Supervised Learning - Evaluation

by Thio Perdana

OUTLINE

Evaluasi Model Machine Learning

Evaluasi Model Klasifikasi

Evaluasi Model Regresi

Cross-Validation

Overfitting dan Underfitting

Improvisasi Model Machine Learning

Feature Extraction

Feature Selection

Hyperparameter Tuning

Evaluasi Model Machine Learning

Evaluasi Model Klasifikasi

Metrics Evaluasi Umum

Confusion Report adalah Laporan tertulis yang menggunakan Confusion Metrix sebagai basis dasar. Dengan Confusion Report kita bisa mengetahui banyak informasi terkait evaluasi model kita.

```
from sklearn.metrics import classification_report

y_true = [0, 1, 2, 2, 0]

y_pred = [0, 0, 2, 1, 0]

target_names = ['class 0', 'class 1', 'class 2']

print(classification_report(y_true, y_pred, target_names=target_names))
```

```
              precision    recall  f1-score   support

     class 0       0.67      1.00      0.80         2
     class 1       0.00      0.00      0.00         1
     class 2       1.00      0.50      0.67         2

    accuracy                           0.60         5
   macro avg       0.56      0.50      0.49         5

weighted avg       0.67      0.60      0.59         5
```

Sekarang akan kita coba bahas satu persatu terkait parameter yang ada pada confusion report

Precision Mengembalikan nilai benar suatu label terhadap semua perkiraan dari label tersebut (baik benar atau salah)

$$TP/(TP + FP)$$

Recall Mengambalikan nilai benar dari suatu label terhadap semua perkiraan benar dari label (termasuk perkiraan dari label lain yang mengarah ke label tersebut)

TP/(TP + FN)

Accuracy Mengembalikan Nilai Benar dari semua label dibandingkan dengan keseluruhan pelabelan. Accuracy juga menentukan nilai error dari model kita

(TP + TN)/(TP + TN + FP + FN)

F1-Score Nilai yang menggambarkan harmoni antara precision dan recall. Semakin besar nilai f1-score artinya semakin besar pula nilai dari precison dan recall.

(2 x Precision x Recall)/(Precision + Recall)

Support Jumlah observasi/data dari setiap label

Macro avg Nilai rata-rata dari total label

Contohnya jika kita menghitung nilai dari macro average precision di atas,

(0,67+0+1)/3

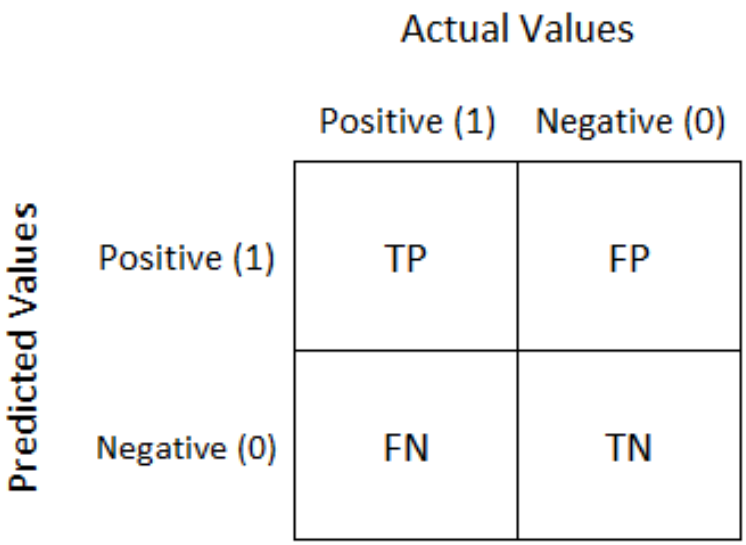
Weighted avg Nilai rata-rata yang memperhitungkan jumlah observasi/data(support)

Contohnya jika kita menghitung nilai dari weighted average precision di atas,

[(0,67*2)+(0*1)+(1*2)]/(2+1+2)

Confusion Matrix

Confusion matrix digunakan untuk mengevaluasi performa model terhadap data tes yang telah diketahui nilai sebenarnya. Confusion matriks berisi jumlah nilai benar dan salah tiap kelas yang ditunjukkan pada gambar seperti berikut:



Dari gambar diatas terdapat 2 nilai yaitu nilai prediksi dan nilai sebenarnya.

- TP = True Positif, dimana hasil prediksi kita bernilai Positif dan nilai sebenarnya juga bernilai positif
- FP = False Positif, dimana hasil prediksi kita positif tetapi nilai sebenarnya adalah negatif
- FN = False Negatif, dimana hasil prediksi kita negatif tetapi nilai sebenarnya adalah positif
- TN = True Negatif, dimana hasil prediksi kita negatif dan nilai sebenarnya adalah negatif

AUC - ROV Curve

Area Di Bawah Kurva-Characteristic Receiver Operating Characteristic (AUC-ROC) adalah metode pengukuran performa untuk masalah klasifikasi pada berbagai ambang batas. Kurva ROC merepresentasikan probabilitas, dan nilai AUC mencerminkan seberapa baik model mampu memisahkan kelas. Semakin tinggi nilai AUC, semakin baik model dalam memprediksi kelas 0 dan 1. Dengan analogi,

semakin tinggi nilai AUC, semakin baik model membedakan antara pasien dengan penyakit dan tanpa penyakit.

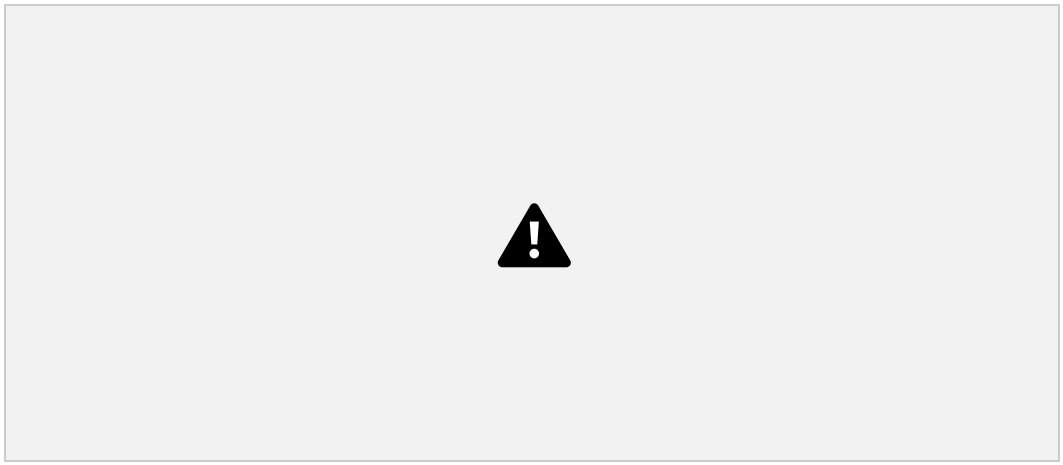
Kurva ROC digambarkan dengan membandingkan Tingkat Positif Benar (TPR) dengan Tingkat Positif Palsu (FPR), di mana TPR berada di sumbu y dan FPR berada di sumbu x. Penekanan pada AUC-ROC adalah kemampuan model dalam membedakan antara hasil positif sejati dan hasil positif palsu.

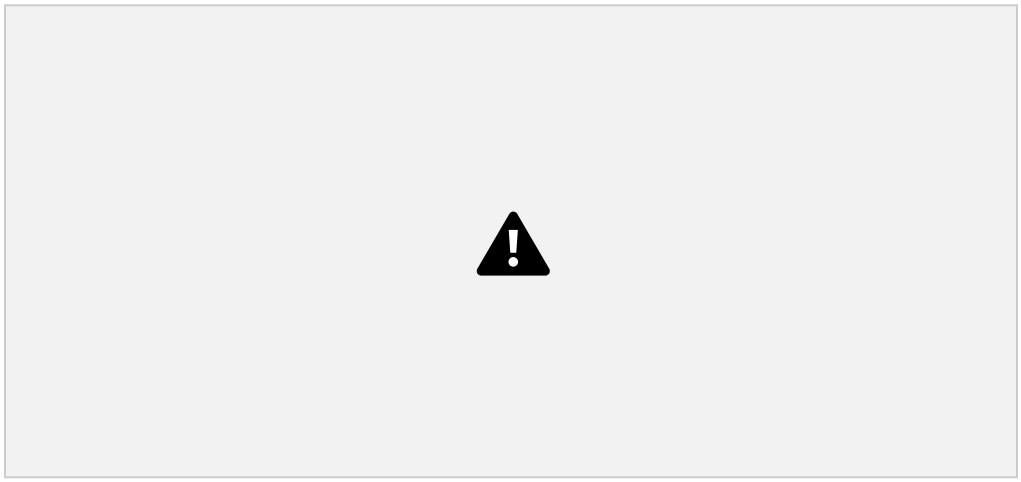
Sebuah model yang sangat baik memiliki nilai AUC mendekati 1, yang menunjukkan kemampuan yang baik dalam memisahkan kelas. Sebuah model yang buruk memiliki nilai AUC mendekati 0, yang berarti memiliki kemampuan pemisahan yang buruk. Bahkan, hal ini berarti model tersebut secara terbalik memprediksi hasil, yakni memprediksi 0 sebagai 1 dan 1 sebagai 0. Ketika AUC adalah 0,5, itu menandakan bahwa model tersebut sama sekali tidak mampu memisahkan kelas.

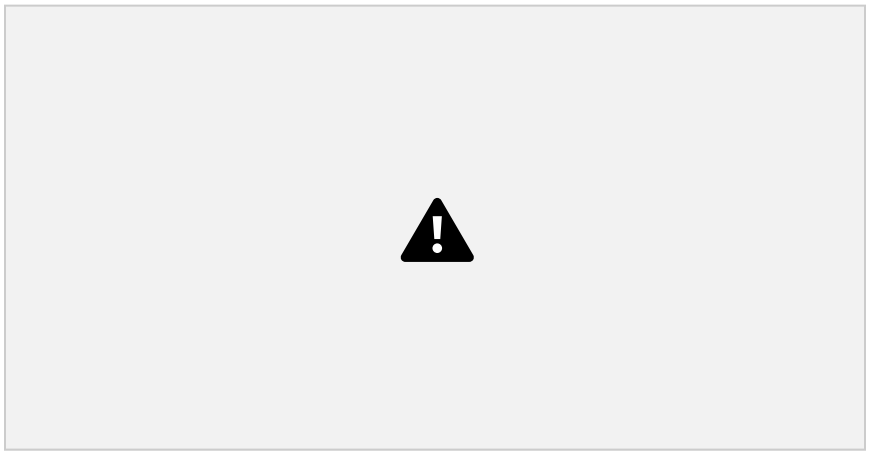
Mari kita terjemahkan pernyataan di atas.

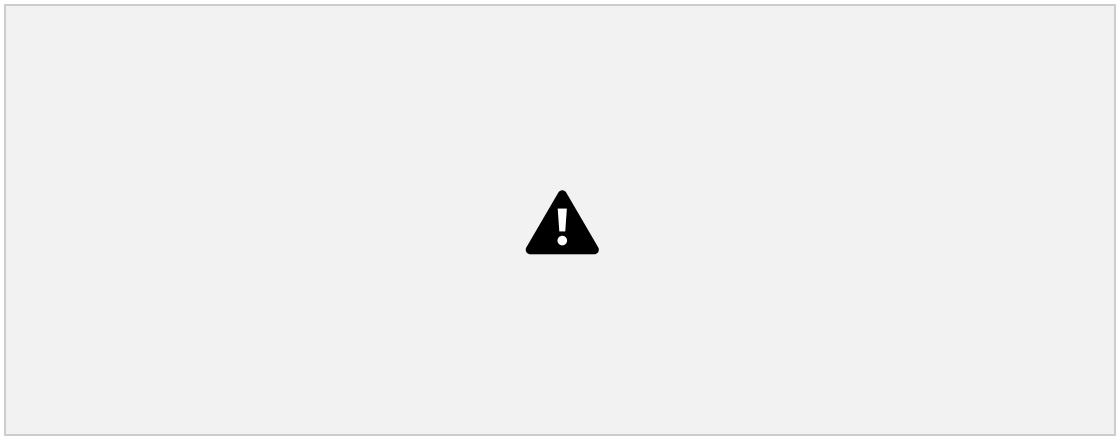
Seperti yang kita tahu, ROC adalah kurva probabilitas. Jadi, mari kita gambarkan distribusi probabilitas tersebut:

Catatan: Kurva distribusi merah merupakan kelas positif (pasien dengan penyakit) dan kurva distribusi hijau merupakan kelas negatif (pasien tanpa penyakit).









Evaluasi Model Regresi

Metrics Evaluasi Umum

MAE

MAE adalah kepanjangan dari Mean Absolute Error. MAE digunakan untuk mengukur keakuratan suatu model statistik dalam melakukan prediksi atau peramalan.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}\left|\hat{y}_i - y_i\right|$$

$\hat{y}_i$ = *predicted value*

y_i = *actual value*

n = *# of observations*

```
from sklearn.metrics import mean_absolute_error

y_true = [3, -0.5, 2, 7]

y_pred = [2.5, 0.0, 2, 8]

mean_absolute_error(y_true, y_pred)

0.5

y_true = [[0.5, 1], [-1, 1], [7, -6]]

y_pred = [[0, 2], [-1, 2], [8, -5]]

mean_absolute_error(y_true, y_pred)

0.75

mean_absolute_error(y_true, y_pred, multioutput='raw_values')

array([0.5, 1. ])

mean_absolute_error(y_true, y_pred, multioutput=[0.3, 0.7])

0.85...
```

MSE

MSE adalah kepanjangan dari Mean Squared Error. Digunakan untuk mengecek estimasi berapa nilai kesalahan pada peramalan. Nilai Mean Squared Error yang rendah atau nilai mean squared error

mendekati nol menunjukkan bahwa hasil peramalan sesuai dengan data aktual dan bisa dijadikan untuk perhitungan peramalan di periode mendatang.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$y_i = actual\ value$$
$$\hat{y}_i = predicted\ value$$
$$n = \#\ of\ observations$$

```
from sklearn.metrics import mean_squared_error
y_true = [3, -0.5, 2, 7]
y_pred = [2.5, 0.0, 2, 8]
mean_squared_error(y_true, y_pred)
0.375
y_true = [[0.5, 1], [-1, 1], [7, -6]]
y_pred = [[0, 2], [-1, 2], [8, -5]]
mean_squared_error(y_true, y_pred)
0.7083...
```

R2

Sering disebut sebagai Coefficient of Determination, merupakan representasi dari variabel y terhadap variabel bebas di dalam model (x). Digunakan untuk melihat seberapa baik model dalam memprediksi sebuah data di masa depan.

```
from sklearn.metrics import r2_score
```

```
y_true = [3, -0.5, 2, 7]

y_pred = [2.5, 0.0, 2, 8]

r2_score(y_true, y_pred)

0.948...
```

Cross-Validation

Melakukan evaluasi pada model machine learning merupakan hal yang penting tapi kadang sedikit tricky. Pada umumnya kita membagi data kita menjadi dua komponen utama yaitu data training dan data testing. Data training merupakan data yang digunakan untuk melatih model kita agar dapat mengetahui karakteristik dari data yang kita miliki. Data yang kita gunakan untuk training haruslah data minim noise yang jumlahnya harus cukup banyak. Pada mayoritas kasus, semakin banyak data maka akan semakin baik. Data testing merupakan data yang digunakan untuk menguji performa dan akurasi dari model kita. Untuk pembagiannya sendiri, biasanya dalam rentang 70% - 80% untuk data training dan sisanya digunakan untuk data testing.

Masalahnya adalah pengambilan data saat kita melakukan pembagian menjadi sangat penting karena akan menentukan performa dari data kita.

Misalkan kita mempunyai 10 data dengan dua label A dan B :

```
[A, A, A, A, A, B, B, B, B, B]
```

Lalu kita membaginya dengan proporsi 80 % data training dan 20% data testing. Idealnya adalah kita mendapatkan 4 data berlabel A dan 4 data berlabel B sehingga model kita dapat belajar karakteristik data A dan B secara seragam.

```
Data Training = [A,A,A,A,B,B,B,B]
```

```
Data Testing = [A,B]
```

Pada kenyataannya ini tidak bisa selalu terjadi, untuk mengikuti karakteristik data di dunia nyata maka kita biasanya melakukan pengacakan dalam hal ini dan biasanya pengacakan tidak menghasilkan data yang ideal.

```
Data Training = [A,A,A,A,A,B,B,B]
```

```
Data Testing = [B,B]
```

Pada kasus diatas kita, model kita akan lebih terbiasa pada data berlabel A daripada data berlabel B sehingga model kita akan memiliki kecenderungan pada data berlabel A. Masalah lainnya adalah karena data yang tersisa adalah data B sehingga model kita tidak memiliki hasil evaluasi model kita terhadap data A.

Bagaimana solusinya?

Solusi termudah adalah dengan memperbanyak data testing yang kita miliki sehingga kemungkinan ini terjadi semakin kecil karena tingkat kepadatan data yang tinggi. Akan tetapi, masalah lainnya adalah karena ini belum selalu bisa dilakukan. Misalkan kita tidak dapat menambah data kita karena terbatasnya cara untuk mendapatkan data tersebut, maka kita bisa menambah validitas model kita dengan menambah rasio data testing terhadap data training. Optimalnya? tentu saja 100%. Akan tetapi, masalah lain muncul karena kita tidak memiliki data testing untuk menguji model kita.

Bagaimana jika kita tidak dapat melakukan penambahan data karena alasan apapun. Lalu kita tetap ingin menggunakan seluruh data sebagai data training dan tetap memiliki data untuk testing ? Solusinya adalah ***Cross Validation***.

Data

Bayangkan jika gambar di atas adalah data kita, pada umumnya kita akan membagi data tersebut menjadi empat bagian, dimana 3 bagian dibagi menjadi data training (75%) dan 1 bagian menjadi data testing (25%). Masalahnya adalah hasil dapat berbeda jika kita menggunakan bagian 1,2,3 atau 1, 2, 4 sebagai data training. Daripada kebingungan akan hal tersebut, cross validation memberikan kita kemampuan untuk menggunakan semua bagian untuk testing dan training seperti pada gambar di bawah ini.

cross validation method

Bagian berwarna kuning merupakan data testing dan bagian berwarna biru merupakan data training. Hasil dari perhitungan tersebut akan dijumlahkan dan dicari nilai rata-ratanya sebagai nilai utama.

Saat kita membagi data menjadi 4 bagian seperti contoh di atas itu dinamakan four fold cross validation, jika kita menggunakan 10 bagian maka dinamakan ten fold cross validation. Karena itu jika kita menggunakan K bagian maka dinamakan ***K-fold cross validation***.

K-Fold Cross Validation

Dengan memanfaatkan library scikit learn kita memiliki pilhan untuk melakukan k-fold cross validation pada berbagai model supervised learning. Pada kesempatan kali ini kita asumsikan kita menggunakan metode K-NN.

Beberapa langkah yang harus dilakukan adalah:

Import Library yang dibutuhkan

```
from sklearn.model_selection import KFold
from sklearn.neighbors import KNeighborsClassifier

import numpy as np
```

KFlod digunakan untuk menjalankan proses pembagian data kita dan *KNeighborsClassifier* merupakan metode KNN kita. Jika kita membutuhkan Library lain maka kita bisa menambah library sesuai kebutuhan.

Buka dan Baca data set kita

Lakukan pra-proses data pada dataset kita

Kita bersihkan dan kita masukkan pada variabel x dan y

```
X = [data feature]

y = [data label]
```

Lakukan kfold

```
scores = []
```

```
model_k = KNeighborsClassifier(n_neighbors=5)
kf = KFold(n_splits=10, random_state=42, shuffle=False)
for train_index, test_index in kf.split(X):

    X_train, X_test, y_train, y_test = X[train_index], X[test_index], y[train_index], y[test_index]
    model_k.fit(X_train, y_train)

    scores.append(model_k.score(X_test, y_test))
```

Kita dapat melakukan k-fold cross validation dengan kode di atas. n_splits menunjukan berapa data kita ingin dibagi (pada contoh di atas adalah 10 bagian).

best_knn.fit(X_train, y_train) adalah kode untuk menggunakan data kita ke dalam metode knn.

scores.append(model_k.score(X_test, y_test)) adalah kode yang digunakan untuk melihat akurasi dari model kita. untuk mendapatkan nilai rata-ratanya kita dapat menggunakan.

```
print(np.mean(scores))
```

K-Fold Cross Validation (Easy Way)

Terdapat cara lebih mudah untuk menjalan kfold cross validation. Kita dapat memanfaatkan metode cross_val_score pada library scikit learn.

```
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier

knn= KNeighborsClassifier(n_neighbors=5)
score= cross_val_score(knn, x, y, cv=10, scoring='accuracy')
print(score)

print(score.mean())
```

Overfitting dan Underfitting

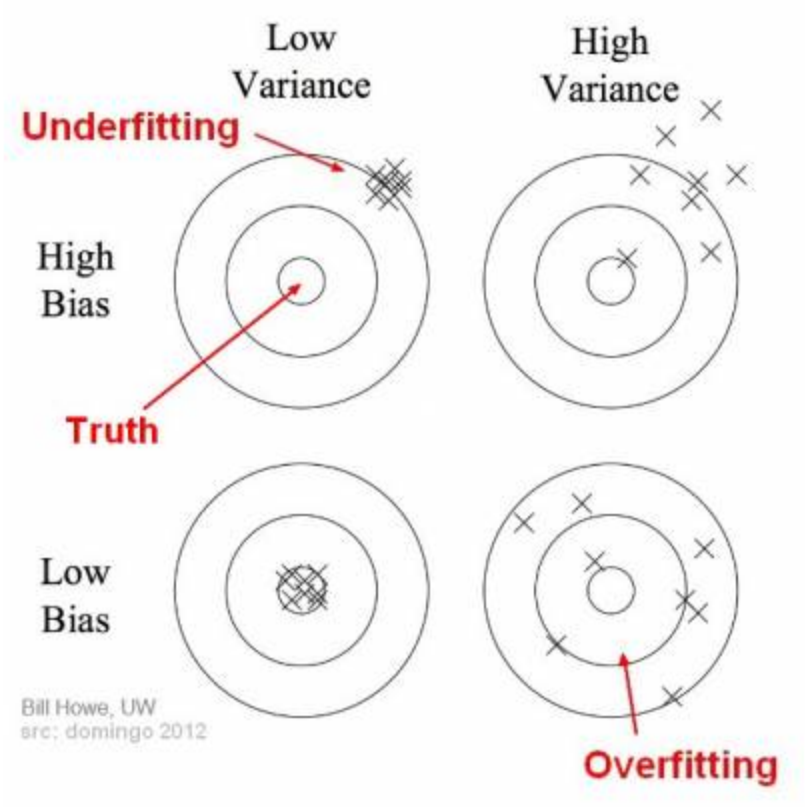
Bias adalah perbedaan antara rata-rata hasil prediksi model kita dengan nilai yang sebenarnya. Model yang memiliki bias yang tinggi, sedikit belajar terhadap data latih dan ini akan membuat error yang tinggi terhadap data training dan data set.

Variance merupakan variability prediksi model kita terhadap data point yang menunjukkan sebaran datanya. Model dengan variance yang tinggi, banyak belajar terhadap data latih dan sedikit terhadap data yang belum pernah ia lihat. Performa modelnya bagus terhadap data latih dan error yang tinggi terhadap data test.

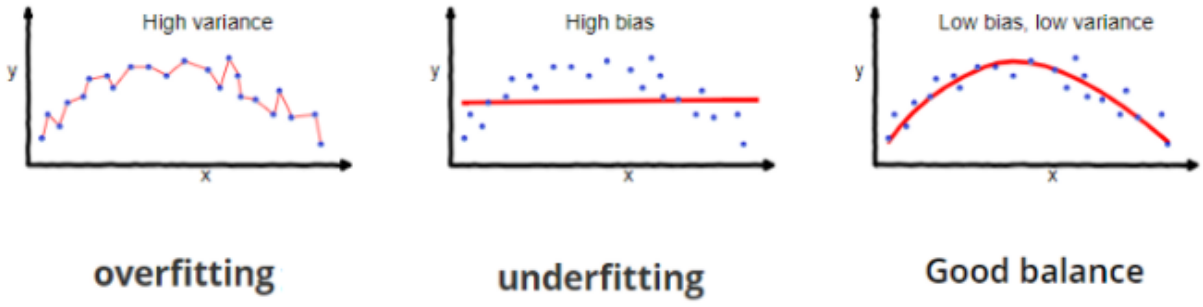
Dalam ML, terdapat error yang didefinisikan seperti berikut:

$Err(x) = Bias^2 + Variance + Irreducable\ error$

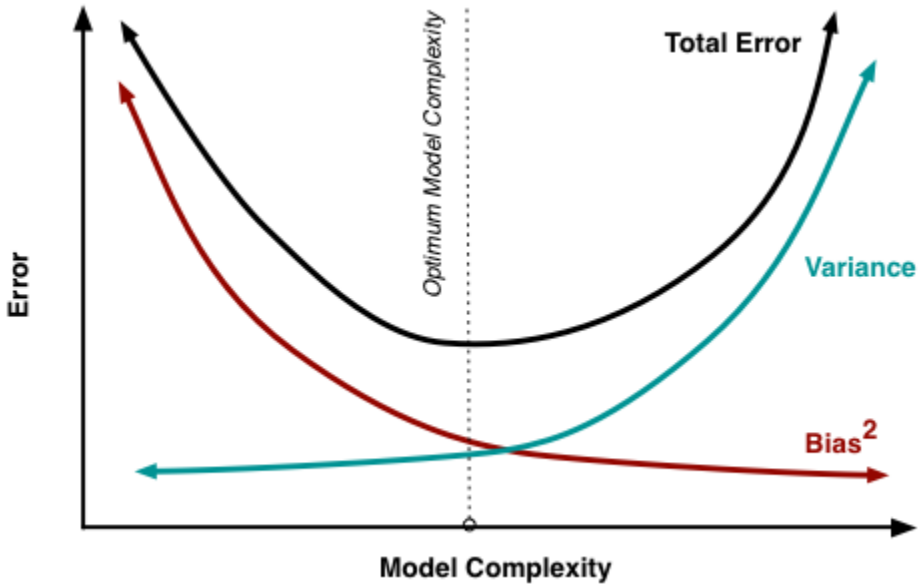
- Error Bias memiliki error yang tinggi terhadap data training dan data test
- Error Variance memiliki error yang tinggi terhadap data training dan rendah terhadap data test
- Irreducable error merupakan error yang tidak bisa sikurangu dengan membuat model yang baik. Biasanya error ini merepresentasikan jumlah noise dari data kita.



Dalam supervised learning, **Underfitting** terjadi ketika model tidak mampu membaca data pattern. Biasanya, model-model ini memiliki Bias yang tinggi dan Variance yang rendah. Hal ini terjadi ketika kita memiliki sedikit data dan ingin membuat model yang akurat, atau kita mencoba membangun algoritma model linear untuk data yang bukan linear, atau kita menggunakan algoritma model yang simple untuk membaca data yang kompleks, seperti linear dan logistic regression. Sedangkan **Overfitting** terjadi ketika model membaca noise dalam data kita, hal ini terjadi ketika kita memerintahkan model untuk mempelajari data yang memiliki banyak noise. Model ini memiliki Bias yang sangat rendah dan Variance yang tinggi dan model yang kompleks seperti decision tree yang sering overfitting.



Lalu bagaimana kita memilih model yang baik? Model yang baik model yang mampu menemukan trade-off dari Bias dan Variance. Mari kita lihat diagram berikut:



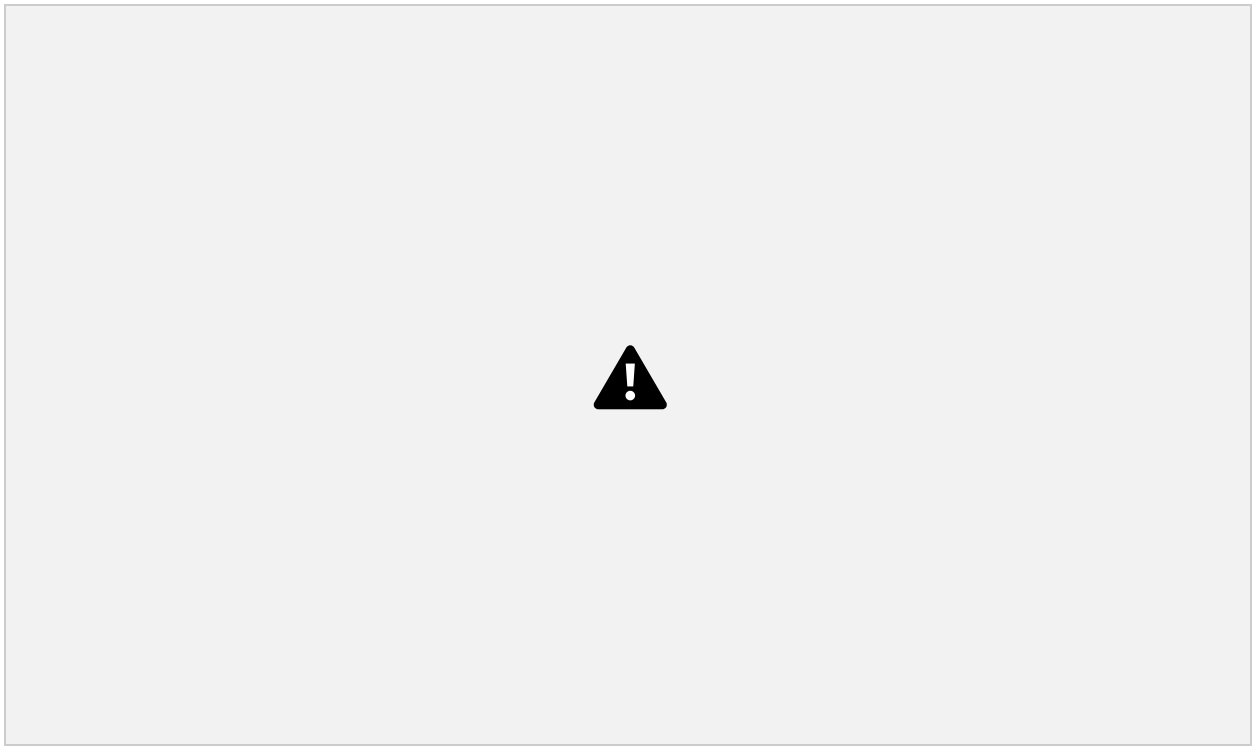
Dari grafik diatas terlihat bahwa jika kita memiliki model yang terlalu simple, maka akan memiliki error Bias yang tinggi dan error Variance yang rendah. Jika kita memiliki model yang terlalu kompleks maka akan memiliki error Variance yang tinggi dan error Bias yang rendah. Maka model yang baik adalah model yang mempu menemukan trade-off dari kedua error tersebut, artinya menemukan keseimbangan antara error Bias dan error Variance sehingga tidak terjadi overfitting maupun underfitting.

Improvisasi Model Machine Learning

Feature Extraction

Principal Component Analysis (PCA)

Principal Component Analysis atau PCA adalah metode reduksi dimensionalitas yang umum digunakan. PCA bekerja dengan menghitung komponen utama dan melakukan perubahan basis. Ini mempertahankan data dalam arah varians maksimum. Komponen utama adalah kunci dalam PCA; mereka mewakili apa yang ada di dalam data Anda. Dalam istilah yang lebih sederhana, ketika data diproyeksikan ke dimensi yang lebih rendah (misalnya tiga dimensi) dari ruang yang lebih tinggi, tiga dimensi tersebut tidak lebih dari tiga Komponen Utama yang menangkap sebagian besar varians (informasi) dari data Anda.



Komponen utama memiliki arah dan magnitudo. Arah mewakili sepanjang sumbu utama mana data sebagian besar tersebar atau memiliki sebagian besar varians, dan magnitudo menunjukkan jumlah varians yang Komponen Utama tangkap dari data ketika diproyeksikan ke sumbu tersebut. Komponen utama adalah garis lurus, dan komponen utama pertama menyimpan sebagian besar varians dalam data. Setiap komponen utama berikutnya bersifat ortogonal terhadap yang terakhir dan memiliki varians yang lebih kecil. Dengan cara ini, dengan kumpulan variabel x yang berkorelasi di atas y sampel, Anda mencapai seperangkat u komponen utama yang tidak berkorelasi di atas sampel y yang sama.

Fitur yang telah direduksi tidak berkorelasi satu sama lain. Fitur-fitur ini dapat digunakan untuk clustering dan klasifikasi tanpa supervisi. Untuk mereduksi dimensionalitas, autoencoder adalah metode lain yang umum digunakan. Namun, ruang laten dari autoencoder tidak selalu tidak berkorelasi. Sementara PCA menjamin bahwa semua fitur tidak berkorelasi satu sama lain. Pertama, PCA menghitung matriks kovarians. Kemudian kita menemukan vektor eigen dan nilai eigen dari matriks kovarians tersebut.

Setelah itu, kita proyeksikan data sepanjang vektor eigen. Jika dimensi data asli adalah n, kita dapat mereduksi dimensi menjadi k, dengan syarat $k \leq n$.

Langkah-Langkah PCA

a. Standarisasi Data: Langkah pertama dalam PCA adalah standarisasi data untuk menghilangkan skala yang berbeda antar variabel.

b. Hitung Matriks Kovarians: Menghitung matriks kovarians dari data standar untuk mengevaluasi hubungan antar variabel.

c. Hitung Vektor dan Nilai Eigen: Menghitung vektor dan nilai eigen dari matriks kovarians. Vektor eigen mewakili arah utama variabilitas, sedangkan nilai eigen menunjukkan seberapa besar variabilitas dalam arah tersebut.

d. Pilih Komponen Utama: Memilih sejumlah komponen utama berdasarkan nilai eigen yang menjelaskan sebagian besar variabilitas data.

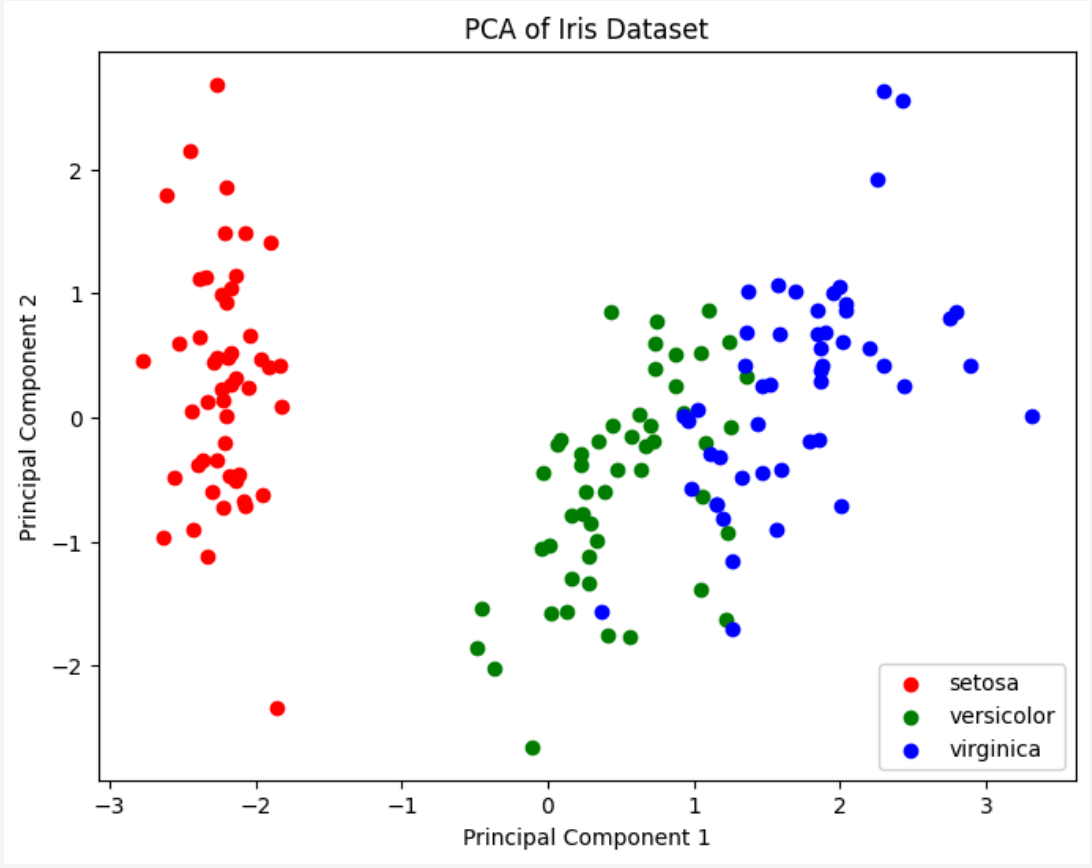
e. Transformasi Data: Menggunakan vektor eigen yang dipilih untuk mentransformasi data asli ke dalam ruang komponen utama.

Contoh Bekerja dengan Python

```
# Impor library

import numpy as np

from sklearn.decomposition import PCA

from sklearn.datasets import load_iris

import matplotlib.pyplot as plt



# Muat dataset iris sebagai contoh

iris = load_iris()
```

```
X = iris.data

y = iris.target


# Standarisasi data

from sklearn.preprocessing import StandardScaler

X_std = StandardScaler().fit_transform(X)


# Inisialisasi objek PCA dengan jumlah komponen utama yang diinginkan

pca = PCA(n_components=2)


# Terapkan PCA ke data

X_pca = pca.fit_transform(X_std)


# Visualisasi hasil PCA

plt.figure(figsize=(8, 6))


for i, c in zip(range(3), ['red', 'green', 'blue']):

    plt.scatter(X_pca[y == i, 0], X_pca[y == i, 1], c=c, label=iris.target_names[i])


plt.title('PCA of Iris Dataset')

plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2')

plt.legend()
```

```
plt.show()
```



Keuntungan dan Kerugian PCA

Keuntungan

Reduksi Dimensi: PCA membantu mengurangi dimensi data dengan mempertahankan sebagian besar variabilitas.

Menghilangkan Korelasi: Komponen utama bersifat tidak berkorelasi, menghilangkan masalah multikolinearitas.

Visualisasi Data: Memudahkan visualisasi data dalam ruang yang lebih rendah.

Mempercepat Algoritma: Mengurangi dimensi dapat mempercepat kinerja algoritma machine learning.

Kerugian

Kehilangan Interpretasi Variabel: Setelah transformasi, interpretasi variabel menjadi lebih sulit.

Sensitif terhadap Skala: PCA sangat sensitif terhadap perbedaan skala antar variabel.

Asumsi Linearitas: Membutuhkan asumsi bahwa hubungan antar variabel adalah linear.

b. t-Distributed Stochastic Neighbor Embedding (t-SNE)

2. Cara Kerja t-SNE

3. Penerapan t-SNE dalam Machine Learning

c. Autoencoders

2. Struktur Autoencoders

3. Penerapan Autoencoders dalam Feature Extraction

Feature Selection

Feature Selection digunakan untuk memilah fitur-fitur yang akan kita gunakan pada model kita. Kita sering dihadapkan pada dataset yang memiliki banyak fitur di dunia nyata, akan tetapi tidak berarti semakin banyak fitur akan selalu membuat model kita semakin akurat. Tidak semua fitur berpengaruh besar dalam pembuatan model, fitur yang tidak relevan hanya akan menjadi noise dan mengurangi keakuratan model kita. Selain untuk menambah akurasi, fitur selection juga bisa juga dipakai untuk mengurangi overfitting dan beban komputasi dengan berkurangnya jumlah data yang harus diolah. Salah satu metode yang bisa digunakan untuk melakukan feature selection adalah Recursive Feature Elimination (RFE).

Cara kerja dari metode RFE ini adalah dengan menghilangkan fitur satu persatu dan secara rekursif akan membangun model. lalu mengevaluasi keluaran model berdasarkan feature yang tersisa. Ini dilakukan berkali-kali hingga ditemukan kombinasi fitur mana saja yang akan menghasilkan kontribusi maksimal dari model tersebut.

RFE pada python bisa dilakukan dengan memanfaatkan modul RFE() seperti yang diperlihatkan pada contoh

```
from sklearn.feature_selection import RFE
```

```
rfe = RFE(model) #model bisa diisi dengan objek algoritma machine learning yang kita akan kerjakan, misalnya LogisticRegression()

rfe_value = rfe.fit(X_train, y_train)
```

```
print('Support=', rfe.support_)

print('Ranking=', rfe.ranking_)
```

Hyperparameter Tuning

Ketika Anda melakukan pelatihan model machine learning, setiap set data dan model memerlukan kumpulan hyperparameter yang berbeda, yang berperan sebagai jenis variabel tertentu. Menentukan hyperparameter ini hanya dapat dilakukan melalui serangkaian eksperimen, di mana Anda memilih suatu set hyperparameter dan menjalankannya melalui model. Proses ini dikenal sebagai penyetelan hyperparameter. Pada dasarnya, Anda melatih model secara berurutan dengan berbagai set hyperparameter. Penyetelan ini dapat dilakukan secara manual atau menggunakan salah satu dari beberapa metode penyetelan hyperparameter otomatis yang tersedia.

Bagaimanapun metodenya, penting untuk melacak hasil eksperimen tersebut. Analisis statistik, seperti fungsi kerugian, harus diterapkan untuk menentukan set hyperparameter yang memberikan hasil terbaik. Penyetelan hyperparameter merupakan proses yang krusial dan membutuhkan sumber daya komputasi yang signifikan.

Hyperparameter, yang merupakan variabel konfigurasi eksternal, digunakan oleh ilmuwan data untuk mengelola pelatihan model machine learning. Disebut juga sebagai hyperparameter model, hyperparameter ini diatur secara manual sebelum proses pelatihan dimulai. Perlu dicatat bahwa hyperparameter model berbeda dari parameter, yang merupakan nilai internal yang secara otomatis disesuaikan selama proses pembelajaran dan tidak diatur oleh ilmuwan data.

Contoh hyperparameter mencakup jumlah simpul dan lapisan dalam jaringan neural, serta jumlah cabang dalam pohon keputusan. Hyperparameter ini menentukan fitur utama, seperti arsitektur model, tingkat pembelajaran, dan kompleksitas model.

Penting untuk memilih set hyperparameter yang sesuai untuk mencapai performa dan akurasi model yang optimal. Sayangnya, tidak ada aturan baku tentang hyperparameter mana yang bekerja paling baik atau nilai default atau optimalnya. Oleh karena itu, eksperimen diperlukan untuk menemukan set hyperparameter yang optimal, suatu kegiatan yang dikenal sebagai penyetelan hyperparameter atau optimisasi hyperparameter.

Hyperparameter memiliki pengaruh langsung terhadap struktur, fungsi, dan performa model. Penyetelan hyperparameter memungkinkan ilmuwan data untuk mengoptimalkan performa model sesuai kebutuhan. Proses ini menjadi bagian yang sangat penting dari machine learning, dan pemilihan nilai hyperparameter yang tepat menjadi kunci keberhasilan.

Sebagai contoh, anggaplah tingkat pembelajaran sebagai hyperparameter model. Jika nilainya terlalu tinggi, model dapat konvergen terlalu cepat dengan hasil yang suboptimal. Sebaliknya, jika tingkatnya terlalu rendah, pelatihan memakan waktu terlalu lama dan hasilnya mungkin tidak memuaskan. Oleh karena itu, pemilihan hyperparameter yang bijaksana dan seimbang dapat menghasilkan model yang akurat dan unggul dalam kinerja.

Metode Tuning Hyperparameter

Manual Tuning: Metode ini melibatkan penyesuaian hyperparameter secara manual berdasarkan pengalaman dan intuisi. Seorang praktisi machine learning akan menguji berbagai nilai hyperparameter secara berulang untuk mencari konfigurasi yang memberikan hasil terbaik.

Grid Search: Grid Search mencakup definisi kumpulan nilai yang mungkin untuk setiap hyperparameter yang akan diuji. Selanjutnya, model akan dievaluasi untuk setiap kombinasi nilai hyperparameter dalam kisi tersebut. Ini dapat membantu menemukan kombinasi hyperparameter yang memberikan kinerja optimal, tetapi dapat menjadi mahal secara komputasi karena menguji semua kombinasi.

Random Search: Random Search, sebaliknya, secara acak memilih set nilai hyperparameter untuk diuji. Meskipun ini tidak menjamin pencarian yang optimal, Random Search dapat lebih efisien secara komputasi dibandingkan Grid Search, terutama jika ruang hyperparameter besar.

Contoh Kode dengan Python

```
import seaborn as sns

from sklearn.model_selection import GridSearchCV

from sklearn.svm import SVC


# Load dataset

iris = sns.load_dataset('iris')


# Pisahkan fitur dan target

X = iris.drop('species', axis=1)

y = iris['species']


# Definisikan model SVC

model = SVC()


# Tentukan kumpulan nilai hyperparameter yang akan diuji

param_grid = {'C': [0.1, 1, 10], 'gamma': [0.01, 0.1, 1]}


# Gunakan GridSearchCV untuk mencari kombinasi hyperparameter terbaik
```

```
grid_search = GridSearchCV(model, param_grid, cv=5)

grid_search.fit(X, y)



# Cetak hyperparameter terbaik

print("Hyperparameter Terbaik:", grid_search.best_params_)
```

Referensi Eksternal

- Evaluasi Model Cross Validation
- Confusion Matrix
- Metriks ROC AUC
- Confusion Matrix dan ROC AUC
- Bias dan Variance
- Overfitting dan Underfitting
- Overfitting dan Underfitting 2