Assignment 03

Hyperparameter Tunning - Group 05

Haoqi Wang Kaiyue Wei Yanyan Li Xulu Wang

September 29, 2023

Instructions (Remove the instructions before submission)

This assignment will deal with tuning the hyperparameters for the online shopping dataset. Make sure to remove the instructions and only keep Q6 onward. The qmd file of this assignment is located in the files folder.

- This is a group assignment with independent submission on Canvas. Collaboration is essential. Use Git for version control.
- Begin by setting your random seed as the last four digits of your GUID.
- Prefix each variable with 'g#groupnumber' (e.g., g01_variableName) to ensure uniqueness and to demonstrate originality in your group's work.
- add the names of all group members to the YAML header above.
- Use of Generative AI tools, including but not restricted to GPT-3 is strictly prohibited.

Git Commit and Collaboration

- This is a group assignment. Collaboration is essential. Use Git for version control.
- Regular and meaningful commit messages are expected, indicating steady progress and contributions from all group members.
- Avoid large, infrequent commits. Instead, aim for more minor, frequent updates showing your code's evolution and thoughts.
- Collaboration tools, especially Git, should be used as a backup tool and a truly collaborative platform. Discuss, review, and merge each other's contributions.

Grading Criteria

- The assignment is worth 75 points.

- There are three grading milestones in the assignment.
  - Adherence to Requirements, Coding Standards, Documentation, Runtime, and Efficiency (22 Points)
    - Adherence to Requirements (5 Points): Ensure all the given requirements of the assignment, including Git commits and collaboration, are met.
    - Coding Standards (5 Points): Code should be readable and maintainable. Ensure appropriate variable naming and code commenting.
    - Documentation (6 Points): Provide explanations or reasoning for using a particular command and describe the outputs. Avoid vague descriptions; aim for clarity and depth.
    - Runtime (3 Points): The code should execute without errors and handle possible exceptions.
    - Efficiency (3 Points): Implement efficient coding practices, avoid redundancy, and optimize for performance where applicable.
  - Collaborative Programming (13 Points)
    - GitHub Repository Structure (3 Points): A well-organized repository with clear directory structures and meaningful file names.
    - Number of Commits (3 Points): Reflects steady progress and contributions from all group members.
    - Commit Quality (3 Points): Clear, descriptive commit messages representing logical chunks of work. Avoid trivial commits like "typo fix."
    - Collaboration & Contribution (4 Points): Demonstrated teamwork where each member contributes significantly. This can be seen through pull requests, code reviews, and merge activities.
  - Assignment Questions (40 Points)

Adherence to Requirements, Coding Standards, Documentation, Runtime, and Efficiency (22 Points)

This section is graded based on adherence to Requirements, Coding Standards, Documentation, Runtime, and Efficiency.

Collaborative Programming (13 Points)

This section is graded based on the Github submission. Each person needs to have made commits to the repository. GitHub Repository Structure, Number of Commits, Commit Quality, Collaboration, and Contribution are generally graded based on the group's overall performance. However, if there is a significant difference in the number of commits or contributions between group members, the instructor may adjust the grade accordingly.

Assignment Questions (40 Points)

Data Preparation (7 Points):

```
# Add data preparation code here
import random
import pandas as pd

# set random seed
random.seed(1494)

# load dataset
g05_url = "https://archive.ics.uci.edu/ml/machine-learning-databases/00468/online_shoppers
g05_df = pd.read_csv(g05_url)

# check the data info
g05_df.head()
g05_df.info()


# check any missing values and remove it
g05_df.isnull().sum()
g05_df.dropna(inplace=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12330 entries, 0 to 12329
Data columns (total 18 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Administrative           12330 non-null  int64
 1   Administrative_Duration  12330 non-null  float64
 2   Informational            12330 non-null  int64
 3   Informational_Duration   12330 non-null  float64
 4   ProductRelated           12330 non-null  int64
 5   ProductRelated_Duration  12330 non-null  float64
 6   BounceRates              12330 non-null  float64
 7   ExitRates                12330 non-null  float64
 8   PageValues               12330 non-null  float64
 9   SpecialDay               12330 non-null  float64
 10  Month                    12330 non-null  object
```

```
 11  OperatingSystems         12330 non-null  int64
 12  Browser                  12330 non-null  int64
 13  Region                   12330 non-null  int64
 14  TrafficType              12330 non-null  int64
 15  VisitorType              12330 non-null  object
 16  Weekend                  12330 non-null  bool
 17  Revenue                  12330 non-null  bool
dtypes: bool(2), float64(7), int64(7), object(2)
memory usage: 1.5+ MB
```

Load the dataset and display the dataframe (2 Points).

```
# Add code here
g05_df.head()
```

	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated	I
0	0	0.0	0	0.0	1	0
1	0	0.0	0	0.0	2	0
2	0	0.0	0	0.0	1	0
3	0	0.0	0	0.0	2	2
4	0	0.0	0	0.0	10	0

Use `describe` to provide statistics on the pandas Dataframe (2 Points).

```
# Add code here
g05_df.describe()
```

	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelate
count	12330.000000	12330.000000	12330.000000	12330.000000	12330.000000
mean	2.315166	80.818611	0.503569	34.472398	31.731468
std	3.321784	176.779107	1.270156	140.749294	44.475503
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	7.000000
50%	1.000000	7.500000	0.000000	0.000000	18.000000
75%	4.000000	93.256250	0.000000	0.000000	38.000000
max	27.000000	3398.750000	24.000000	2549.375000	705.000000

Split the dataset into a Training set and a Test set. Justify your preferred split (3 Points).

```
# Add code here

from sklearn.model_selection import train_test_split

# Assuming the target variable is 'Revenue'
g05_X = g05_df.drop('Revenue', axis=1)
g05_y = g05_df['Revenue']

# Splitting the dataset into training (80%) and test set (20%)
g05_X_train, g05_X_test, g05_y_train, g05_y_test = train_test_split(g05_X, g05_y, test_siz

print(f"Training set has {g05_X_train.shape[0]} samples.")
print(f"Test set has {g05_X_test.shape[0]} samples.")
```

```
Training set has 9864 samples.
Test set has 2466 samples.
```

Classification Routine (12 Points):

Execute a classification routine using RandomForestClassifier(), BaggingClassifier(), and XGboostclassifier(). Independently output the accuracy box plot as discussed in class. Use any package you are comfortable with (seaborn, matplotlib).

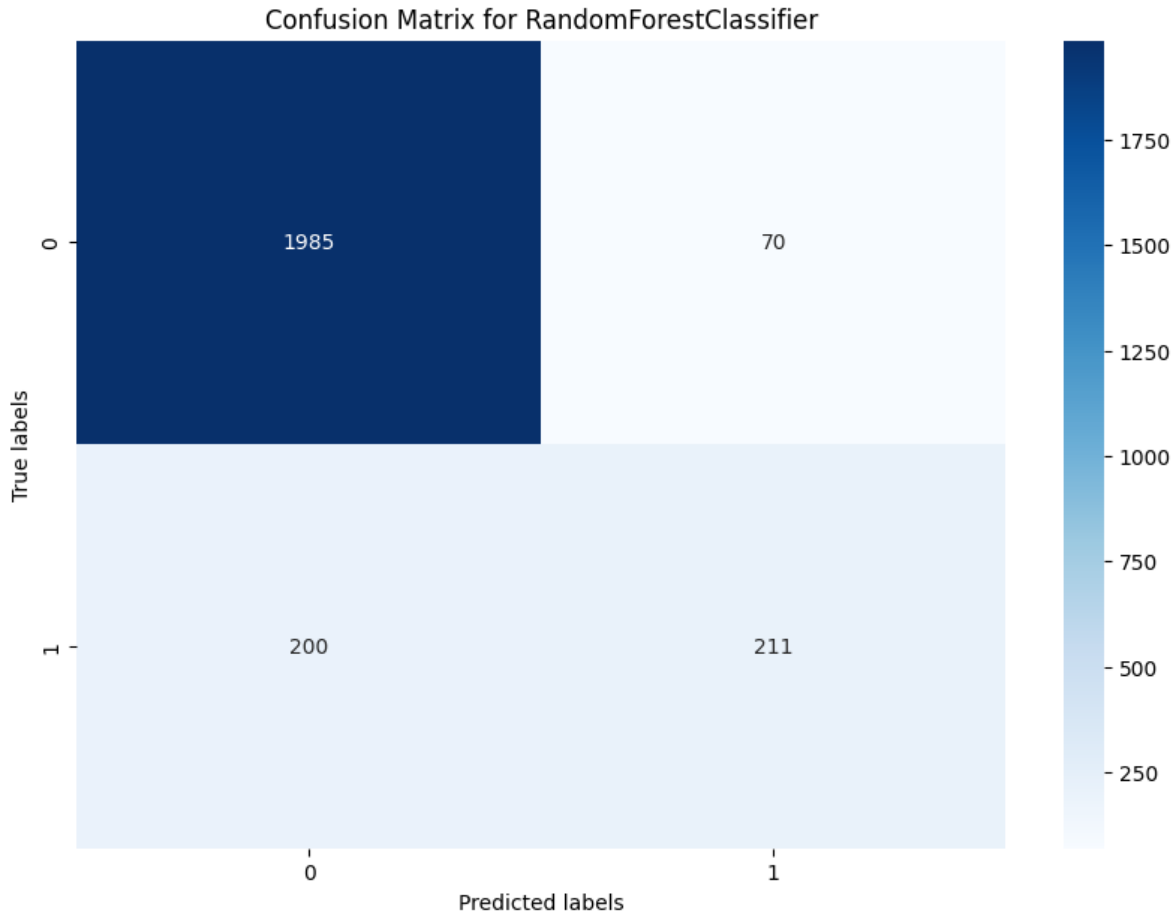
RandomForestClassifier():

```
# Add code here
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# one hot encoding
g05_X_train = pd.get_dummies(g05_X_train)
g05_X_test = pd.get_dummies(g05_X_test)
```

```python
# built rf model
g05_rf = RandomForestClassifier(n_estimators=100)
g05_rf.fit(g05_X_train, g05_y_train)

# make prediction
g05_y_rf_pred = g05_rf.predict(g05_X_test)

# calculate the accuracy
print(f"Accuracy: {accuracy_score(g05_y_test, g05_y_rf_pred) * 100:.2f}%")
print("\nClassification Report:\n", classification_report(g05_y_test, g05_y_rf_pred))

# Calculating the confusion matrix
g05_cm = confusion_matrix(g05_y_test, g05_y_rf_pred)

# Printing the confusion matrix using seaborn for better visualization
plt.figure(figsize=(10,7))
sns.heatmap(g05_cm, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix for RandomForestClassifier')
plt.show()
```

```
Missing colon in file PosixPath('/Users/iris/.matplotlib/stylelib/mycustom.mplstyle'), line 1
Missing colon in file PosixPath('/Users/iris/.matplotlib/stylelib/lab-1.2.mplstyle'), line 2
```

```
Accuracy: 89.05%

Classification Report:
              precision    recall  f1-score   support

       False       0.91      0.97      0.94      2055
        True       0.75      0.51      0.61       411

    accuracy                           0.89      2466
   macro avg       0.83      0.74      0.77      2466
weighted avg       0.88      0.89      0.88      2466
```

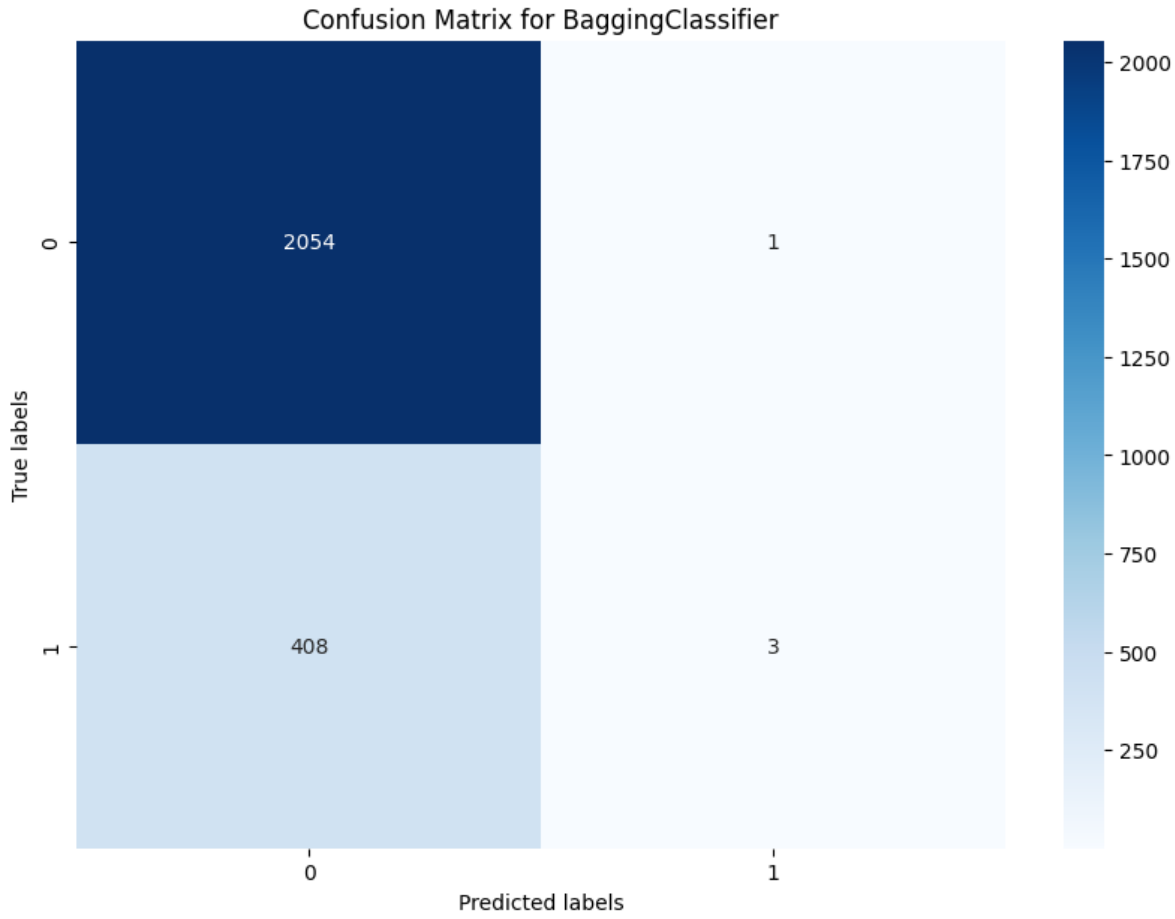


BaggingClassifier():

```
# Add code here
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import make_scorer, accuracy_score, classification_report, confusion_

seed = 3225
num_folds = 5
scoring = {'AUC': 'roc_auc', 'Accuracy': make_scorer(accuracy_score)}

# built rf model
# g05_bagg = BaggingClassifier(n_estimators=100)
g05_bagg = BaggingClassifier(DecisionTreeClassifier(max_features="auto",
```

```
                                                    splitter="random",
                                                    max_leaf_nodes=16,
                                                    random_state=seed),
                                n_estimators=100)

g05_bagg.fit(g05_X_train, g05_y_train)

# make prediction
g05_y_bagg_pred = g05_bagg.predict(g05_X_test)

# calculate the accuracy
print(f"Accuracy: {accuracy_score(g05_y_test, g05_y_bagg_pred) * 100:.2f}%")
print("\nClassification Report:\n", classification_report(g05_y_test, g05_y_bagg_pred))

# Calculating the confusion matrix
g05_bagg_cm = confusion_matrix(g05_y_test, g05_y_bagg_pred)

# Printing the confusion matrix using seaborn for better visualization
plt.figure(figsize=(10,7))
sns.heatmap(g05_bagg_cm, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix for BaggingClassifier')
plt.show()
```

```
Accuracy: 83.41%

Classification Report:
              precision    recall  f1-score   support

       False       0.83      1.00      0.91      2055
        True       0.75      0.01      0.01       411

    accuracy                           0.83      2466
   macro avg       0.79      0.50      0.46      2466
weighted avg       0.82      0.83      0.76      2466
```

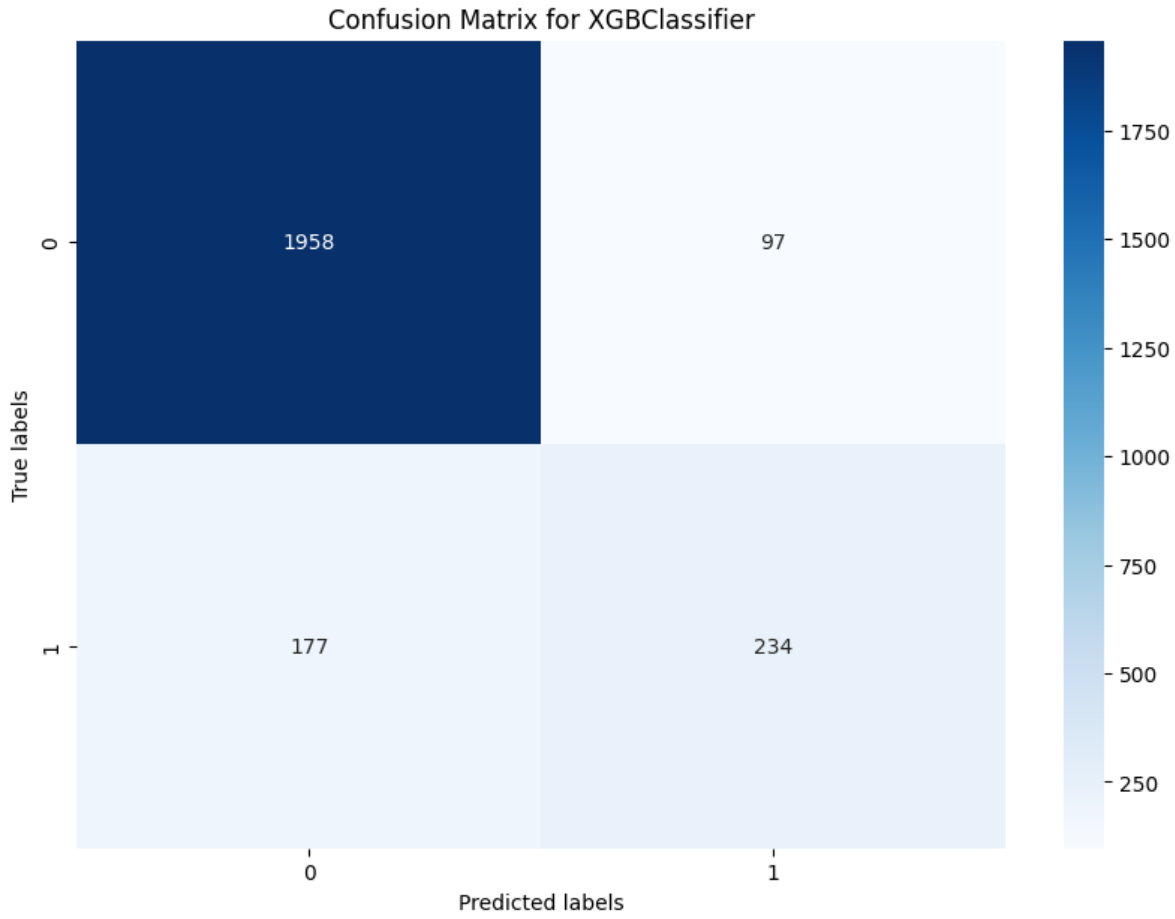


XGboostclassifier():

```
# Add code here
from xgboost import XGBClassifier

# Intialize XGBClassifier
g05_xgb = XGBClassifier(n_estimators=100)
# Fit model
g05_xgb_fit = g05_xgb.fit(g05_X_train, g05_y_train)
# Make predictions
g05_pred_xgb = g05_xgb_fit.predict(g05_X_test)

# Accuracy and classification report
g05_xgb_accuracy = accuracy_score(y_true=g05_y_test, y_pred=g05_pred_xgb)
```

```
print("XGBoost Classifier Accuracy: {:.2%}".format(g05_xgb_accuracy))
print(classification_report(g05_y_test, g05_pred_xgb))

# confusion matrix
g05_cm_xgb = confusion_matrix(g05_y_test, g05_pred_xgb)
# Printing the confusion matrix using seaborn for better visualization
plt.figure(figsize=(10,7))
sns.heatmap(g05_cm_xgb, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix for XGBClassifier')
plt.show()
```

```
XGBoost Classifier Accuracy: 88.89%
              precision    recall  f1-score   support

       False       0.92      0.95      0.93      2055
        True       0.71      0.57      0.63       411

    accuracy                           0.89      2466
   macro avg       0.81      0.76      0.78      2466
weighted avg       0.88      0.89      0.88      2466
```



Classification with GridSearchCV (8 Points):

Replicate the classification from Q2 using GridsearchCV().

RandomForestClassifier with GridSearchCV

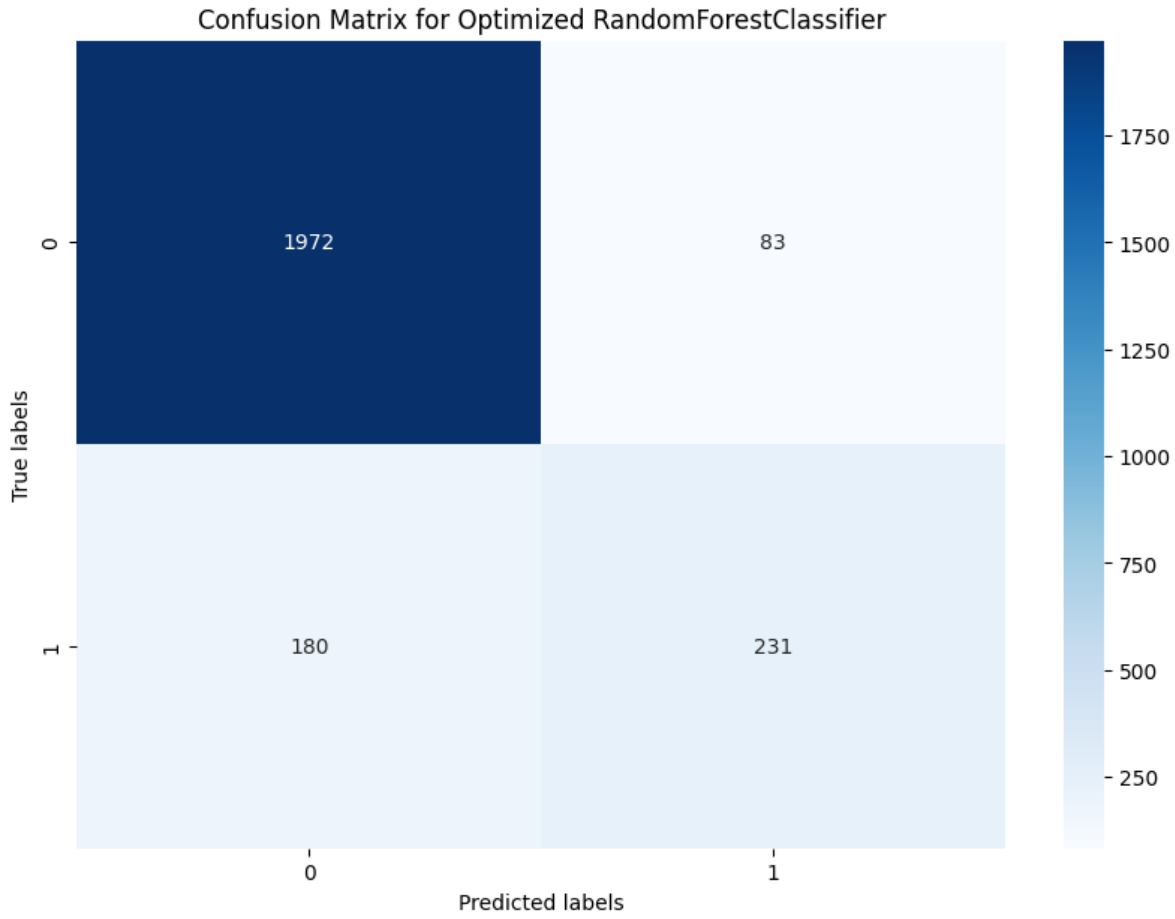
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.model_selection import GridSearchCV
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd  # Make sure to import pandas
```

```
# Define the parameter grid
g05_param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Create a RandomForestClassifier
g05_rf = RandomForestClassifier()

# Instantiate GridSearchCV with the model and the parameter grid
g05_grid_search = GridSearchCV(estimator=g05_rf, param_grid=g05_param_grid,
                               cv=5, n_jobs=-1, verbose=2, scoring='accuracy')

# Fit the grid search to the data
g05_grid_search.fit(g05_X_train, g05_y_train)

# Display the best score and hyperparameters
print("Best: %f using %s" % (g05_grid_search.best_score_, g05_grid_search.best_params_))

# Get the best model
g05_best_rf = g05_grid_search.best_estimator_

# Make prediction
g05_y_rf_pred = g05_best_rf.predict(g05_X_test)
```

```
# Calculate and display the accuracy
accuracy = accuracy_score(g05_y_test, g05_y_rf_pred)
print("\nAccuracy: {:.2%}\n".format(accuracy))

# Display the classification report
print("Classification Report:")
print(classification_report(g05_y_test, g05_y_rf_pred))

# Calculating the confusion matrix
g05_cm = confusion_matrix(g05_y_test, g05_y_rf_pred)

# Printing the confusion matrix using seaborn for better visualization
```

```
plt.figure(figsize=(10,7))
sns.heatmap(g05_cm, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix for Optimized RandomForestClassifier')
plt.show()
```

```
Accuracy: 89.33%

Classification Report:
              precision    recall  f1-score   support

       False       0.92      0.96      0.94      2055
        True       0.74      0.56      0.64       411

    accuracy                           0.89      2466
   macro avg       0.83      0.76      0.79      2466
weighted avg       0.89      0.89      0.89      2466
```



BaggingClassifier with GridSearchCV

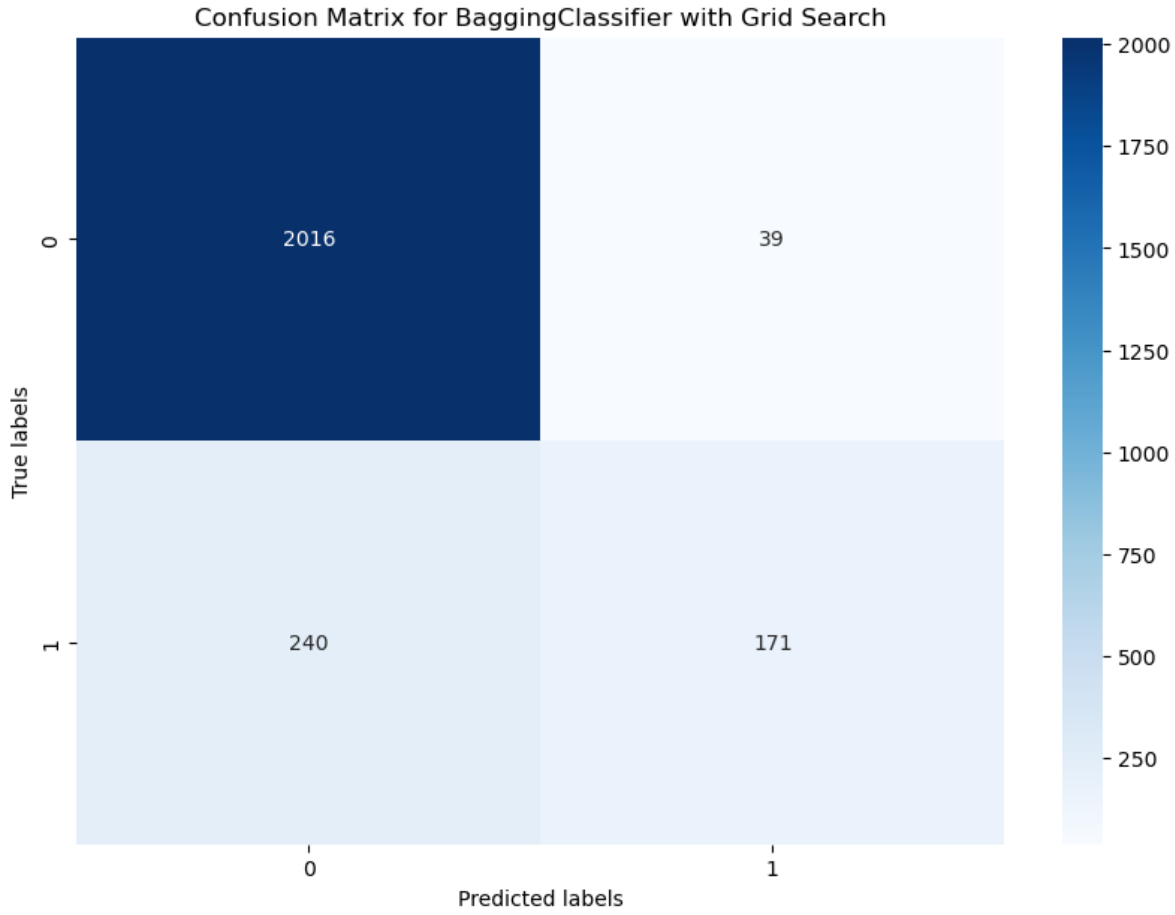
```
# Add code here
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import StratifiedKFold
# BaggingClasifie
g_05_bagg_grid_para = {
    "n_estimators" : [10, 25, 50, 100, 250, 500, 1000],
    "bootstrap" : [True],
    "max_samples" : [0.05, 0.1, 0.2, 0.5, 0.7],
    "oob_score" : [True]  # Only for bootstrap equals to True
}
```

```
kfold = StratifiedKFold(n_splits=num_folds, random_state=seed, shuffle=True)

g_05_bagg_grid = GridSearchCV(estimator=g05_bagg,
                     param_grid=g_05_bagg_grid_para,
                     cv=kfold,
                     scoring=scoring,
                     return_train_score=True,
                     n_jobs=-1,
                     refit="AUC",
                     verbose=10)

g_05_best_bagg_grid = g_05_bagg_grid.fit(g05_X_train, g05_y_train)
# print("Best: %f using %s" % (g_05_best_bagg_grid.best_score_,g_05_best_bagg_grid.best_pa
```

```
print("Best: %f using %s" % (g_05_best_bagg_grid.best_score_,g_05_best_bagg_grid.best_para

g05_y_bagg_grid_pred = g_05_best_bagg_grid.predict(g05_X_test)

# calculate the accuracy
print(f"Accuracy: {accuracy_score(g05_y_test, g05_y_bagg_grid_pred) * 100:.2f}%")
print("\nClassification Report:\n", classification_report(g05_y_test, g05_y_bagg_grid_pred

# Calculating the confusion matrix
g05_bagg_grid_cm = confusion_matrix(g05_y_test, g05_y_bagg_grid_pred)

# Printing the confusion matrix using seaborn for better visualization
plt.figure(figsize=(10,7))
sns.heatmap(g05_bagg_grid_cm, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix for BaggingClassifier with Grid Search')
plt.show()
```

```
Best: 0.921183 using {'bootstrap': True, 'max_samples': 0.7, 'n_estimators': 1000, 'oob_score
Accuracy: 88.69%

Classification Report:
              precision    recall  f1-score   support

       False       0.89      0.98      0.94      2055
        True       0.81      0.42      0.55       411
```

accuracy			0.89	2466
macro avg	0.85	0.70	0.74	2466
weighted avg	0.88	0.89	0.87	2466

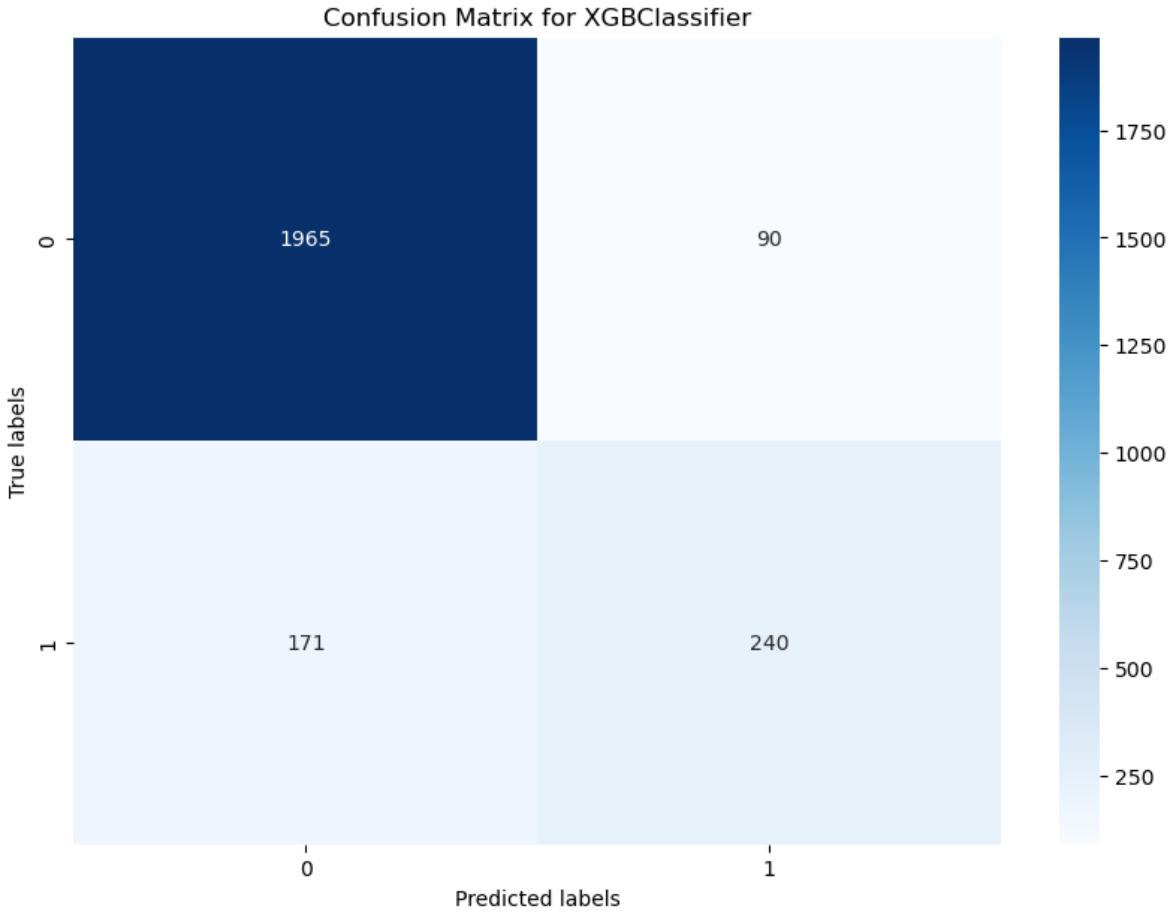


XGBClassifier with GridSearchCV

```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RepeatedStratifiedKFold

# define model
g05_xgb_model = XGBClassifier()
# define search space
g05_xgb_space = dict()
```

```
g05_xgb_space['booster'] = ['gbtree', 'gblinear', 'dart']
g05_xgb_space['learning_rate'] = [1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1, 10]
g05_xgb_space['gamma'] = [1e-4, 1e-3, 1e-2, 1e-1, 1, 10]

# define evaluation
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)

# define grid search
g05_xgb_grid_search = GridSearchCV(g05_xgb_model, g05_xgb_space, scoring='accuracy', n_job

g05_xgb_grid_result = g05_xgb_grid_search.fit(g05_X_train, g05_y_train)
g05_xgb_grid_pred = g05_xgb_grid_result.predict(g05_X_test)
# summarize result
print('Best Score: %s' % g05_xgb_grid_result.best_score_)
print('Best Hyperparameters chosen by GridSearchCV: %s' % g05_xgb_grid_result.best_params_

# Accuracy and classification report
g05_xgb_grid_accuracy = accuracy_score(y_true=g05_y_test, y_pred=g05_xgb_grid_pred)
print("XGBoost Classifier Accuracy: {:.2%}".format(g05_xgb_grid_accuracy))
print(classification_report(g05_y_test, g05_xgb_grid_pred))

# confusion matrix
g05_cm_grid_xgb = confusion_matrix(g05_y_test, g05_xgb_grid_pred)
# Printing the confusion matrix using seaborn for better visualization
plt.figure(figsize=(10,7))
sns.heatmap(g05_cm_grid_xgb, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix for XGBClassifier')
plt.show()
```

```
Best Score: 0.9053462274855237
Best Hyperparameters chosen by GridSearchCV: {'booster': 'gbtree', 'gamma': 1, 'learning_rat
XGBoost Classifier Accuracy: 89.42%
              precision    recall  f1-score   support

       False       0.92      0.96      0.94      2055
        True       0.73      0.58      0.65       411

    accuracy                           0.89      2466
   macro avg       0.82      0.77      0.79      2466
weighted avg       0.89      0.89      0.89      2466
```



Classification with RandomSearchCV (8 Points):

Replicate the classification from Q2 using RandomSearchCV().

RandomForestClassifier with RandomSearchCV

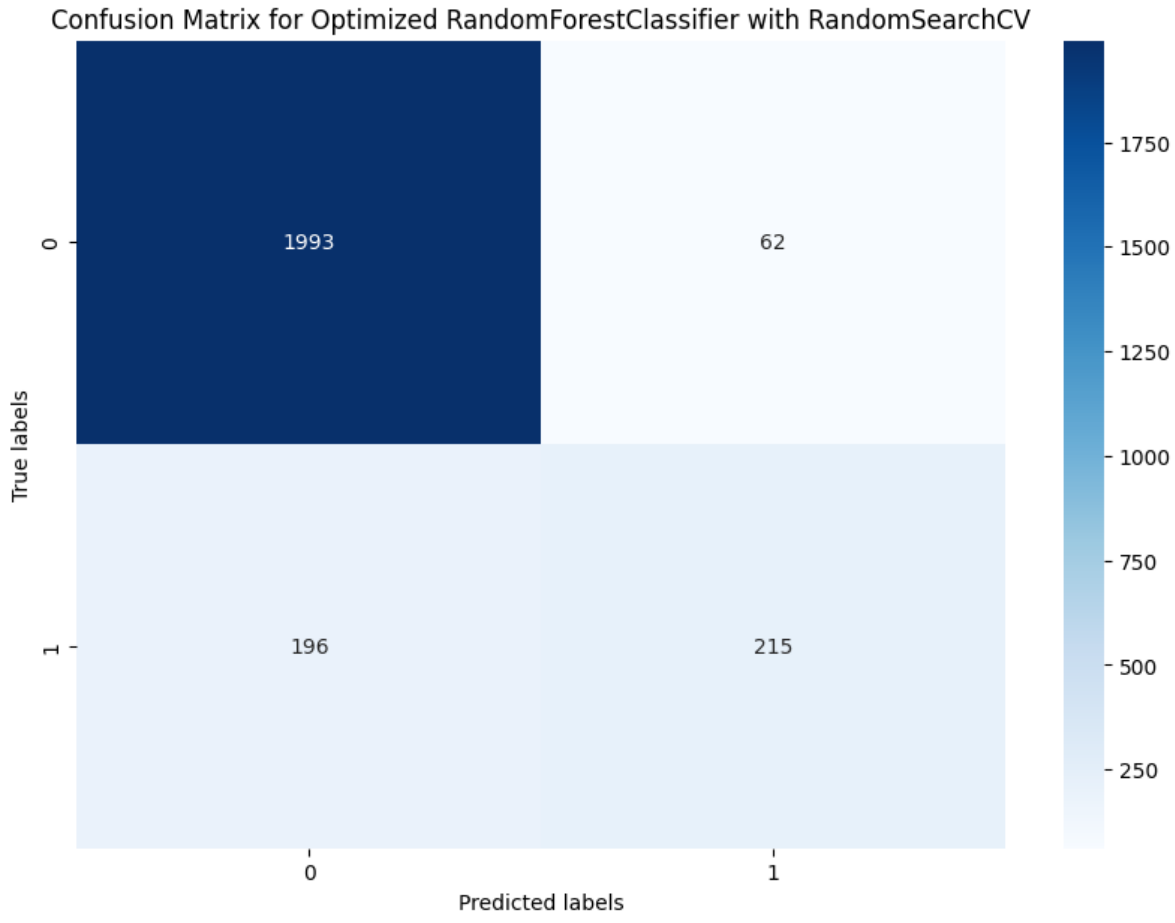
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint
import seaborn as sns
import matplotlib.pyplot as plt
```

```
# Define the parameter space
g05_param_dist = {
    'n_estimators': randint(10, 200),
    'max_depth': [None] + list(randint(1, 20).rvs(10)),
    'min_samples_split': randint(2, 11),
    'min_samples_leaf': randint(1, 11),
    'bootstrap': [True, False]
}

# Create a RandomForestClassifier
g05_rf = RandomForestClassifier()

# Instantiate RandomizedSearchCV with the model and the parameter distribution
random_search = RandomizedSearchCV(estimator=g05_rf, param_distributions=g05_param_dist,
                                   n_iter=100, cv=5, n_jobs=-1, verbose=2, scoring='accura

# Fit the random search to the data
random_search.fit(g05_X_train, g05_y_train)

# Display the best score and hyperparameters
print("Best: %f using %s" % (random_search.best_score_, random_search.best_params_))

# Get the best model
g05_best_rf = random_search.best_estimator_

# Make prediction
g05_y_rf_pred = g05_best_rf.predict(g05_X_test)
```

```
# Display results
print("\nAccuracy: {:.2%}".format(accuracy_score(g05_y_test, g05_y_rf_pred)))
print("Classification Report:")
print(classification_report(g05_y_test, g05_y_rf_pred))

# Calculating the confusion matrix
g05_cm = confusion_matrix(g05_y_test, g05_y_rf_pred)

# Printing the confusion matrix using seaborn for better visualization
plt.figure(figsize=(10,7))
sns.heatmap(g05_cm, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
```

```
plt.title('Confusion Matrix for Optimized RandomForestClassifier with RandomSearchCV')
plt.show()
```

```
Accuracy: 89.54%
Classification Report:
              precision    recall  f1-score   support

       False       0.91      0.97      0.94      2055
        True       0.78      0.52      0.62       411

    accuracy                           0.90      2466
   macro avg       0.84      0.75      0.78      2466
weighted avg       0.89      0.90      0.89      2466
```



BaggingClassifier with RandomSearchCV

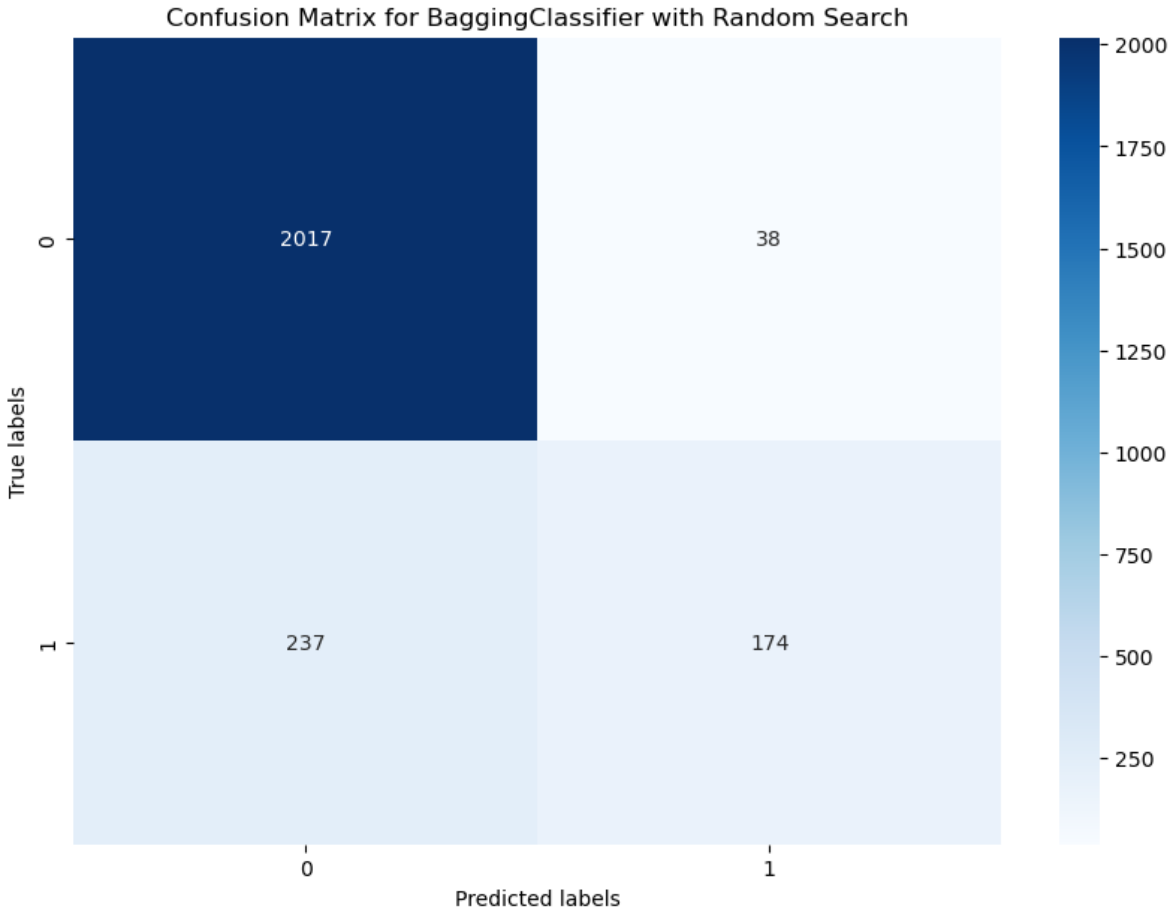
```
# Add code here
import numpy as np
from sklearn.model_selection import RandomizedSearchCV
```

```
num_folds = 5
kfold = StratifiedKFold(n_splits=num_folds, random_state=seed, shuffle=True)

# Number of trees in bagging classifer base model - decision tree
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = [0.5, 0.7, 1.0]
# max samples to consider at every split
max_samples = [0.05, 0.1, 0.2, 0.5, 0.6, 0.7]

# Create the random grid
g_05_bagg_random_para = {'n_estimators' : n_estimators,
                         'max_features': max_features,
                         'max_samples' : max_samples}

g_05_bagg_random=RandomizedSearchCV(estimator=g05_bagg,
                                param_distributions=g_05_bagg_random_para,
                                n_iter=50,
                                cv=kfold,
                                verbose=2,
                                random_state=seed,
                                n_jobs=-1)

g_05_best_bagg_random = g_05_bagg_random.fit(g05_X_train, g05_y_train)
```

```
best_random_grid=g_05_best_bagg_random.best_estimator_

print("Best: %f using %s" % (g_05_best_bagg_random.best_score_,g_05_best_bagg_random.best_

g05_y_bagg_random_pred = g_05_best_bagg_random.predict(g05_X_test)

# calculate the accuracy
print(f"Accuracy: {accuracy_score(g05_y_test, g05_y_bagg_random_pred) * 100:.2f}%")
print("\nClassification Report:\n", classification_report(g05_y_test, g05_y_bagg_random_pr
```

```
# Calculating the confusion matrix
g05_bagg_random_cm = confusion_matrix(g05_y_test, g05_y_bagg_random_pred)

# Printing the confusion matrix using seaborn for better visualization
plt.figure(figsize=(10,7))
sns.heatmap(g05_bagg_random_cm, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix for BaggingClassifier with Random Search')
plt.show()
```

```
Best: 0.902980 using {'n_estimators': 400, 'max_samples': 0.6, 'max_features': 1.0}
Accuracy: 88.85%

Classification Report:
              precision    recall  f1-score   support

       False       0.89      0.98      0.94      2055
        True       0.82      0.42      0.56       411

    accuracy                           0.89      2466
   macro avg       0.86      0.70      0.75      2466
weighted avg       0.88      0.89      0.87      2466
```

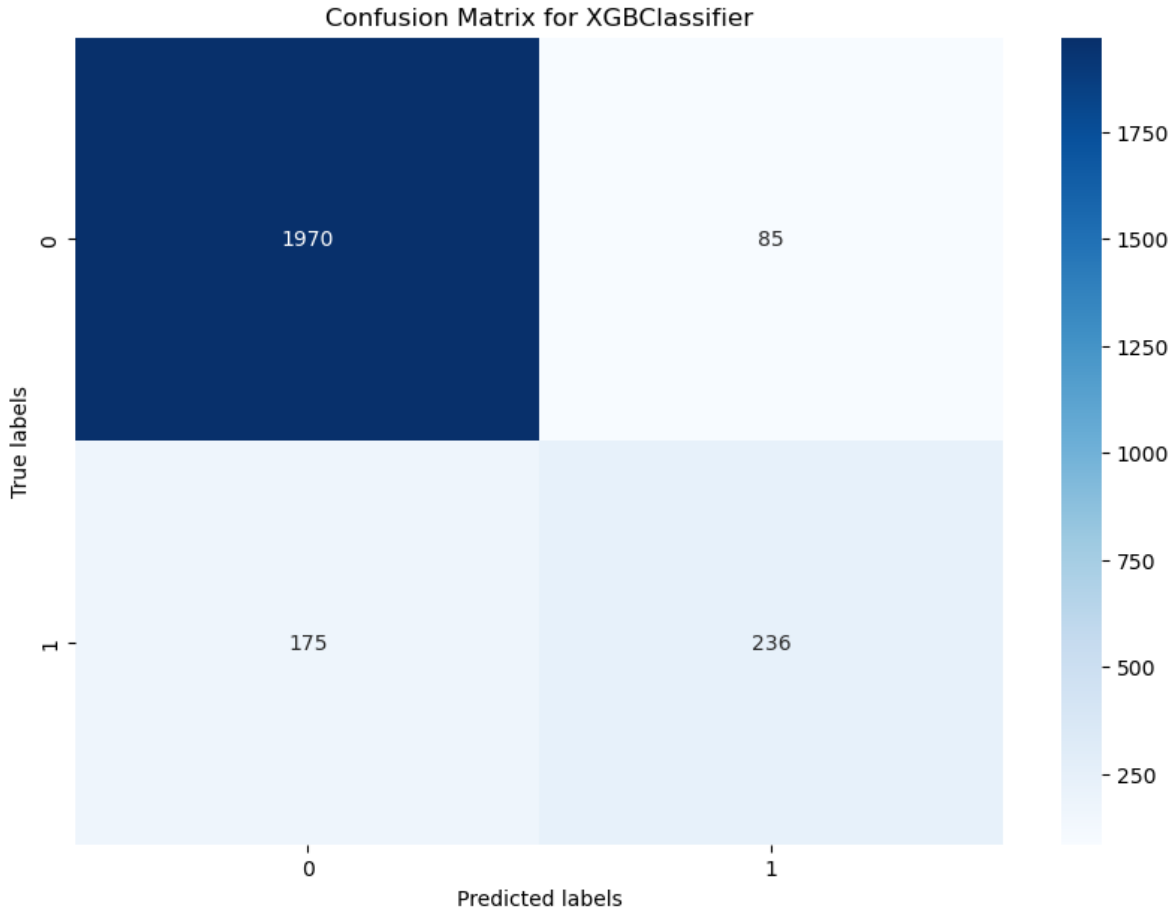


XGBClassifier with RandomSearchCV

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

# define search space
g05_xgb_space = dict()
g05_xgb_space['booster'] = ['gbtree', 'gblinear', 'dart']
g05_xgb_space['learning_rate'] = loguniform(1e-5, 10)
g05_xgb_space['gamma'] = loguniform(1e-5, 100)

# define random search
g05_xgb_random_search = RandomizedSearchCV(g05_xgb_model, g05_xgb_space, n_iter=300, scori
```

```
g05_xgb_random_result = g05_xgb_random_search.fit(g05_X_train, g05_y_train)
g05_xgb_random_pred = g05_xgb_random_result.predict(g05_X_test)
# summarize result
print('Best Score: %s' % g05_xgb_random_result.best_score_)
print('Best Hyperparameters: %s' % g05_xgb_random_result.best_params_)

# Accuracy and classification report
g05_xgb_random_accuracy = accuracy_score(y_true=g05_y_test, y_pred=g05_xgb_random_pred)
print("XGBoost Classifier Accuracy: {:.2%}".format(g05_xgb_random_accuracy))
print(classification_report(g05_y_test, g05_xgb_random_pred))

# confusion matrix
g05_cm_random_xgb = confusion_matrix(g05_y_test, g05_xgb_random_pred)
# Printing the confusion matrix using seaborn for better visualization
plt.figure(figsize=(10,7))
sns.heatmap(g05_cm_random_xgb, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix for XGBClassifier')
plt.show()
```

```
Best Score: 0.9061571581461291
Best Hyperparameters: {'booster': 'gbtree', 'gamma': 4.972389468490119, 'learning_rate': 0.1
XGBoost Classifier Accuracy: 89.46%
              precision    recall  f1-score   support

       False       0.92      0.96      0.94      2055
        True       0.74      0.57      0.64       411

    accuracy                           0.89      2466
   macro avg       0.83      0.77      0.79      2466
weighted avg       0.89      0.89      0.89      2466
```



Comparison and Analysis (5 Points):

Compare the results from Q2, Q3, and Q4. Describe the best hyperparameters for all three experiments.

RandomForestClassifier: All three models have a similar accuracy, hovering around 89%. The model optimized with RandomSearchCV has the highest accuracy at 89.54%, but the differences are marginal. The precision for the "True" class has increased slightly in the RandomSearchCV model, indicating that it has reduced the number of false positives relative to the original model. The F1-score considers both precision and recall. The model with GirdSearchCV has the highest F1-score for the "True" class, suggesting that it strikes the best balance between precision and recall among the three models. Thus, If you prioritize accuracy and minimizing false predictions, then the model optimized with RandomSearchCV is the best choice.However, if you want a better balance between precision and recall, the model with GridSearchCV seems to strike a better balance.

BaggingClassifier: For the overall accuracy, the models with RandomSearchCV(88.85%) and GridSearchCV(88.69%) improve a lot compared to original model(83.41%).The base BaggingClassifier struggles with predicting the 'True' class, having a recall of just 0.01. Both Grid Search and Random Search considerably enhance the performance, with Random Search having a minor edge over Grid Search in this context.In terms of optimization techniques, both Grid Search and Random Search have shown to be effective in improving the classifier's performance. However, the difference between the two optimization methods is marginal. Thus, Considering accuracy, precision, and recall, the BaggingClassifier with Random Search offers the best performance among the three. However, the difference between Grid Search and Random Search is slight, so the choice may come down to computational resources and time, as Grid Search can be more computationally expensive.

XGBboostClassifier: All three methods produce comparable accuracies, but both Grid Search and Random Search have a slight edge over the original model. Grid Search offers the best precision for the True class, while its recall is comparable to the original method. Random Search's precision for the True class is slightly better than the original but falls a bit behind Grid Search.Fewer False Positives and False Negatives are generally desirable. Grid Search has the fewest False Positives, while Random Search has the fewest False Negatives. In conclusion, while the differences are not massive, both Grid Search and Random Search offer improvements over the original setup. Grid Search seems to provide the best balance in terms of precision, recall, and F1-Score, especially for the class labeled as True. However, the choice between the methods might also depend on computational resources and time, as Grid Search can be more computationally intensive.