Bitlabs

Metode Boosting dan Terapannya

Oleh: M Mirza Fahmi

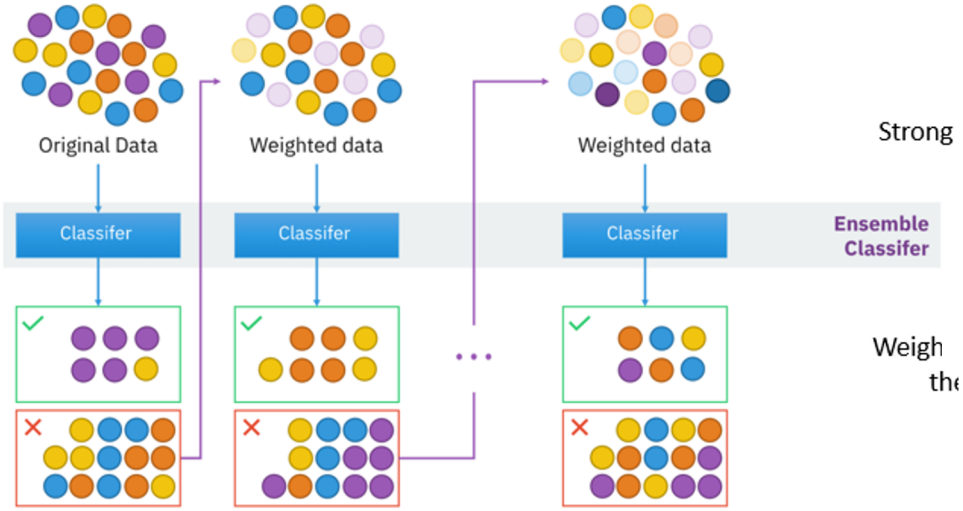
Dengan motivasi yang sama dengan metode bagging, boosting juga **berupaya mengkombinasikan model lemah untuk memperoleh model yang lebih kuat.**

Berbeda dengan bagging, boosting bersifat sekuensial, artinya setiap estimator bergantung pada estimator sebelumnya untuk meningkatkan hasil prediksi.

Cara Kerja Metode Boosting

Logika dibalik konsep ini, sebuah model lemah yang sering dilatih akan menjadi lebih kuat.

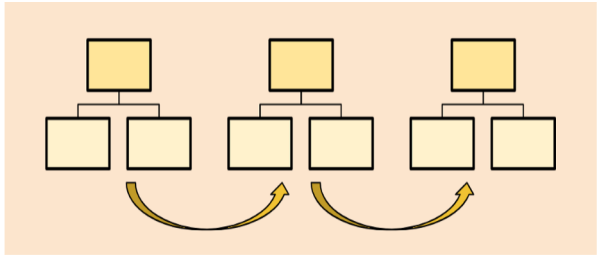
Cara Kerja Metode Boosting



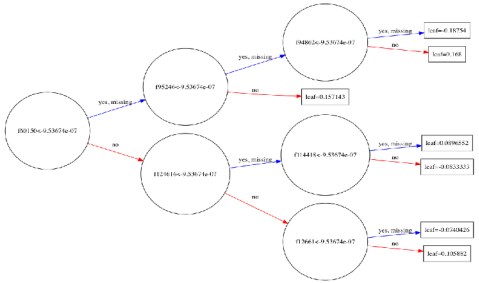
1. Buat sebuah potongan data dari dataset asli (seluruh titik data diberi bobot yang sama.
2. Buat dan latih sub model dengan sub data tersebut.
3. Sub model digunakan untuk membuat prediksi dari data asli.
4. Hitung kesalahan prediksi.
5. Titik data dengan kesalahan prediksi yang tinggi diberikan bobot lebih tinggi.
6. Model lain dibuat dengan titik-titik data yang telah diberi bobot yang tinggi.
7. Ulangi langkah-langkah diatas sesuai keperluan.
8. Model akhir diperoleh dengan mengagregasi sub-sub model.

Varian Metode Boosting

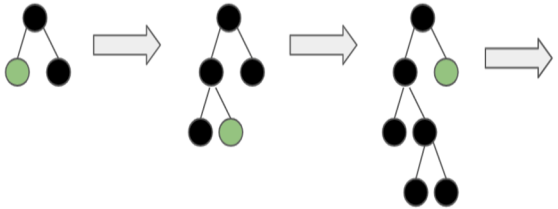
AdaBoost



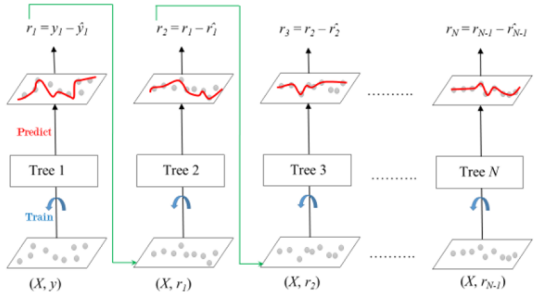
XGBoost



LightGBM

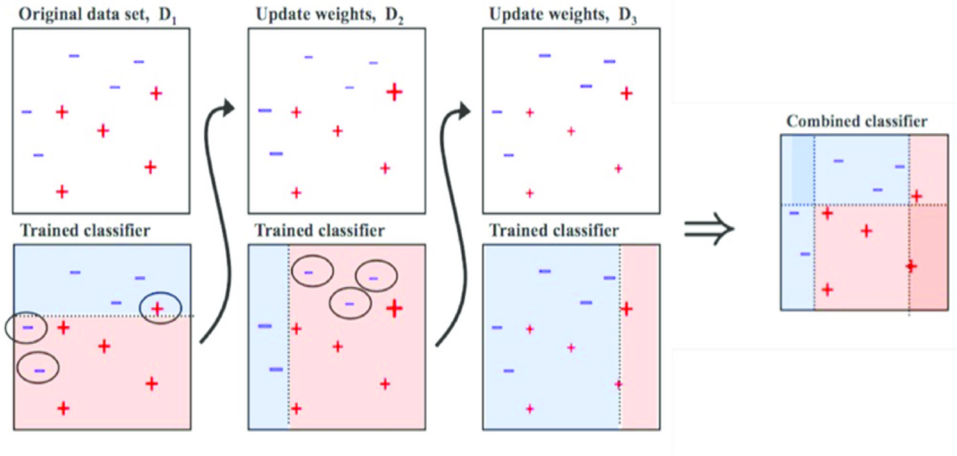


Gradient Boosting



AdaBoost (Adaptive Boosting)

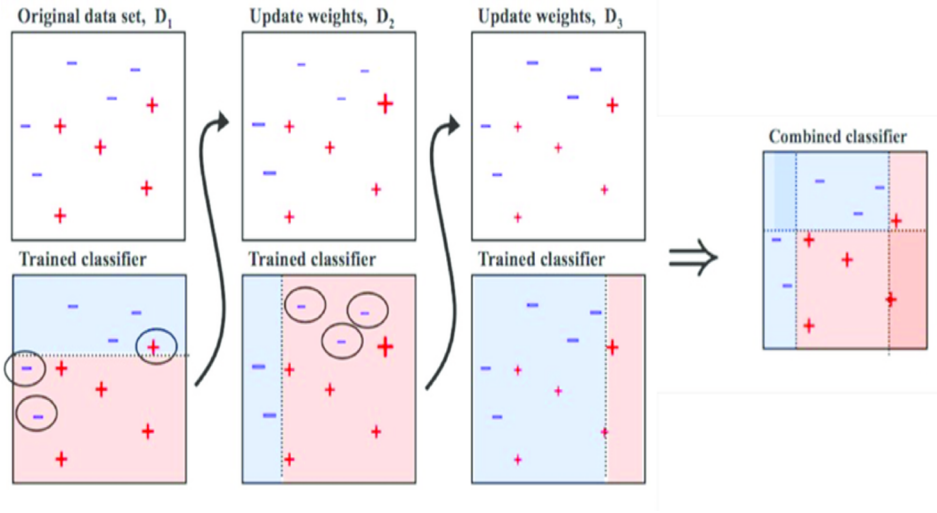
Definisi: Metode yang digunakan dalam algoritma ini yaitu dengan memperbaiki pendahulunya dengan lebih memperhatikan sampel yang underfit oleh sub-model sebelumnya. Sampel-sampel yang underfit selanjutnya akan diberikan pembobotan.



AdaBoost (Adaptive Boosting)

Langkah-langkah:

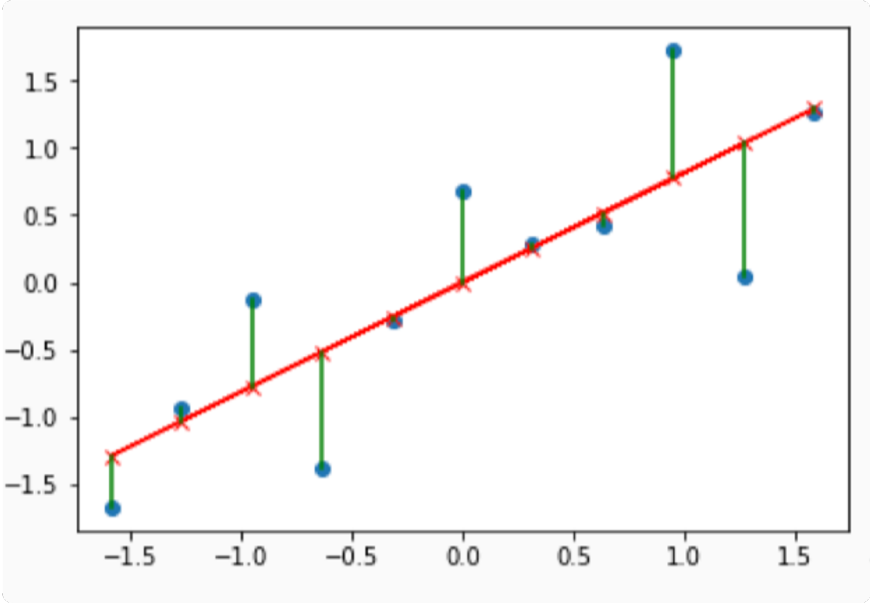
1. Inisiasi bobot untuk setiap titik data. Jika data latih ada 100 titik data, maka setiap titik bobot awalnya 1/100 = 0.01
2. Latih classifier nya (decision tree).
3. Hitung prediksi yang salah dan perlakukan berbeda tergantung bobot titik datanya.
4. Hitung bobot classifier.
5. Perbarui bobot untuk titik data yang salah prediksi.
6. Ulangi langkah 1
7. Buat prediksi final



AdaBoost (Adaptive Boosting)

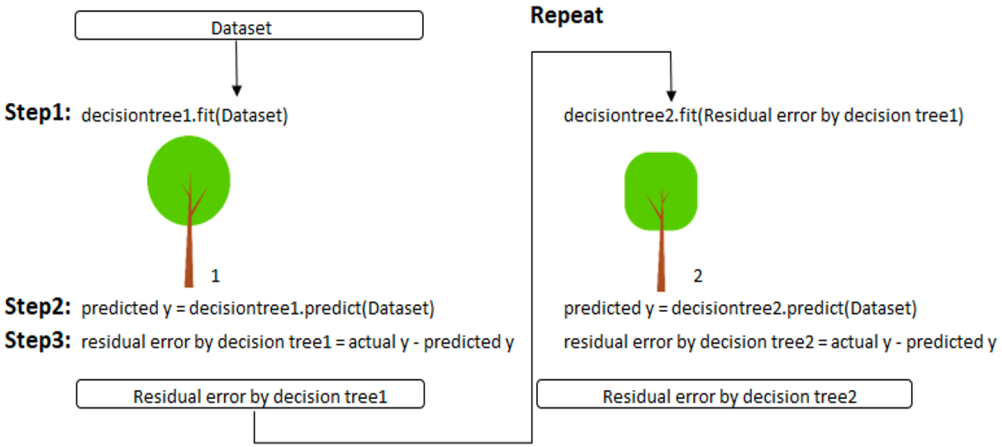
```
'''
X: independent variables - array-like matrix
y: target variable - array-like vector
M: number of boosting rounds. Default is 100 - integer
'''

# Clear before calling
self.alphas = []
self.training_errors = []
self.M = M

# Iterate over M weak classifiers
for m in range(0, M):

    # Set weights for current boosting iteration
    if m == 0:
        w_i = np.ones(len(y)) * 1 / len(y)  # At m = 0, weights are all the same and equal to 1 / N
    else:
        # (d) Update w_i
        w_i = update_weights(w_i, alpha_m, y, y_pred)

    # (a) Fit weak classifier and predict labels
    G_m = DecisionTreeClassifier(max_depth = 1)     # Stump: Two terminal-node classification tree
    G_m.fit(X, y, sample_weight = w_i)
    y_pred = G_m.predict(X)

    self.G_M.append(G_m) # Save to list of weak classifiers

    # (b) Compute error
    error_m = compute_error(y, y_pred, w_i)
    self.training_errors.append(error_m)

    # (c) Compute alpha
    alpha_m = compute_alpha(error_m)
    self.alphas.append(alpha_m)

assert len(self.G_M) == len(self.alphas)
```

Gradient Boosting



Definisi: Metode boosting yang belajar dari kesalahan sebelumnya yang langsung dari error residualnya. Model yang diboosting adalah decision tree. Prediksi akhirnya akan menjumlahkan seluruh prediksi-prediksi yang telah dibuatnya.

Gradient Boosting



Langkah-langkah:

1. Latih classifier.
2. Buat prediksi dari classifier yang sudah dilatih.
3. Hitung residual dari classifier, simpan kesalahan residual sebagai target baru.
4. Ulangi langkah 1.
5. Buat prediksi akhir.

Gradient Boosting

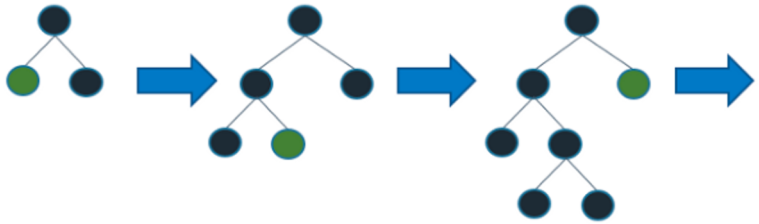
```
xi = x  # initialization of input
yi = y  # initialization of target
# x,y --> use where no need to change original y
ei = 0  # initialization of error
n = len(yi)  # number of rows
predf = 0  # initial prediction 0

for i in range(30):  # loop will make 30 trees (n_estimators).
    # DecisionTree scratch code can be found on www.kaggle.com/grroverpr/gradient-boosting-simplified
    tree = DecisionTree(xi, yi)
    # It just create a single decision tree with provided min. sample leaf
    # For selected input variable, this splits (<n and >n) data so that std. deviation of
    tree.find_better_split(0)
    # target variable in both splits is minimum as compared to all other splits

    # finds index where this best split occurs
    r = np.where(xi == tree.split)[0][0]

    left_idx = np.where(xi <= tree.split)[0]  # index lhs of split
    right_idx = np.where(xi > tree.split)[0]  # index rhs of split

# predictions by ith decisision tree

predi = np.zeros(n)
# replace left side mean y
np.put(predi, left_idx, np.repeat(np.mean(yi[left_idx]), r))
np.put(predi, right_idx, np.repeat(
    np.mean(yi[right_idx]), n-r))  # right side mean y

predi = predi[:, None]  # make long vector (nx1) in compatible with y
# final prediction will be previous prediction value + new prediction of residual
predf = predf + predi

ei = y - predf  # needed originl y here as residual always from original y
yi = ei  # update yi as residual to reloop
```

LightGBM

Light Gradient Boosting Machine

Level-wise tree growth

Level-wise tree growth in XGBOOST.



Leaf-wise tree growth

Leaf wise tree growth in Light GBM.

Definisi: Metode ini adalah perbaikan dari gradient boosting untuk mengatasi data - data berukuran besar.
Jika pada gradient boosting, model berbasis tree tumbuhnya *Level-wise* maka LightGBM tumbuhnya *Leaf-wise*.

Kelebihan	Kekurangan
Proses training lebih cepat dan mendukung GPU.	Rentan overfitting.
Fokus kepada akurasi	Tidak disarankan untuk dataset kecil.

Bitlabs

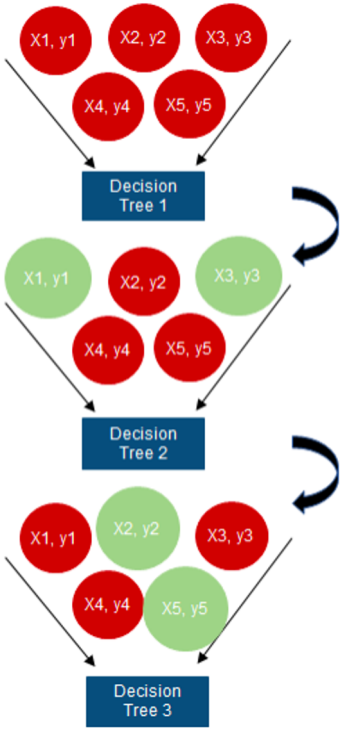
LightGBM

Light Gradient Boosting Machine

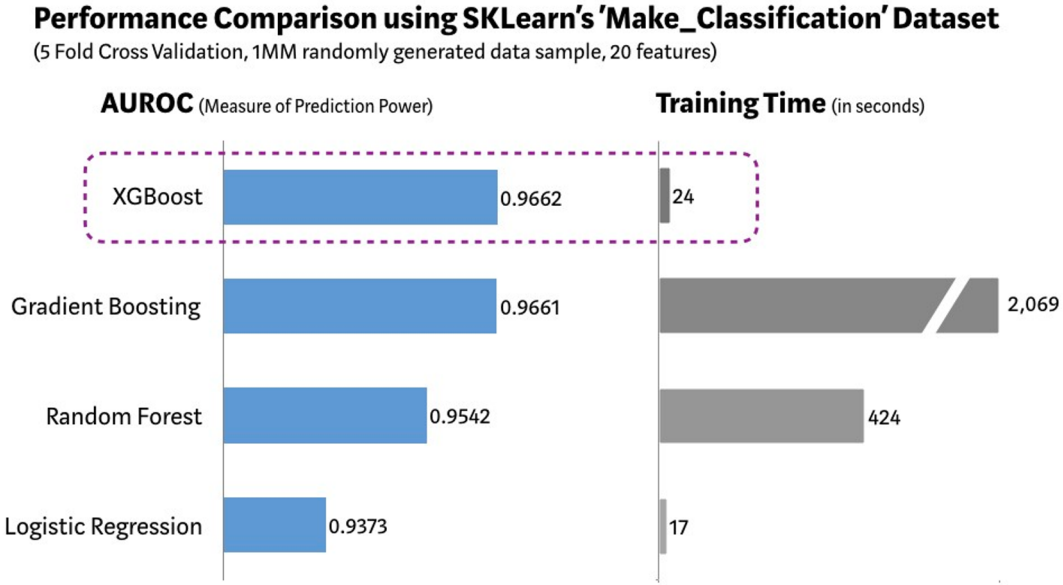
```
# Importing Required Library
import pandas as pd
import lightgbm as lgb

# Similarly LGBMRegressor can also be imported for a regression model.
from lightgbm import LGBMClassifier

# Reading the train and test dataset
data = pd.read_csv("cancer_prediction.csv)

# Removing Columns not Required
data = data.drop(columns = ['Unnamed: 32'], axis = 1)
data = data.drop(columns = ['id'], axis = 1)

# Skipping Data Exploration
# Dummification of Diagnosis Column (1-Benign, 0-Malignant Cancer)
data['diagnosis']= pd.get_dummies(data['diagnosis'])

# Spliiting Dataset in two parts
train = data[0:400]
test = data[400:568]

# Separating the independent and target variable on both data set
x_train = train.drop(columns =['diagnosis'], axis = 1)
y_train = train_data['diagnosis']
x_test = test_data.drop(columns =['diagnosis'], axis = 1)
y_test = test_data['diagnosis']

# Creating an object for model and fitting it on training data set
model = LGBMClassifier()
model.fit(x_train, y_train)

# Predicting the Target variable
pred = model.predict(x_test)
print(pred)
accuracy = model.score(x_test, y_test)
print(accuracy)
```

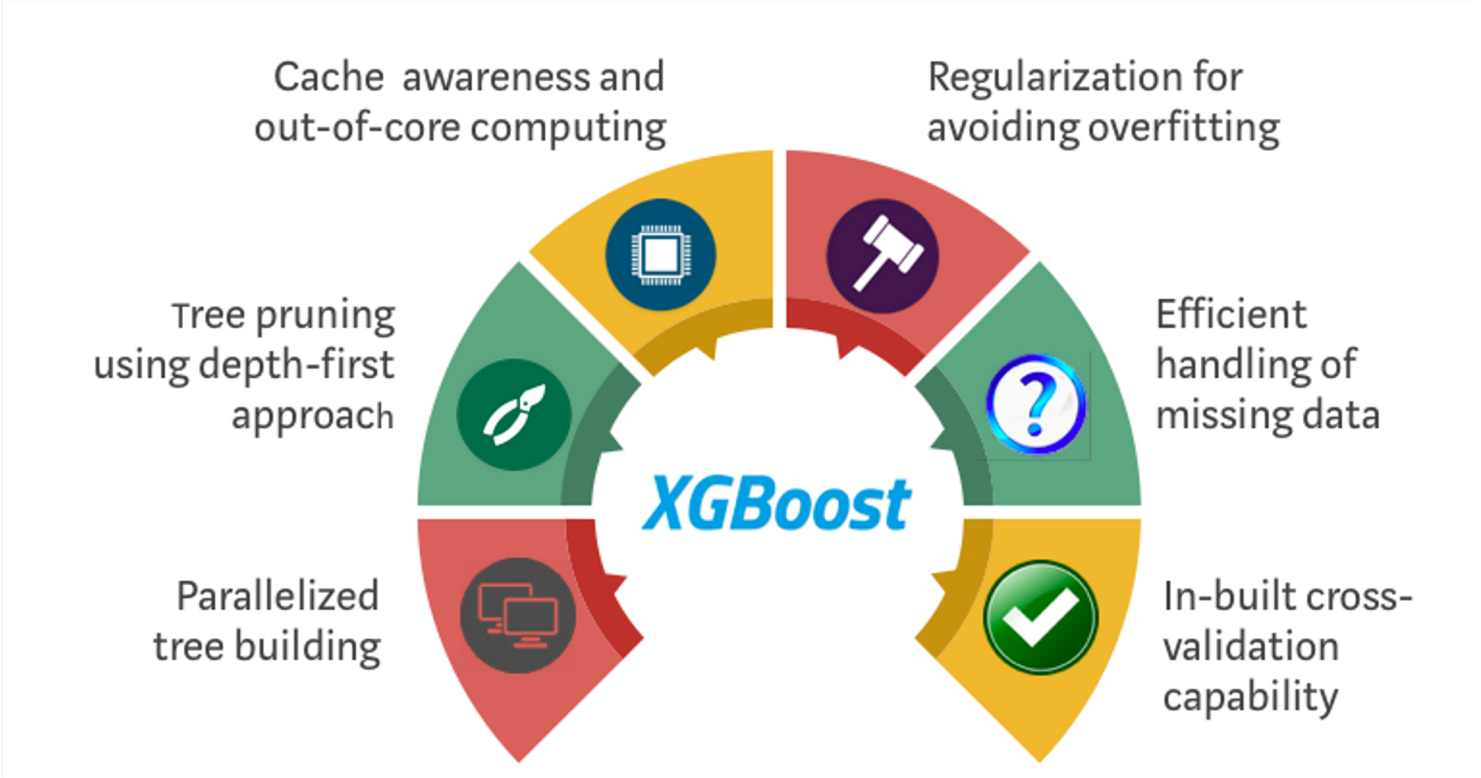
XGBoost

eXtreme Gradient Boosting



Definisi: extreme gradient boosting merupakan versi algoritma gradient boosting yang dioptimalkan.




Cache awareness and out-of-core computing
Regularization for avoiding overfitting
Tree pruning using depth-first approach
Efficient handling of missing data
XGBoost
Parallelized tree building
In-built cross-validation capability

XGBoost

eXtreme Gradient Boosting

```
import pickle
import xgboost as xgb

import numpy as np
from sklearn.model_selection import KFold, train_test_split, GridSearchCV
from sklearn.metrics import confusion_matrix, mean_squared_error
from sklearn.datasets import load_iris, load_digits, load_boston

rng = np.random.RandomState(31337)

print("Zeros and Ones from the Digits dataset: binary classification")
digits = load_digits(n_class=2)
y = digits['target']
X = digits['data']
kf = KFold(n_splits=2, shuffle=True, random_state=rng)
for train_index, test_index in kf.split(X):
    xgb_model = xgb.XGBClassifier(n_jobs=1).fit(
        X[train_index], y[train_index]
    )
    predictions = xgb_model.predict(X[test_index])
    actuals = y[test_index]
    print(confusion_matrix(actuals, predictions))
```

Quiz

Apakah perbedaan mendasar antara LightGBM dan Gradient Boosting?

Sebutkan kekurangan LightGBM?

Bagaimana Adaboost memperbarui bobot titik-titik data training?