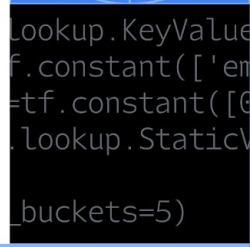
Google Developer Groups

Study Jam Machine Learning 7: Training & Evaluation



Google Developer Groups

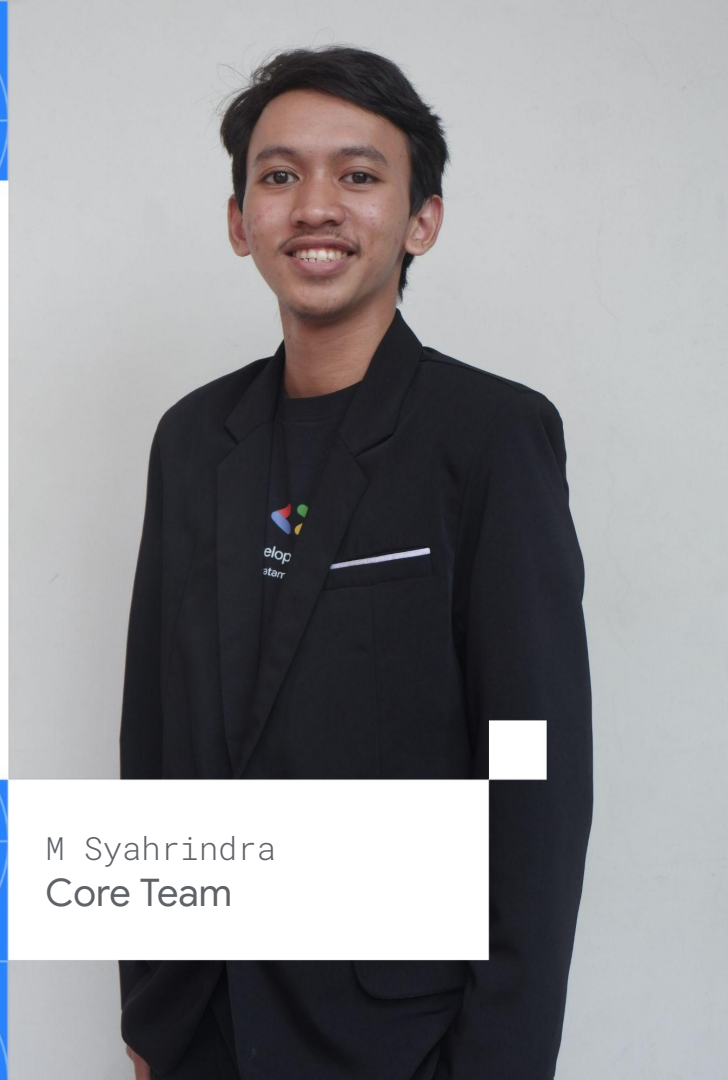
On Campus Widyatama University

- **Co-Lead GDGoC Widyatama University**
- **Informatics student at Widyatama University**
- **Bangkit Alumni 2024 batch 1**

instagram.com/m.syahrindra

linkedin.com/in/m-syahrindra/

github.com/syahrindra



M Syahrindra

Core Team

Outline

1. Split, Train, and Validate: Train-Test Split and Cross-Validation
2. Understand the Bias-Variance Tradeoff
3. What's Worse? Overfitting or Underfitting?
4. Learning rate
5. Loss
6. Optimizer
7. Tuning Your Hyperparameters
8. Accuracy, Pression, and F1-score

Split, Train and Validate: Train-Test Split and Cross-validation

Train-Test Split

Konsep sederhana dimana kita membagi dataset menjadi dua bagian:

- Train data -> Data yang digunakan untuk melatih model
- Test data -> Data yang digunakan untuk menguji model setelah selesai dilatih

Biasanya, pembagian dilakukan dengan rasio 80% train - 20% test atau 70% train - 30% test.

Contoh Implementasi

```
from sklearn.model_selection import
train_test_split
from sklearn.datasets import load_iris
from sklearn.ensemble import
RandomForestClassifier
from sklearn.metrics import accuracy_score

# Load Dataset
data = load_iris()
X, y = data.data, data.target
```

```
# Split Data into Train and Test Sets
X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.3,
random_state=42)

# Model Training
model = RandomForestClassifier()
model.fit(X_train, y_train)

# Model Evaluation
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

Cross-Validation (CV)

Cross-validation digunakan untuk mengatasi kelemahan train-test split dengan membagi data menjadi beberapa bagian dan melatih model beberapa kali.

K-Fold Cross-Validation (paling umum)

- Data dibagi menjadi K bagian (misalnya, 5 bagian).
- Model dilatih menggunakan K-1 bagian dan diuji dengan bagian yang tersisa.
- Proses diulang K kali, sehingga setiap bagian pernah menjadi test set.
- Hasil akhirnya dirata-rata untuk mendapatkan skor yang lebih stabil.

K-Fold Cross Validation

Let no of observations be 1000

sumber

Contoh Implementasi

```
from sklearn.model_selection import
cross_val_score


# Cross-Validation with K=5

scores = cross_val_score(model, X, y, cv=5)

print(f"Cross-Validation Scores: {scores}")

print(f"Mean Score: {scores.mean():.2f}")
```

Understand the Bias-Variance Tradeoff

- **Bias ->** Model terlalu sederhana, tidak cukup belajar dari data. (**Underfitting**)
- **Variance ->** Model terlalu kompleks, terlalu peka terhadap data latih (Overfitting)

Bias-Variance Tradeoff

Ada keseimbangan antara bias dan variance:

- **Bias tinggi (Underfitting)** → Model tidak cukup belajar dari data, prediksinya buruk.
- **Variance tinggi (Overfitting)** → Model terlalu hafal data latih, tapi buruk di data baru.
- **Ideal** → Model dengan keseimbangan bias dan variance, yang bisa generalisasi dengan baik.

Analoginya,

Bayangkan kamu belajar bermain catur:

- Bias tinggi (Underfitting) → Kamu hanya menghafal aturan dasar, tapi tidak bisa menang karena strategimu terlalu lemah.
- Variance tinggi (Overfitting) → Kamu menghafal setiap langkah dari satu lawan, tapi saat main dengan orang lain, kamu kalah karena strateginya berbeda.
- Ideal → Kamu memahami prinsip dasar catur dan bisa beradaptasi dengan berbagai lawan.

Contoh Implementasi

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.preprocessing import
PolynomialFeatures
from sklearn.linear_model import
LinearRegression
from sklearn.pipeline import make_pipeline
from sklearn.metrics import
mean_squared_error

# Sample Data
np.random.seed(42)
X = np.random.rand(100, 1) * 10
y = 2 * X.squeeze() + 1 +
np.random.randn(100) * 2
```

```
# Polynomial Regression with Different Degrees
train errors, test errors = [], []
degrees = range(1, 10)

for degree in degrees:
   model = make pipeline(PolynomialFeatures(degree),
LinearRegression())
   model.fit(X, y)

   y_train_pred = model.predict(X)
   train errors.append(mean_squared_error(y,
y_train_pred))

   # Simulate test data
   X test = np.random.rand(50, 1) * 10
   y_test = 2 * X_test.squeeze() + 1 +
np.random.randn(50) * 2
   y test pred = model.predict(X test)
   test_errors.append(mean_squared_error(y_test,
y_test_pred))
```

```
# Plot Training and Test Errors

plt.plot(degrees, train_errors,
label='Training Error')

plt.plot(degrees, test_errors, label='Test
Error')

plt.xlabel('Model Complexity (Degree)')

plt.ylabel('Mean Squared Error')

plt.legend()

plt.show()
```

Learning rate

Sederhananya, learning rate adalah seberapa besar langkah yang diambil oleh model machine learning saat belajar dari data.

Tidak semua algoritma dipengaruhi oleh learning rate. Learning rate hanya berpengaruh pada algoritma yang menggunakan optimasi berbasis gradien (gradient-based optimization)

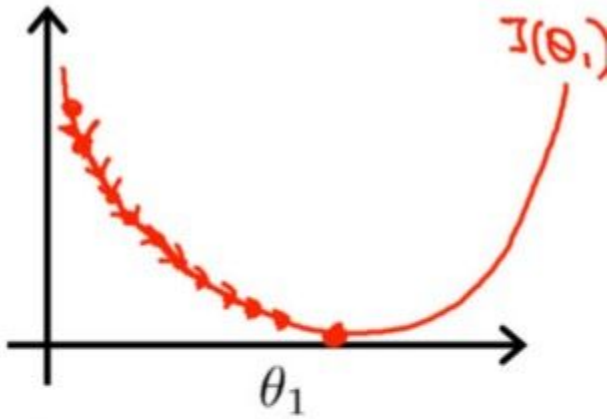
Learning rate

Efek Learning Rate:
✅ Terlalu kecil → Model belajar lambat, bisa nyangkut di titik yang kurang optimal (konvergensi lambat).
❌ Terlalu besar → Model bisa bolak-balik tanpa pernah menemukan titik terbaik (divergen).
✔️ Pas → Model cepat menemukan solusi terbaik dengan stabil.

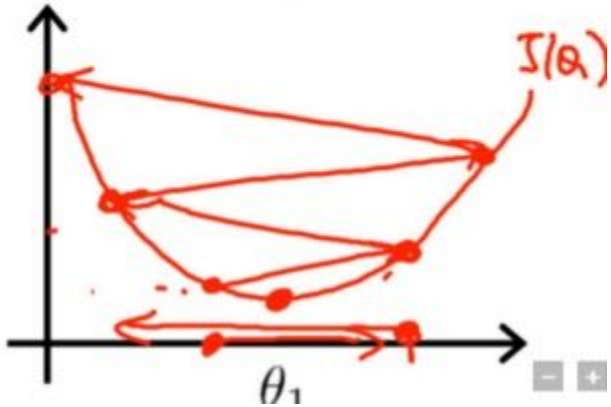
Biasanya, learning rate diuji dengan beberapa nilai (misalnya 0.1, 0.01, 0.001) untuk menemukan yang paling optimal

$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

If α is too small, gradient descent can be slow.



If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.



Loss

Loss adalah angka yang menunjukkan seberapa jauh prediksi model dari nilai sebenarnya.

- Jika loss kecil, berarti model bagus dalam memprediksi.
- Jika loss besar, berarti model masih banyak melakukan kesalahan.

Loss vs Accuracy

📌 Loss mengukur seberapa buruk prediksi model dalam angka.

📌 Accuracy (akurasi) mengukur berapa banyak prediksi yang benar.

🛠️ Contoh:

- Model memprediksi 95% benar → Akurasi 95%
- Tapi beberapa prediksi yang salah sangat jauh dari nilai sebenarnya → Loss bisa tetap tinggi!

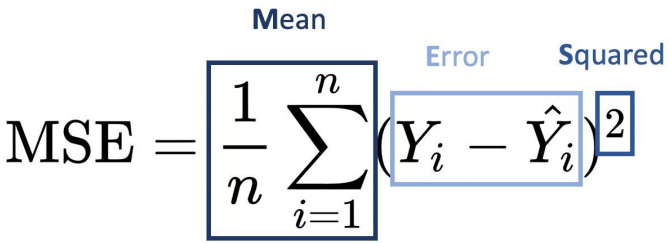
Jadi, model dengan akurasi tinggi tidak selalu memiliki loss yang kecil, tergantung seberapa besar kesalahannya.

Jenis-Jenis Loss Function

Loss function tergantung pada **jenis tugas ML**:

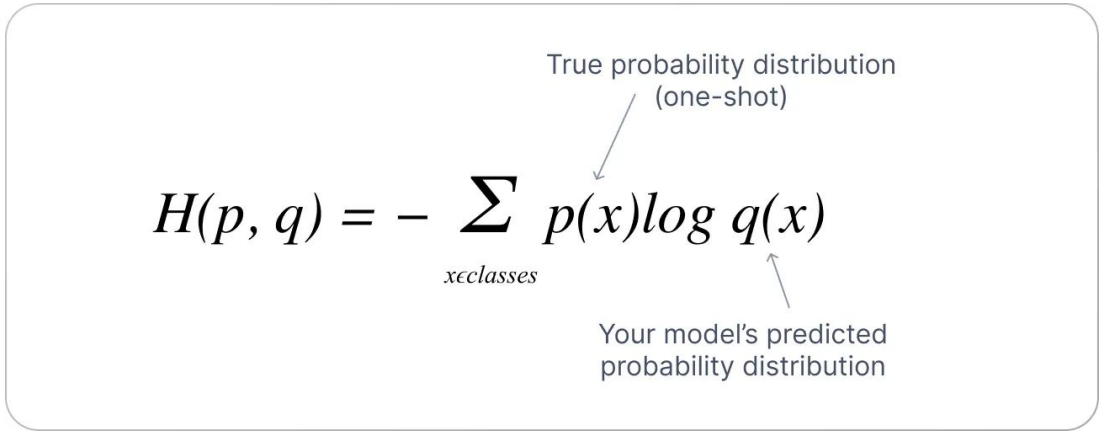
1 **Regresi (Prediksi Angka)**

- **Mean Squared Error (MSE)** → Menghitung rata-rata kuadrat selisih antara prediksi dan nilai asli.
- **Mean Absolute Error (MAE)** → Menghitung rata-rata selisih absolut antara prediksi dan nilai asli.



Klasifikasi (Prediksi Kategori)

- **Cross-Entropy Loss (Log Loss)** → Digunakan untuk klasifikasi biner dan multi-kelas. Jika model salah tebak, loss lebih besar.



sumber

Optimizer

Optimizer adalah algoritma yang digunakan untuk memperbarui bobot model agar loss menjadi lebih kecil dan model semakin baik.

Tujuan utama optimizer:

- Menyesuaikan bobot model berdasarkan gradient descent.
- Meminimalkan loss function.
- Meningkatkan akurasi prediksi.

Cara Kerja Optimizer (Gradient Descent)

Optimizer bekerja dengan konsep Gradient Descent, yaitu:

- Hitung loss → Seberapa jauh prediksi model dari nilai asli.
- Hitung turunan (gradien) → Menentukan arah untuk memperbaiki model.
- Update bobot → Menggunakan learning rate untuk memperbarui bobot model sedikit demi sedikit.
- Ulangi hingga loss cukup kecil atau model stabil.

Jenis-jenis Optimizer

1. **Stochastic Gradient Descent (SGD)**
    a. Mengupdate bobot setiap satu sampel data
    b. Cepat tapi bisa tidak stabil karena naik-turun
2. **Momentum**

Perbaikan dari SGD dengan menambahkan "momentum" untuk menghindari osilasi (naik-turun yang tidak perlu)

3. **RMSprop (Root Mean Square Propagation)**

Menggunakan rata-rata kuadrat dari gradient untuk menyesuaikan learning rate secara adaptif.

4. **Adam (Adaptive Moment Estimation)**

Kombinasi Momentum + RMSprop.

Buat lebih jelasnya:
https://youtu.be/NE88eqLngkg?si=WemFRuSabLZSHRJb

Kondisi	Gunakan Optimizer
Dataset kecil, sederhana	SGD
Loss naik-turun tidak stabil	Momentum
Deep Learning, CNN, RNN	Adam atau RMSprop
Data time-series, NLP	RMSprop
Eksperimen pertama, tidak tahu mana yang terbaik	Adam

Tuning Your Hyperparameters: Using Grid Search and Random Search

Saat melatih model ML, kita punya dua jenis parameter:

- Parameter → Dipelajari dari data (contoh: bobot dalam neural network).
- Hyperparameter → Diatur sebelum pelatihan, tidak dipelajari langsung dari data (contoh: learning rate, jumlah pohon di Random Forest).

Tuning hyperparameter berarti mencari kombinasi terbaik agar model bekerja optimal.

Grid Search

Grid Search mencoba semua kombinasi hyperparameter dari daftar yang kita tentukan.

Gimana, tuh?

1. Kita tentukan beberapa nilai untuk setiap hyperparameter.
2. Grid Search menguji semua kemungkinan kombinasi.
3. Model dengan kombinasi terbaik dipilih berdasarkan evaluasi (misalnya, akurasi tertinggi pada validasi).

Contoh Implementasi

```
from sklearn.model_selection import GridSearchCV

# Parameter Grid
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10]
}

# Grid Search with Cross-Validation
grid_search = GridSearchCV(RandomForestClassifier(), param_grid, cv=5)
grid_search.fit(X_train, y_train)

print(f"Best Parameters: {grid_search.best_params_}")
print(f"Best Cross-Validation Score: {grid_search.best_score_:.2f}")
```

Random Search

Random Search memilih kombinasi hyperparameter secara acak dalam rentang yang ditentukan.

Kumaha, tah?

- Kita tentukan rentang nilai untuk setiap hyperparameter.
- Algoritma memilih beberapa kombinasi secara acak dan mengujinya.
- Model terbaik dipilih berdasarkan hasil validasi.

Contoh Implementasi

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

# Parameter Distributions
param_dist = {
    'n_estimators': randint(50, 200),
    'max_depth': [None, 10, 20, 30],
    'min_samples_split' : randint(2, 11)
}

# Randomized Search with Cross-Validation
random_search = RandomizedSearchCV(RandomForestClassifier(),
param_distributions=param_dist, n_iter= 10, cv=5, random_state= 42)
random_search.fit(X_train, y_train)

print(f"Best Parameters: {random_search.best_params_} ")
print(f"Best Cross-Validation Score: {random_search.best_score_: .2f}")
```

Accuracy, Precision and F1-score

Akurasi

Akurasi adalah metrik yang mengukur seberapa sering model membuat prediksi yang benar dari total keseluruhan prediksi (baik yang benar maupun yang salah.

Akurasi = (Jumlah Prediksi Benar) / (Total Jumlah Prediksi)

Presisi

Presisi adalah metrik yang mengukur seberapa sering model membuat prediksi positif yang benar dari total keseluruhan prediksi positif (baik yang benar maupun yang salah).

Presisi = (Jumlah Prediksi Positif yang Benar) / (Total Jumlah Prediksi Positif)

Misalkan Anda memiliki model yang memprediksi apakah seorang pasien terkena penyakit tertentu atau tidak. Dari 100 prediksi, model membuat 80 prediksi yang benar. Maka, akurasi model tersebut adalah 80%.

Dalam kasus yang sama seperti di atas, misalkan dari 50 prediksi positif, model membuat 40 prediksi yang benar. Maka, presisi model tersebut adalah 80%.

F1-Score

F1-score adalah metrik yang menggabungkan presisi dan recall (proporsi prediksi positif yang benar dari total data positif sebenarnya) dalam satu angka. F1-score mencapai nilai tertinggi (1) ketika presisi dan recall sama-sama tinggi.

F1-score = 2 * (Presisi * Recall) / (Presisi + Recall)

Jika presisi model adalah 80% dan recall-nya adalah 75%, maka F1-score model tersebut adalah sekitar 77,4%.

Credit:
https://medium.com/@murmuarpan530/model-training-and-evaluation-2f86037c9509