# 

# Code

import pandas as pd

import matplotlib.pyplot as plt

import time # to check time for modeling

from sklearn.model\_selection import train\_test\_split # Split into training and testing

from sklearn.linear\_model import LogisticRegression # LogisticRegression

from sklearn.tree import DecisionTreeClassifier # Decison Tree

from sklearn.neighbors import KNeighborsClassifier # KNN

from xgboost import XGBClassifier # XGBoost

from sklearn.metrics import (

    accuracy\_score, precision\_score, recall\_score, f1\_score,

    roc\_auc\_score, confusion\_matrix, ConfusionMatrixDisplay, roc\_curve # for evaluation

)

# Load data

df = pd.read\_csv("final\_matrix.csv") # Data after preprocessing but no PCA

X = df.drop(columns=["status"]) # Features

y = df["status"] # Target

X\_train, X\_test, y\_train, y\_test = train\_test\_split( # Split dataset into training and testing

    X, y, stratify=y, test\_size=0.2, random\_state=42 # 20% of data will be used for testing

)

models = { # Classifiers

    'XGBoost': XGBClassifier(use\_label\_encoder=False, eval\_metric='logloss'),

    'Logistic Regression': LogisticRegression(max\_iter=1000),

    'Decision Tree': DecisionTreeClassifier(),

    'KNN': KNeighborsClassifier()

}

results = {} # Evaluation Result

roc\_data = {} # ROC curve data

for name, model in models.items():

    print(f"\n Model: {name}")

    start = time.time() # Record start time

    model.fit(X\_train, y\_train) # Train the model

    y\_pred = model.predict(X\_test) # Predict Labels

    y\_proba = model.predict\_proba(X\_test)[:, 1] # Predict probabilities for ROC AUC

    end = time.time() # Record end time

    elapsed = end – start # Calculate elapsed time

    # Calculate evaluation metrics

    acc = accuracy\_score(y\_test, y\_pred) # Accuracy

    prec = precision\_score(y\_test, y\_pred) # Precision

    rec = recall\_score(y\_test, y\_pred) # Recall

    f1 = f1\_score(y\_test, y\_pred) # F1 score

    auc = roc\_auc\_score(y\_test, y\_proba) # ROC AUC score

# Save metrics to result{}

    results[name] = {

        'Time': elapsed,

        'Accuracy': acc,

        'Precision': prec,

        'Recall': rec,

        'F1 Score': f1,

        'ROC AUC': auc

    }

    # Confusion Matrix

    ConfusionMatrixDisplay.from\_predictions(y\_test, y\_pred)

    plt.title(f"{name} - Confusion Matrix")

    plt.show()

    # ROC Curve

    fpr, tpr, \_ = roc\_curve(y\_test, y\_proba)

    roc\_data[name] = (fpr, tpr, auc) # Store ROC data

    plt.plot(fpr, tpr, label=f"{name} (AUC={auc:.2f})")

    plt.plot([0, 1], [0, 1], 'k--') # Reference line

    plt.title(f"{name} - ROC Curve")

    plt.xlabel("False Positive Rate") # FP

    plt.ylabel("True Positive Rate") # TP

    plt.legend()

    plt.grid(True)

    plt.show()

    # Feature Importance (if available)

    if hasattr(model, "feature\_importances\_"):

        import numpy as np

        importances = model.feature\_importances\_

        feature\_names = X.columns if hasattr(X, "columns") else [f"f{i}" for i in range(X.shape[1])]

        # Get top 5 Features

        top\_indices = np.argsort(importances)[::-1][:5]

        top\_features = [feature\_names[i] for i in top\_indices]

        top\_importances = importances[top\_indices]

# Plot top 5 Features

        plt.figure(figsize=(10, 6))

        plt.barh(range(5), top\_importances[::-1])

        plt.yticks(range(5), top\_features[::-1])

        plt.xlabel("Importance")

        plt.title(f"{name} - Top 5 Feature Importances")

        plt.tight\_layout()

        plt.show()

    # Evaluation Summary

    print(f" Evaluation for {name}:")

    print(f"Time: {elapsed:.4f} sec")

    print(f"Accuracy: {acc:.4f}")

    print(f"Precision: {prec:.4f}")

    print(f"Recall: {rec:.4f}")

    print(f"F1 Score: {f1:.4f}")

    print(f"ROC AUC: {auc:.4f}")

# Combined ROC Curve

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

for name, (fpr, tpr, auc) in roc\_data.items():

    plt.plot(fpr, tpr, label=f"{name} (AUC={auc:.2f})")

plt.plot([0, 1], [0, 1], 'k--')

plt.title("ROC Curve Comparison")

plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

plt.legend()

plt.grid(True)

plt.show()

# Summary Table

df\_result = pd.DataFrame(results).T

df\_result

# Outputs

텍스트, 스크린샷, 사각형, 다채로움이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

텍스트, 라인, 그래프, 스크린샷이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다. 스크린샷, 텍스트, 디스플레이, 소프트웨어이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다. 텍스트, 폰트, 스크린샷, 디자인이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다. 텍스트, 스크린샷, 도표, 사각형이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다. 텍스트, 라인, 그래프, 도표이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다. 텍스트, 폰트, 스크린샷, 블랙이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다. 텍스트, 스크린샷, 사각형, 직사각형이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다. 텍스트, 라인, 그래프, 도표이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다. 스크린샷, 텍스트, 디스플레이, 소프트웨어이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다. 텍스트, 폰트, 스크린샷이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다. 텍스트, 스크린샷, 사각형, 직사각형이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다. 텍스트, 라인, 그래프, 도표이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다. 텍스트, 폰트, 스크린샷, 디자인이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다. 텍스트, 스크린샷, 그래프, 라인이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다. 텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

# Individual review

Through the Term Project in this Data Science course, I gained a solid understanding of how to extract meaningful information from a dataset. In particular, I realized the importance of preprocessing, especially when working with datasets that contain a large number of features.