1. SVM model

import pandas as pd

from sklearn.model\_selection import train\_test\_split, GridSearchCV, StratifiedKFold

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

from sklearn.metrics import classification\_report

from imblearn.over\_sampling import SMOTE

from sklearn.pipeline import Pipeline

from sklearn.feature\_selection import RFECV

from imblearn.pipeline import make\_pipeline

from google.colab import drive

# Load the data from the CSV file

drive.mount('/content/drive')

data = pd.read\_csv('/content/drive/MyDrive/mac\_combined\_May\_17.csv')

# Separate the features (X) and Flags (y)

X = data.drop(columns=['Website', 'Flag'])

y = data['Flag']

# Split the data into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Define the pipeline to handle class imbalance using SMOTE, standardize features, and train the SVM model

pipe = make\_pipeline(SMOTE(random\_state=42), StandardScaler(), SVC(kernel='rbf', class\_weight='balanced'))

# Hyperparameter tuning using GridSearchCV with wider range of hyperparameters

param\_grid = {'svc\_\_C': [0.01, 0.1, 1, 10, 100, 1000, 10000],

'svc\_\_gamma': [10, 1, 0.1, 0.01, 0.001, 0.0001, 0.00001]}

grid = GridSearchCV(pipe, param\_grid, verbose=3, cv=StratifiedKFold(5))

grid.fit(X\_train, y\_train)

# Print the best parameters

print("Best parameters found by GridSearchCV:")

print(grid.best\_params\_)

# Make predictions on the test set

y\_pred = grid.predict(X\_test)

# Print classification report

print(classification\_report(y\_test, y\_pred))  
  
Best parameters found by GridSearchCV:

{'svc\_\_C': 10000, 'svc\_\_gamma': 1}

precision recall f1-score support

0 0.97 0.64 0.77 4070

1 0.73 0.98 0.84 4121

accuracy 0.81 8191

macro avg 0.85 0.81 0.80 8191

weighted avg 0.85 0.81 0.80 8191

1. Logistic Regression Model

from google.colab import drive

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

from sklearn import metrics

import time

from tabulate import tabulate

from imblearn.over\_sampling import SMOTE

from sklearn.feature\_selection import RFE

from sklearn.model\_selection import GridSearchCV

# Load the data

print("Loading data...")

start\_time = time.time()

drive.mount('/content/drive')

df = pd.read\_csv('/content/drive/MyDrive/mac\_combined\_May\_17.csv')

print(f"Data loaded in {time.time() - start\_time:.2f} seconds.\n")

# Extract feature columns

features = df.drop(['Website', 'Flag'], axis=1)

# Extract target column 'Flag'

target = df['Flag']

# Split the data into training set and test set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(features, target, test\_size=0.3, random\_state=42)

# Initialize the StandardScaler

scaler = StandardScaler()

# Scale the features

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Balance the dataset with SMOTE

print("Balancing the dataset...")

sm = SMOTE(random\_state=42)

X\_train\_res, y\_train\_res = sm.fit\_resample(X\_train, y\_train)

# Feature selection with Recursive Feature Elimination

print("Applying Recursive Feature Elimination...")

model = LogisticRegression(max\_iter=1000)

rfe = RFE(estimator=model, n\_features\_to\_select=20) # choose the top 10 features

rfe = rfe.fit(X\_train\_res, y\_train\_res)

X\_train\_res = rfe.transform(X\_train\_res)

X\_test = rfe.transform(X\_test)

# Hyperparameter tuning with GridSearchCV

print("Tuning hyperparameters...")

parameters = {'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]}

clf = GridSearchCV(model, parameters, cv=5)

clf.fit(X\_train\_res, y\_train\_res)

print(f"Best parameters: {clf.best\_params\_}")

# Train the model with best parameters

print("Training the model with best parameters...")

model = LogisticRegression(max\_iter=1000, C=clf.best\_params\_['C'])

model.fit(X\_train\_res, y\_train\_res)

# Make predictions

y\_pred = model.predict(X\_test)

# Evaluate the model

accuracy = metrics.accuracy\_score(y\_test, y\_pred)

classification\_report = metrics.classification\_report(y\_test, y\_pred)

print("Model Evaluation:\n")

print(f"Accuracy: {accuracy}")

print("\nClassification Report:")

print(tabulate(pd.DataFrame(metrics.classification\_report(y\_test, y\_pred, output\_dict=True)).transpose(), headers='keys', tablefmt='psql'))  
  
Model Evaluation:

Accuracy: 0.7814674642900745

Classification Report:

+--------------+-------------+----------+------------+-------------+

| | precision | recall | f1-score | support |

|--------------+-------------+----------+------------+-------------|

| 0 | 0.985106 | 0.568796 | 0.721184 | 4070 |

| 1 | 0.699538 | 0.991507 | 0.820317 | 4121 |

| accuracy | 0.781467 | 0.781467 | 0.781467 | 0.781467 |

| macro avg | 0.842322 | 0.780151 | 0.770751 | 8191 |

| weighted avg | 0.841433 | 0.781467 | 0.771059 | 8191 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |

1. Naïve Bayes Model

from google.colab import drive

import pandas as pd

from sklearn.model\_selection import train\_test\_split, GridSearchCV, StratifiedKFold

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score, classification\_report

from sklearn.preprocessing import StandardScaler

from sklearn.feature\_selection import SelectKBest, f\_classif

drive.mount('/content/drive')

data = pd.read\_csv('/content/drive/MyDrive/mac\_combined\_May\_17.csv')

# Drop the columns with constant values

constant\_columns = [col for col in data.columns if data[col].nunique() <= 1]

data = data.drop(columns=constant\_columns)

# Feature Selection - Select the top K most informative features

X = data.drop(columns=['Website', 'Flag'])

y = data['Flag']

selector = SelectKBest(score\_func=f\_classif, k=10)

X\_selected = selector.fit\_transform(X, y)

# Data Preprocessing - Scale numerical features

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X\_selected)

# Split into train and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.3, random\_state=42)

# Hyperparameter Tuning - Perform GridSearchCV to find the best hyperparameters

param\_grid = {'var\_smoothing': [1e-9, 1e-8, 1e-7]}

model = GaussianNB()

grid\_search = GridSearchCV(model, param\_grid, cv=5)

grid\_search.fit(X\_train, y\_train)

# Train the model with the best hyperparameters

best\_model = grid\_search.best\_estimator\_

best\_model.fit(X\_train, y\_train)

# Cross-Validation - Evaluate the model using StratifiedKFold cross-validation

cv = StratifiedKFold(n\_splits=5)

accuracy\_scores = []

classification\_reports = []

for train\_index, test\_index in cv.split(X\_scaled, y):

X\_train\_cv, X\_test\_cv = X\_scaled[train\_index], X\_scaled[test\_index]

y\_train\_cv, y\_test\_cv = y.iloc[train\_index], y.iloc[test\_index]

model\_cv = GaussianNB(var\_smoothing=best\_model.var\_smoothing)

model\_cv.fit(X\_train\_cv, y\_train\_cv)

y\_pred\_cv = model\_cv.predict(X\_test\_cv)

accuracy\_scores.append(accuracy\_score(y\_test\_cv, y\_pred\_cv))

classification\_reports.append(classification\_report(y\_test\_cv, y\_pred\_cv))

# Predict on the test set

y\_pred = best\_model.predict(X\_test)

# Print accuracy and classification report

accuracy = accuracy\_score(y\_test, y\_pred)

class\_report = classification\_report(y\_test, y\_pred)

# Print results

print("Model Performance:")

print("Metric Score")

print("--------------------- -----------------------------------------------------")

print(f"Accuracy {accuracy}")

print("Classification Report\n", class\_report)

# Print cross-validation results

print("Cross-Validation Results:")

print("Fold Accuracy")

print("----------------")

for i in range(len(accuracy\_scores)):

print(f"{i+1:<8} {accuracy\_scores[i]:.4f}")

Model Performance:

Metric Score

--------------------- -----------------------------------------------------

Accuracy 0.7651080454157002

Classification Report

precision recall f1-score support

0 1.00 0.53 0.69 4070

1 0.68 1.00 0.81 4121

accuracy 0.77 8191

macro avg 0.84 0.76 0.75 8191

weighted avg 0.84 0.77 0.75 8191

Cross-Validation Results:

Fold Accuracy

----------------

1 0.7982

2 0.7790

3 0.7663

4 0.7509

5 0.7496

1. KNN Model

from google.colab import drive

import pandas as pd

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

from sklearn.preprocessing import StandardScaler, PolynomialFeatures

from sklearn.decomposition import PCA

from tabulate import tabulate

import matplotlib.pyplot as plt

import time

# Load the dataset

print("Loading data...")

start\_time = time.time()

drive.mount('/content/drive')

data = pd.read\_csv('/content/drive/MyDrive/mac\_combined\_May\_17.csv')

print(f"Data loaded in {time.time() - start\_time:.2f} seconds.\n")

# Drop the columns with constant values

constant\_columns = [col for col in data.columns if data[col].nunique() <= 1]

data = data.drop(columns=constant\_columns)

# Split into features and target

X = data.drop(columns=['Website', 'Flag'])

y = data['Flag']

# Split 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)

# Scale the features

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

# Apply Polynomial Features

poly\_features = PolynomialFeatures(degree=2)

X\_train\_poly = poly\_features.fit\_transform(X\_train\_scaled)

X\_test\_poly = poly\_features.transform(X\_test\_scaled)

# Apply PCA for dimensionality reduction

pca = PCA(n\_components=0.95) # Retain 95% of the variance

X\_train\_pca = pca.fit\_transform(X\_train\_poly)

X\_test\_pca = pca.transform(X\_test\_poly)

# Define the parameter grid for KNN

param\_grid\_knn = {'n\_neighbors': [3, 5, 7]}

# Perform grid search for KNN

grid\_search\_knn = GridSearchCV(KNeighborsClassifier(), param\_grid\_knn)

grid\_search\_knn.fit(X\_train\_pca, y\_train)

# Get the best KNN model and its parameters

best\_knn\_model = grid\_search\_knn.best\_estimator\_

best\_knn\_params = grid\_search\_knn.best\_params\_

# Train the best KNN model

print("Training the model...")

start\_time = time.time()

best\_knn\_model.fit(X\_train\_pca, y\_train)

print(f"Model trained in {time.time() - start\_time:.2f} seconds.\n")

# Predict on the test set

y\_pred = best\_knn\_model.predict(X\_test\_pca)

# Evaluate the model

accuracy = accuracy\_score(y\_test, y\_pred)

class\_report = classification\_report(y\_test, y\_pred)

confusion\_mat = confusion\_matrix(y\_test, y\_pred)

# Print accuracy, classification report, and confusion matrix

print("Model Performance:")

print("Accuracy:", accuracy)

print("Classification Report:\n", class\_report)

print("Confusion Matrix:\n", confusion\_mat)

# Print results in a tabular format

print("Model Performance:")

print(tabulate([

["Accuracy", accuracy],

["Classification Report", "\n" + class\_report]

], headers=["Metric", "Score"]))

# Plot confusion matrix

plt.figure()

plt.imshow(confusion\_mat, cmap='Blues')

plt.title("Confusion Matrix")

plt.colorbar()

plt.xticks([0, 1], ['Class 0', 'Class 1'])

plt.yticks([0, 1], ['Class 0', 'Class 1'])

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

Model Performance:

Accuracy: 0.8042973995849103

Classification Report:

precision recall f1-score support

0 0.94 0.64 0.77 4070

1 0.73 0.96 0.83 4121

accuracy 0.80 8191

macro avg 0.84 0.80 0.80 8191

weighted avg 0.84 0.80 0.80 8191

Confusion Matrix:

[[2625 1445]

[ 158 3963]]

Model Performance:

Metric Score

--------------------- -----------------------------------------------------

Accuracy 0.8042973995849103

Classification Report precision recall f1-score support

0 0.94 0.64 0.77 4070

1 0.73 0.96 0.83 4121

accuracy 0.80 8191

macro avg 0.84 0.80 0.80 8191

weighted avg 0.84 0.80 0.80 8191

1. Decision Tree

from google.colab import drive

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score, classification\_report

from tabulate import tabulate

import time

# Load the dataset

print("Loading data...")

start\_time = time.time()

drive.mount('/content/drive')

data = pd.read\_csv('/content/drive/MyDrive/mac\_combined\_May\_17.csv')

print(f"Data loaded in {time.time() - start\_time:.2f} seconds.\n")

# Drop the columns with constant values

constant\_columns = [col for col in data.columns if data[col].nunique() <= 1]

data = data.drop(columns=constant\_columns)

# Split into features and target

X = data.drop(columns=['Website', 'Flag'])

y = data['Flag']

# Split 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)

# Train the model

print("Training the model...")

start\_time = time.time()

model = DecisionTreeClassifier()

model.fit(X\_train, y\_train)

print(f"Model trained in {time.time() - start\_time:.2f} seconds.\n")

# Predict on the test set

y\_pred = model.predict(X\_test)

# Print accuracy and classification report

accuracy = accuracy\_score(y\_test, y\_pred)

class\_report = classification\_report(y\_test, y\_pred)

# Print results in a tabular format

print("Model Performance:")

print(tabulate([

["Accuracy", accuracy],

["Classification Report", "\n" + class\_report]

], headers=["Metric", "Score"]))

Model Performance:

Metric Score

--------------------- -----------------------------------------------------

Accuracy 0.8315224026370407

Classification Report precision recall f1-score support

0 0.97 0.68 0.80 4070

1 0.76 0.98 0.85 4121

accuracy 0.83 8191

macro avg 0.86 0.83 0.83 8191

weighted avg 0.86 0.83 0.83 8191

1. Random Forest

from google.colab import drive

import pandas as pd

from sklearn.model\_selection import train\_test\_split, RandomizedSearchCV

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, classification\_report

from tabulate import tabulate

from sklearn.preprocessing import LabelEncoder

import time

import numpy as np

# Load the dataset

print("Loading data...")

start\_time = time.time()

drive.mount('/content/drive')

data = pd.read\_csv('/content/drive/MyDrive/mac\_combined\_May\_17.csv')

print(f"Data loaded in {time.time() - start\_time:.2f} seconds.\n")

# Convert categorical features to numeric

le = LabelEncoder()

for col in data.columns:

if data[col].dtype == 'object':

data[col] = le.fit\_transform(data[col].astype(str))

# Split into features and target

target\_col = 'Flag' # Update this to your actual target column name

X = data.drop(columns=[target\_col])

y = data[target\_col]

# Train a RandomForest model to compute feature importance

print("Computing feature importance...")

model = RandomForestClassifier(n\_estimators=100)

model.fit(X, y)

# Select the top 20 most important features

important\_features = np.argsort(model.feature\_importances\_)[-20:]

# Select only the important features from X

X = X[X.columns[important\_features]]

# Split 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)

# Define the parameter grid for the random search

param\_grid = {

'n\_estimators': [100, 200, 500, 1000],

'max\_depth': [10, 15, 20, None],

'min\_samples\_split': [2, 5, 10],

'min\_samples\_leaf': [1, 2, 4],

'bootstrap': [True, False]

}

# Initialize a RandomizedSearchCV object

random\_search = RandomizedSearchCV(

estimator=RandomForestClassifier(),

param\_distributions=param\_grid,

cv=5,

n\_iter=10, # Number of random combinations to try

n\_jobs=-1,

random\_state=42

)

print("Performing random search...")

start\_time = time.time()

random\_search.fit(X\_train, y\_train)

print(f"Random search completed in {time.time() - start\_time:.2f} seconds.\n")

# Print the best parameters

print(f"Best parameters: {random\_search.best\_params\_}")

# Train the model with the best parameters

print("Training the model...")

start\_time = time.time()

model = RandomForestClassifier(\*\*random\_search.best\_params\_)

model.fit(X\_train, y\_train)

print(f"Model trained in {time.time() - start\_time:.2f} seconds.\n")

# Evaluate cross-validated results

cv\_results = random\_search.cv\_results\_

print("Cross-Validation Results:")

print(tabulate([

["Mean Train Score", "-"],

["Mean Test Score", np.mean(cv\_results['mean\_test\_score'])]

], headers=["Metric", "Score"]))

# Predict on the test set

y\_pred = model.predict(X\_test)

# Print accuracy and classification report

accuracy = accuracy\_score(y\_test, y\_pred)

class\_report = classification\_report(y\_test, y\_pred)

# Print results in a tabular format

print("Model Performance:")

print(tabulate([

["Accuracy", accuracy],

["Classification Report", "\n" + class\_report]

], headers=["Metric", "Score"]))

Best parameters: {'n\_estimators': 100, 'min\_samples\_split': 5, 'min\_samples\_leaf': 1, 'max\_depth': None, 'bootstrap': False}

Training the model...

Model trained in 3.37 seconds.

Cross-Validation Results:

Metric Score

---------------- ------------------

Mean Train Score -

Mean Test Score 0.8500545364728316

Model Performance:

Metric Score

--------------------- -----------------------------------------------------

Accuracy 0.8791356366743988

Classification Report precision recall f1-score support

0 0.88 0.88 0.88 4070

1 0.88 0.88 0.88 4121

accuracy 0.88 8191

macro avg 0.88 0.88 0.88 8191

weighted avg 0.88 0.88 0.88 8191

1. XGBoost

from google.colab import drive

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from xgboost import XGBClassifier

from sklearn.metrics import accuracy\_score, classification\_report

from tabulate import tabulate

from sklearn.preprocessing import LabelEncoder

import time

import numpy as np

# Load the dataset

print("Loading data...")

start\_time = time.time()

drive.mount('/content/drive')

data = pd.read\_csv('/content/drive/MyDrive/mac\_combined\_May\_17.csv')

print(f"Data loaded in {time.time() - start\_time:.2f} seconds.\n")

# Convert categorical features to numeric

le = LabelEncoder()

for col in data.columns:

if data[col].dtype == 'object':

data[col] = le.fit\_transform(data[col].astype(str))

# Split into features and target

target\_col = 'Flag' # Update this to your actual target column name

X = data.drop(columns=[target\_col])

y = data[target\_col]

# Train an XGBoost model to compute feature importance

print("Computing feature importance...")

model = XGBClassifier(use\_label\_encoder=False, eval\_metric='logloss', tree\_method='gpu\_hist')

model.fit(X, y)

# Select the top 20 most important features

important\_features = np.argsort(model.feature\_importances\_)[-20:]

# Select only the important features from X

X = X[X.columns[important\_features]]

# Split 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)

# Train the model

print("Training the model...")

start\_time = time.time()

model = XGBClassifier(use\_label\_encoder=False, eval\_metric='logloss', tree\_method='gpu\_hist')

model.fit(X\_train, y\_train)

print(f"Model trained in {time.time() - start\_time:.2f} seconds.\n")

# Predict on the test set

y\_pred = model.predict(X\_test)

# Print accuracy and classification report

accuracy = accuracy\_score(y\_test, y\_pred)

class\_report = classification\_report(y\_test, y\_pred)

# Print results in a tabular format

print("Model Performance:")

print(tabulate([

["Accuracy", accuracy],

["Classification Report", "\n" + class\_report]

], headers=["Metric", "Score"]))

Model Performance:

Metric Score

--------------------- -----------------------------------------------------

Accuracy 0.8550848492247589

Classification Report precision recall f1-score support

0 0.92 0.78 0.84 4070

1 0.81 0.93 0.87 4121

accuracy 0.86 8191

macro avg 0.86 0.85 0.85 8191

weighted avg 0.86 0.86 0.85 8191

1. Ensemble\_Random\_forest\_XGBoost

# Import necessary libraries

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from sklearn.preprocessing import LabelEncoder

from sklearn.ensemble import RandomForestClassifier, VotingClassifier

from sklearn.metrics import accuracy\_score, classification\_report

from imblearn.over\_sampling import SMOTE

from xgboost import XGBClassifier

from google.colab import drive

# Load the dataset

print("Loading data...")

start\_time = time.time()

drive.mount('/content/drive')

data = pd.read\_csv('/content/drive/MyDrive/mac\_combined\_May\_17.csv')

print(f"Data loaded in {time.time() - start\_time:.2f} seconds.\n")

# Define target and drop it from main data

y = data['Flag']

X = data.drop(['Flag'], axis=1)

# Encoding the Website feature

le = LabelEncoder()

X['Website'] = le.fit\_transform(X['Website'])

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Define classifiers with 'balanced' class weights

rf\_clf = RandomForestClassifier(class\_weight='balanced', random\_state=42)

xgb\_clf = XGBClassifier(use\_label\_encoder=False, eval\_metric='logloss', verbosity=2, tree\_method='gpu\_hist')

# Parameters for GridSearchCV

parameters\_rf = {

'n\_estimators': [200, 300, 500],

'max\_depth': [None, 10, 15, 20],

'min\_samples\_split': [2, 5, 10]

}

parameters\_xgb = {

'n\_estimators': [200, 300, 500],

'max\_depth': [10, 15, 20],

'learning\_rate': [0.01, 0.05, 0.1],

'subsample': [0.6, 0.8, 1.0],

'colsample\_bytree': [0.6, 0.8, 1.0]

}

# Grid search for hyperparameter tuning

grid\_rf = GridSearchCV(rf\_clf, parameters\_rf, cv=5)

grid\_xgb = GridSearchCV(xgb\_clf, parameters\_xgb, cv=5)

# Training the models

for clf, label in zip([grid\_rf, grid\_xgb], ['Random Forest', 'XGBoost']):

start\_time = time.time()

clf.fit(X\_train, y\_train)

print(f"\nTraining time for {label}: {time.time() - start\_time:.2f} seconds.\n")

# Predict on the test set

y\_pred = clf.predict(X\_test)

# Print accuracy and classification report

accuracy = accuracy\_score(y\_test, y\_pred)

class\_report = classification\_report(y\_test, y\_pred)

print(f"{label} Model Performance:")

print(f"Best Parameters: {clf.best\_params\_}")

print(f"Accuracy: {accuracy}")

print(f"Classification Report: \n{class\_report}")

print("\n---------------------------------\n")

# Create the ensemble model

ensemble\_model = VotingClassifier(estimators=[('rf', grid\_rf.best\_estimator\_), ('xgb', grid\_xgb.best\_estimator\_)], voting='hard')

ensemble\_model.fit(X\_train, y\_train)

# Make predictions

y\_pred\_ensemble = ensemble\_model.predict(X\_test)

# Print accuracy and classification report

accuracy\_ensemble = accuracy\_score(y\_test, y\_pred\_ensemble)

class\_report\_ensemble = classification\_report(y\_test, y\_pred\_ensemble)

print(f"Ensemble Model Performance:")

print(f"Accuracy: {accuracy\_ensemble}")

print(f"Classification Report: \n{class\_report\_ensemble}")

Random Forest Model Performance:

Best Parameters: {'max\_depth': None, 'min\_samples\_split': 2, 'n\_estimators': 300}

Accuracy: 0.87950189232084

Classification Report:

precision recall f1-score support

0 0.89 0.86 0.88 4070

1 0.87 0.90 0.88 4121

accuracy 0.88 8191

macro avg 0.88 0.88 0.88 8191

weighted avg 0.88 0.88 0.88 8191

XGBoost Model Performance:

Best Parameters: {'colsample\_bytree': 0.6, 'learning\_rate': 0.1, 'max\_depth': 15, 'n\_estimators': 200, 'subsample': 1.0}

Accuracy: 0.8559394457331218

Classification Report:

precision recall f1-score support

0 0.90 0.80 0.85 4070

1 0.82 0.91 0.86 4121

accuracy 0.86 8191

macro avg 0.86 0.86 0.86 8191

weighted avg 0.86 0.86 0.86 8191

---------------------------------

/usr/local/lib/python3.10/dist-packages/xgboost/sklearn.py:1395: UserWarning: `use\_label\_encoder` is deprecated in 1.7.0.

warnings.warn("`use\_label\_encoder` is deprecated in 1.7.0.")

Ensemble Model Performance:

Accuracy: 0.8730313759003785

Classification Report:

precision recall f1-score support

0 0.86 0.90 0.88 4070

1 0.89 0.85 0.87 4121

accuracy 0.87 8191

macro avg 0.87 0.87 0.87 8191

weighted avg 0.87 0.87 0.87 8191