9

Seleksi Fitur dan Metode Evaluasi

> "The key to artificial intelligence has always been the representation."
>
> Jeff Hawkins

Bab ini membahas beberapa tips dan trik yang sebenarnya sudah disinggung pada bab 5. Hanya saja, beberapa hal lebih baik dijelaskan secara lebih mendalam ketika sudah mempelajari beberapa algoritma pembelajaran mesin, yaitu *feature engineering*, *feature selection* dan penjelasan lebih lanjut *cross validation*. Kamu dapat menganggap bab ini sebagai kelanjutan tips dan trik pembelajaran mesin lanjutan bab 5.

9.1 Feature Engineering

Record (data) pada pembelajaran mesin pada umumnya dikonversi menjadi suatu vektor (*feature vector*) yang merepresentasikannya dalam bentuk matematis. Fitur-fitur biasanya tersusun atas variabel-variabel yang kita anggap memiliki pengaruh terhadap *output*. Sebagai contoh, tekanan darah diprediksi berdasarkan usia, jenis kelamin dan BMI. Seringkali, seseorang membutuhkan keahlian suatu bidang agar dapat memilih fitur yang tepat. Proses untuk mencari (atau me-*list*) kandidat fitur, disebut sebagai aktivitas ***feature engineering***.

Seperti kata mutiara yang kami berikan pada awal bab ini, kunci pembelajaran mesin adalah representasi permasalahan. Fitur yang dipilih untuk merepresentasikan data adalah bagaimana cara kamu merepresentasikan masalah juga. Karena membutuhkan tingkat keahlian tertentu, proses memilih fitur tidaklah mudah. Bisa jadi (sering), orang memilih fitur yang tidak representatif! Dengan demikian, tidak heran apabila seseorang mempublikasikan makalah ilmiah atas kontribusinya menentukan fitur baru pada domain tertentu.

Konon tetapi, proses pemilihan fitur yang bersifat manual ini cukup bahaya karena rentan dengan *bias*, yaitu kemampuan seseorang pada domain tertentu. Alangkah baiknya, apabila kita dapat memilih fitur yang memang benar-benar diperlukan (murni) secara otomatis. Hal tersebut akan dibahas lebih lanjut pada materi *artificial neural network*. Hal ini salah satu alasan yang membuatnya populer.

9.2 High Dimensional Data

Pada kenyataan, fitur yang kita gunakan untuk membangun model pembelajaran mesin tidaklah sesederhana yang dicontohkan pada subbab sebelumnya. Seringkali, kita berhadapan dengan data yang memiliki sangat banyak fitur (*high dimensional data*). Sebagai contoh, seorang *marketing analyst* mungkin saja ingin mengerti pola seseorang berbelanja pada *online shop*. Untuk mengerti pola tersebut, ia menganalisis seluruh kata kunci pencarian (*search term*) *item*. Misalnya, seseorang yang ingin membeli meja makan juga mungkin akan membeli kursi (sebagai satu paket). Data pada analisis semacam ini berdimensi besar. Seberapa besar dimensi yang dapat disebut *high dimension* adalah hal yang relatif.

Sayangnya, data dengan dimensi yang sangat besar membawa beberapa masalah pada pembelajaran mesin. Pertama, model pembelajaran susah untuk memiliki kinerja yang optimal pada data berdimensi tinggi. Semakin banyak fitur yang dipakai, semakin kompleks suatu model pembelajaran mesin harus memodelkan permasalahan. Berhubung kita memiliki banyak fitur, *search space* untuk mencari konfigurasi parameter optimal sangatlah luas. Hal ini dapat dianalogikan seperti mencari seseorang pada gedung berlantai satu vs. gedung berlantai 20. Kedua, hal ini menyebabkan mudah terjadi *overfitting* karena ada sangat banyak konfigurasi fitur walaupun kita hanya memiliki data yang terbatas[1]. Ketiga, data dengan dimensi yang besar susah untuk diproses secara komputasi (*computationally expensive*), baik dari segi memori dan waktu. Karenanya hal ini, kita ingin agar fitur-fitur yang kita gunakan sesedikit mungkin. Dengan kata lain, kita ingin representasi permasalahan sesederhana mungkin dari sesi memori dan *computational processing*.

9.3 Feature Selection

Pada pembelajaran mesin, pada umumnya kita menggunakan banyak (lebih dari satu) fitur. Artinya kita merepresentasikan setiap *record* (instans) atau *input* sebagai suatu vektor $\mathbf{x} \in \mathbb{R}^{1 \times F}$; dimana F melambangkan dimensi vektor atau banyaknya fitur. Seringkali, F bernilai besar sehingga model yang kita miliki kompleks. Kita tentunya ingin mengurangi kompleksitas dengan

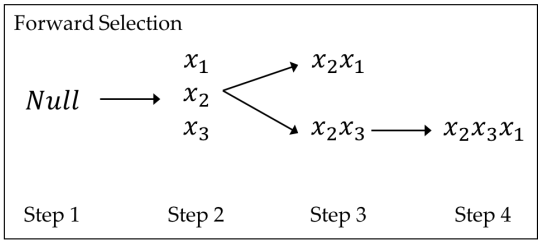
[1] `Curse of Dimensionality`

alasan-alasan yang sudah disebutkan pada subbab 9.2. Alasan lainnya karena belum tentu semua fitur berguna. Cara termudah adalah dengan menghapus fitur yang memiliki nilai varians = 0. Sayang sekali, hal ini tidak selalu terjadi. Subbab ini membahas teknik-teknik yang dapat digunakan untuk menyederhanakan fitur (mengurangi dimensi input).

9.3.1 Subset Selection (Feature Ablation)

Cara paling intuitif untuk mencari tahu kombinasi fitur terbaik adalah dengan mencoba seluruh kombinasi fitur. Misal kita mempunyai fitur sebanyak F, kita bisa pilih untuk menggunakan atau tidak menggunakan masing-masing fitur, menghasilkan kombinasi sebanyak 2^F. Dari keseluruhan kombinasi tersebut, kita pilih suatu kombinasi fitur yang memerikan kinerja terbaik. Akan tetapi, metode *brute force* ini terlalu memakan waktu. Kita dapat juga menggunakan teknik *greedy* yaitu *forward selection* dan *backward selection*. ***Forward*** dan ***backward selection*** sering juga disebut sebagai ***feature ablation***.

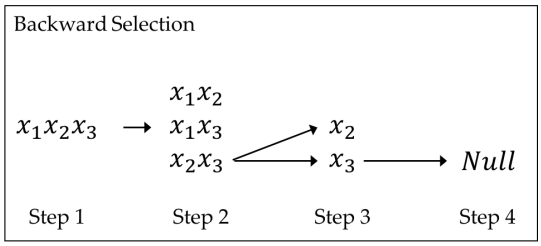
Pada *forward selection*, kita mulai dengan suatu model yang tidak menggunakan fitur apapun sama sekali, yaitu meng-*assign* kelas yang paling sering muncul di dataset, pada *input*. Setelah itu, kita tambahkan satu per satu fitur pada setiap langkah. Langkah berikutnya, kita gunakan satu fitur. Diantara F pilihan fitur, kita cari fitur yang memberi nilai terbaik. Kemudian, pada tahap berikutnya, kita kombinasikan fitur yang kita pilih pada langkah sebelumnya dengan fitur yang tersisa. Hal ini terus diulang sampai kita sudah menggunakan seluruh fitur pada model kita. Untuk mencari model terbaik, kita hanya perlu mencari kombinasi fitur yang memberikan nilai kinerja terbaik. *Forward selection* diilustrasikan pada Gambar 9.1. Dibanding *brute force* yang bersifat eksponensial, *forward selection* membentuk suatu deret aritmatika kombinasi fitur yang dicoba, yaitu $1+F+(F-1)+\cdots+1 = 1+F(F+1)/2$. Apabila kita memiliki fitur sebanyak $F = 10$, kombinasi *brute force* menghasilkan $1,024$ kombinasi, sementara *forward selection* hanya sebanyak 56!



Gambar 9.1. Ilustrasi *forward selection* untuk tiga fitur

Backward selection adalah kebalikan dari *forward selection*. Apabila pada *forward selection*, kita menambahkan satu fitur tiap langkah, *backward selection* mengurangi satu fitur pada tiap langkah. Ilustrasi diberikan pada Gam-

bar 9.2. Seperti *forward selection*, *backward selection* juga hanya mencoba sebanyak $1 + F(F + 1)/2$ kombinasi. Kita juga dapat menggabungkan *forward* dan *backward selection* menjadi metode *hybrid*, yaitu menambah satu fitur pada tiap langkah, serta memperbolehkan untuk menghilangkan fitur juga [17].



Gambar 9.2. Ilustrasi *backward selection* untuk tiga fitur

9.3.2 Shrinkage

Ingat kembali materi bab 5 tentang *regularization*. Seperti yang sudah dijelaskan, kita ingin agar model kita sesederhana mungkin. Mengurangi dimensi fitur adalah cara mengurangi kompleksitas. Ingat kembali, dengan menggunakan suatu fungsi regularisasi, objektif pembelajaran adalah meminimalkan *loss* dan kompleksitas, seperti pada persamaan 9.1. Kompleksitas model dapat dihitung menggunakan L_2 (Ridge, persamaan 9.2) atau L_1 (Lasso, persamaan 9.3) norm. Karena kita ingin meminimalkan norm, artinya kita juga membuat parameter model pembelajaran mesin bernilai dekat dengan nol. Pada metode Ridge, kita ingin meminimalkan fungsi eksponensial, sementara fungsi skalar pada Lasso. Artinya, Lasso lebih cenderung untuk menghasilkan suatu model yang bersifat *sparse*. Dengan kata lain, Lasso melakukan *subset feature selection* seperti yang sudah dijelaskan pada subbab sebelumnya. Sementara itu, Ridge cenderung tidak meng-nol-kan parameter, melainkan hanya **dekat** dengan nol [17]. Kamu mungkin berpikir, kenapa kita tidak menggunakan Lasso saja, berhubung ia mengeleminasi fitur. Pada metode Ridge, semua fitur tetap digunakan walaupun nilainya diturunkan (*shrink*) agar dekat dengan nol. Hal ini, walaupun tidak mengeleminasi fitur, dapat mengurangi variasi kinerja model[2] (dijelaskan pada subbab 9.4).

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \mathcal{L}(\mathbf{w}) + \lambda \mathrm{R}(\mathbf{w}) \tag{9.1}$$

$$\mathrm{R}(\mathbf{w}) = \|\mathbf{w}\|_2^2 = \sum_i (w_i)^2 \tag{9.2}$$

[2] baca buku [17] untuk pembuktiannya

$$\mathrm{R}(\mathbf{w}) = \|\mathbf{w}\| = \sum_i |w_i| \tag{9.3}$$

Secara singkat, Ridge digunakan untuk menghasilkan model yang varian kinerjanya kecil. Sementara itu, Lasso digunakan untuk menghasilkan model yang mudah dimengerti (*interpretability*) dengan mengeliminasi fitur.

9.3.3 Principal Components Analysis (Dimension Reduction)

Teknik-teknik sebelumnya berfokus pada cara mengeleminasi fitur. Akan tetapi, penghapusan suatu fitur berpotensi pada model yang tidak mampu mengerti kompleksitas permasalahan. Dengan kata lain, *oversimplification*. Dibanding menghapus fitur, cara lain untuk mengurangi kompleksitas komputasi adalah mentransformasi data ke dalam dimensi lebih kecil. Untuk *input* yang memiliki F fitur, kita kurangi dimensi *input* menjadi $M < F$ (*dimension reduction*). Apabila kita mampu mengurangi dimensi *input*, maka kita juga dapat mengurangi jumlah parameter pada model pembelajaran mesin, yang artinya mengurangi kompleksitas komputasi dan meningkatkan *interpretability*.

Ide utama *dimension reduction* adalah mentransformasi data dari suatu *space* ke *space* lainnya, dimana data direpresentasikan dengan dimensi lebih kecil. Dengan catatan, data dengan dimensi lebih kecil harus mampu merepresentasikan karakteristik data pada dimensi aslinya! Dengan demikian, satu fitur pada dimensi yang baru mungkin memuat informasi beberapa fitur pada dimensi aslinya. Walaupun model yang kita hasilkan lebih *interpretable* secara jumlah parameter, tetapi kita membutuhkan usaha ekstra untuk mengerti representasi fitur-fitur pada dimensi yang baru.

Teknik yang digunakan untuk mengurangi dimensi adalah *principal component analysis*. Ide dasarnya adalah mencari bentuk data (pada dimensi lebih kecil) yang memiliki nilai varians tinggi untuk tiap fiturnya, karena varians yang tinggi berarti kemampuan fitur yang tinggi dalam melakukan diskriminasi (klasifikasi). Kita ingin mencari suatu arah (vektor, matriks, tensor) dimana data kita memiliki varians tertinggi. Pada dimensi yang baru, kita berharap data kita tersebar menjadi beberapa grup yang mudah dibedakan (*easily separable*). Arah ini dikenal sebagai **eigenvector**, dan nilai varians pada arah tersebut dikenal sebagai **eigenvalue**. Kami harap kamu ingat materi kuliah aljabar linear. Eigenvector yang memiliki nilai eigenvalue tertinggi disebut sebagai ***principal components***, yaitu semacam bentuk data yang ringkas. Selain mengurangi dimensi, teknik ini juga mengurangi (atau meniadakan) interaksi antar-fitur. Dengan demikian, kita dapat menggunakan *additive assumption* saat melakukan pembelajaran (ingat kembali materi bab 5).

Saat melakukan analisis ini, kita membuat suatu asumsi bahwa sejumlah *principal components* cukup untuk merepresentasikan variasi yang ada pada data, serta hubungan data asli dengan respons [17]. Dengan kata lain,

kita mengasumsikan arah (eigenvector) ketika data memiliki varians tertinggi, adalah arah yang berasosiasi dengan *output*. *Singular Value Decomposition* adalah salah satu teknik untuk melakukan *principal component analysis*. Hal tersebut akan dibahas pada bab 12. Buku ini juga akan memberi contoh konkret *dimensionality reduction* dengan *artificial neural network* pada bab 12.

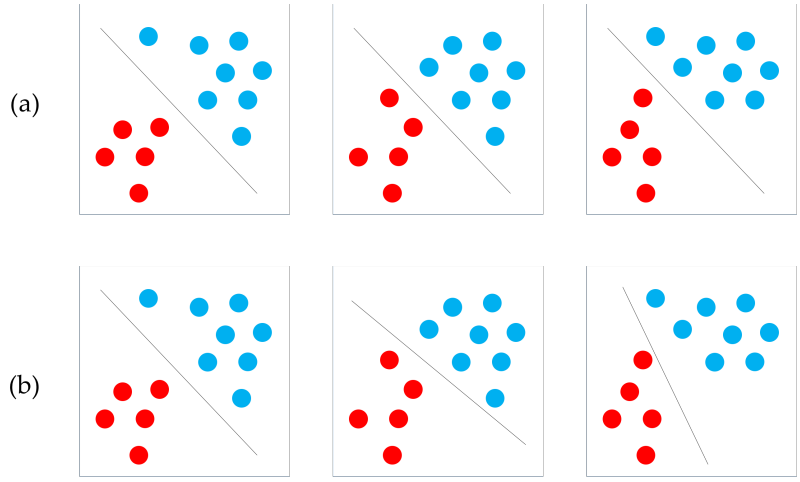
9.4 Cross Validation

Ingat kembali materi bab-bab sebelumnya bahwa pada umumnya kita membagi dataset menjadi tiga grup: *training*, *validation/development* dan *testing*. Melatih dan mengukur kinerja model menggunakan *training data* adalah hal yang tidak bijaksana karena kita tidak dapat mengetahui kemampuan generalisasi model. Kita melatih model pembelajaran mesin menggunakan *training data* yang dievaluasi kinerjanya (saat *traning*) menggunakan *validation data*. Setelah itu, kita uji model kembali menggunakan *testing data*. Pada umumnya, ketiga grup tersebut memiliki data dengan karakteristik yang sama. Artinya, dataset disampel dari distribusi yang sama. Misal pada *training data*, ada pembagian 30:30:40 untuk data kelas pertama, kedua dan ketiga. Distribusi persebaran tersebut juga harus sama pada *validation* dan *testing data*. Walaupun tidak boleh ada data yang sama (*overlap*) pada ketiga grup tersebut, memiliki *validation* dan *testing data* yang dibedakan adalah hal yang cukup bijaksana.

Pada dunia nyata, kita belum tentu memiliki ketiga grup tersebut. Penyebabnya ada banyak, misal dataset yang kecil (hanya memiliki sedikit *records*) atau pengadaan *testing data* mahal. Apabila kita tidak memiliki *testing data*, *validation data* dapat menjadi pengganti (*proxy*). Tetapi ada masalah apabila kita hanya menguji model pada satu set *validation data*, yaitu *bias*[3]. *Validation data* yang tetap menyebabkan kita tidak mengetahui variasi kinerja model [17]. Sebisanya, kita ingin mengetes kinerja model pada beberapa dataset, kemudian mengevaluasi bagaimana variasi kinerja model. Kegiatan semacam ini membuat kita mengetahui apakah suatu model cukup stabil[4] dan fleksibel. Stabil berarti kinerja model tidak begitu berubah, diberikan *input* yang sedikit berubah (ilustrasi diberikan pada Gambar 9.3). Fleksibel berarti model yang mampu menangkap karakteristik data dengan baik. Misal kita menggunakan model linear dengan persamaan polinomial orde-1, orde-2 dan orde-3. Saat kita menaikkan orde persamaan sampai dengan orde-3, kinerja model terus meningkat pada *validation* data, walaupun kurang lebih sama pada *training data*. Lalu, kita mencoba menggunakan persamaan polinomial orde-4, dan kinerja model menurun pada *validation* data. Artinya, persamaan polinomial orde-3 adalah yang paling optimal untuk memodelkan permasalahan yang kita miliki. Apabila kita tidak memiliki *validation* data, kita tidak

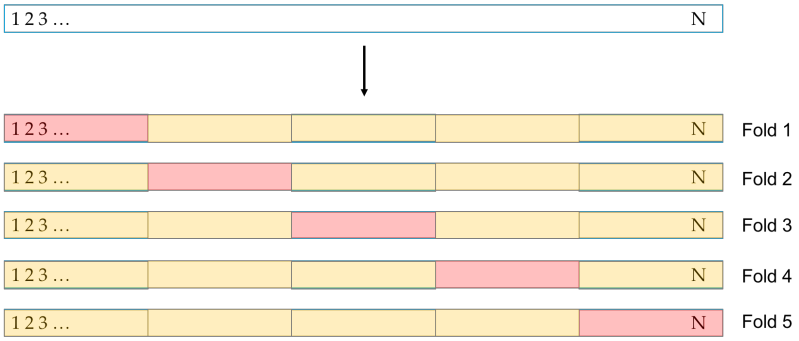
[3] *statistical bias*

[4] `https://en.wikipedia.org/wiki/Stability_(learning_theory)`



Gambar 9.3. Model yang stabil (a) memberikan *decision boundary* yang serupa walaupun *input* sedikit diubah-ubah (variasi kinerja yang kecil, untuk set *input* yang berbeda). Model yang kurang stabil (b) memberikan *decision boundary* yang berbeda untuk *input* yang sedikit diubah-ubah (variasi kinerja yang besar, untuk set *input* yang berbeda)

akan mengetahui hal ini. Stabil berkaitan dengan nilai varians, sedangkan fleksibel berkaitan dengan perubahan terhadap ukuran kinerja. Fleksibilitas dapat dianalisis dengan mem-*plot* grafik kinerja model pada *training data* dan *validation/testing data* untuk mengetahui *underfitting* dan *overfitting* seperti yang telah dijelaskan pada bab 5.



Gambar 9.4. Ilustrasi 5-*fold-cross-validation*. Kotak berwarna merah melambangkan subset yang dipilih sebagai *validation data*

Kita dapat menganalisis stabilitas dengan menggunakan teknik *cross validation* seperti yang sudah dijelaskan pada bab 3. Intinya adalah, kita

membagi-bagi *dataset* yang kita miliki menjadi K bagian. Kita latih model menggunakan $K-1$ bagian, kemudian menguji model dengan menggunakan satu bagian lainnya sebagai *validation data* yang mengaproksimasi kinerja pada *testing data* (Ilustrasi pada Gambar 9.4). Hal ini disebut sebagai K-***fold-cross-validation***. Untuk $K = N$ yaitu jumlah data, kita sebut teknik tersebut sebagai ***leave-one-out-cross-validation***. Perlu diperhatikan, distribusi tiap-tiap subset data haruslah sama (*stratified sampling*). Dengan teknik *cross validation*, kita memiliki sejumlah K model pembelajaran mesin dan K buah nilai kinerja. Kita dapat mengukur stabilitas model dengan menghitung varians kinerja dari K model. Nilai rata-rata kinerja yang diberikan adalah sebuah estimasi terhadap kinerja model pada *testing data* (yang sesungguhnya tidak ada), diberikan pada persamaan 9.4. Kita ingin model yang stabil, karena nilai varians yang kecil berarti eksperimen dapat diulangi, dan kita mendapatkan kinerja yang sama saat eksperimen diulang. Varians yang terlalu besar dapat berarti kita mendapatkan suatu nilai kinerja secara *random chance* (untung-untungan belaka).

$$\mathrm{CV}_{(K)} = \frac{1}{K}\sum_{i=1}^{K}\mathrm{KinerjaModel}_i \tag{9.4}$$

9.5 Replicability, Overclaiming dan Domain Dependence

Pada dunia pembelajaran mesin, *replicability* adalah hal yang sangat penting. Artinya, eksperimen yang kamu lakukan dapat diulangi kembali oleh orang lain, serta mendapat kinerja yang kurang lebih sama. Untuk ini, biasanya dataset dipublikasi pada domain publik agar dapat digunakan oleh banyak orang, atau mempublikasi kode program. Selain *replicability*, kamu juga harus memperhatikan *overclaiming*. Banyak orang yang menggunakan *toy dataset* (berukuran sangat kecil) yang tidak merepresentasikan permasalahan aslinya. Akan tetapi, mereka mengklaim bahwa model mereka memiliki generalisasi yang baik. Kamu harus menginterpretasikan kinerja model secara bijaksana. Kinerja yang baik pada *toy dataset* belum tentu berarti kinerja yang baik pada dunia nyata. Sebagai contoh, artikel yang dibahas pada *post*[5] ini adalah contoh *overclaiming*.

Perlu kamu perhatikan juga, mendapatkan kinerja yang bagus pada suatu dataset belum tentu berarti model yang sama dapat mencapai kinerja yang baik pada dataset lainnya. Misalnya, model yang dilatih untuk mengklasifikasikan kategori berita belum tentu dapat mengklasifikasikan laporan medis. Hal ini banyak terjadi pada *supervised learning* [35].

[5] `Pranala Post Yoav Goldberg`

Soal Latihan

9.1. Seleksi Fitur
Jelaskanlah algoritma seleksi fitur selain yang sudah dijelaskan pada bab ini! (Saran: baca *survey paper*)

9.2. AIC dan BIC
Jelaskan *Akaike Information Criterion* (AIC) dan *Bayesian Information Criterion* (BIC)!

9.3. Bootstrapping
Jelaskan apa itu teknik *bootstrapping* (evaluasi model)! Bagaimana perbedaanya *bootstrapping* dan *cross validation*?

9.4. Varians
Jelaskan mengapa fitur yang baik memiliki varians yang tinggi, sementara kinerja model yang baik memiliki varians yang rendah!