{9.1}$$

yaitu nilai rata-rata setiap elemen *feature vector* untuk seluruh anggota *cluster* tersebut, dimana n adalah banyaknya anggota *cluster* dan d_i adalah dokumen ke-i. Dengan ini, centroid secara umum bisa jadi tidak merupakan elemen anggota *cluster* (centroid bukan sebuah instans data).

Pada bab 5 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 9.2, yaitu *dot product* antara dua vektor dibagi dengan

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

perkalian *norm* kedua vektor.

$$cosSim(d_i, d_j) = \frac{d_i d_j}{\|d_i\| \, \|d_j\|} \tag{9.2}$$

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

9.2 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* [29]:

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 9.1 yang disebut **dendogram**. Dendogram melambangkan taksonomi, sebagai contoh taksonomi dokumen berita *sport*, dipecah menjadi *baseball* dan *soccer*.

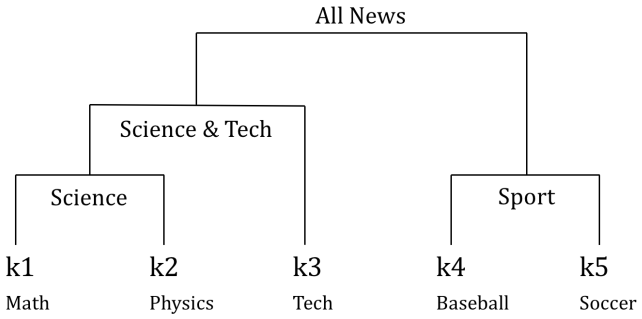

Fig. 9.1. Ilustrasi Hierarchical Clustering

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 9.1). Perhitungan kemiripan *clusters* dapat dihitung dengan tiga metode [31] (untuk data numerik):

1. ***Single Link***. Nilai kemiripan dua *clusters* dihitung berdasarkan nilai kemiripan **maksimum** diantara anggota kedua *clusters* tersebut.

$$Sim_{single-link}(K_i, K_j) = \max_{d_i \in K_i, d_j \in K_j} cosSim(d_i, d_j) \tag{9.3}$$

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

$$Sim_{complete-link}(K_i, K_j) = \min_{d_i \in K_i, d_j \in K_j} cosSim(d_i, d_j) \tag{9.4}$$

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

$$Sim_{UPGMA}(K_i, K_j) = \frac{1}{n_i n_j} \sum_{d_i \in K_i, d_j \in K_k} cosSim(d_i, d_j) = \frac{c_i c_j}{n_i n_j} \tag{9.5}$$

9.3 Evaluasi

Diberikan sekumpulan data $\{d_1, d_2, \cdots, d_n\}$ untuk suatu *cluster*. Saat tidak tersedianya informasi label/kelas untuk setiap data, kualitas hasil *clustering* dapat dihitung dengan dua kriteria yaitu:

1. ***Intra-cluster similarity***, yaitu menghitung rata-rata kedekatan antara suatu anggota dan anggota *cluster* lainnya.

$$I = \frac{1}{n^2} \sum_{d_i d_j, i \neq j} cosSim(d_i, d_j) \tag{9.6}$$

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

$$I = \frac{1}{n^2} \sum_{d_i d_j, i \neq j} cosSim(d_i, d_j) = \|c\|^2 \tag{9.7}$$

Perhitungan ini dapat dinormalisasi sesuai dengan banyaknya anggota cluster

$$I' = \frac{\|c\|^2}{n} \tag{9.8}$$

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 [30].

$$E = \sum_{k=1}^{K} n_k \frac{c_k C}{\|c_k\|} \tag{9.9}$$

dimana c_k adalah centroid *cluster* k, C adalah centroid (*mean*) dari seluruh data, dan n_k adalah banyaknya anggota *cluster* k. 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:

$$H = \frac{\sum_{k=1}^{K} I'_k}{E} = \frac{\sum_{k=1}^{K} \frac{\|c_k\|^2}{n_k}}{\sum_{k=1}^{K} n_k \frac{c_k C}{\|c_k\|}} = \sum_{k=1}^{K} \frac{\|c_k\|}{n_k^2 c_k C} \tag{9.10}$$

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

9.1. Intra-cluster Evaluation
Buktikan kebenaran persamaan 9.7.

9.2. Entropy
Bagaimana cara menghitung kualitas algoritma *clustering*, jika diberikan informasi label/ kelas setiap data menggunakan: (hint, baca [29])

(a) Entropy
(b) Purity

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

9.4. 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.

9.5. 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)?

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

Part III

Topik Lanjutan

10

Autoencoder (Dimensionality Reduction)

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

Bab ini memuat materi yang relatif cukup sulit (karena agak *high level*), kamu boleh melewati bab ini bila dianggap susah. 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 [32]. Teknik yang dibahas pada bab ini adalah **autoencoder** untuk melakukan pengurangan dimensi pada *feature vector* (*dimensionality reduction*), teknik ini dapat tergolong *semi-supervised* atau *unsupervised learning*. Kami tidak memberikan golongan yang jelas karena memang berada diantara kedua hal tersebut. Teknik autoencoder juga sering dikenal dengan *representation learning*. Artinya, mengubah suatu representasi menjadi bentuk representasi lain, sedemikian sehingga informasi yang terdapat pada representasi asli tidak hilang/terjaga.

10.1 Curse of Dimensionality

Curse of dimensionality dapat dipahami secara mendalam apabila kamu membaca buku [33]. 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 relatif "besar". Karena dimensi *feature vector* besar, kita butuh sumber daya komputasi yang besar

juga. Untuk itu, terdapat metode-metode ***feature selection*** 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*.

Untuk masalah yang kehilangan informasi, kita bisa menggunakan cara lain untuk mengurangi kompleksitas komputasi adalah dengan melakukan kompresi *feature vector*. **Autoencoding** adalah metode untuk melakukan **kompresi** *feature vector* menggunakan *neural network*. Proses melakukan kompresi disebut ***encoding***, hasil *feature vector* dalam bentuk terkompres disebut ***coding***, proses mengembalikan hasil kompresi ke bentuk awal disebut ***decoding***. *Neural network* yang mampu melakukan hal ini disebut **autoencoder** [34, 35, 36, 37, 38]. Ilustrasi autoencoder dapat dilihat pada Fig. 10.1. Karena tujuan autoencoder untuk kompresi, jumlah neuron pada *hidden layer* sebaiknya lebih sedikit dibandingkan neuron pada *input layer*. *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%.

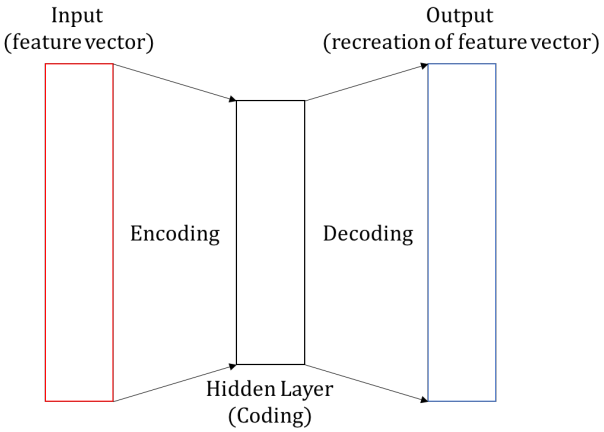

Fig. 10.1. Contoh autoencoder sederhana

10.2 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[1]. 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 antar kata adalah ***statistical semantics hypothesis*** yang menyebutkan pola penggunaan kata dapat digunakan untuk menemukan arti kata. Contoh sederhana, kata yang muncul pada "**konteks**" yang sama cenderung memiliki makna yang sama [39]. Perhatikan "konteks" dalam artian NLP adalah kata-kata sekitar (*surrounding words*); 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 [32]. Arti kata ***embedding*** adalah transformasi kata menjadi bentuk matematis (vektor). "Kedekatan hubungan makna" (***semantic relationship***) antar kata kita harapkan dapat tercermin pada operasi vektor. Salah satu metode sederhana untuk merepresentasikan kata sebagai vektor adalah **Vector Space Model**.

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*.

10.3 Vector Space Model

Vector space model (VSM) 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 antar kata juga mampu menangkap hubungan antar dokumen (*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 Fig. 10.2.

[1] https://wordnet.princeton.edu/

	Document 1	Document 2	Document 3	Document 4	...
King	1	0	0	0	...
Queen	0	1	0	0	
Prince	0	0	1	0	
Princess	0	0	0	1	

Fig. 10.2. Contoh 1-of-V encoding

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 10.1. 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.

$$IDF(t, D) = log\left(\frac{N}{\|d \epsilon D; t \epsilon d\|}\right) \tag{10.1}$$

Dengan menggunakan perhitungan TF-IDF yaitu $TF * IDF$ untuk mengisi sel pada matriks Fig. 10.2, 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.

Statistical semantics hypothesis diturunkan lagi menjadi empat macam hipotesis [39]:

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

Silahkan pembaca mencari sumber tersendiri untuk mengerti keempat hipotesis tersebut atau membaca paper [39].

10.4 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* adalah diketahuinya sekuens kedepan secara penuh atau tidak. Menurut saya, beda paling utama adalah tentang waktu. Data *time series* berkaitan dengan perubahan data berdasarkan waktu, sementara data *sequential* bukan berkaitan dengan waktu, melainkan urutan (CMIIW).

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 Fig. 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.

10.5 Distributed Word Representation

Seperti yang disebutkan pada bagian sebelumnya, kita ingin hubungan kata dapat direpresentasikan sebagai operasi vektor seperti pada ilustrasi Fig. 10.4. Kata "raja" memiliki sifat-sifat yang dilambangkan oleh suatu vektor (misal 90% aspek loyalitas, 80% kebijaksanaan, 90% aspek kebangsaan, dst), begitu

[2] source: Pinterest

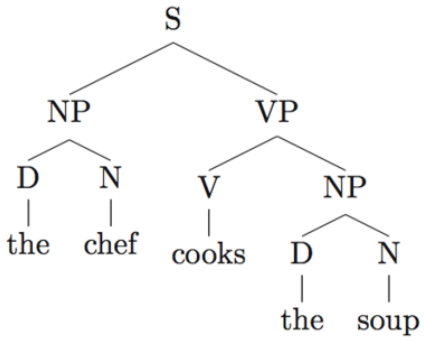

Fig. 10.3. Contoh Constituent Tree[2]

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.

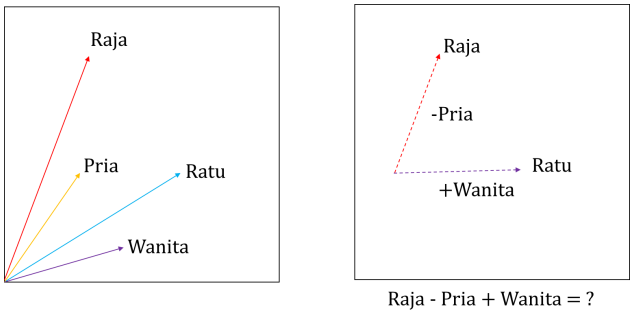

Fig. 10.4. Contoh Operasi Vektor Kata

Selain Vector Space Model, apakah ada cara lain yang mampu merepresentasikan kata dengan lebih baik? Salah satu kekurangan VSM adalah tidak memadukan sifat *sequential* pada konstruksi vektornya. Cara lebih baik ditemukan oleh [22, 23] dengan ekstensi pada [38]. Idenya adalah menggunakan teknik autoencoder 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. Autoencodernya 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 Fig. 10.5. Kedua arsitektur ini sangat erat dengan prinsip autoencoder, bagian *projection layer* pada Fig. 10.5 adalah *coding* layer. Kami akan memberikan contoh CBOW secara lebih detil.

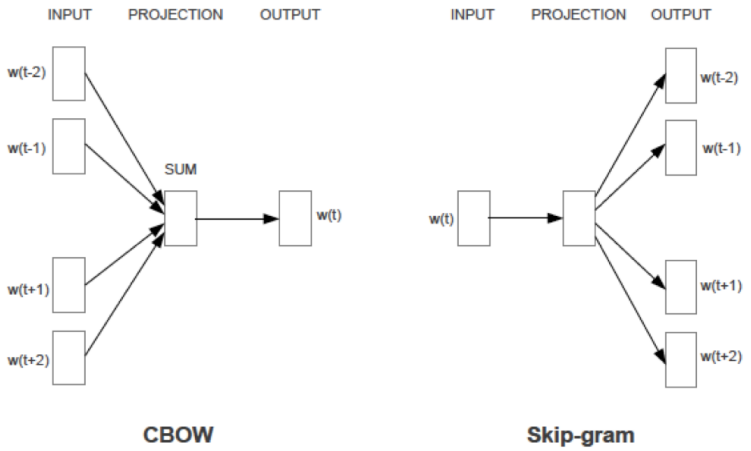

Fig. 10.5. CBOW vs Skip Gram [23]

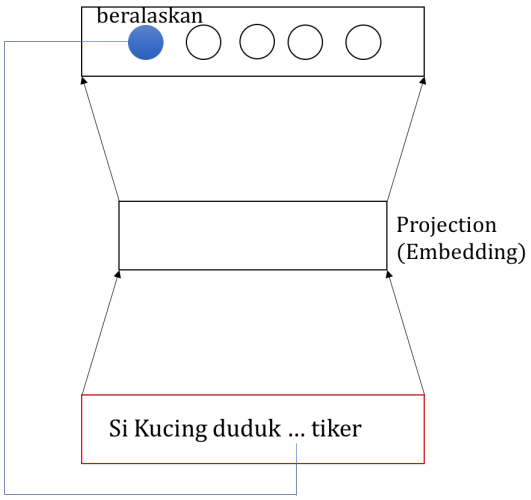

Fig. 10.6. CBOW

Perhatikan Fig. 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 *sequential atau time series* dan *statistical semantics hypothesis* terpenuhi (*to a certaint 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 menggunakan mencoba-coba kode pada pranala [3] dengan bahasa pemrograman Python 2.7.

10.6 Distributed Sentence Representation

Kita sudah dapat merepresentasikan kata menjadi vektor, selanjutnya kita ingin mengkonversi unit lebih besar (kalimat) menjadi vektor. 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 mengkonversi kata menjadi vektor, untuk mengkonversi kalimat menjadi vektor cara sederhananya adalah merata-ratakan nilai vektor kata-kata pada kalimat tersebut. Tetapi dengan cara sederhana ini, sifat *sequential* tidak terpenuhi. Selain itu, kalimat juga memiliki sifat *compositionality*.

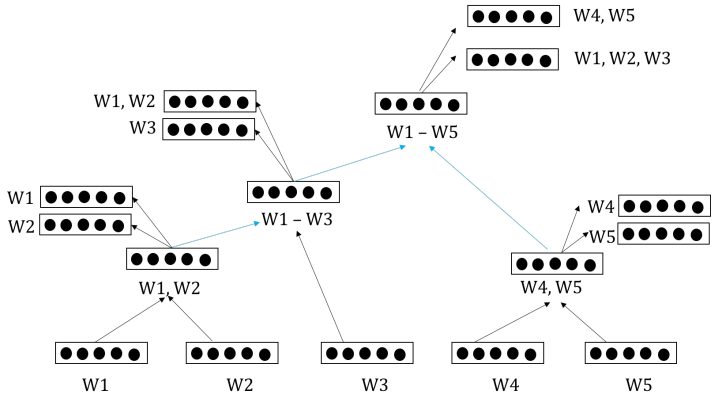

Fig. 10.7. Contoh recursive autoencoder

[3] https://github.com/wiragotama/GloVe_Playground

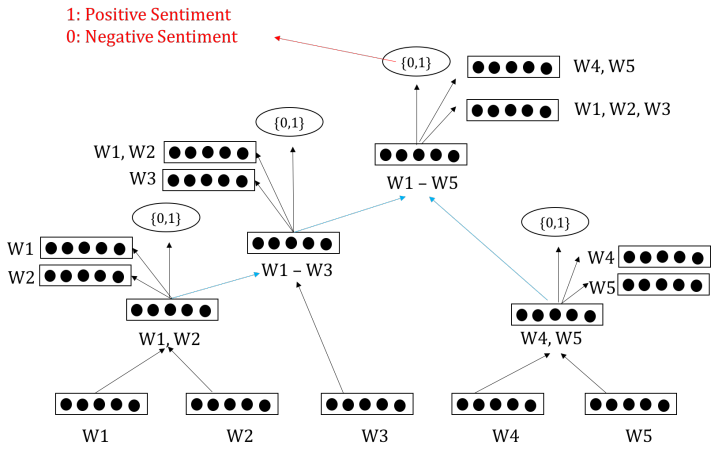

Fig. 10.8. Contoh recursive autoencoder dengan sentiment[35]

Cara lainnya adalah meng-*encode* kalimat sebagai *vektor* menggunakan *recursive autoencoder*. Kamu sudah belajar *recursive neural network*, sekarang kita modifikasi *recursive neural network* agar bisa digunakan untuk *encoding*. Kami sarankan kamu untuk mencari sumber bagaimana cara melatih *recursive neural network*. Penggunaan *recursive autoencoder* sangat rasional berhubung 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 Fig. 10.7. Pada setiap langkah *recursive*, *hidden layer/coding layer* berusaha men-*decode* atau merekonstruksi kembali vektor *input*. Anda dapat mencoba analisis sentimen menggunakan recursive autoencoder pada pranala[4].

Lebih jauh, untuk sentimen analisis pada kata, kita dapat menambahkan output pada setiap *hidden layer*, yaitu sentimen unit gabungan, seperti pada Fig. 10.8. 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 mengkonversi kalimat menjadi vektor, lalu bagaimana dengan paragraf, satu dokumen, atau satu frasa saja? Teknik umum untuk mengkonversi teks menjadi vektor dapat dibaca pada [37] yang lebih dikenal dengan nama ***paragraph vector*** atau ***doc2vec***.

[4] https://nlp.stanford.edu/sentiment/

10.7 Akhiran

Bab ini menyampaikan penggunaan *neural network* untuk melakukan *kompresi data* dengan teknik *semi-supervised* atau *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, kami 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 Vector Space Model?

10.3. Recurrent Autoencoder
(a) Jelaskanlah recurrent autoencoder!
(b) Apa persamaan dan perbedaan antara recurrent dan recursive autoencoder?
(c) Pada kasus apa kita lebih baik menggunakan recurrent atau recursive neural network?

11

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.

11.1 Sistem Rekomendasi

*Bagian sistem rekomendasi ditulis oleh **Candy Olivia Mawalim**. 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/penggunaan pada 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 mengerti kelakuan pengguna secara spesifik (mengerti kelakukan masing-masing pengguna). Teknik kedua berfokus pada selera terhadap suatu *item*. Teknik *machine learning*, berguna untuk mencari prediksi *item* yang disukai pengguna. Dua subbab (11.1.1 dan 11.1.2) berikutnya memuat pada bagian apa teknik *machine learning* dapat digunakan pada masing-masing teknik pembuatan sistem rekomendasi.

11.1.1 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 [40]. 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* adalam 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 (ekuvalen 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 10.5).

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 11.1). 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

Table 11.1. 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 11.2! 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

Table 11.2. Representasi *user embedding* berdasarkan *rating* yang diberikan pengguna.

Berdasarkan *item embedding* dan *user embedding* yang kita punya, kita ganti kolom "MovieId" pada Tabel 11.2 menggunakan baris pada *item embedding*. Sekarang, kita memiliki dataset *behavior* pengguna dengan UserId

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

= 1 (Tabel 11.3). 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

Table 11.3. Dataset *behavior* pengguna dengan UserId = 1

seperti pada Tabel 11.3, 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.

11.1.2 Collaborative Filtering

Teknik ini diperkenalkan oleh Paul Resnick dan Hal Varian pada 1997 [41]. Prinsip *collaborative filtering* adalah asumsi bahwa selera penggunaan terhadap suatu *item* cenderung sama dari waktu ke waktu [42]. Pada contoh kasus sederhana untuk sistem rekomendasi film, teknik ini memanfaatkan informasi *rating* dari banyak pengguna. Kita dapat merepresentasikan ***behavior*** seluruh user menggunakan matriks utilitas dimana baris merepresentasikan profil pengguna dan kolom merepresentasikan *item*. Sebagai contoh, perhatikanlah Tabel 11.4.

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

Table 11.4. 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* [43] (pada Tabel 11.4, pengguna yang mirip memiliki baris yang mirip). Selain itu, *item* yang memiliki kemiripan, akan memiliki pola *rating* yang mirip (pada Tabel 11.4, *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* [43]. Untuk suatu sistem dengan data yang *sparse*, matriks utilitas seperti Tabel 11.4 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 jawaban kamu pada soal-soal latihan pada bab 10).

11.2 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 [44, 45, 46]. 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*) [47] atau banyak dokumen menjadi satu ringkasan (*multi-document summarization*) [48].
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*) [49], 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) [44]. *query-based* artinya menghasilkan ringkasan dokumen berdasarkan *query* yang diberikan *user* (ringkasan adalah informasi yang dibutuhkan oleh *user* yang ditemukan di dokumen) [50]. *taioler*. *Taoilored* berarti menyajikan informasi dengan tipe spesifik. Misalkan pada berita terdapat informasi 5W1H, kita hanya ingin mencari tahu informasi *how*.
5. Ekstraktif [51] atau abstraktif [46]. 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 [44, 52, 53, 45, 54, 49, 48, 47, 46, 55]:

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 [56]. Kita pun dapat menggali hubungan antara unit informasi (misal direpresentasikan sebagai graf) untuk kemudian mencari unit mana yang sentral [49, 57].
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 11.2.1 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 11.2.2), artinya keseluruhan proses tersebut terjadi bersamaan.

11.2.1 Pipelined Approach

Subbab ini akan memuat cerita singkat tentang peringkasan *paper* [52, 55]. Berdasarkan teori *argumentative zoning* [52], *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* [58].

Ringkasan, dalam bentuk abstrak atau judul *paper* memiliki pola [59, 55]. 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 [58, 55]. Pertama-tama setiap kalimat direpresentasikan menjadi *feature vector* (ingat kembali materi bab 4 dan bab 5). Sebagai contoh, *paper* [55] 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.

11.2.2 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*) [60, 46, 61, 62, 63, 64]. 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 11.1.

$$\arg\max_{\mathbf{r}} P(\mathbf{r} \mid \mathbf{u}) \tag{11.1}$$

Pada umumnya, terdapat suatu variabel tambahan θ yang mengendalikan (*govern*) probabilitas tersebut. Sehingga secara lebih tepat, persamaan 11.1 diubah menjadi persamaan 11.2.

$$\arg\max_{\mathbf{r}} P(\mathbf{r} \mid \mathbf{u}, \theta) \tag{11.2}$$

Teknik *machine learning*, khususnya *sequence to sequence encoder–decoder* digunakan untuk mengaproksimasi persamaan 11.2. Diktat ini tidak akan membahas secara detil bagaimana aproksimasi tersebut dilakukan. Silahkan mempelajari lebih jauh bidang NLP untuk mengerti *sequence to sequence encoder–decoder*.

Soal Latihan

11.1. 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?

11.2. Eksplorasi Sistem Rekomendasi
Bangunlah suatu sistem rekomendasi dengan teknik *collaborative filtering* pada dataset MovieLens dengan memanfaatkan library *recommenderlab* [2]!

11.3. Peringkasan Dokumen
(a) Presentasikanlah di kelasmu, *paper* sistem peringkasan dokumen otomatis oleh Kupiec et al. (1995) [56] [3] yang menggunakan *pipelined approach*!
(b) Presentasikanlah di kelasmu, *paper* sistem peringkasan dokumen otomatis oleh Cheng and Lapata [64] [4] yang menggunakan *single-view approach*!
(c) Jelaskan perbedaan, kelebihan, dan kelemahan pendekatan-pendekatan sistem peringkasan dokumen!

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

References

1. Stuart Russel and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.
2. Jonathan Gratch and Stacell Marsella. Computationally modelling human emotion. *Communications of the ACM*, 57(12):71–99, 2014.
3. Masayu Leylia Khodra and Dessi Puji Lestari. Odd semester lecture on machine learning. Lecture of Institute Teknologi Bandung, 2015.
4. Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.
5. Sumio Watanabe and Hidetoshi Nishimori. Fall lecture note on statistical learning theory. Lecture note for Tokyo Institute of Technology, 2016.
6. Brian Caffo. *Statistical Inference for Data Science*. Lean Publishing, 2015.
7. Daniel Jurafsky and James H. Martin. *Speech and Language Processing Second Edition*. Prentice Hall, 2009.
8. Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley, 1991.
9. Hidetoshi Nishimori. *Statistical Physics of Spin Glasses and Information Processing: An Introduction*. Clarendon Press, 2001.
10. Sharon L. Myres Ronald E. Walpole, Raymond H. Myers and Keying Ya. *Probability and Statistics for Engineers and Scientists*. Prentice Hall, 2012.
11. Jeff Leek. *The Elements of Data Analytic Style*. Leanpub, 2015.
12. Takao Terano and Tsuyoshi Murata. Spring lecture on machine learning. Lecture of Tokyo Institute of Technology, 2017.
13. Tom Mitchell. *Machine Learning*. McGraw-Hill, 1997.
14. J. R. Quilan. *Discovering rules by induction from large collections of examples*. Edinburgh University Press, 1979.
15. J.R. Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106, March 1986.
16. C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.
17. Takao Terano and Tsuyoshi Murata. Spring lecture on machine learning. Lecture of Tokyo Institute of Technology, 2017.
18. Jack D. Cowan. Neural networks: The early days. In *Proceedings of Advances in Neural Information Processing Systems 2*, 1989.

19. Amir Atiya. *Learning Algorithms for Neural Network*. PhD thesis, California Institute of Technology, 1994.
20. Al. Cripps. Using artificial neural nets to predict academic performance. In *Proceedings of the 1996 ACM Symposium on Applied Computing*, pages 33–37, 1996.
21. Thomas Mikolov. *Statistical Language Models Based on Neural Networks*. PhD thesis, Brno University of Technology, 2012.
22. 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.
23. Gred Corrado Thomas Mikolov, Kai Chen and Jeffrey Dean. Efficient estimation of word representations in vector space. In *Proceedings of CoRR*, 2013.
24. Kai Yu. Large-scale deep learning at baidu. In *Proceedings of the 22^{nd} ACM International Conference on Information and Knowledge Management*, pages 2211–2212, 2013.
25. Jeffrey L. Elman. Learning and development in neural networks: The importance of starting small. *Journal of Cognition*, (48):71–99, 1993.
26. 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.
27. James Allen. *Natural Language Understanding*. Benjamin-Cummings Publishing Co., Inc., 1995.
28. Quoc V. Le Ilya Sutskever, Oriol Vinyals. Sequence to sequence learning with neural networks. In *Proceedings of the 27^{th} International Conference on Neural Information Processing Systems*, pages 3104–3112, 2014.
29. George Karypis Michael Steinbach and Vipin Kumar. A comparison of document clustering techniques. In *KDD Workshop on Text Mining*, pages 525 – 526, 2000.
30. 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.
31. 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.
32. Christopher D. Manning and Hinrich Schutze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.
33. Prabhakar Raghavan Christopher D. Manning and Hinrich Schutze. *An Introduction to Information Retrieval*. Cambridge UP, 2009.
34. 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 28^{th} International Conference on Machine Learning*, 2011.
35. 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.
36. 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.

37. Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31$_{st}$ International Conference on Machine Learning*, 2014.
38. 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.
39. Peter D. Turney and Patrick Pantel. From frequency to meaning: Vector space models of semantics. *Journal of Artificial Intelligence Research*, (37):141–188, 2010.
40. Daniel Billsus and Michael J. Pazzani. The adaptive web. chapter Adaptive News Access, pages 550–570. Springer-Verlag, Berlin, Heidelberg, 2007.
41. Paul Resnick and Hal R. Varian. Recommender systems. *Commun. ACM*, 40(3):56–58, March 1997.
42. Daniar Asanov. Algorithms and methods in recommender systems. Berlin Institute of Technology, 2011.
43. Charu C. Aggrawal. *Recommender Systems: The Textbook*. Springer International Publishing Switzerland, 2016.
44. 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.
45. 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.
46. 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.
47. 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.
48. 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.
49. 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.
50. Shufeng Xiong and Donghong Ji. Query-focused multi-document summarization using hypergraph-based ranking. *Inf. Process. Manage.*, 52(4):670–681, July 2016.
51. 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.

52. 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.
53. 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.
54. Jurij Leskovec, Natasa Milic-Frayling, and Marko Grobelnik. Extracting summary sentences based on the document semantic graph. *Microsoft Research*, 2005.
55. Jan Wira Gotama Putra. Rhetorical sentence classification for automatic title generation in scientific article. *TELKOMNIKA*, 15(2):656–664, 2017.
56. 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.
57. 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.
58. Simone Teufel and Marc Moens. Summarizing scientific articles: Experiments with relevance and rhetorical status. *Comput. Linguist.*, 28(4):409–445, December 2002.
59. 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.
60. 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.
61. 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.
62. 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.
63. 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.
64. Jianpeng Cheng and Mirella Lapata. Neural summarization by extracting sentences and words. *CoRR*, abs/1603.07252, 2016.