Richard Rivaldo
13519185 – K04

Supervised Learning

1. Apa itu supervised learning?

   Supervised Learning adalah salah satu jenis algoritma pembelajaran dalam *machine learning* dan *deep learning*. Pada *supervised learning*, perhitungan prediksi dan pembangunan aturan-aturan sebagai kriteria prediksi dibangun dengan menggunakan data yang telah memiliki label tertentu. Ketika data masukan diberikan kepada model *supervised learning*, maka model akan berusaha untuk memahami dan mencari pola pemetaan fungsi $y = f(x)$. Dalam hal ini, *y* adalah label yang terdapat pada semua data, dan *x* adalah semua ftur yang ada pada data tersebut.
   Supervised Learning biasanya menggunakan *training dataset* untuk mengajari model tersebut. Penggunaan *test set* adalah untuk memeriksa keberhasilan model tersebut dalam memprediksi data baru, dan *test set* harus berbeda dengan *training set* sehingga tidak terjadi *data leak.* Proses pembelajarannya sendiri dilakukan dengan mencari fungsi yang bisa meminimumkan *loss* atau *cost function* untuk data yang diberikan. Hal tersebut dapat dicapai dengan memodifikasi parameter-parameter yang digunakan dalam melakukan *fitting* terhadap suatu data.
   Secara umum, algoritma ini bisa dibagi menjadi dua kelompok permasalahan, yaitu persoalan klasifikasi dan regresi. Permasalahan klasifikasi merupakan permasalahan untuk mengelompokkan suatu data ke dalam kategori tertentu sehingga label keluaran yang diberikan biasanya berupa nilai diskrit. Sebaliknya, regresi merupakan permasalahan untuk mencari hubungan fitur dari suatu data dengan nilai keluarannya yang bertipe kontinu. Nantinya, hubungan yang ditemukan ini bisa digunakan untuk memprediksi nilai keluaran tersebut.
   Keuntungan dari algoritma ini adalah cara penentuan labelnya yang berdasarkan pengamatan pada data yang diberikan. Selain itu, algoritma ini merupakan algoritma yang cukup efektif untuk menyelesaikan banyak permasalahan. Namun, algoritma ini tidak terlalu baik untuk memecahkan permasalahan yang kompleks. Selain itu, proses pembelajarannya bisa memakan waktu yang sangat lama, tergantung jenis algoritma dan banyak data yang digunakan. Contoh dari algoritma ini adalah *Linear* dan *Logistic Regression*, *Naïve Bayes*, *K-Nearest Neighbors*, *Support Vector Machine*, dan *Random Forest* serta *Neural Network.*

2. Jelaskan bagaimana cara kerja dari algoritma yang anda implementasikan!
   a. *K-Nearest Neighbor*
      Algoritma ini dibuat sebagai sebuah kelas *Classifier* dengan atribut yang berisikan konstanta K, dan dua dataset yang masing-masing merupakan *training* dan *test* dataset. Dalam hal ini, konstanta K menandakan jumlah tetangga terdekat yang ingin dicari dari suatu input data tertentu. Adapun *training* dan *test* dataset diambil dari satu dataset besar yang sama, namun di-*slice* sehingga dataset yang awalnya berukuran 10.000 *tuple* dibagi menjadi masing-masing 9900 dan 100 *tuple*.
      Untuk mencari tetangga terdekat, maka bisa digunakan beberapa metode perhitungan. Secara sederhana, setiap data dapat dikatakan sebagai sebuah titik pada dimensi yang tinggi karena ukuran *feature* nya yang besar. Oleh karena itu, jarak dari suatu data ke data lainnya bisa dihitung dengan menggunakan *Euclidean Distance* seperti pada implementasi yang dilakukan, atau dengan *Manhattan Distance*. Hal ini bisa dilakukan karena tidak ada *feature* yang bertipe kategorikal, dan data fitur bertipe demikian biasanya dihitung dengan menggunakan *Hamming Distance*.

Untuk menghitung jarak *Euclidean* tersebut, maka perhitungan dilakukan tanpa memasukkan kolom terakhir yang merupakan label data. Dalam hal ini, perhitungan dibantu dengan menggunakan fungsi *norm* pada pustaka aljabar linear Numpy.

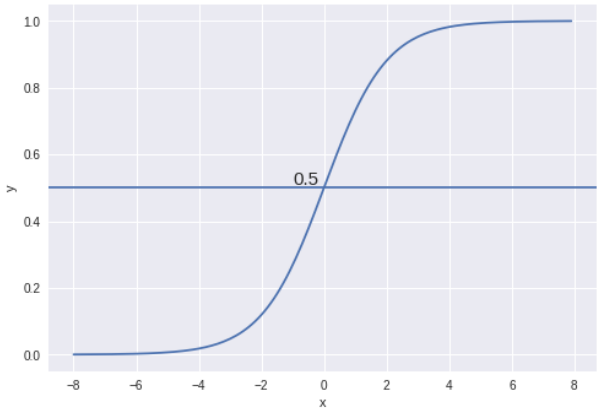
Setelah mendefinisikan fungsi tersebut, maka untuk suatu masukan data tertentu, akan dicari jarak dari masukan ini dengan semua data yang ada pada data *training*. *Tuple* yang mengandung jarak hasil perhitungannya serta data *training* yang berkorespondensi dimasukkan ke dalam suatu larik. Elemen pada larik ini nantinya akan diurutkan secara menaik berdasarkan komputasi jarak sebelumnya. Setelah itu, karena semua data sudah terurut dari yang paling dekat (jarak terkecil) sampai terjauh (jarak terbesar) untuk data masukan, maka daftar tabel bisa dipotong sebanyak K buah elemen saja. Artinya, hasil akhir untuk pemrosesan ini berupa daftar *K-Nearest Neighbor* untuk data masukan.

Dari daftar tetangga tersebut, kemudian dilakukan *majority voting* untuk menentukan kategori dari data masukan. *Majority voting* dilakukan dengan menghitung jumlah kategori yang paling banyak untuk semua tetangga yang ada. Pengimplementasian *voting* dibantu dengan menggunakan kelas Counter yang terdapat pada pustaka *Collections*. Dengan demikian, sudah didapat hasil prediksi dari algoritma KNN ini dan hanya perlu ditampilkan kepada pengguna saja ketika membuat suatu prediksi tertentu.

b. *Logistic Regression*

Algoritma ini dimuat di dalam sebuah kelas dengan beberapa atribut. Dalam hal ini, pengguna harus memberikan argumen berupa *learning rate* dan jumlah *epochs* yang akan digunakan dalam komputasi model, karena tidak ditentukan nilai *default* untuk kedua atribut. Tidak seperti KNN, atribut *training* dan *test dataset* masih disimpan dalam bentuk *dataframe* Pandas. Atribut lain yang berkaitan dengna proses perhitungan algoritma ini adalah *weight*, *intercept*, dan *cost*.

Model ini merupakan model yang matematis dan tidak terlalu intuitif jika dibandingkan dengan KNN. Fungsi utama yang digunakan untuk memetakan suatu input yang diterima model ini adalah fungsi Sigmoid, yaitu fungsi yang memetakan input sehingga nilainya selalu berada di antara 0 dan 1. Fungsi ini diberikan oleh pemetaan berikut.



$$\sigma(t) = \frac{1}{1 + e^{-t}}$$

Sebelum melakukan prediksi, model ini harus menjalani proses *training* dan *fitting* data terlebih dahulu. Pada proses ini, model akan mencari *weight* dan *intercept* terbaik yang bisa memberikan *fitting line* terbaik untuk data yang diberikan. Di balik pencarian tersebut, terdapat beberapa fungsi statistik untuk menyelesaikan permasalahan yang ada.

Parameter yang perlu diperbaiki untuk setiap algoritma adalah *weight* dan *intercept*. *Weight* atau bobot adalah sebuah larik (diinisialisasi dengan 0) yang berisikan koefisien-koefisien untuk semua *feature* yang ada di dalam dataset. *Weight* menentukan seberapa penting pengaruh dari sebuah *feature* terhadap hasil prediksi yang dilakukan nantinya. *Intercept* adalah konstanta `b` seperti yang biasanya ada pada persamaan regresi linear biasa.
Kedua parameter ini di-*update* dengan menggunakan bantuan *Gradient Descent*. *Gradient descent* adalah salah satu algoritma optimisasi yang paling sering digunakan untuk menemukan nilai minimum lokal yang meminimumkan *error* dari suatu model. Dengan nilai *error* yang minimum, maka secara intuitif dapat dikatakan bahwa model tersebut bekerja dengan baik.
Perhitungan *gradient descent* dalam algoritma ini membutuhkan nilai dari fungsi aktivasi dari parameter yang dimiliki pada saat itu. Karena fungsi aktivasi yang digunakan adalah fungsi *sigmoid*, maka nilai fungsi aktivasi untuk parameter ini merupakan hasil pengaplikasian *weight* terhadap semua data *training* yang ada ditambah dengan nilai *intercept* nya. Pengaplikasian *weight* terhadap semua data *training* dapat dilakukan dengan menggunakan perkalian dot biasa. Nilai yang didapat dari fungsi ini sering disebut sebagai nilai hipotesis (*y hat*).
Setelah mendapatkan nilai aktivasi tersebut, maka nilai gradien, masing-masing untuk *weight* (dW) dan *intercept* (db) bisa dicari dengan menggunakan formula berikut.

$$dW = \left(\frac{1}{m}\right) \times ((A - y)^T \,.\, X)$$
$$db = \left(\frac{1}{m}\right) \times \sum (A - y)$$

Dengan m = jumlah data, A adalah nilai aktivasi, X adalah semua *training data*, dan y adalah label untuk semua *training data*.
Untuk memperbarui kedua parameter tadi, maka kedua parameter tadi tinggal dikurangi dengan gradien masing-masing parameter yang telah dikalikan dengan tingkat pembelajaran (*learning rate*) yang diberikan. Oleh karena itu, *learning rate* akan menentukan seberapa cepat perubahan nilai dengan metode ini. Pembaruan nilai ini akan dilakukan sebanyak jumlah *epochs* yang diberikan pada awal pemanggilan.
Pada setiap *epoch*, akan terdapat *cost function* yang berbeda-beda. Fungsi biaya ini dapat dikatakan sebagai rataan dari akumulasi *loss function* untuk setiap data pada *training data*. Tujuan penggunaan *gradient descent* tidak lain adalah mencari parameter yang meminimumkan *cost function.* Pada kasus tertentu, *gradient descent* bisa saja mencapai konvergensi dengan sangat cepat sehingga nilai *cost function* untuk *epoch* berikutnya cenderung konstan. Nilai dari *cost function* ini dapat dihitung dengan menggunakan rumus berikut.

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L\left(\hat{y}^{(i)}, y^{(i)}\right) = -\frac{1}{m}\sum_{i=1}^{m}\left[\left(y^{(i)}\log\left(\hat{y}^{(i)}\right) + \left(1 - y^{(i)}\right)\log(1 - \hat{y}^{(i)})\right]\right.$$

Setelah proses *training* telah selesai, maka model akan langsung memilih salah satu data yang ada pada *test set* untuk diprediksi. Prediksi dilakukan dengan mengaplikasikan *weight* yang didapat dari hasil *training* ditambahkan dengan nilai *intercept*. Jumlah keduanya dimasukkan lagi ke dalam fungsi *sigmoid* dan dapat dikatakan bahwa hasil fungsi ini akan menjadi *confidence level* untnk prediksi data

tersebut. Untuk menentukan hasil prediksi, digunakan *threshold* 0.5 sehingga tingkat kepercayaan yang melebihi 0,5 menandakan bahwa label yang dihasilkan adalah *True*, dan sebaliknya.
Perlu diperhatikan bahwa semua *features* yang ada pada *training* dan *test set* harus dinormalisasi terlebih dahulu sebelum diberikan kepada model. Hal ini untuk mencegah nilai yang terlalu besar ataupun terlalu kecil sehingga mengakibatkan *overflow* dan *zero division* pada fungsi *sigmoid* dan logaritma biasa. Normalisasi ini bisa dilakukan dengan menggunakan rumus berikut.

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

c. *Iterative Dichotomiser 3*
Pembangkitan pohon keputusan dengan menggunakan algoritma *Iterative Dichotomiser 3* dibangun dalam sebuah kelas dengan beberapa atribut. Pada kelas yang dibangun, *training* dan *test dataset* dimuat secara utuh dalam bentuk *data frame*. Perbedaan model ini dari kedua model sebelumnya adalah pada model ini data yang diberikan hanya mengandung atribut kategorikal saja, sedangkan pada kedua model sebelumnya hanya berupa atribut numerik. Tidak ada atribut lain yang diberikan untuk model ini.
Pembangkitan setiap simpulnya sendiri dihitung dengan menggunakan *information gain* berbasis tingkat entropi. Tingkat entropi menyatakan tingkat ketidakteraturan, sehingga semakin tinggi nilai entropi pada suatu data, berarti bahwa data tersebut cenderung heterogen, dan sebaliknya. Untuk suatu fitur tertentu, nilai entropinya merupakan minus dari jumlah semua nilai probabilitas kemunculan suatu nilai unik dikali dengan nilai $\log_2$ dari probabilitas tersebut. Dari sini dapat diketahui bahwa semakin banyak nilai yang unik di suatu data, maka entropinya akan semakin tinggi, sehingga data dengan fitur bertipe numerik akan membentuk pohon dengan banyak sekali cabang.

$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

Adapun *information gain* secara sederhana dapat dikatakan sebagai seberapa berpengaruh suatu atribut terhadap penentuan suatu label, mirip dengan *weight* pada *logistic regression*. Nilai ini untuk sebuah atribut bisa dihitung dengan mengurangi total entropi untuk atribut tersebut dengan total entropi untuk atribut tersebut pada nilai tertentu.

$$Gain(T, X) = Entropy(T) - Entropy(T, X)$$

Seperti pada algoritma berbasis pohon lainnya, pembangkitan pohon keputusan pada algoritma ini dilakukan dengan prinsip rekursivitas. Basis pertama yang akan diperiksa terlebih dahulu adalah tingkat entropi dari label yang ingin diprediksi.

Jika ternyata hanya ada satu buah nilai unik, hal ini berarti bahwa fitur ini murni sehingga bisa langsung dikembalikan nilai unik label tersebut.
Selain itu, jika tidak ada data sama sekali untuk sebuah pencabangan, maka nilai yang akan diberikan untuk data ini adalah nilai mayoritas dari label data pada status tersebut. Terakhir, karena pencabangan pada pohon dilakukan dengan memeriksa daftar fitur yang ada, maka jika sudah tidak ada lagi fitur yang sedang diperiksa, maka akan dikembalikan nilai mayoritas dari *parent node* nya.
Adapun proses rekursif dari algoritma yang dibangun adalah sebagai berikut. Terlebih dahulu akan ditetapkan nilai *parent class* yang penentuannya dilakukan berdasarkan nilai mayoritas dari label target yang ada. Simpul akar dari pohon merupakan simpul yang berisikan fitur dengan *information gain* tertinggi. Oleh karena itu, pada setiap pembangunan pohon, akan dicari fitur dengan *information gain* tertinggi. Fitur ini akan ditambahkan ke pohon yang dibuat dalam representasi *dictionary*.
Jika sudah ditambahkan, maka fitur kemudian dihapus dari daftar fitur yang ada sehingga tidak mengakibatkan *infinite recursion*. Untuk setiap nilai unik pada fitur tersebut, akan ditambahkan ke dalam pohon keputusan sebagai *child nodes*. Untuk setiap anak ini pula, nantinya akan ditambahkan *tree* baru yang proses pembentukannya menggunakan daftar fitur yang telah berkurang tadi secara rekursif. Pada akhirnya, telah dihasilkan pohon keputusan dengan simpul dalam menyatakan `kondisi` sesuai dengan nilai unik yang ada pada semua fiturnya, dan simpul daun merupakan nilai label yang diprediksi nantinya.
Pembentukan pohon akan dilakukan secara otomatis pada saat pembentukan objek berhasil dilakukan. Untuk memprediksi suatu data, data yang awalnya berupa *data frame* diubah dulu ke dalam bentuk *list of dictionaries*. Untuk setiap *record* data yang ada, penentuan labelnya dilakukan dengan menelusuri semua atribut yang ada pada data tersebut dan juga pada *decision tree*. Penelusuran ini juga dilakukan secara rekursif menggunakan nilai setiap fitur pada data yang diprediksi.

3. Bandingkan ketiga algoritma tersebut, kemudian tuliskan kelebihan dan kelemahannya!
Algoritma KNN dan ID3 merupakan sebuah algoritma yang bersifat *non-parametric*, sedangkan algoritma *Logistic Regression* merupakan algoritma yang bersifat *parametric*. Algoritma *parametric* berarti algoritma atau model yang dibangun memiliki struktur yang tetap, sedangkan pada model *non-parametric* kompleksitasnya berubah-ubah sesuai dengan jumlah data yang diberikan kepadanya.
Secara *default*, model *Logistic Regression* tidak bisa digunakan untuk *Multiclass Classification*, sebaliknya untuk ID3 dan KNN. Namun, ada ekstensi lain untuk algoritma *Logistic Regression*, yaitu *One-vs-All Model* yang memungkinkan *Logistic Regression* untuk menangani kasus klasifikasi multikelas. Di samping itu, dari ketiga model ini, hanya KNN yang juga bisa digunakan untuk permasalahan regresi biasa, yaitu dengan mengubah *majority vote* menjadi rataan tetangga terdekat.
Di sisi lain, dapat dikatakan bahwa model *Logistic Regression* merupakan model yang paling cepat di antara ketiganya dalam melakukan prediksi tertentu. Hal ini dikarenakan pada kedua model lainnya melakukan perhitungan yang cukup berat dan iteratif sehingga secara komparatif, perhitungan *Logistic Regression* yang cukup sederhana membuat algoritma ini sangat cepat.
Algoritma KNN dan *Logistic Regression* merupakan algoritma yang bisa disuplai dengan fitur bertipe ordinal maupun nominal. Namun, algoritma ID3 biasanya hanya digunakan untuk fitur bertipe nominal dan tidak terlalu efektif untuk data ordinal. Selain itu, *Logistic*

Regression hanya akan efektif digunakan untuk data linear, tidak seperti KNN yang juga mendukung data non-linear.
Terakhir, algoritma *Logistic Regression* dan ID3 memiliki *confidence level* yang digunakan dalam penentuan prediksinya. Hal ini tidak dapat dilihat pada algoirtma KNN yang hanya mengeluarkan label saja.

Algoritma	Kelebihan	Kelemahan
K-Nearest Neighbor	1. Algoritma ini cukup mudah untuk diimplementasikan. 2. Merupakan *lazy learning algorithm*, tidak membutuhkan *training* sehingga lebih cepat dibandingkan algoritma lainnya yang perlu dilatih terlebih dahulu. 3. Karena tidak membutuhkan *training*, maka data baru bisa langsung ditambahkan. 4. Memiliki *hyperparameter* yang cukup sedikit untuk proses *tuning*.	1. Algoritma ini memiliki biaya yang tinggi untuk membuat prediksi tertentu. 2. Membutuhkan *scaling* yang baik uutnk setiap fitur sehingga perhitungan jarak yang dihasilkan tidak subjektif. 3. Sangat bergantung pada *hyperparameter* yang dipilih, dan cukup sulit melakukan perhitungan jarak untuk data dengan fitur kategorikal.
Logistic Regression	1. Algoritma ini cukup mudah untuk diimplementasikan. 2. Merupakan algoritma yang cukup cepat dalam hal *training* dan membuat prediksi data baru. 3. Karena adanya penentuan *weight* tiap *features*, maka bisa diketahui tingkat korelasi serta apakah korelasinya positif atau negatif dari fitur tersebut untuk setiap label.	1. Algoritma ini hanya bekerja secara efektif untuk data yang linear. 2. Mudah terkena *overfitting* untuk *high-dimensional data*, biasanya diminimalisir dengan regularisasi seperti pada SKLearn. 3. Sensitif terhadap *outliers*, membutuhkan seleksi fitur yang tepat serta jumlah data yang cukup banyak.
Iterative Dichotomiser 3	1. Algoritma ini bisa diaplikasikan untuk data dengan *missing values*, tetapi bisa menjadi kelemahan jika data yang diberikan terlalu *sparse*. 2. *Rules* yang dihasilkan dari algoritma ini mudah dimengerti layaknya algoritma pohon keputusan biasa.	1. Cenderung rentan terhadap *overfitting* dan membutuhkan data dalam jumlah yang cukup besar. 2. Kriteria *information gain* cenderung memilih fitur dengan nilai yang paling banyak, meskipun atribut tersebut belum tentu memiliki

	3. Menghasilkan pohon yang ukurannya tidak terlalu besar, meskipun belum tentu menjamin bahwa ukuran pohon adalah minimal.	pengaruh signifikan terhadap label. 3. Perlu dilakukan *pruning* agar prediksi tidak jauh melenceng. 4. Hanya menggunakan satu atribut saja ketika melakukan penentuan *rules*.

4. Jelaskan penerapan dari algoritma supervised! (misal di bidang kesehatan atau industri)

   Salah satu aplikasi *supervised learning* yang sekarang sedang ramai diperbincangkan adalah *image recognition*. Dengan menggunakan *supervised learning*, maka suatu gambar bisa dideteksi dengan *supervised learning*, bahkan dikelompokkan ke dalam kategori tertentu. Adapun pengembangan aplikasi ini biasanya menggunakan ilmu dalam bidang *computer vision* dan pengolahan citra untuk menganalisis suatu citra dan mendapatkan suatu *insight* tertentu.
   Selain itu, *supervised learning* juga bisa digunakan untuk melakukan *predictive analysis*. Dengan algoritma ini, maka bisa diprediksi nilai suatu hal pada masa depan berdasarkan data-data yang ada pada saat ini. Hasil prediksi bisa menjadi data baru, atau menjadi faktor pembantu dalam menarik suatu keputusan bisnis tertentu. Hasil prediksi juga bisa digunakan untuk mendapatkan *insight* tertentu mengenai bisnis atau hal lain yang diprediksi.
   Pada *email* juga bisa digunakan algoritma ini untuk mengelompokkan suatu *email*. Google, melalui *gmail*, menggunakan isi *email* untuk mempelajari pola dan kemungkinan anomali yang ada pada suatu *email*. Nantinya, model akan mengelompokkan *email* berdasarkan suatu kriteria tertentu, misalnya seberapa penting *email* tersebut untuk orang yang dituju, atau misalnya melakukan *spam detection*.
   Terakhir, aplikasi yang sudah pernah Saya implementasikan adalah *sentiment analysis*. *Sentiment Analysis* adalah proses mengelompokkan suatu *review* berdasarkan emosi dan konteks yang terdapat pada *review text* suatu *customer*. Nantinya, dari data yang diberikan, perusahaan bisa mengetahui letak kekurangan suatu produk berdasarkan hasil analisis ini, dan dengan demikian mengaplikasikan perbaikan servis atau produknya untuk mendapatkan *review* dan tingkat kepuasan konsumen yang lebih baik.

Referensi Lain

Comparative Study on Classic Machine learning Algorithms oleh Danny Varghese.
What is Supervised Learning? | IBM oleh IBM Cloud Education.

Perbedaan Supervised Learning and Unsupervised Learning oleh Universitas Ciputra.

Supervised and Unsupervised Machine Learning Algorithms oleh Jason Browniee.

Supervised Learning oleh David Petersson.

Supervised Machine Learning oleh JavaTPoint.

What is the Difference between A Parametric Learning Algorithm and A Nonparametric Learning Algorithm? oleh Sebastian Raschka.