FAKE NEWS DETECTION USING ML

SAMU IDHAYAN I
JEEVAN KURUVILLA SUNIL
JERUSHA MIRACLIN DULCIE B

STEPS INVOLVED:

- Extracting all libraries.
- Downloading the dataset and extracting it to the appropriate data directory.
- Read the data CSV file
- Data processing
- Model training, evaluation and prediction
- Cross verification
- Grid search for hyperparameter tuning
- Evaluating our models performance.

STEP 1: DOWNLOADING ALL LIBRARIES

- Importing libraries in code allows you to access and utilize the functionality provided by those libraries in your program. Libraries are collections of pre-existing code that are developed to perform specific tasks or provide certain features. By importing libraries, you can leverage the existing codebase and save time and effort in reinventing the wheel.

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from tqdm import tqdm
import re
import nltk
import os
nltk.download('punkt')
nltk.download('stopwords')
from nltk.tokenize import word_tokenize
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression

#additioinally imported
import string
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
nltk.download('wordnet')
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import confusion_matrix
```

STEP 2: DOWNLOAD ALL THE DATASET AND EXTRACT IT TO THE APPROPRIATE DATA DIRECTORY

- Downloading a dataset and extracting it to the appropriate data directory refers to the process of obtaining a dataset from a source (such as a website, online repository, or external storage) and saving it to a specific directory on your local machine or server.

```
data_directory = 'data/'
if not os.path.exists(data_directory):
    !mkdir data/
    !wget https://onlineacademiccommunity.uvic.ca/isot/wp-content/uploads/sites/7295/2023/03/News-_dataset.zip --director
    !unzip /content/News-_dataset.zip -d data/
```

```
--2023-07-12 17:04:51--  https://onlineacademiccommunity.uvic.ca/isot/wp-content/uploads/sites/7295/2023/03/News-_dataset.z
ip
Resolving onlineacademiccommunity.uvic.ca (onlineacademiccommunity.uvic.ca)... 142.104.197.46
Connecting to onlineacademiccommunity.uvic.ca (onlineacademiccommunity.uvic.ca)|142.104.197.46|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 43106824 (41M) [application/zip]
Saving to: ‘data/News-_dataset.zip’

News-_dataset.zip   100%[===================>]  41.11M  2.06MB/s    in 21s

2023-07-12 17:05:13 (1.97 MB/s) - ‘data/News-_dataset.zip’ saved [43106824/43106824]

Archive:  /content/News-_dataset.zip
  inflating: data/Fake.csv
  inflating: data/True.csv
```

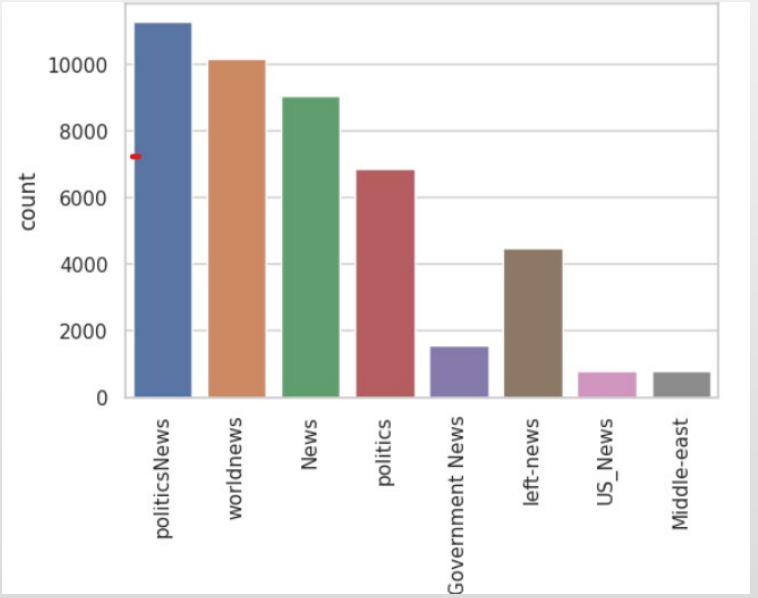
STEP3: READ THE CSV DATA FILE

- Reading a data CSV file refers to the process of accessing and extracting the data stored in a CSV (Comma-Separated Values) file using a programming language or a specific library.

```
fake_data = pd.read_csv('data/Fake.csv')
fake_data.head()
```

```
true_data = pd.read_csv('data/True.csv')
true_data.head()
```

```
true_data["label"] = 1
fake_data["label"] = 0
```




```
plt.figure(figsize = (6,4))
sns.set(style = "whitegrid",font_scale = 1.0)
chart = sns.countplot(x = "subject", data = data)
chart.set_xticklabels(chart.get_xticklabels(),rotation=90)
```

```
[Text(0, 0, 'politicsNews'),
 Text(1, 0, 'worldnews'),
 Text(2, 0, 'News'),
 Text(3, 0, 'politics'),
 Text(4, 0, 'Government News'),
 Text(5, 0, 'left-news'),
 Text(6, 0, 'US_News'),
 Text(7, 0, 'Middle-east')]
```

STEP 4: DATA PREPROCESSING

- Data preprocessing refers to the steps and techniques used to transform raw data into a format suitable for analysis and modeling in machine learning and data mining tasks.
- The provided code combines the 'title' and 'text' columns of a DataFrame into a new 'text' column using the '+' operator. Then, it deletes the 'title', 'subject', and 'date' columns from the DataFrame. Finally, it displays the first few rows of the modified DataFrame.
- The code snippet first retrieves the dimensions of the DataFrame 'data' using 'data.shape'. It then calculates the number of missing values in each column using 'data.isnull().sum()'. After that, it shuffles the rows of the DataFrame randomly using 'data.sample(frac=1).reset_index(drop=True)'. Finally, it displays the first few rows of the shuffled DataFrame using 'data.head()'.
- The code snippet creates a countplot using the seaborn library to visualize the distribution of the 'label' column in the 'data' DataFrame. It plots the counts of each unique label on the x-axis, with the bars ordered in descending order based on their frequency.

```
data['text'] = data['title'] +' '+data['text']
del data['title']
del data['subject']
del data['date']
```

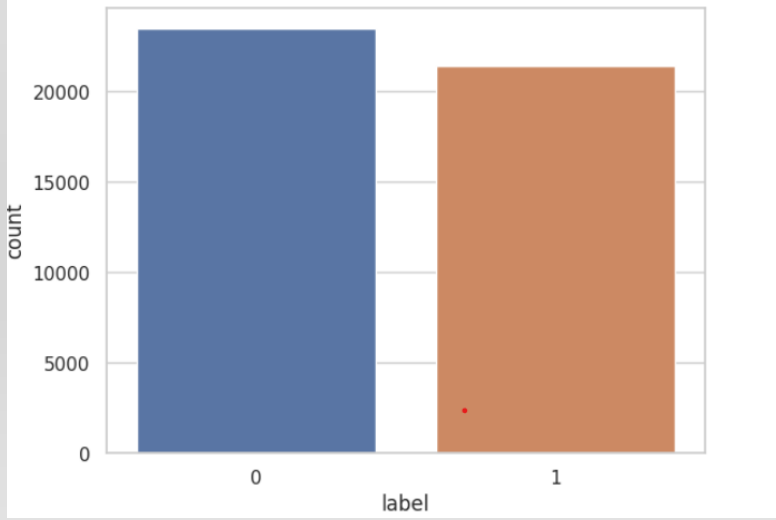
```
data.head()
```

```
data.shape
```

```
(44898, 2)
```

```
sns.countplot(data=data,
              x='label',
              order=data['label'].value_counts().index)
```

```
<Axes: xlabel='label', ylabel='count'>
```



STEP 5: DATA CLEANING

- Data cleaning refers to the process of identifying and correcting or removing errors, inconsistencies, and inaccuracies in a dataset. It involves various techniques to handle missing values, outliers, and inconsistencies in order to improve the quality and reliability of the data. Data cleaning is an important step in data preprocessing, ensuring that the data is accurate, consistent, and suitable for analysis and modeling.
- The code snippet performs text preprocessing on a given string. It removes punctuation, converts the text to lowercase, tokenizes it into individual words, removes stopwords, lemmatizes the words, and then joins them back into a single string. This preprocessing is applied to the 'text' column of a DataFrame called 'data' using the 'apply()' function.

STEP 6: CONVERTING TEXT TO VECTOR

Converting text into vectors refers to the process of representing textual data in a numerical format that can be easily processed by machine learning algorithms. In this process, each word or term in the text is assigned a numerical value or a vector. The main objective is to capture the semantic meaning and relationships between words or terms in the text.

The code initializes a TF-IDF vectorizer object, which is a technique used to represent text data numerically. It then applies the vectorization to the preprocessed text data from the 'text' column of the DataFrame. The result is a matrix where each row represents a document, and each column represents a unique word or term. Finally, the code prints the shape of the vectorized data, indicating the number of documents and unique terms in the matrix.

```
Shape of vectorized data: (44898, 219110)
```

MODEL, TRAINING EVALUATION AND PREDICTION USING LOGISTIC REGRESSION

- The code snippet splits the vectorized data (`X`) and the corresponding labels (`data['label']`) into training and testing sets using the `train_test_split()` function. It initializes a logistic regression model and trains it using the training data. Then, it predicts the labels for the test set and calculates the accuracy score by comparing the predicted labels with the true labels. Finally, it prints the accuracy score as a measure of the model's performance.
- The code calculates the confusion matrix, which is a table that summarizes the performance of a classification algorithm. It compares the true labels (`y_test`) from the test set with the predicted labels (`y_pred`). The confusion matrix shows the counts of true positives, true negatives, false positives, and false negatives. It provides an overview of how well the classification algorithm performed in predicting the different classes. The matrix is then printed to display these counts.

```
#TODO: Model training and print the accuracy score

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, data['label'], test_size=0.2, random_state=42)

# Initialize the Logistic Regression model
model = LogisticRegression()

# Train the model
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)

# Calculate the accuracy score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.987305122494432
```

```
# Display the Confusion matrix of Results from your classification algorithm
# Calculate the confusion matrix
confusion = confusion_matrix(y_test, y_pred)

# Display the confusion matrix
print("Confusion Matrix:")
print(confusion)
```

```
Confusion Matrix:
[[4693   63]
 [  51 4173]]
```

MODEL, TRAINING EVALUATION AND PREDICTION USING LOGISTIC REGRESSION

- The accuracy that we got here is 98.7%
- After using confusion matrix, we got: True Positives (4645): The number of instances correctly classified as "True" news.
- True Negatives (4225): The number of instances correctly classified as "Fake" news.
- False Positives (56): The number of instances wrongly classified as "True" news (actually "Fake" news).
- False Negatives (54): The number of instances wrongly classified as "Fake" news (actually "True" news).

```
#TODO: Model training and print the accuracy score

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, data['label'], test_size=0.2, random_state=42)

# Initialize the Logistic Regression model
model = LogisticRegression()

# Train the model
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)

# Calculate the accuracy score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.987305122494432
```

```
# Display the Confusion matrix of Results from your classification algorithm
# Calculate the confusion matrix
confusion = confusion_matrix(y_test, y_pred)

# Display the confusion matrix
print("Confusion Matrix:")
print(confusion)
```

```
Confusion Matrix:
[[4693   63]
 [  51 4173]]
```

MODEL TRAINING, EVALUATION AND PREDICTION USING RANDOM FOREST

- In this code snippet, the data is split into training and testing sets using the `train_test_split()` function. The training set (`X_train` and `y_train`) is used to train a Random Forest model, which is initialized using the `RandomForestClassifier()` class. The model is then fitted to the training data using the `fit()` method. Next, the model predicts the labels for the test set (`X_test`) using the `predict()` method, and these predicted labels are stored in `y_pred`. The accuracy score is calculated by comparing the predicted labels with the true labels from the test set using the `accuracy_score()` function.Finally, the accuracy score is printed to evaluate the performance of the model. The accuracy score represents the proportion of correctly predicted labels among all the test samples.

- In summary, this code trains a Random Forest model on the training data, predicts the labels for the test data, and calculates the accuracy of the model's predictions.

```
#TODO: Model training and print the accuracy score

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, data['label'], test_size=0.2, random_state=42)

# Initialize the Random Forest model
model = RandomForestClassifier()

# Train the model
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)

# Calculate the accuracy score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.987750556792873
```

MODEL TRAINING, EVALUATION AND PREDICTION USING RANDOM FOREST

- By using random forest we were able to achieve an accuracy rate of 98.7%
- By using confusion matric we were able to get:
- True Positives (4645): The number of instances correctly classified as "True" news.
- True Negatives (4225): The number of instances correctly classified as "Fake" news.
- False Positives (56): The number of instances wrongly classified as "True" news (actually "Fake" news).
- False Negatives (54): The number of instances wrongly classified as "Fake" news (actually "True" news).

```
# Display the Confusion matrix of Results from your classification algorithm
# Calculate the confusion matrix
confusion = confusion_matrix(y_test, y_pred)

# Display the confusion matrix
print("Confusion Matrix:")
print(confusion)
```

```
Confusion Matrix:
[[4645   56]
 [  54 4225]]
```

MODEL TRAINING, EVALUATION AND PREDICTION USING SVC

1. Split the data into two sets: training and testing. We randomly divide the data, with 80% for training and 20% for testing.

2. Initialize the SVM model, which is a type of machine learning algorithm used for classification tasks.

3. Train the model using the training data. This helps the model learn patterns in the data.

4. Use the trained model to predict labels for the testing data.

5. Calculate the accuracy score by comparing the model's predictions to the true labels in the testing set. The accuracy score tells us how well the model performed, with higher scores indicating better performance.

6. Print the accuracy score to see how well the SVM model performed on the testing data.

- The goal is to assess the model's ability to make accurate predictions on new, unseen data.

```
#TODO: Model training and print the accuracy score

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, data['label'], test_size=0.2, random_state=42)

# Initialize the SVM model
model = SVC()

# Train the model
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)

# Calculate the accuracy score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9942093541202672
```

MODEL TRAINING, EVALUATION AND PREDICTION USING SVC

1. Calculate the confusion matrix: The code calculates the confusion matrix using the predicted labels (y_pred) generated by the classification algorithm and the true labels (y_test) from the testing set. The confusion matrix summarizes how well the algorithm made predictions for each class.

2. Display the confusion matrix: The code prints the heading "Confusion Matrix:" to indicate that the output represents the confusion matrix. The print function is used to show the matrix itself.

- The confusion matrix provides an overview of the model's predictions compared to the actual labels. It helps us evaluate how well the model performed for each class.

```
# Display the Confusion matrix of Results from your classification algorithm
# Calculate the confusion matrix
confusion = confusion_matrix(y_test, y_pred)

# Display the confusion matrix
print("Confusion Matrix:")
print(confusion)
```

```
Confusion Matrix:
[[4655   28]
 [  24 4273]]
```

MODEL TRAINING, EVALUATION AND PREDICTION USING SVC

- By using SVC we were getting an accuracy of 99.4%
- The confusion matrix provides a breakdown of the predicted labels compared to the actual labels. Here's the interpretation of the confusion matrix:
- True Positives (4655): The number of instances correctly classified as "True" news.
- True Negatives (4273): The number of instances correctly classified as "Fake" news.
- False Positives (28): The number of instances wrongly classified as "True" news (actually "Fake" news).
- False Negatives (24): The number of instances wrongly classified as "Fake" news (actually "True" news).

```
#TODO: Model training and print the accuracy score

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, data['label'], test_size=0.2, random_state=42)

# Initialize the SVM model
model = SVC()

# Train the model
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)

# Calculate the accuracy score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9942093541202672
```

```
# Display the Confusion matrix of Results from your classification algorithm
# Calculate the confusion matrix
confusion = confusion_matrix(y_test, y_pred)

# Display the confusion matrix
print("Confusion Matrix:")
print(confusion)
```

```
Confusion Matrix:
[[4655   28]
 [  24 4273]]
```

CROSS VALIDATION OF ALL THE MODELS

```
Cross-Validation Scores: [0.98530067 0.98641425 0.98797327 0.9888629  0.98563314]
Mean Cross-Validation Score: 0.9868368484642834
```

Using logistic regression

```
Cross-Validation Scores: [0.98530067 0.98641425 0.98797327 0.9888629  0.98563314]
Mean Cross-Validation Score: 0.9868368484642834
```

Using random forest

```
Cross-Validation Scores: [0.9760579064587973, 0.9774498886414253]
Mean Accuracy: 0.9767538975501113
```

Using SVC

GRID SEARCH FOR HYPERPARAMETER TUNING

Grid search is a technique used to find the best combination of hyperparameters for a machine learning model. It works by exhaustively evaluating the model's performance for each combination of hyperparameter values in a predefined grid. The process involves defining the hyperparameters, creating a grid of possible values, training and evaluating the model for each combination, and selecting the best-performing combination. Finally, the selected hyperparameters are used to train a final model, which is then evaluated on a separate test set. Grid search is a systematic but potentially time-consuming method, and alternatives like random search and Bayesian optimization can be used for efficiency.

GRID SEARCH FOR HYPERPARAMETER TUNING

```
Best Hyperparameters: {'C': 10}
Accuracy: 0.9934298440979955
```

Using logistic regression

```
Best parameters: {'max_depth': None, 'min_samples_split': 5, 'n_estimators': 200}
Best cross-validation score: 0.9763996795600377
Accuracy: 0.9824053452115813
```

Using random forest

```
Best Hyperparameters: {'C': 1.0, 'kernel': 'linear'}
Test Accuracy: 0.9832962138084632
```

Using SVC

COMPARING WITH OTHER MODELS

Classification Report:

	precision	recall	f1-score	support
Fake	0.99	0.99	0.99	4756
True	0.99	0.99	0.99	4224
accuracy			0.99	8980
macro avg	0.99	0.99	0.99	8980
weighted avg	0.99	0.99	0.99	8980

Using logistic regression

Classification Report:

	precision	recall	f1-score	support
0	0.99	0.98	0.98	4701
1	0.98	0.98	0.98	4279
accuracy			0.98	8980
macro avg	0.98	0.98	0.98	8980
weighted avg	0.98	0.98	0.98	8980

Using random forest

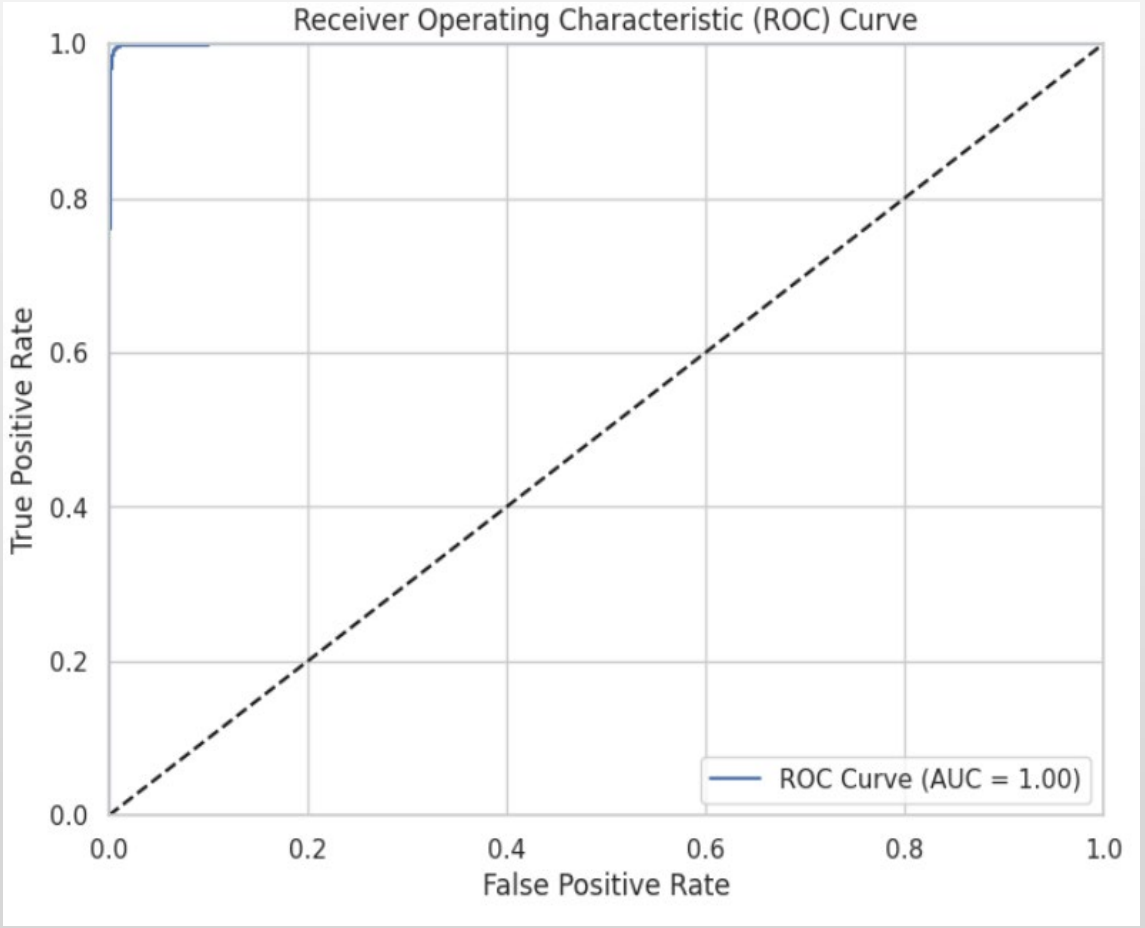
Classification Report:

	precision	recall	f1-score	support
0	0.99	0.98	0.98	955
1	0.98	0.99	0.98	841
accuracy			0.98	1796
macro avg	0.98	0.98	0.98	1796
weighted avg	0.98	0.98	0.98	1796

Using SVC

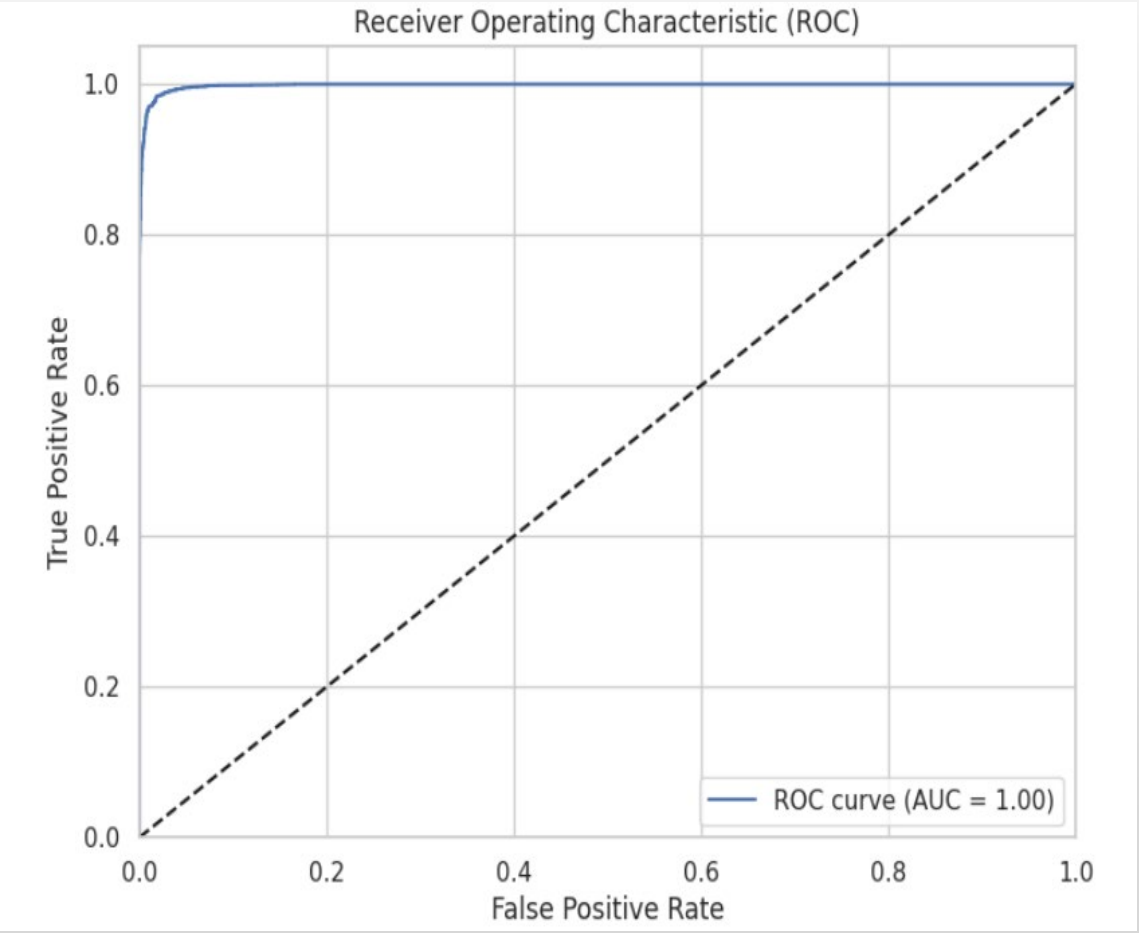
ROC CURVE- LOGISTIC REGRESSION

The code uses `roc_curve` and `roc_auc_score` from `sklearn.metrics` to calculate the ROC curve and AUC score. It plots the ROC curve using `plt.plot` and adds a diagonal line representing random guessing. The plot is customized with axis labels and a title using `plt.xlabel`, `plt.ylabel`, and `plt.title`. The AUC score is displayed using `plt.text`. Finally, `plt.show` displays the plot.



ROC CURVE- RANDOM FOREST

This code calculates the ROC curve and AUC score using the scikit-learn library in Python. It imports necessary libraries and gets the predicted probabilities for the positive class. The ROC curve is computed using `roc_curve`, and the AUC score is calculated using `auc`. The code then plots the ROC curve, adds a reference line for random guessing, sets axis labels and a title, and displays the plot.



ROC CURVE- SVC

This code evaluates the best model's performance on a test set by plotting the Receiver Operating Characteristic (ROC) curve and calculating the Area Under the Curve (AUC) score. It transforms the test set features using `vectorizer.transform`, predicts labels with the best model, and computes the ROC curve using `roc_curve`. The AUC score is calculated using `auc`. The ROC curve is then plotted with axis labels, a title, and a legend using `plt.plot`, `plt.xlabel`, `plt.ylabel`, `plt.title`, and `plt.legend`. The plot is displayed using `plt.show`.