Jan Wira Gotama Putra

Pengenalan Konsep Pembelajaran Mesin

Not for dummies, because you are not!

Edisi 0.3

May 16, 2017

Untuk Tuhan, Bangsa, dan Almamater

Preface

Saya banyak mendengar baik dari teman, junior, senior, dll; tentang suatu kalimat "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 lah 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 *practical*. Barangkali lebih asyik membaca buku yang berjudul "Pembelajaran mesin menggunakan <*bahasa pemrograman*>". Sebagai informasi, mereka yang bekerja di tim Google Translate, IBM Watson, Tensor Flow, Deep Mind (*bukan promosi*) memiliki gelar PhD. Saya ingin mengajak saudara/i untuk memahami konsep *machine learning* secara dalam sebelum memanfaatkan.

Lecture note atau diktat ini ditujukan sebagai penunjang mata kuliah *machine learning* untuk mahasiswa tingkat sarjana di Indonesia. Seperti yang kita ketahui, kebanyakan buku perkuliahan berasal dari luar negeri. 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 itu, saya ingin berkontribusi pada pendidikan melalui *lecture note* ini. *Lecture note* ini adalah ringkasan catatan kuliah saya. Tentunya *lecture note* ini memiliki banyak kekurangan maka, kritik dan saran yang membangun sangat saya hargai. *Lecture note* ini tidak dapat dijadikan sebagai acuan utama karena hanya bersifat sebagai pengantar/pelengkap. Semoga *lecture note* ini dapat memberikan gambaran apa itu *machine learning*. Anggap saja membaca *lecture note* ini seperti sedang membaca novel. Dengan membaca *lecture note* ini, diharapkan kawan-kawan mengetahui harus belajar apa (lebih jauhnya) dalam bidang *machine learning*.

Pembaca disarankan sudah memahami/mengambil setidaknya mata kuliah statistika, kalkulus, aljabar linier/geometri, pengenalan kecerdasan buatan,

dan logika fuzzy. Hati-hati membedakan vektor dan variabel biasa. Saat membaca buku ini, disarankan membaca dari bagian pendahuluan. Gaya penulisan *lecture note* ini diusahakan **santai/semiformal**, dengan notasi matematis yang minimal, dan mudah-mudahan tanpa mengurangi esensi materi.

Pembaca dipersilahkan menyebar/mencetak buku ini untuk alasan **NON KOMERSIAL**, tetapi dimohon untuk tidak menyalin/meniru isi *lecture note*. Bila ingin memuat konten *lecture note* ini pada media yang pembaca kelola, dimohon untuk mengontak pengarang terlebih dahulu. Tidak semua istilah bahasa asing diterjemahkan ke Bahasa Indonesia agar esensinya tidak hilang (atau penulis tidak tahu versi Bahasa Indonesia yang baku). Karena *lecture note* ini bersifat draf, dimohon untuk **tidak mengutip**. Kami memuat soal latihan pada diktat ini, tujuan latihan tersebut adalah untuk mengarahkan apa yang harus dibaca/dipahami oleh para pembaca berikutnya.

Diktat ini dibuat menggunakan template monograph (LaTex) dari Springer. Formatting diatur secara omomatis 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.

Saya ingin mengucapkan terimakasih pada Bapak/Ibu/Saudara/i yang telah memberikan bantuan dalam bentuk apapun pada penulisan buku ini. Mohon maaf karena namanya tidak dapat disebut satu per satu.

Tokyo, Jepang *Jan Wira Gotama Putra*

gotama.w.aa@m.titech.ac.jp
https://icemerly.wordpress.com
http://www.cl.cs.titech.ac.jp/en/take/index

Contents

Part I

Pengetahuan Dasar

1

Statistical Learning Theory

> "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 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 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.

1.2 Intelligent Agent

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

[1] KBBI Web, Accessed on 10 October 2016

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

- ***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.

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

- ***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 tidak dapat memodelkan cara berpikir/tingkah laku manusia ke dalam bentuk deterministik. Dengan hal itu, saat ini kita hanya mempu mengoptimalkan agen yang mempunyai dimensi *acting rationally* dan *thinking rationally*.

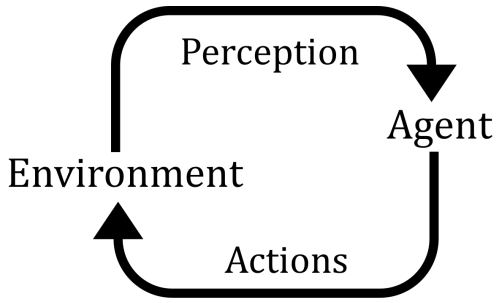

Fig. 1.1. Agent vs environment [3]

Perhatikan Fig. 1.1! Agen mengumpulkan informasi dari lingkungannya, kemudian memberikan respon berupa aksi. 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 matema-

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

1.3 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*); dan/atau **memperoleh ilmu pengetahuan** atau *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.

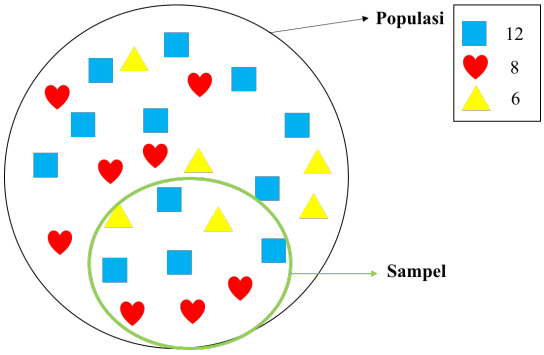

Fig. 1.2. Ilustrasi makanan pesta 1

Perhatikan Fig. 1.2 (permasalahan yang disederhanakan). Misalkan kamu diundang ke suatu pesta. Pada pesta tersebut ada 3 jenis kue yang dis-

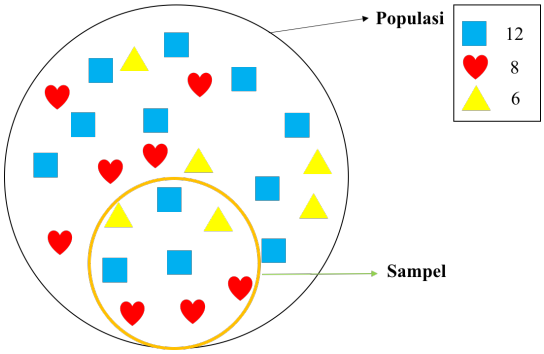

Fig. 1.3. Ilustrasi makanan pesta 2

ajikan. Kamu ingin mengetahui berapa rasio kue yang disajikan dibandingkan masing-masing jenisnya (seluruh populasi). Tapi kamu terlalu malas untuk menghitung semua kue yang ada. Karena itu, 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.

1.4 Supervised Learning

Jika diterjemahkan secara literal, *supervised learning* adalah pembalajaran terarah. Artinya, pada pembelajaran ini, ada guru yang mengajar 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).

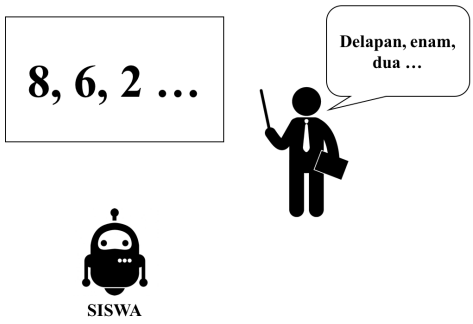

Fig. 1.4. Supervised Learning

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* $p(y|x, \vec{\mathbf{w}})$, dimana y adalah target (*output*), x adalah input dan vektor $\vec{\mathbf{w}}$ adalah ***learning parameter***. Proses belajar ini, yaitu mengoptimalkan $\vec{\mathbf{w}}$ disebut sebagai training. Semakin kamu membaca *lecture note* ini, konsep ini akan menjadi semakin jelas.

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

Selain *supervised learning*, masih ada metode pembelajaran lainnya yaitu *unsupervised learning*, *semi-supervised learning*, dan *reinforcement learning*.

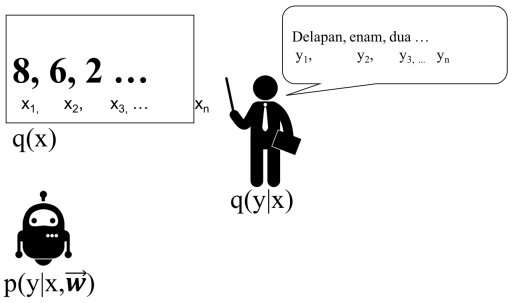

Fig. 1.5. Supervised learning - mathematical explanation

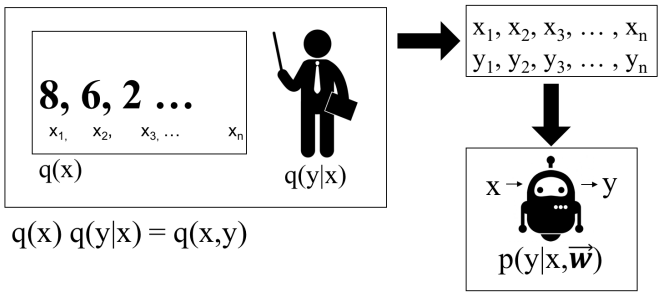

Fig. 1.6. Supervised learning - mathematical explanation 2

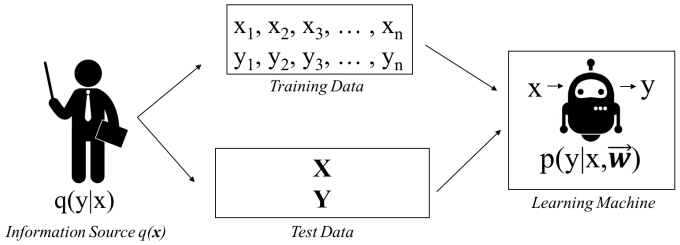

Fig. 1.7. Supervised learning framework

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*.

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

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

1.5 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***.

1.6 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*.

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 $p(x|\overrightarrow{\mathbf{w}})$ yang mengoptimasi parameter $\overrightarrow{\mathbf{w}}$. Perbedaan antara estimasi dan fungsi asli disebut sebagai ***generalization loss***.

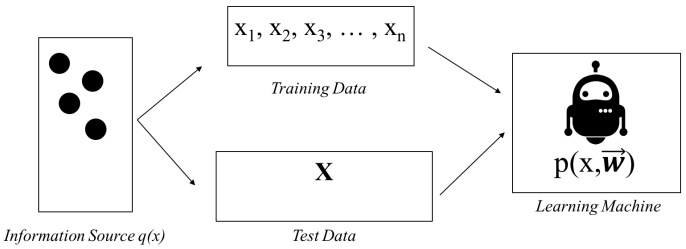

Fig. 1.8. Unsupervised learning framework

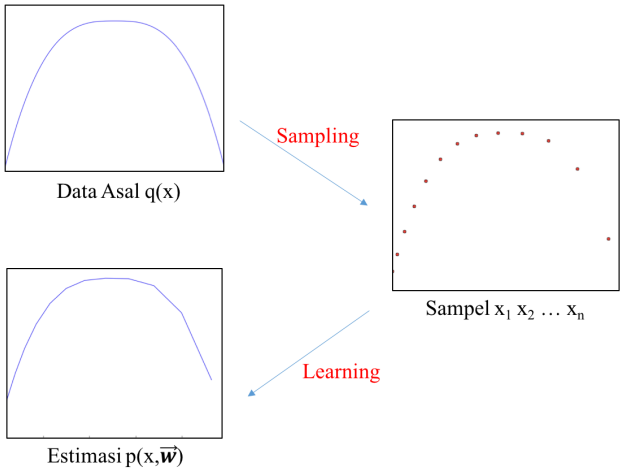

Fig. 1.9. Generalization error of unsupervised learning

$$p(x|\vec{\mathbf{w}}) \tag{1.2}$$

Kunci pemahaman *unsupervised learning* adalah mengingat persamaan 1.2. Ada dua hal penting yaitu: *input* dan *learning parameters*.

1.7 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 data asli $q(y|x)$. Dari distribusi tersebut, kita diberikan beberapa sampel pasangan *input-output* $\{z_1, z_2, z_3, ..., z_n\}; z = (x, y)$. Kita membuat learning machine $p(y|x, \overrightarrow{\mathbf{w}})$. Awalnya disodorkan 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. Semakin lama, kita jadi makin percaya bahwa *learning machine* semakin optimal (mampu memprediksi fungsi aslinya).

1.8 Tipe Permasalahan di Dunia

Ada dua tipe permasalahan *(debateable)*, 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.

1.9 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:

- Matematika Diskrit dan Teori Bilangan
- Aljabar Linier dan Geometri (vektor, matriks, skalar, decomposition, transformasi, tensor, dsb)
- Calculus (diferensial dan integral)
- Optimasi (*Lagrange Multiplier, Convex, Gradient Descent, Integer Linear Problem,* dsb)
- Probabilitas dan Statistika (Probabilitas, Probability Densities, Hypothesis Testing, Inter-rater agreement, Bayesian, Statistical Mechanics)
- Teori Fuzzy

1.10 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). Berikut adalah beberapa artikel berita menarik:

- https://techcrunch.com/2016/03/15/google-ai-beats-go-world-champion-again-to-complete-historic-4-1-series-victory/
- http://www-formal.stanford.edu/jmc/whatisai/node3.html
- https://www.google.com/selfdrivingcar/
- 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. Klasifikasi
Jelaskan *multi-level classification*!

2

Mathematical Foundation

> "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. Hal ini 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 $P(x = X, y = Y)$. Apabila x dan y independen (tidak bergantung satu sama lain), maka $P(x = X, y = Y) = P(X) * P(Y)$. Dalam kasus ini, kejadiannya adalah saling lepas, artinya mengambil bola dari kotak biru, pada saat yang bersamaan tidak akan mempengaruhi hasil pengambilan kotak merah. Sebaliknya, apabila X tidak saling lepas 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.

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.

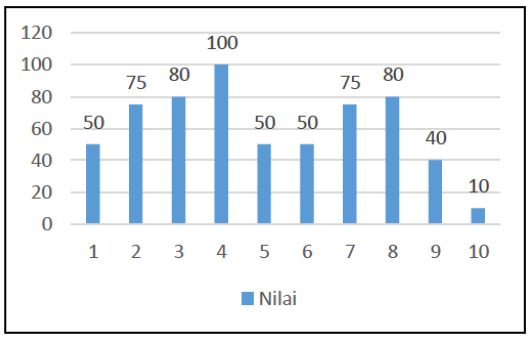

Fig. 2.1. Nilai siswa

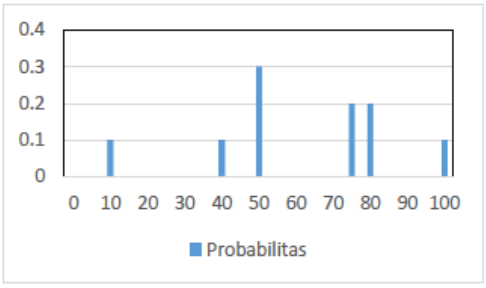

Fig. 2.2. Persebaran probabilitas nilai siswa

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 Expectations dan Variance

Salah satu operasi paling penting dalam probabilitas adalah menemukan nilai rata-rata terbobot (*weighted average*) sebuah fungsi [4]. Hal ini disebut menghitung ekspektasi (*expectation*). Untuk sebuah fungsi $f(x)$ dengan distribusi probabilitas *random variable* adalah $p(x)$, nilai expectation diberikan pada persamaan 2.4.

$$E(f) = \begin{cases} \sum_x p(x)f(x); & diskrit \\ \int p(x)f(x)dx; & kontinu \end{cases} \tag{2.4}$$

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

$$E(f) \simeq \frac{1}{N} \sum_{i=1}^{N} f(x_i) \tag{2.5}$$

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.

$$var[f] = E[f(x) - E[f(x)]^2]) \tag{2.6}$$

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 tiap kombinasi variabel.

2.4 Bayesian Probability

Dalam subbab sebelumnya, kita menghitung probabilitas dengan frekuensi kejadian yang dapat diulang. Pada pandangan Bayesian, kita ingin menguantifikasi ketidakpastian. 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.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{2.7}$$

$P(A)$ 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*).

$$posterior \propto likelihood * prior \tag{2.8}$$

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 kemudian).

Perhatikan kembali persamaan 2.7, 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.

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 2.9

$$N(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{2.9}$$

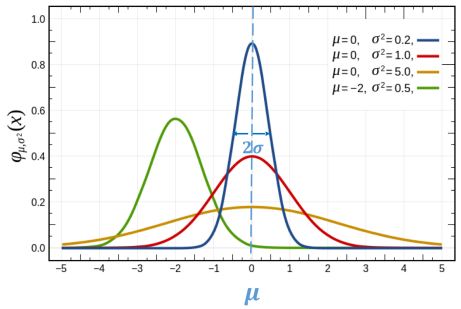

Fig. 2.3. Univariate Gaussian [1]

Perhatikan Fig. 2.3, nilai N (persamaan 2.9) 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 2.10.

$$p(x|\mu, \sigma^2) = \prod_{i=1}^{N} N(x|\mu, \sigma^2) \tag{2.10}$$

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

[1] source: wikimedia.org

logaritma sebanding dengan meminimalkan *error*, hal ini diberikan pada persamaan 2.11.

$$ln(p(x|\mu,\sigma^2)) = -\frac{1}{2\sigma^2}\sum_{i=1}^{N}(x_i-\mu)^2 - \frac{N}{2}ln(\sigma^2) - \frac{N}{2}ln(2\pi) \tag{2.11}$$

solusi 2.11 diberikan pada 2.12.

$$\mu = \frac{1}{N}\sum_{i=1}^{N}x_i; \sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i-\mu)^2 \tag{2.12}$$

Perhatikan baik-baik interpretasi berikut! Artinya kita dapat **mengestimasi distribusi asli menggunakan sampel data yang kita miliki**. Mean distribusi asli diestimasi dengan mean sampel. *Variance* distribusi asli 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. 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 asli berwarna merah, sedangkan GMM berwarna biru.

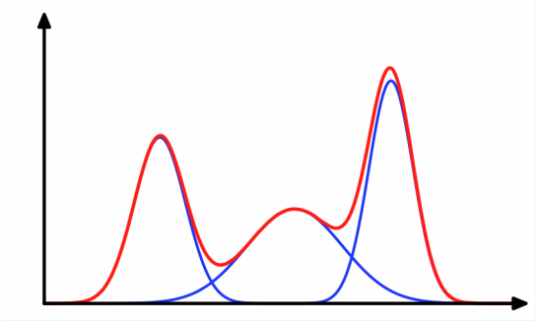

Fig. 2.4. Gaussian Mixture Model [2]

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

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 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 2.13 dan persamaan 2.14.

$$G = \int q(x) log(p(x)) dx \tag{2.13}$$

$$T = \frac{1}{N} \sum_{i=1}^{N} log(p(x)) \tag{2.14}$$

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 asli $q(x)$, misalnya **maximimum 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*** 2.15.

$$S(x) = -\sum_{i=1}^{N} p(x_i) log(p(x_i)) \tag{2.15}$$

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

$$S = -8\frac{1}{8} log(\frac{1}{8}) = 3 \tag{2.16}$$

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 tidak uniform, memiliki *entropy* yang lebih besar dibanding distribusi yang 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 Divergence** (KL-divergence). Secara konseptual, dirumuskan sebagai persamaan 2.17.

$$KL(q||p) = -\int q(x) log\left(\frac{q(x)}{p(x)}\right) dx \tag{2.17}$$

Persamaan tersebut dapat diminimalkan jika dan hanya jika $q(x) = p(x)$. Kita dapat menganggap KL-divergence sebagai ukuran seberapa jauh aproksimasi dan distribusi asli. 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

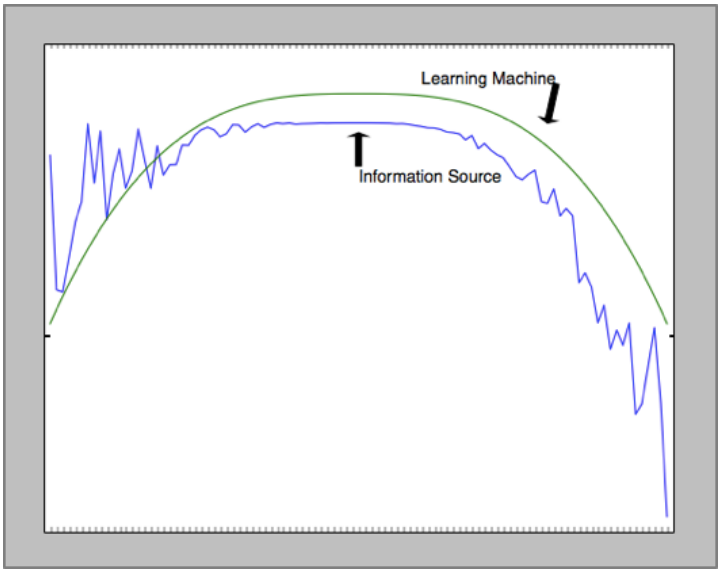

Fig. 2.5. Information source vs learning machine

learning machine $p(x|\vec{\mathbf{w}})$. Ekspektasi terhadap $q(x)$ dapat diaproksimasi dengan menggunakan data sampel ini, sehingga menjadi persamaan 2.18 [4].

$$KL(q||p) \approx \frac{1}{N}\sum_{i=1}^{N}\left(-log(p(x_i|\vec{\mathbf{w}}) + log(q(x_i))\right) \tag{2.18}$$

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 hubun-

gan 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 $f(x)$!

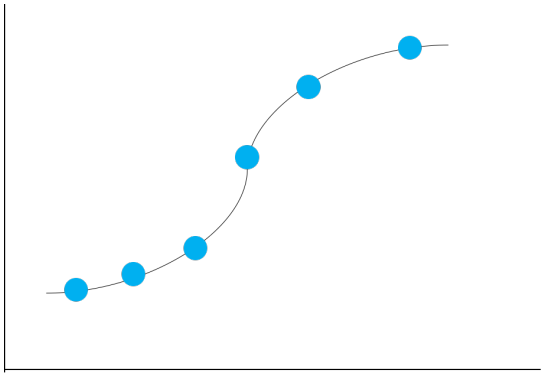

Fig. 3.1. Contoh fungsi Sigmoid

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*.

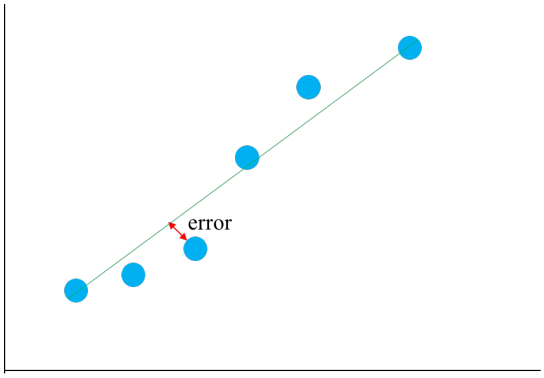

Fig. 3.2. Pendekatan fungsi sigmoid

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.

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

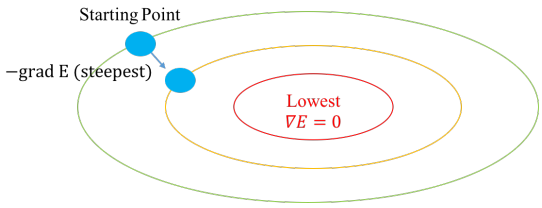

Fig. 3.3. Steepest Gradient Descent

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

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.

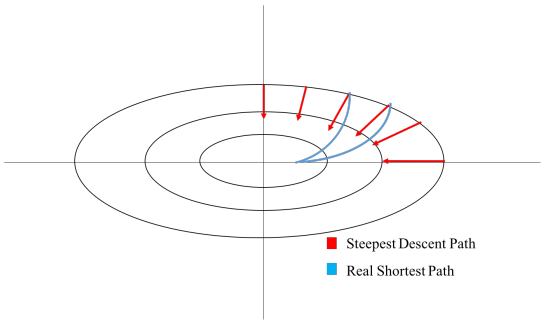

Fig. 3.4. Steepest Gradient Descent 2

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.

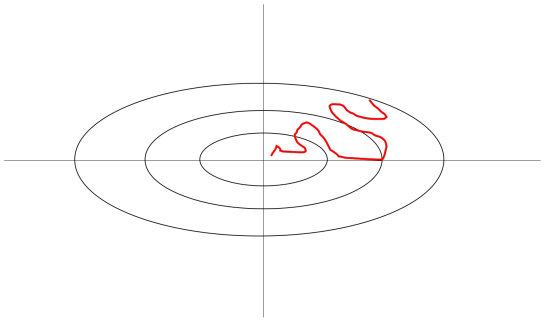

Fig. 3.5. Swing

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.

- http://users.ics.aalto.fi/jhollmen/dippa/node22.html
- 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 konsep *gradient descent* selain *steepest gradient descent*, misalnya *stochastic gradient descent*!

Part II

Algoritma Pembelajaran Mesin

4

Algoritma Dasar

> "It is a capital mistake to
> theorize before one has data."
>
> Arthur Conan Doyle

Sebelum masuk ke algoritma *machine learning* yang cukup modern, 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.

4.1 Data

Pada bab 1, kamu telah mengetahui bahwa tujuan *machine learning* adalah menginferensi informasi berdasarkan data. Inferensi tersebut dapat berupa aturan atau hal lainnya. 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** tertentu dan **range** tertentu. Sebagai contoh, fitur *outlook* memiliki tipe data **nominal** yaitu nilainya tersusun oleh suatu **himpunan** terbatas = {*sunny, overcast, rainy*}. *sunny* disebut salah satu atribut (salah satu kemungkinan nilai) untuk fitur *outlook*.

Tipe data lainnya adalah **numerik** yaitu memiliki range nilai **bilangan riil**. Seumpama kita ingin memprediksi, diberikan suatu informasi keadaan variabel, apakah seorang anak ingin bermain tenis atau tidak. Variabel yang kita prediksi *play* disebut **kelas/*class***. Setiap baris pada tabel disebut

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* [3]

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

Table 4.2. Contoh dataset *linearly separable*

instance, contohnya *instance* dengan id_4 adalah {*outlook=rainy, temperature=mild, humidity=high, windy=false, play=yes*}. Untuk setiap baris, 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*}. Pada *supervsied learning* kita memprediksi kelas berdasarkan *feature vector* ini. *feature vector* bisa diibaratkan sifat-sifat atau keadaan yang diasosiasikan dengan kelas. Pada *supervised learning* untuk setiap *feature vector* terdapat kelas yang berkorespondisi. Di lain pihak, pada *unsupervised learning* tidak ada kelas yang berkorespondensi, melainkan kita harus menemukan "kelas". Karena kelas pada Table. 4.1 hanya ada dua, klasifikasi data ini disebut ***binary classification***. Apabila kelas klasifikasi lebih dari dua, disebut ***multi-class classification/multi-label classification***. Mohon bedakan antara ***multi-class classification*** dan ***multi-level/hierarchical classification***. Pada *multi-level/hierarchical classification*, pertama-tama kita melakukan klasifikasi untuk suatu kelas generik, lalu dilanjutkan mengklasifikan data ke kelas yang lebih spesifik. Contoh *multi-level classification* adalah kingdom (biologi), pertama diklasifikasikan ke kingdom animalia, lalu lebih spesifiknya ke phylum Vertebrata, dst. *Multi-class/multi-label classification* hanya tentang ada banyak "kelas" tanpa tinjauan hirarkis.

Sekarang kamu diberikan data lain, seperti pada Table. 4.2. Data pada tabel tersebut kita sebut ***linearly separable***. Untuk suatu nilai variabel tertentu, iya 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, maka memisahkan kelas satu dan kelas lainnya dapat diperoleh dengan cara menciptakan garis linier (*linear line*) atau bidang datar. Ilustrasi dapat dilihat pada 4.1.

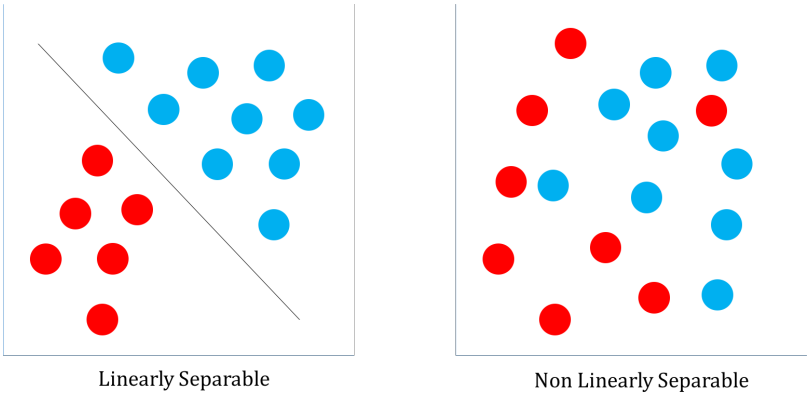

Fig. 4.1. Linearly vs Non Linearly Separable

Untuk merubah data *non-linearly separable*, kita dapat menggunakan teknik transformasi data seperti *radial basis function*. Silahkan membaca lebih lanjut pada literatur lain.

4.2 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 4.1 dimana c_i adalah suatu kelas, C adalah kelas (*variabel kelas*), t adalah fitur dan F adalah banyaknya fitur. Kita memprediksi kelas berdasarkan probabilitas kemunculan atribut pada kelas tersebut.

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

Mari kita bangun model Naive Bayes untuk Table. 4.1. Dengan hal ini, kita sebut Table. 4.1 sebagai ***training data***. Untuk menghitung probabili-

	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 4.3. 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 4.4. Frekuensi setiap nilai atribut

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

Table 4.5. Contoh testing data *play tennis* [3]

tas, pertama-tama kita hitung terlebih dahulu frekuensi nilai atribut seperti pada Table. 4.3, setelah itu kita bangun model probabilitasnya seperti pada Table. 4.4.

Untuk menguji kebenaran model yang telah kita bangun, kita menggunakan ***testing data***, diberikan pada Table. 4.5. *testing data* berisi *unseen example* yaitu contoh yang tidak ada pada *training data*.

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

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).

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 4.6. Contoh dataset orang kaya

id	$centroid_1$	$centroid_2$	assignment
2	2	2	k_1
3	1	3	k_1
4	3	1	k_2
5	2	2	k_1

Table 4.7. Assignment K-means Langkah 1

4.3 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:

1. Tentukan jumlah kelompok yang kita inginkan.
2. Inisiasi **centroid** untuk setiap kelompok 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. 4.6, 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

id	$centroid_1$	$centroid_2$	assignment
1	2	3	k_1
3	1	3	k_1
5	3	1	k_2
6	2	2	k_1

Table 4.8. Assignment K-Means Langkah 2

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. 4.8. Hasil langkah ke-2 adalah perubahan anggota kelompok, $k_1 = \{id_1, id_2, id_3, id_5\}$ dan $k_2 = \{id_4, id_6\}$. Anggap pada langkah ke-3 kita memilih kembali id_2 dan id_4 sebagai centroid masing-masing kelompok sehingga tidak ada perubahan keanggotaan.

Bila kamu membaca buku literatur lain, kemungkinan besar akan dijelaskan bahwa *clustering* itu memiliki **hubungan erat dengan gaussian mixture model**. Secara sederhana, **satu *cluster* (atau satu kelas)** sebenarnya seolah-olah dapat dipisahkan dengan kelas lainnya oleh distribusi gaussian. Perhatikan Fig. 4.2! Suatu *cluster* atau kelas, seolah olah "dibungkus" oleh suatu distribusi gaussian. Distribusi seluruh dataset dapat diaproksimasi dengan *gaussian mixture model*.

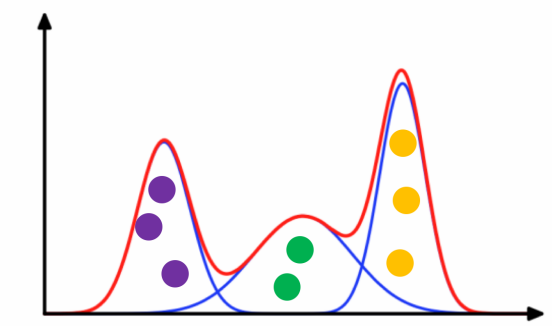

Fig. 4.2. Ilustrasi Hubungan Clustering, Kelas, dan Gaussian

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

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

Table 4.9. Perbedaan data baru vs data orang kaya

hubungan yang sangat erat dengan distribusi gaussian karena pola tersebut dapat diaproksimasi dengan distribusi gaussian.

4.4 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 4.3, 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. 4.9. *feature vector* termirip dimiliki oleh data dengan id_1, id_5, id_6 seperti diilustrasikan pada 4.9. 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

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

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

4.3. K-means
(a) Cari tahu cara lain untuk memilih **centroid** pada algoritma K-means (selain cara acak) baik untuk data nominal dan numerik!
(b) Cari tahu cara lain untuk menghitung kedekekatan suatu data dengan **centroid** baik untuk data nominal dan numerik! Hint: *euclidian distance*, *manhattan distance*, *cosine similarity*.

4.4. Konversi atribut
Sebutkan dan jelaskan macam-macam fungsi konversi atribut! Baik numerik-nominal dan nominal-numerik.

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

5

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*.

5.1 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*) [11, 12, 13, 14]. Kami yakin banyak yang sudah tidak asing dengan istilah ini, berhubung *deep learning* sedang populer dan banyak yang membicarakannya (dan digunakan sebagai trik pemasaran). Silahkan belajar biologi untuk lebih mengerti tentang saraf manusia. *Artificial neural network* adalah salah satu algoritma *supervised learning* yang

populer dan bisa juga digunakan untuk *semi-supervised* atau *unsupervised learning* [12, 14, 26, 27, 15].

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 upabab berikutnya. Sebagai gambaran, ANN berbentuk seperti Fig. 5.1.

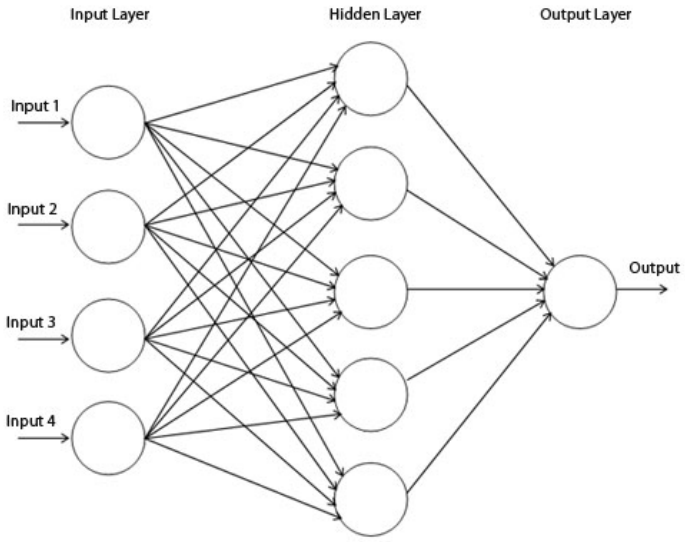

Fig. 5.1. Multilayer Perceptron

5.2 Single Perceptron

Bentuk terkecil (minimal) sebuah ANN adalah *single perceptron* yang hanya terdiri dari sebuah neuron. Sebuah neuron diilustrasikan pada Fig. 5.2. Secara matematis, terdapat *feature vector* $\vec{\mathbf{x}}$ yang menjadi *input* bagi neuron tersebut. Neuron akan memproses *input* $\vec{\mathbf{x}}$ 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 5.1. Bila di-*plot* menjadi grafik, fungsi ini memberikan bentuk seperti huruf S.

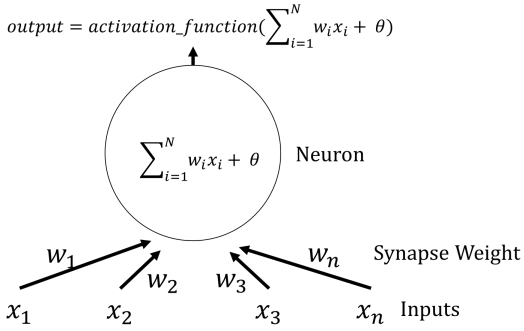

Fig. 5.2. Single Perceptron

$$\sigma(u) = \frac{1}{1 + e^{-u}} \tag{5.1}$$

Untuk melakukan pembelajaran *single perceptron*, *training* dilakukan berdasarkan ***perceptron training rule***. Prosesnya sebagai berikut [16]:

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 ***feedfoward***.
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 5.2 dan perubahan *synapse weight* diberikan pada persamaan 5.3. y melambangkan *desired output*, o melambangkan *actual output*. η disebut sebagai *learning rate*. Pada kasus *single perceptron* nilai K adalah 1.

$$E(\overrightarrow{\mathbf{w}}) = \frac{1}{2}\sum_{k=1}^{K}(y_k - o_k)^2; K = \#output\ neurons \tag{5.2}$$

$$\Delta w_i = \eta(y - o)x_i \tag{5.3}$$

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.

5.3 Multilayer Perceptron

Kamu sudah belajar *training* untuk *single perceptron*. Selanjutnya kita akan mempelajari *multilayer perceptron* (MLP). Seperti ilustrasi pada Fig. 5.3, *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* [12, 16]. Hal serupa berlaku untuk *artificial neural network* dengan lebih dari 3 *layers*.

$$o_j = \sigma(\sum_{k=1}^{M} w_{jk}x_k + \theta_j) \tag{5.4}$$

$$f_i = \sigma(\sum_{j=1}^{H} u_{ij}o_j + \gamma_i) = \sigma(\sum_{j=1}^{H} u_{ij}\sigma(\sum_{k=1}^{M} w_{jk}x_k + \theta_j) + \gamma_i) \tag{5.5}$$

Perhatikan persamaan 5.4 dan 5.5 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*

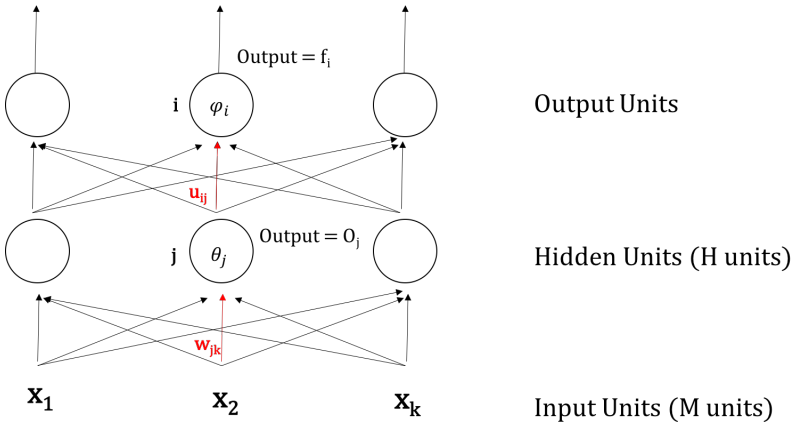

Fig. 5.3. Multilayer Perceptron 2

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* pada ANN diberikan oleh persamaan 5.2.

$$E(\overrightarrow{\mathbf{w}}) = \frac{1}{2}\sum_{k=1}^{K}(y_k - o_k)^2 \tag{5.6}$$

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. 5.3 dapat diberikan seperti pada Fig. 5.4. 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. 5.4. Proses latihan MLP menggunakan *backpropagation*

5.3.1 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 1.

5.3.2 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 [26, 27].

5.4 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*.

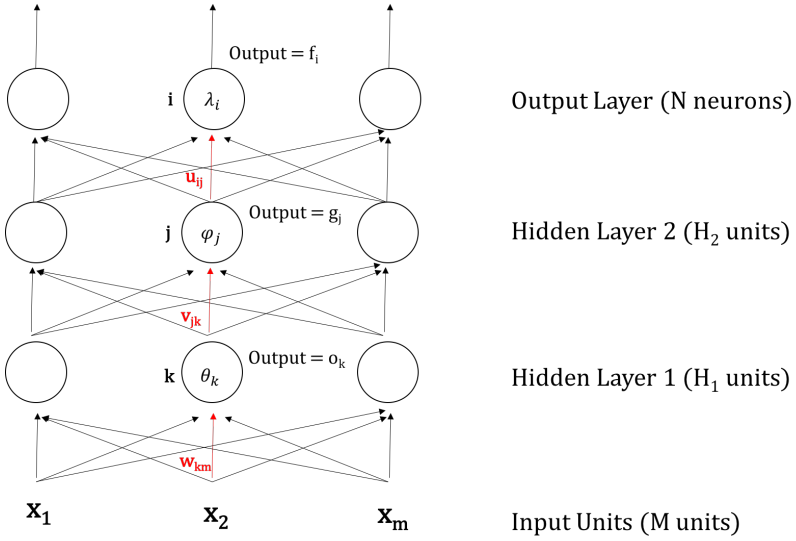

Fig. 5.5. Deep Neural Network

Perhatikan Fig. 5.5 yang memiliki 4 *layers*. Cara menghitung *final output* sama seperti MLP, diberikan pada persamaan 5.7 dimana θ, φ, λ adalah *noise* atau *bias*.

$$f_i = \sigma(\sum_{j=1}^{H_2} u_{ij}\sigma(\sum_{k=1}^{H_1} v_{jk}\sigma(\sum_{m=1}^{M} w_{km}x_m + \theta_k) + \varphi_j) + \lambda_i) \tag{5.7}$$

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 5.6.

(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. 5.6. Proses latihan DNN menggunakan *backpropagation*

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" antara *input* dan *output*, yang tidak dapat diselesaikan menggunakan *multilayer 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 [17], yaitu mulai dari hal kecil. Ilustrasinya dapat dilihat pada Fig. 5.7.

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

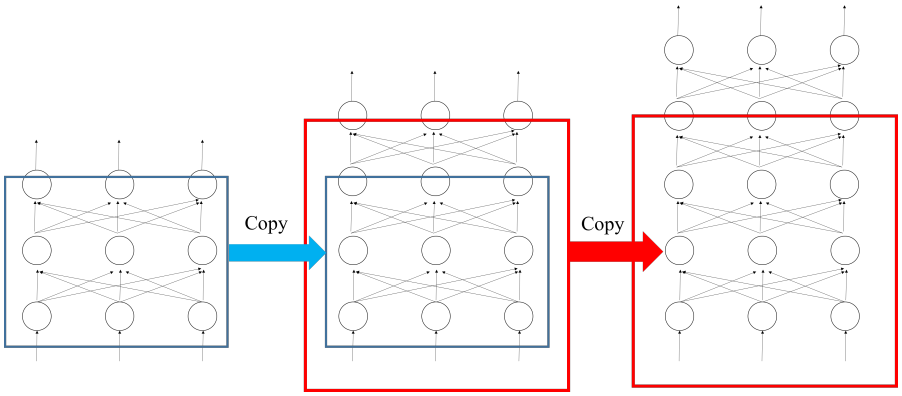

Fig. 5.7. Contoh Succesive Learning

5.5 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) [14]. Ilustrasi *recurrent neural network* dapat lihat pada Fig. 5.8. Perhitungan fungsi aktivasi pada *hidden layer* dipengaruhi oleh *state hidden layer* pada iterasi sebelumnya. Hal ini sesuai dengan konsep *recurrent* yaitu "mengingat" kejadian sebelumnya.

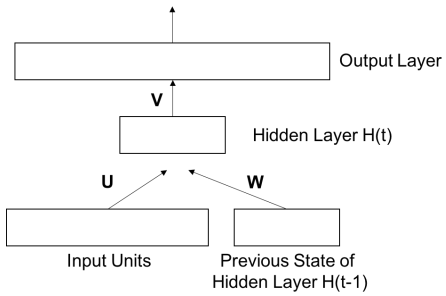

Fig. 5.8. Konsep Recurrent Neural Network

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*. Awalnya kita melakukan *feedfoward* 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 5.9. 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. 5.10.

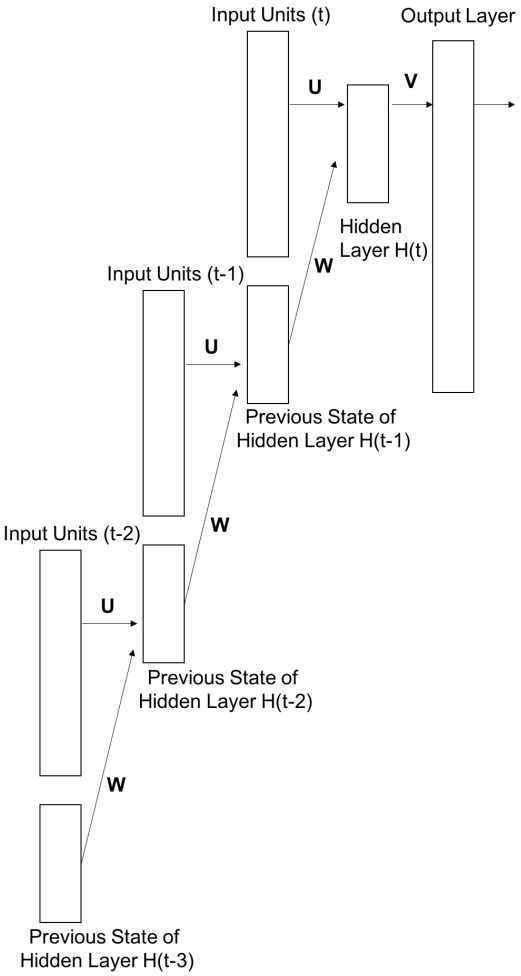

Fig. 5.9. Konsep Backpropagation Through Time [14]

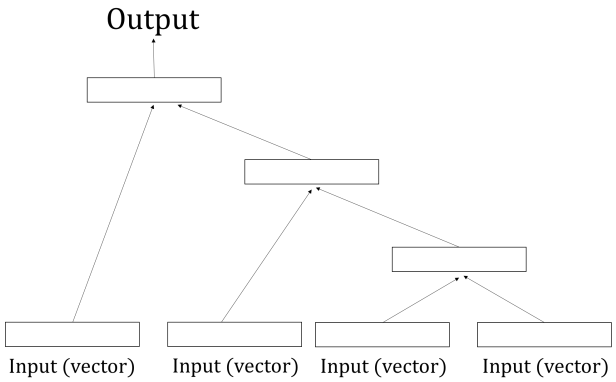

Fig. 5.10. Recurrent Neural Network

5.6 Recursive Neural Network

Seperti namanya, **recursive neural network** memiliki struktur rekursif (contoh: struktur data pohon). Ilustrasi dapat dilihat pada fig 5.11. Bila 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 6. Yang perlu diingat adalah, struktur *recursive neural network* melambangkan hirarki.

5.7 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 *overtraining*. 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 6. 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.

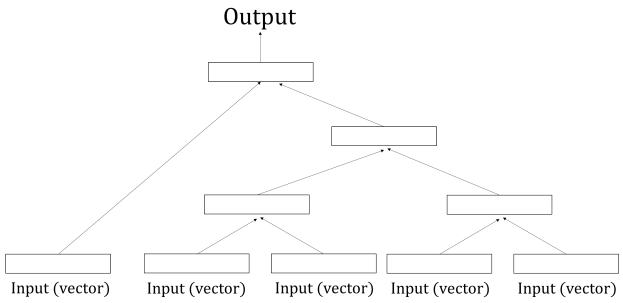

Fig. 5.11. Recursive Neural Network

Soal Latihan

5.1. Turunan

(a) Turunkanlah perubahan *noise/bias* untuk *training* pada MLP.
(b) Turunkanlah proses *training* DNN pada Fig. 5.6 termasuk perubahan *noise/bias*.

5.2. 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?

5.3. 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*.

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

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

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

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

5.8. 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?

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

6

Autoencoder

> "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 ***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 [18].

6.1 Curse of Dimensionality

Curse of dimensionality dapat dipahami secara mendalam apabila kamu membaca buku [19]. 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** [20, 21, 22, 23, 24]. Ilustrasi autoencoder dapat dilihat pada Fig. 6.1. 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%.

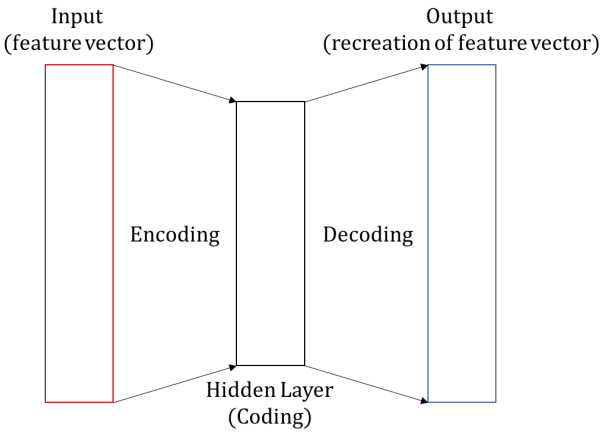

Fig. 6.1. Contoh autoencoder sederhana

6.2 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.

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

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 [25]. 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 [18]. 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*.

6.3 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 encoding*** dimana V adalah ukuran kosa kata. Ilustrasi dapat dilihat pada Fig. 6.2.

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 6.1. Tingkat kepentingan kata berbanding terbalik dengan

	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. 6.2. Contoh 1-of-V encoding

jumlah dokumen dimana kata tersebut dimuat. N adalah banyaknya dokumen, $\|d \epsilon D; t \epsilon d\|$ adalah banyaknya dokumen dimana kata t muncul.

$$IDF(t, D) = log\left(\frac{N}{\|d \epsilon D; t \epsilon d\|}\right) \tag{6.1}$$

Dengan menggunakan perhitungan TF-IDF yaitu $TF * IDF$ untuk mengisi sel pada matriks 6.2, 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 [25]:

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 [25].

6.4 Time Series dan Compositionality

Bahasa manusia memiliki dua macam karakteristik yaitu adalah data berbentuk ***time series*** dan memenuhi sifat ***compositionality***. Data *time series* 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 untuk data yang memiliki sifat *time series* adalah gelombang laut, angin, ucapan manusia, dan cuaca. Kita mampu memprediksi cuaca berdasarkan rekaman parameter cuaca pada hari-hari sebelumnya.

Data yang memenuhi sifat *compositionality* berarti memiliki struktur hirarkis. Struktur hirarkis ini menggambarkan bagaimana unit-unit lebih kecil berinteraksi sebagai satu kesatuan. 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. 6.3. *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.

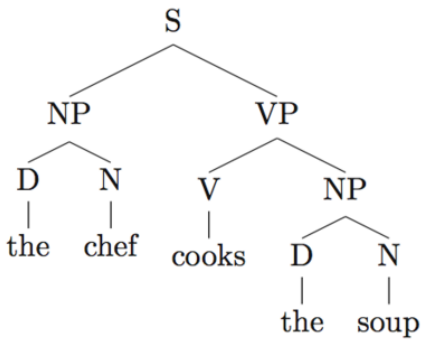

Fig. 6.3. Contoh Constituent Tree[2]

6.5 Distributed Word Representation

Seperti yang disebutkan pada bagian sebelumnya, kita ingin hubungan kata dapat direpresentasikan sebagai operasi vektor seperti pada ilustrasi Fig. 6.4. Kata "raja" memiliki sifat-sifat yang dilambangkan oleh suatu vektor (misal 90% aspek loyalitas, 80% kebijaksanaan, 90% aspek kebangsaan, dst), begitu pula dengan kata "pria", "wanita", dan "ratu". Jika sifat-sifat yang dimiliki "raja" dihilangkan bagian sifat-sifat "pria"-nya, kemudian ditambahkan sifat-sifat "wanita" maka idealnya operasi ini menghasilkan vektor yang dekat

[2] source: Pinterest

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.

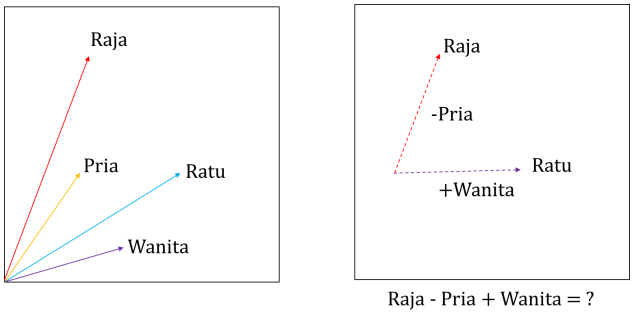

Fig. 6.4. Contoh Operasi Vektor Kata

Selain Vector Space Model, apakah ada cara lain yang mampu merepresentasikan kata dengan lebih baik? Salah satu kekurangan VSM adalah tidak memadukan sifat *time series* pada konstruksi vektornya. Cara lebih baik ditemukan oleh [26, 27] dengan ekstensi pada [24]. 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. 6.5. Kedua arsitektur ini sangat erat dengan prinsip autoencoder, bagian *projection layer* pada Fig. 6.5 adalah *coding* layer. Kami akan memberikan contoh CBOW secara lebih detil.

Perhatikan Fig. 6.6. 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 *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.

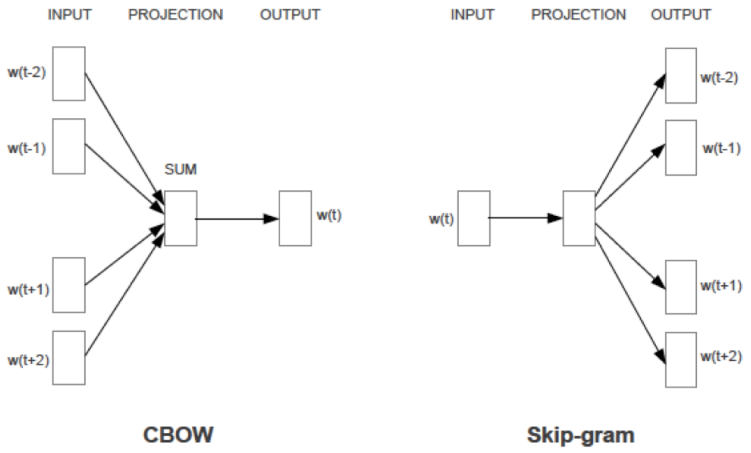

Fig. 6.5. CBOW vs Skip Gram [27]

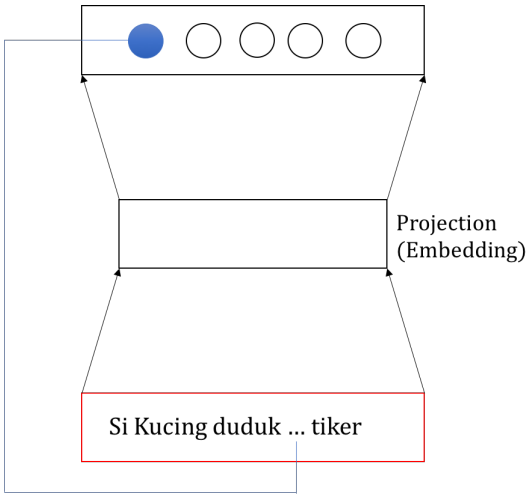

Fig. 6.6. CBOW

Agar lebih tahu kegunaan vektor kata, kamu dapat menggunakan mencoba-coba kode pada pranala [3] dengan bahasa pemrograman Python 2.7.

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

6.6 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 *time series* tidak terpenuhi. Selain itu, kalimat juga memiliki sifat *compositionality*.

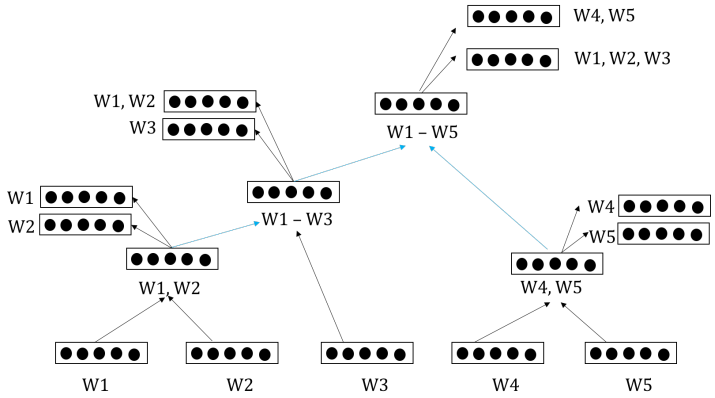

Fig. 6.7. Contoh recursive autoencoder

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. 6.7. 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

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

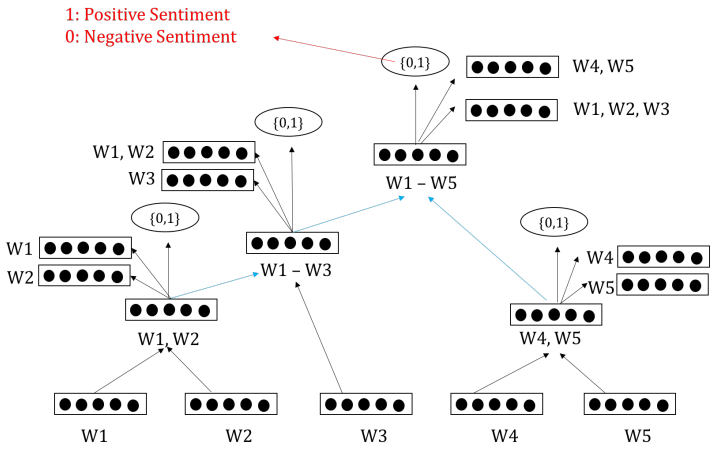

Fig. 6.8. Contoh recursive autoencoder dengan sentiment[21]

Fig. 6.8. 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 [23] yang lebih dikenal dengan nama ***paragraph vector*** atau ***doc2vec***.

6.7 Kesimpulan

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

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

6.2. LSI dan LDA
(a) Jelaskanlah Latent Semantic Indexing (LSI) dan Latent Dirichlet Allocation (LDA)!
(b) Apa persamaan dan perbeadan antara LSI, LDA, dan Vector Space Model?

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

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 note on machine learning. Lecture note for Institute Teknologi Bandung, 2015.
4. Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.
5. Sumio Watanabe. Fall lecture note on statistical learning theory part i. 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. Jack D. Cowan. Neural networks: The early days. In *Proceedings of Advances in Neural Information Processing Systems 2*, 1989.
12. Amir Atiya. *Learning Algorithms for Neural Network*. PhD thesis, California Institute of Technology, 1994.
13. Al. Cripps. Using artificial neural nets to predict academic performance. In *Proceedings of the 1996 ACM Symposium on Applied Computing*, pages 33–37, 1996.
14. Thomas Mikolov. *Statistical Language Models Based on Neural Networks*. PhD thesis, Brno University of Technology, 2012.
15. Kai Yu. Large-scale deep learning at baidu. In *Proceedings of the 22nd ACM International Conference on Information and Knowledge Management*, pages 2211–2212, 2013.
16. Tom Mitchell. *Machine Learning*. McGraw-Hill, 1997.
17. Jeffrey L. Elman. Learning and development in neural networks: The importance of starting small. *Journal of Cognition*, (48):71–99, 1993.

18. Christopher D. Manning and Hinrich Schutze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.
19. Prabhakar Raghavan Christopher D. Manning and Hinrich Schutze. *An Introduction to Information Retrieval*. Cambridge UP, 2009.
20. Andrew Y. Ng Richard Socher, Cliff Chiung-Yu Lin and Christopher D. Manning. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th International Conference on Machine Learning*, 2011.
21. 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.
22. 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.
23. Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31$_{st}$ International Conference on Machine Learning*, 2014.
24. 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.
25. Peter D. Turney and Patrick Pantel. From frequency to meaning: Vector space models of semantics. *Journal of Artificial Intelligence Research*, (37):141–188, 2010.
26. 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.
27. Gred Corrado Thomas Mikolov, Kai Chen and Jeffrey Dean. Efficient estimation of word representations in vector space. In *Proceedings of CoRR*, 2013.