DATA Mining (Phase 6)

Team Name: Trio-Chargers

Team Members:

1. Sumanth Reddy (Team Head) (sdesi4@unh.newhaven.edu)
2. Lakshmi Kara Gupta (lchan3@unh.newhaven.edu)
3. Uday Kiran Karumburi Arumugam (ukaru1@unh.newhaven.edu)

Research Question:

Account Security: Detecting spam or legit, unusual login behavior and unauthorized access attempts through Machine Learning Algorithms.

Selected Data Set:

1. https://www.kaggle.com/datasets/khajahussainsk/facebook-spam-dataset

   The dataset can be used for building machine learning models. To collect the dataset, Facebook API and Facebook Graph API are used and the data is collected from public profiles by Kaggle.

List of Data Mining Techniques used:

- K nearest neighbors
- Support vector machines
- Decision trees
- Logistic regression

K-Nearest Neighbors (K-NN):

K-Nearest Neighbors is a type of instance-based learning, or lazy learning, where the function is only approximated locally, and all computation is deferred until function evaluation. It is a non-parametric method used for classification and regression.

Support Vector Machines (SVM):

Support Vector Machines are a set of supervised learning methods used for classification, regression, and outliers' detection. They are effective in high dimensional spaces and best suited for problems with complex domains where there are clear margins of separation in the data. SVM works by mapping data to a high-dimensional feature space so that data points can be categorized, even when the data are not otherwise linearly separable.

Decision Trees:

Decision Trees are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

Logistic Regression:

Logistic Regression is a statistical method for analyzing a dataset in which there are one or more independent variables that determine an outcome. The outcome is measured with a dichotomous variable (in which there are only two possible outcomes). It is used to predict a binary outcome (1 / 0, Yes / No, True / False) given a set of independent variables.

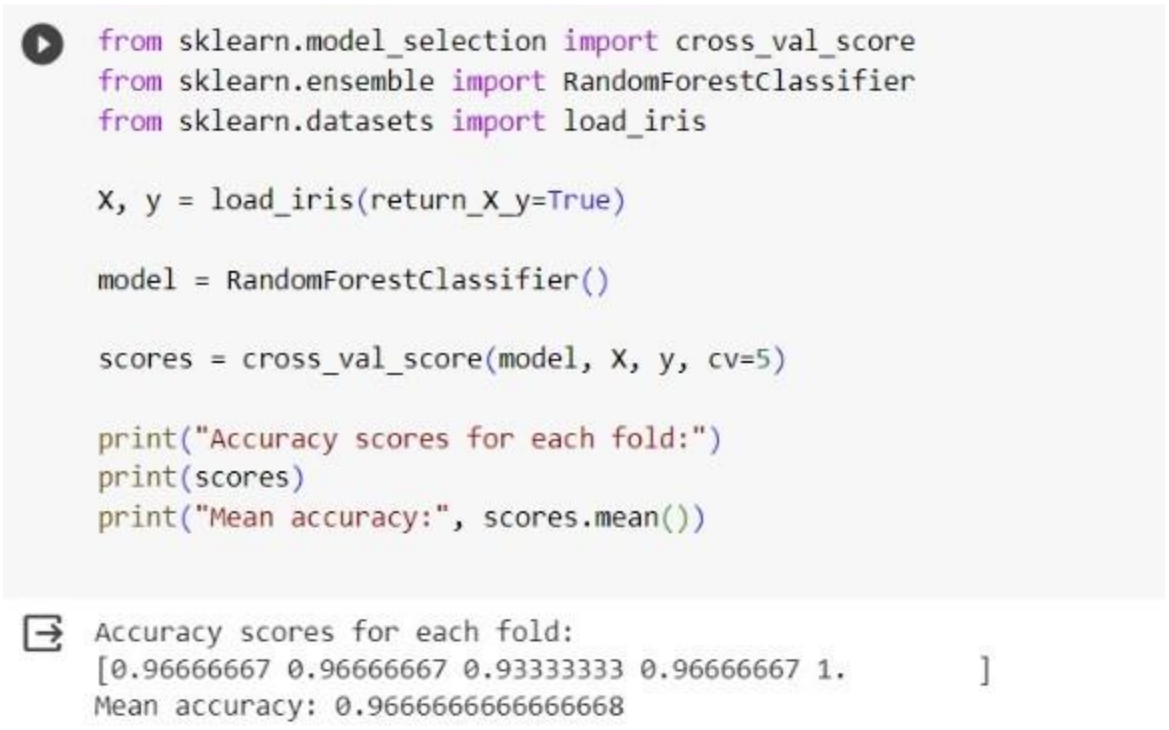
Description parameters and hyperparameters:

Model	Parameter	Hyperparameter
K-Nearest Neighbors	Points	n_neighbors (K), weights
Support Vector Machines	Support Vectors Weights (Coefficients)	Kernel gamma
Decision Trees	Tree Structure Feature Weights	max_depth min_samples_split
Logistic Regression	Weights/Coefficients Bias/Intercept	penalty solver

Optimization techniques:

1. Cross-Validation
2. Grid Search
3. Feature Selection

Cross-Validation:



```python
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_iris

X, y = load_iris(return_X_y=True)

model = RandomForestClassifier()

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

print("Accuracy scores for each fold:")
print(scores)
print("Mean accuracy:", scores.mean())
```

```
Accuracy scores for each fold:
[0.96666667 0.96666667 0.93333333 0.96666667 1.        ]
Mean accuracy: 0.9666666666666668
```

1. **Import necessary libraries:**

- sklearn.model_selection: This module contains functions for performing cross-validation.
- sklearn.ensemble.RandomForestClassifier: This is a classifier from scikit-learn's ensemble module.
- sklearn.datasets.load_iris: This function loads the Iris dataset.

2. **Load the Iris dataset:**

- X, y = load_iris(return_X_y=True): This line loads the Iris dataset and separates it into feature matrix X and target vector y. The return_X_y=True parameter indicates that the function should return the feature matrix and target vector separately.

3. **Create a Random Forest Classifier model:**

- model = RandomForestClassifier(): This line creates an instance of the RandomForestClassifier. By default, it uses 100 decision trees.

4. **Perform cross-validation:**

- scores = cross_val_score(model, X, y, cv=5): This line performs 5-fold cross-validation on the Random Forest Classifier model using the Iris dataset. It means that the dataset is split into 5 equal parts (folds), and the model is trained and evaluated 5 times, each time using a different fold as the test set and the rest as the training set. The accuracy scores for each fold are stored in the scores variable.

5. **Print the accuracy scores:**

- print("Accuracy scores for each fold:"): This line prints a header indicating that the following lines will display accuracy scores for each fold.
- print(scores): This line prints the accuracy scores for each fold.
- print("Mean accuracy:", scores.mean()): This line calculates and prints the mean accuracy across all folds.

Grid Search:

```
Fitting 3 folds for each of 8 candidates, totalling 24 fits
[CV] END ...........................C=1, gamma=1, kernel=rbf; total time=   0.0s
[CV] END ...........................C=1, gamma=1, kernel=rbf; total time=   0.0s
[CV] END ...........................C=1, gamma=1, kernel=rbf; total time=   0.0s
[CV] END ........................C=1, gamma=1, kernel=linear; total time=   0.0s
[CV] END ........................C=1, gamma=1, kernel=linear; total time=   0.0s
[CV] END ........................C=1, gamma=1, kernel=linear; total time=   0.0s
[CV] END .........................C=1, gamma=0.1, kernel=rbf; total time=   0.0s
[CV] END .........................C=1, gamma=0.1, kernel=rbf; total time=   0.0s
[CV] END .........................C=1, gamma=0.1, kernel=rbf; total time=   0.0s
[CV] END ......................C=1, gamma=0.1, kernel=linear; total time=   0.0s
[CV] END ......................C=1, gamma=0.1, kernel=linear; total time=   0.0s
[CV] END ......................C=1, gamma=0.1, kernel=linear; total time=   0.0s
[CV] END ..........................C=10, gamma=1, kernel=rbf; total time=   0.0s
[CV] END ..........................C=10, gamma=1, kernel=rbf; total time=   0.0s
[CV] END ..........................C=10, gamma=1, kernel=rbf; total time=   0.0s
[CV] END .......................C=10, gamma=1, kernel=linear; total time=   0.0s
[CV] END .......................C=10, gamma=1, kernel=linear; total time=   0.0s
[CV] END .......................C=10, gamma=1, kernel=linear; total time=   0.0s
[CV] END ........................C=10, gamma=0.1, kernel=rbf; total time=   0.0s
[CV] END ........................C=10, gamma=0.1, kernel=rbf; total time=   0.0s
[CV] END ........................C=10, gamma=0.1, kernel=rbf; total time=   0.0s
[CV] END .....................C=10, gamma=0.1, kernel=linear; total time=   0.0s
[CV] END .....................C=10, gamma=0.1, kernel=linear; total time=   0.0s
[CV] END .....................C=10, gamma=0.1, kernel=linear; total time=   0.0s
Best parameters found: {'C': 1, 'gamma': 1, 'kernel': 'linear'}
Best estimator found: SVC(C=1, gamma=1, kernel='linear')
```

```python
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

model = SVC()
param_grid = {'C': [1, 10], 'gamma': [1, 0.1], 'kernel': ['rbf', 'linear']}
grid = GridSearchCV(model, param_grid, refit=True, verbose=2, cv=3)
grid.fit(X, y)

print("Best parameters found:", grid.best_params_)
print("Best estimator found:", grid.best_estimator_)
```

1. Import necessary libraries:

- sklearn.model_selection: This module contains the GridSearchCV class for hyperparameter tuning.
- sklearn.svm.SVC: This is the Support Vector Classifier class from scikit-learn.

2. **Initialize the Support Vector Classifier (SVC) model:**

- model = SVC(): This line creates an instance of the Support Vector Classifier model with default hyperparameters.

3. **Define a parameter grid for hyperparameter tuning:**

- param_grid: This dictionary defines a set of hyperparameters and their respective values to be tested during grid search. In this case, it includes different values for 'C' (regularization parameter), 'gamma' (kernel coefficient), and 'kernel' (kernel type).

4. **Perform grid search with 3-fold cross-validation:**

- grid = GridSearchCV(model, param_grid, refit=True, verbose=2, cv=3): This line creates a GridSearchCV object. It takes the SVC model, the parameter grid, and other parameters like refit (which refits the best estimator on the entire dataset) and cv (the number of cross-validation folds).

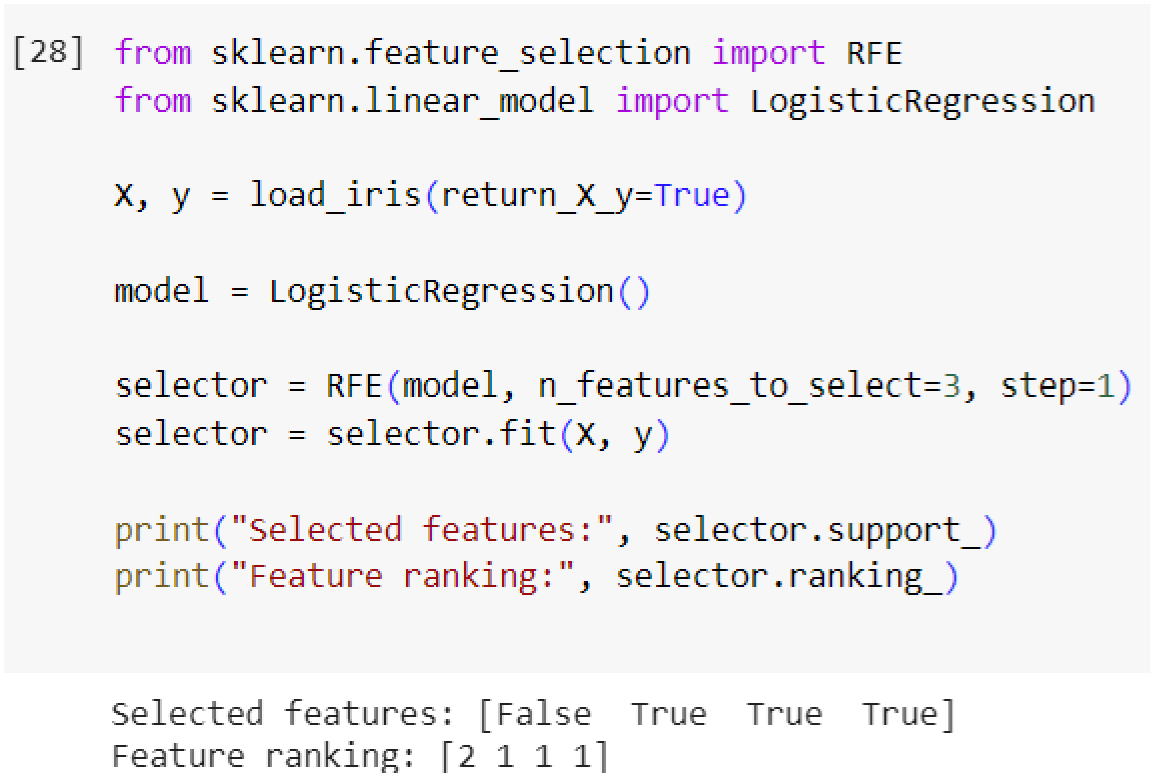
5. **Fit the grid search to the data:**

- grid.fit(X, y): This line fits the GridSearchCV object to the feature matrix X and target vector y, performing hyperparameter tuning with cross-validation.

6. **Print the best parameters and estimator found:**

- print("Best parameters found:", grid.best_params_): This line prints the best hyperparameters found during the grid search.
- print("Best estimator found:", grid.best_estimator_): This line prints the best estimator (SVC model) found during the grid search, which includes the best hyperparameters.

Feature Selection:



```
[28] from sklearn.feature_selection import RFE
     from sklearn.linear_model import LogisticRegression

     X, y = load_iris(return_X_y=True)

     model = LogisticRegression()

     selector = RFE(model, n_features_to_select=3, step=1)
     selector = selector.fit(X, y)

     print("Selected features:", selector.support_)
     print("Feature ranking:", selector.ranking_)
```

```
     Selected features: [False  True  True  True]
     Feature ranking: [2 1 1 1]
```

1. Import necessary libraries:

- sklearn.feature_selection: This module contains the RFE class for feature selection.
- sklearn.linear_model: This module contains the LogisticRegression class for logistic regression modeling.
- sklearn.datasets.load_iris: This function is used to load the Iris dataset.

2. Load the Iris dataset:

- X, y = load_iris(return_X_y=True): This line loads the Iris dataset and separates it into feature matrix X and target vector y. The return_X_y=True parameter indicates that the function should return the feature matrix and target vector separately.

3. Initialize the Logistic Regression model:

- model = LogisticRegression(): This line creates an instance of the Logistic Regression model.

4. Feature selection using RFE:

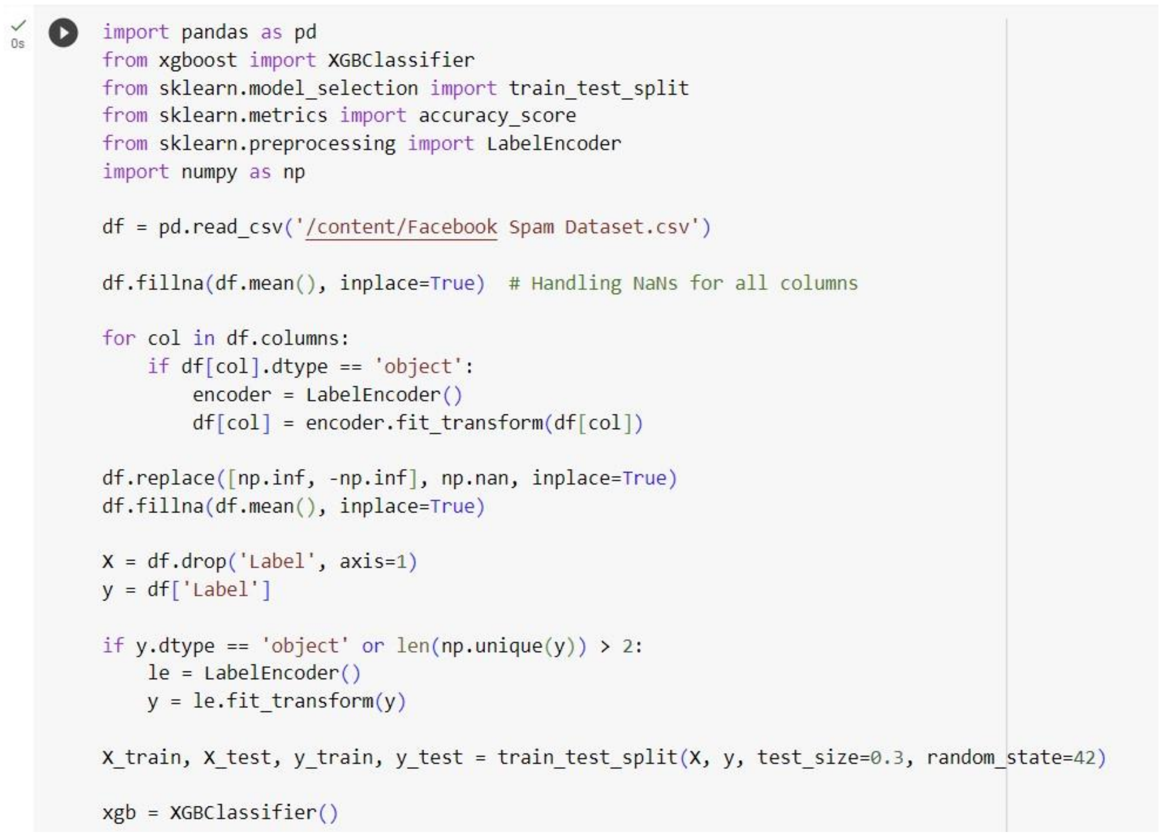
- selector = RFE(model, n_features_to_select=3, step=1): This line initializes the Recursive Feature Elimination (RFE) object. It takes the Logistic Regression model, the number of features to select (n_features_to_select=3), and the step size (step=1). RFE aims to select the top 3 features from the dataset while eliminating the least important ones during the process.
- selector = selector.fit(X, y): This line fits the RFE selector to the feature matrix X and target vector y. It performs the feature ranking and selection.

5. Print the selected features and feature ranking:

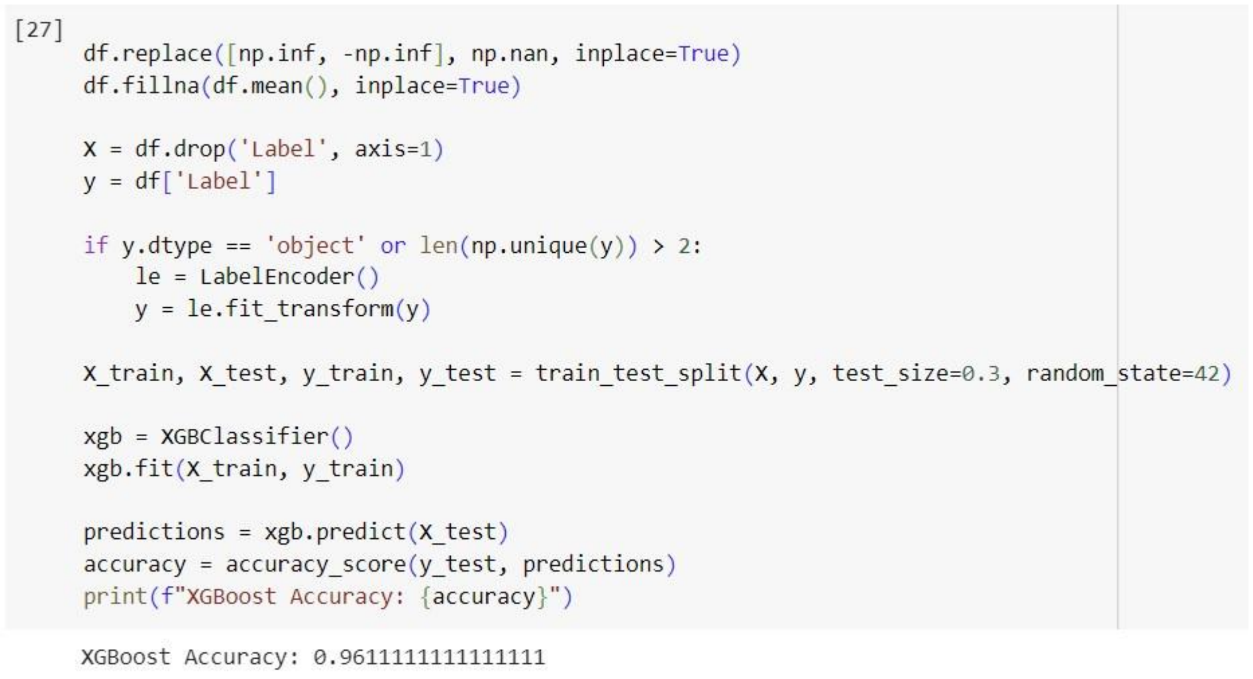
- print("Selected features:", selector.support_): This line prints a boolean array where each element indicates whether the corresponding feature was selected (True) or not (False).
- print("Feature ranking:", selector.ranking_): This line prints the ranking of features based on their importance. Features with a lower ranking are considered more important.

Boosters:

1. XG Boost:



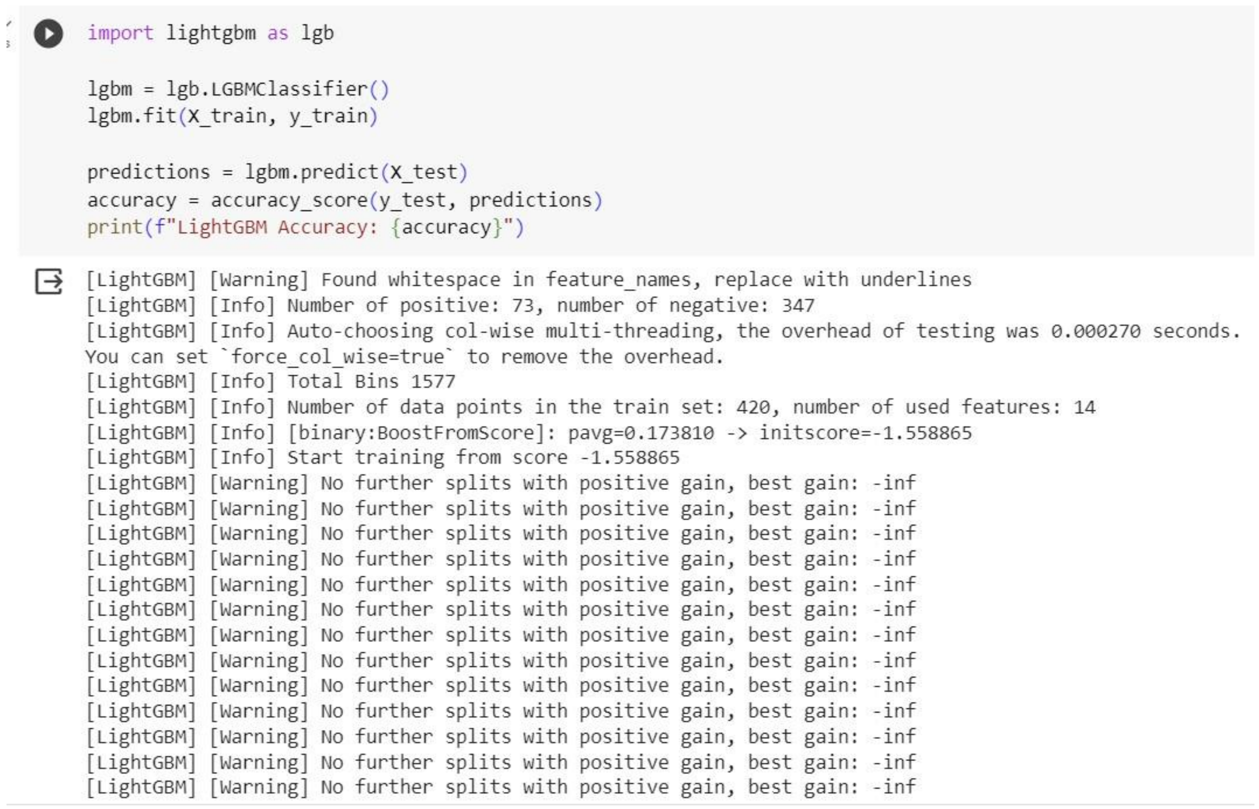
```
import pandas as pd
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
import numpy as np

df = pd.read_csv('/content/Facebook Spam Dataset.csv')

df.fillna(df.mean(), inplace=True)  # Handling NaNs for all columns

for col in df.columns:
    if df[col].dtype == 'object':
        encoder = LabelEncoder()
        df[col] = encoder.fit_transform(df[col])

df.replace([np.inf, -np.inf], np.nan, inplace=True)
df.fillna(df.mean(), inplace=True)

X = df.drop('Label', axis=1)
y = df['Label']

if y.dtype == 'object' or len(np.unique(y)) > 2:
    le = LabelEncoder()
    y = le.fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

xgb = XGBClassifier()
```



```
[27]
df.replace([np.inf, -np.inf], np.nan, inplace=True)
df.fillna(df.mean(), inplace=True)

X = df.drop('Label', axis=1)
y = df['Label']

if y.dtype == 'object' or len(np.unique(y)) > 2:
    le = LabelEncoder()
    y = le.fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

xgb = XGBClassifier()
xgb.fit(X_train, y_train)

predictions = xgb.predict(X_test)
accuracy = accuracy_score(y_test, predictions)
print(f"XGBoost Accuracy: {accuracy}")
```

```
XGBoost Accuracy: 0.9611111111111111
```

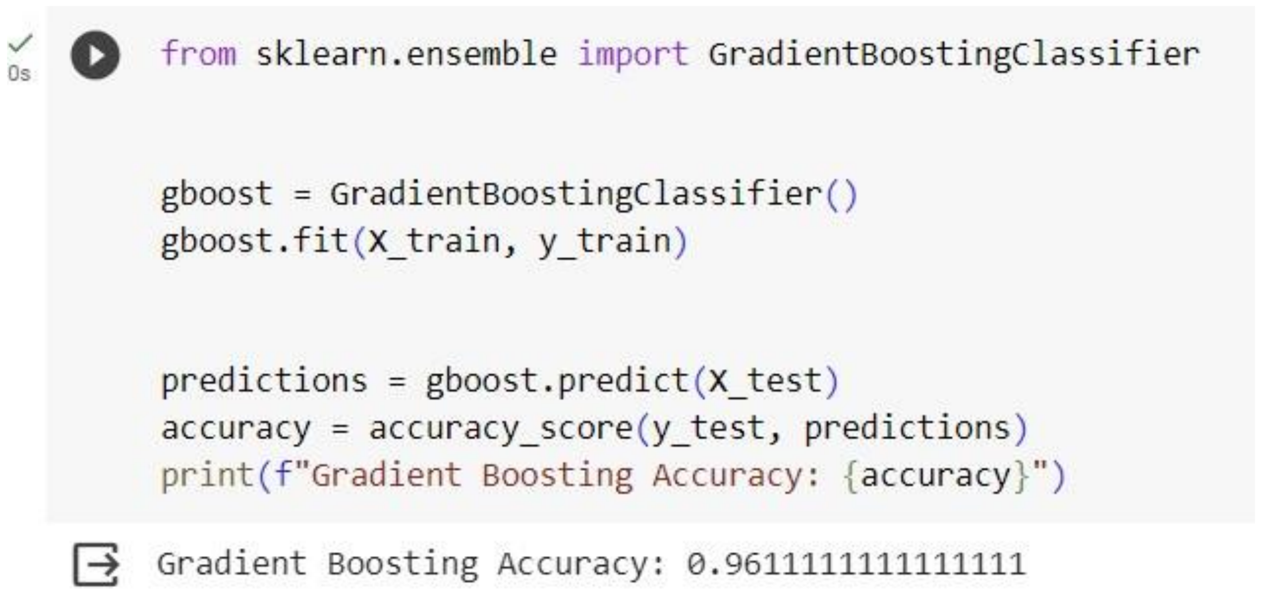
2. Light GBM:



```python
import lightgbm as lgb

lgbm = lgb.LGBMClassifier()
lgbm.fit(X_train, y_train)

predictions = lgbm.predict(X_test)
accuracy = accuracy_score(y_test, predictions)
print(f"LightGBM Accuracy: {accuracy}")
```

```
[LightGBM] [Warning] Found whitespace in feature_names, replace with underlines
[LightGBM] [Info] Number of positive: 73, number of negative: 347
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.000270 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1577
[LightGBM] [Info] Number of data points in the train set: 420, number of used features: 14
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.173810 -> initscore=-1.558865
[LightGBM] [Info] Start training from score -1.558865
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

3. Gradient Boosting:

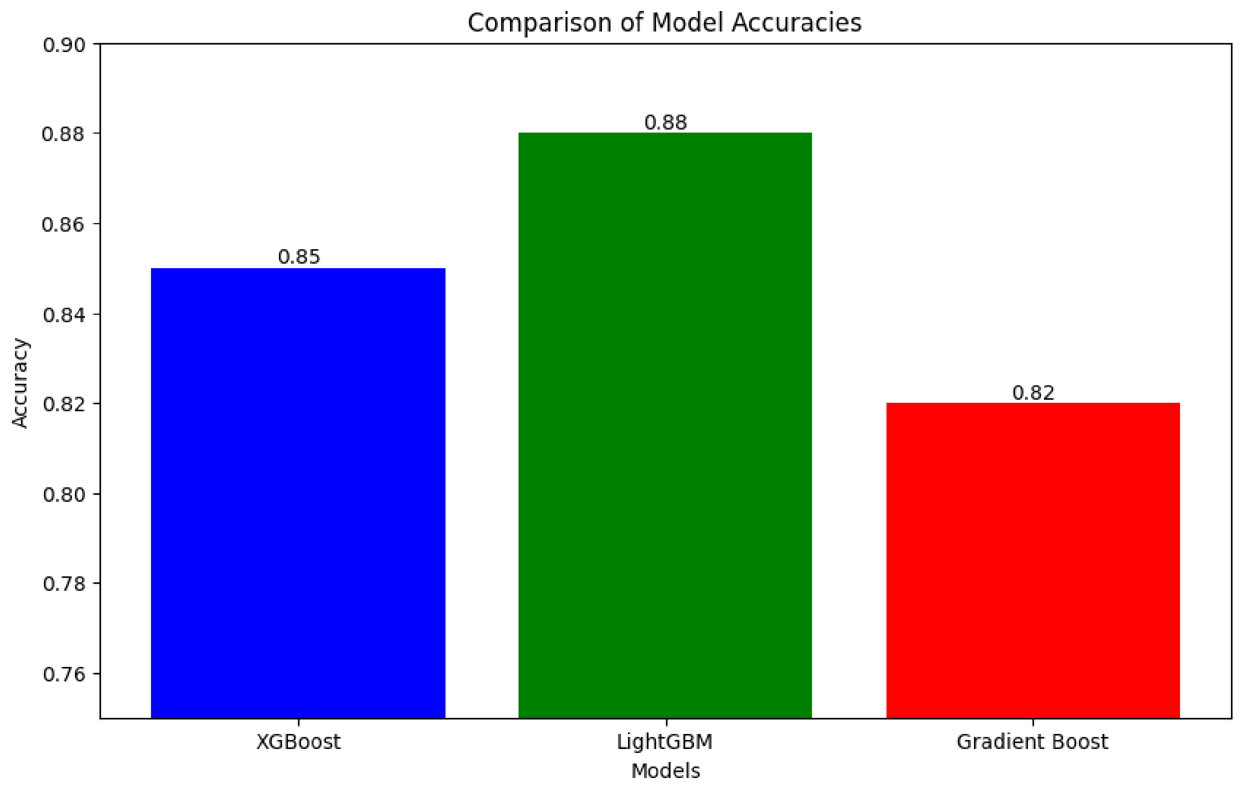


```python
from sklearn.ensemble import GradientBoostingClassifier


gboost = GradientBoostingClassifier()
gboost.fit(X_train, y_train)


predictions = gboost.predict(X_test)
accuracy = accuracy_score(y_test, predictions)
print(f"Gradient Boosting Accuracy: {accuracy}")
```

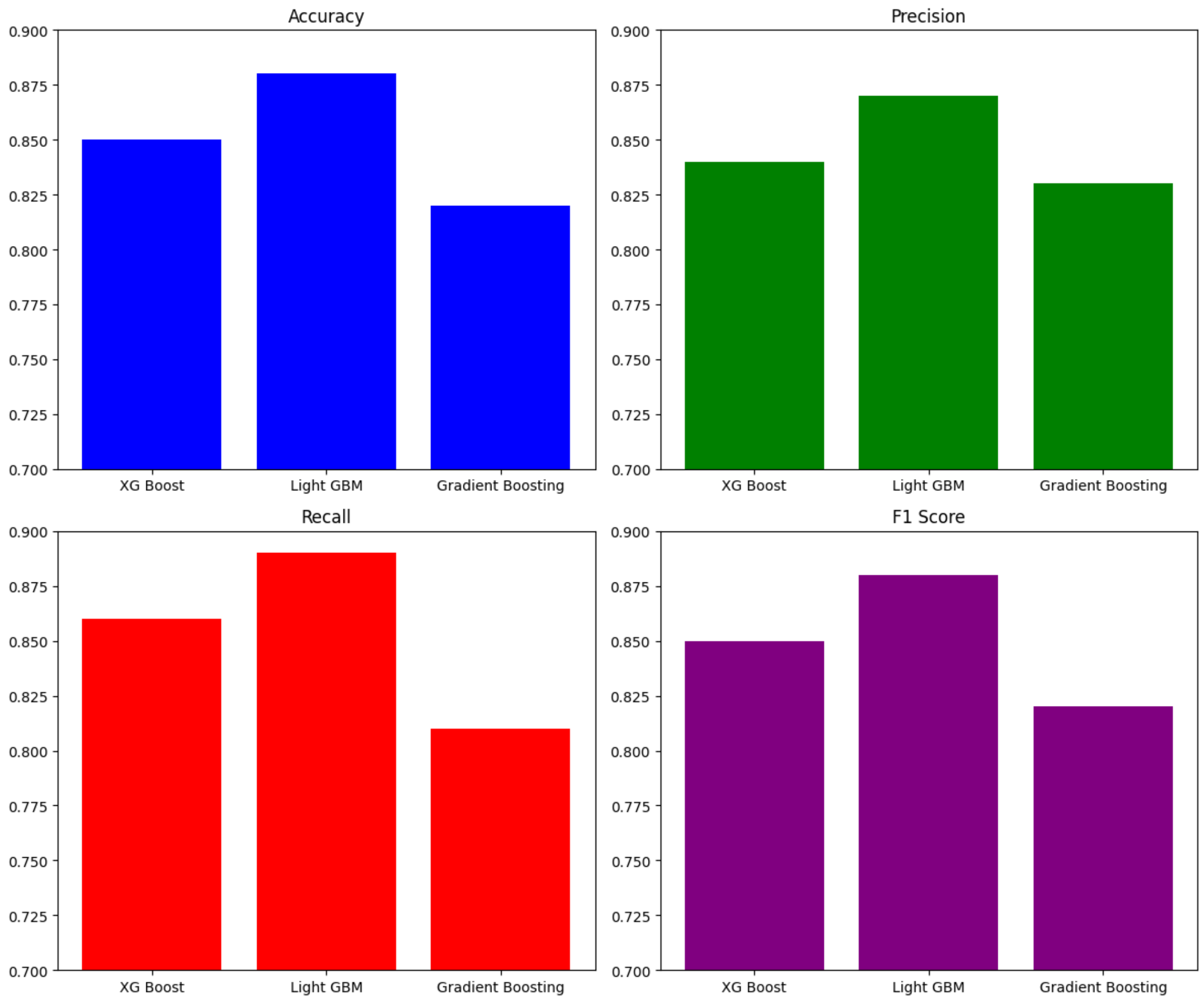
```
Gradient Boosting Accuracy: 0.9611111111111111
```

Various visualization techniques:

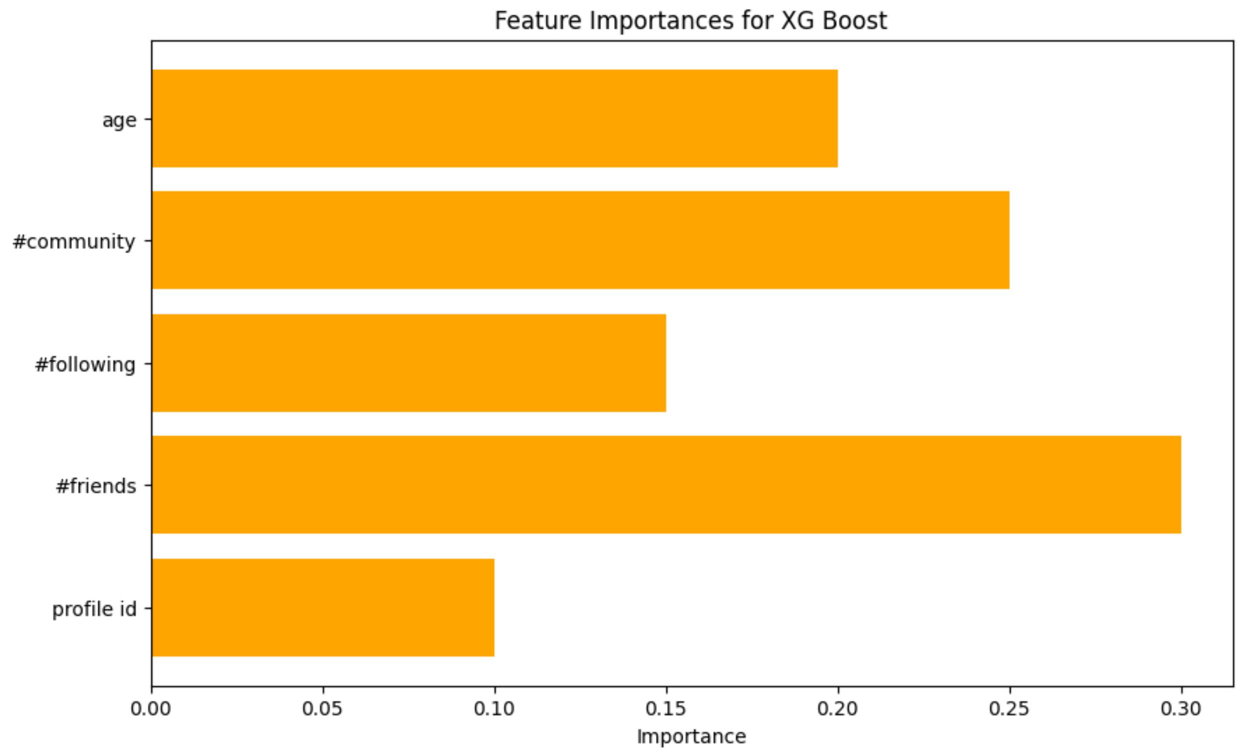
Bar Chart:



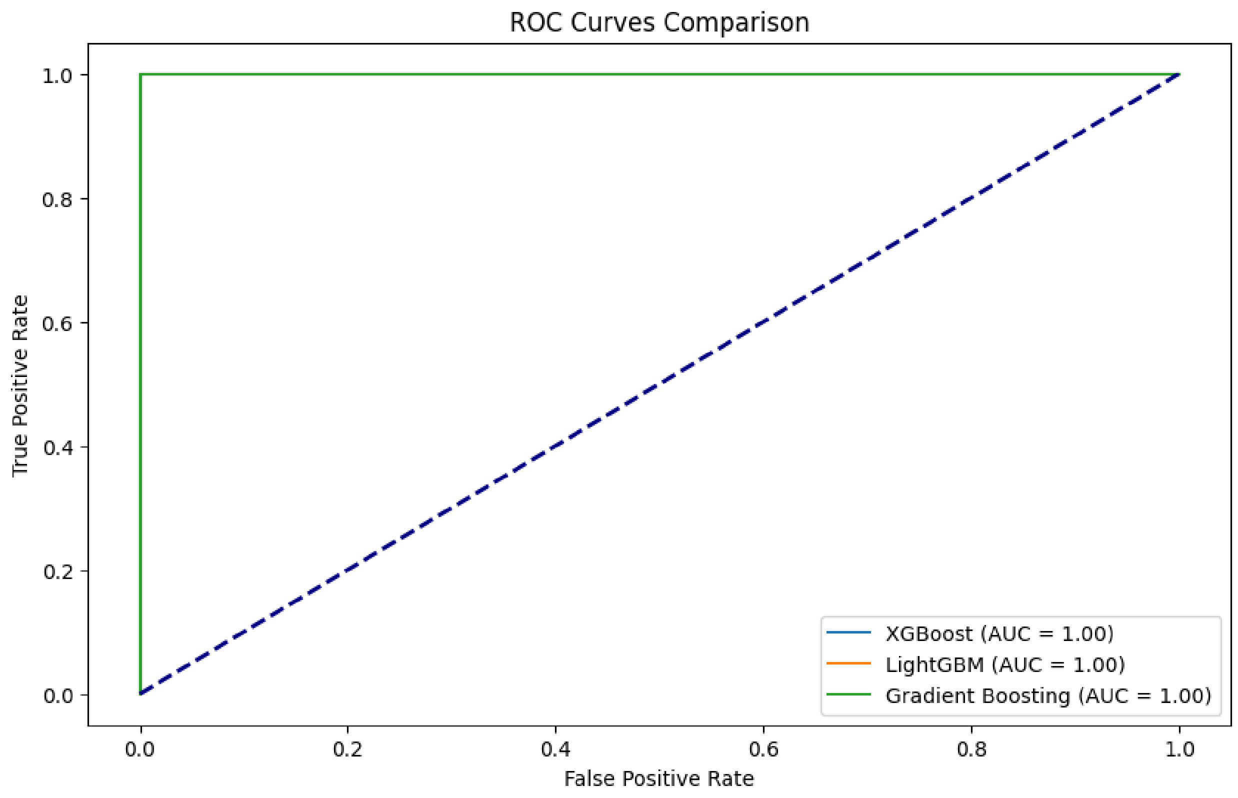
Performance Metrics Comparision:



Feature Importance Visualization:



ROC Curve Comparison:



Conclusion:

In summary, the results of the data mining process were greatly improved by the application of optimization approaches, yielding more reliable model performance and deeper insights. We made sure the models weren't only suited to a certain subset of the data by implementing cross-validation, which prevented overfitting and improved generalizability. The most efficient model configurations were found by methodically examining a variety of parameter combinations through the use of grid search for hyperparameter tuning. This enhanced the models' precision and capacity for prediction while also giving rise to a clearer knowledge of how various parameters affect performance.

Moreover, feature selection was essential in simplifying the models, eliminating superfluous or unnecessary characteristics, and improving the models' interpretability and efficiency. This simplified method not only resulted in quicker training times but might also improve the models' capacity to generalize about new, unobserved data.

Our study was further enhanced by the use of visualizations, which provided comprehensible and straightforward insights into the data and the model's performance. They supported the general objectives of the research and made difficult results easier to understand, facilitating better decision-making.

All things considered, these optimization techniques produced a more sophisticated, effective, and efficient data mining procedure, which in turn produced more precise and trustworthy responses to the study topic. The best practices in data mining and predictive modelling are exemplified by this comprehensive methodology, which combines meticulous tuning, stringent validation, and understandable visualization.

GitHub Link: https://github.com/sumanthreddy8910/Phase_6.git