CELEBAL TECHNOLOGIES

Galgotias University

Task 1: Email Spam Classification

Code by Ansh Shankar

Question:

Classification Problem-Solving (Use all Classification algorithm and find best out of all)

Solution:

1. Introduction:

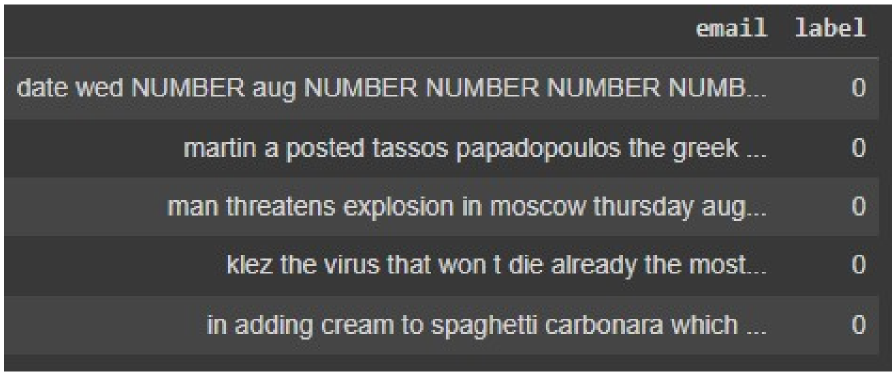
The purpose of this assignment is to perform a classification task on an email dataset to identify whether an email is spam or not spam. The objective is to explore and compare the performance of different machine learning algorithms, including k-Nearest Neighbors (k-NN), Random Forest, Decision Tree, and Support Vector Machine (SVM), for this classification task. The assignment aims to evaluate the effectiveness of these algorithms in distinguishing between spam and legitimate emails.

2. Dataset:

The dataset used in this assignment is named "spam_or_not_spam.csv." The dataset is taken from Kaggle. It consists of emails, with each email represented as a single row in the dataset. The dataset contains two main columns:

1. Email: This column represents the email content. It contains the textual data of the emails, which will be preprocessed and used as input to the machine learning models.

2. Label (Target Variable): This column indicates whether the email is spam (1) or not spam (0). It serves as the target variable for the classification task.



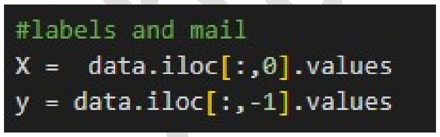
	email	label
	date wed NUMBER aug NUMBER NUMBER NUMBER NUMB...	0
	martin a posted tassos papadopoulos the greek ...	0
	man threatens explosion in moscow thursday aug...	0
	klez the virus that won t die already the most...	0
	in adding cream to spaghetti carbonara which ...	0

Figure 1: Dataset

3. Methodology:

a. Preprocessing Steps:

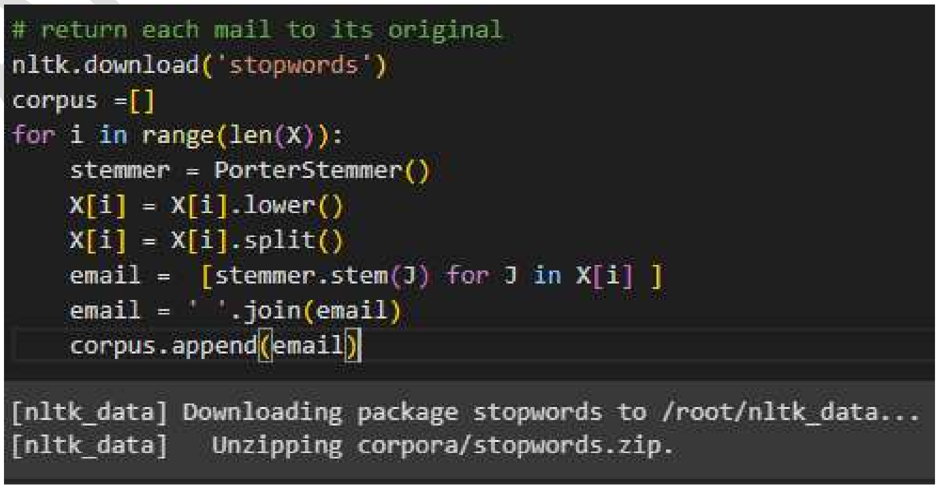
- Now we split the dataset into X and Y columns, where X contains email feature and Y contains the labels.



```
#labels and mail
X =  data.iloc[:,0].values
y = data.iloc[:,-1].values
```

Figure 2: Spliting the Dataset

- Next we download the English stopwords from nltk library. It a collection of common words that are often excluded from text analysis due to their limited semantic value.
- Then the email text is preprocessed to lowercase, tokenized, and then stemmed using the Porter stemming algorithm to reduce words to their base form.

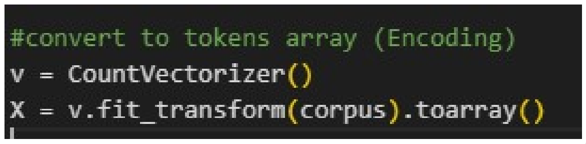


```
# return each mail to its original
nltk.download('stopwords')
corpus =[]
for i in range(len(X)):
    stemmer = PorterStemmer()
    X[i] = X[i].lower()
    X[i] = X[i].split()
    email =  [stemmer.stem(J) for J in X[i] ]
    email = ' '.join(email)
    corpus.append(email)

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

Figure 3: Preprocessing the Dataset

- The preprocessed emails are then combined to form a corpus of processed emails.
- In next step, we use CountVectorizer class from scikit-learn to convert a list of processed email texts into a numerical feature matrix. The fit_transform method learns the vocabulary and transforms the texts into a matrix where each row represents an email, and each column indicates the count of a specific word. The resulting numerical matrix is useful for machine learning tasks and text analysis.



```
#convert to tokens array (Encoding)
v = CountVectorizer()
X = v.fit_transform(corpus).toarray()
```

Figure 4: Vectorizing the email feature

b. Data Splitting:

- The dataset is split into training and testing sets using an 80%-20% split. The training set is used to train the models, while the testing set is used to evaluate their performance. We used random state 42.

c. **Implementation of Each Algorithm**:

i. **k-Nearest Neighbors (k-NN):** The code performs k-fold cross-validation to find the best value of k. It then trains a k-NN classifier with the optimal k value on the training data and evaluates its accuracy on the test set. The step is as follows:

- In this process, we assess the k-Nearest Neighbors (k-NN) classification algorithm and evaluate its performance using cross-validation. We create an array of k values (1 to 20) to represent the number of neighbors. Cross-validation accuracy scores for each k value are computed using a loop with 10-fold cross-validation. The results are then visualized using matplotlib with a line plot, showing how the model's performance varies with different k values and helping identify the optimal k value for the k-NN classifier.

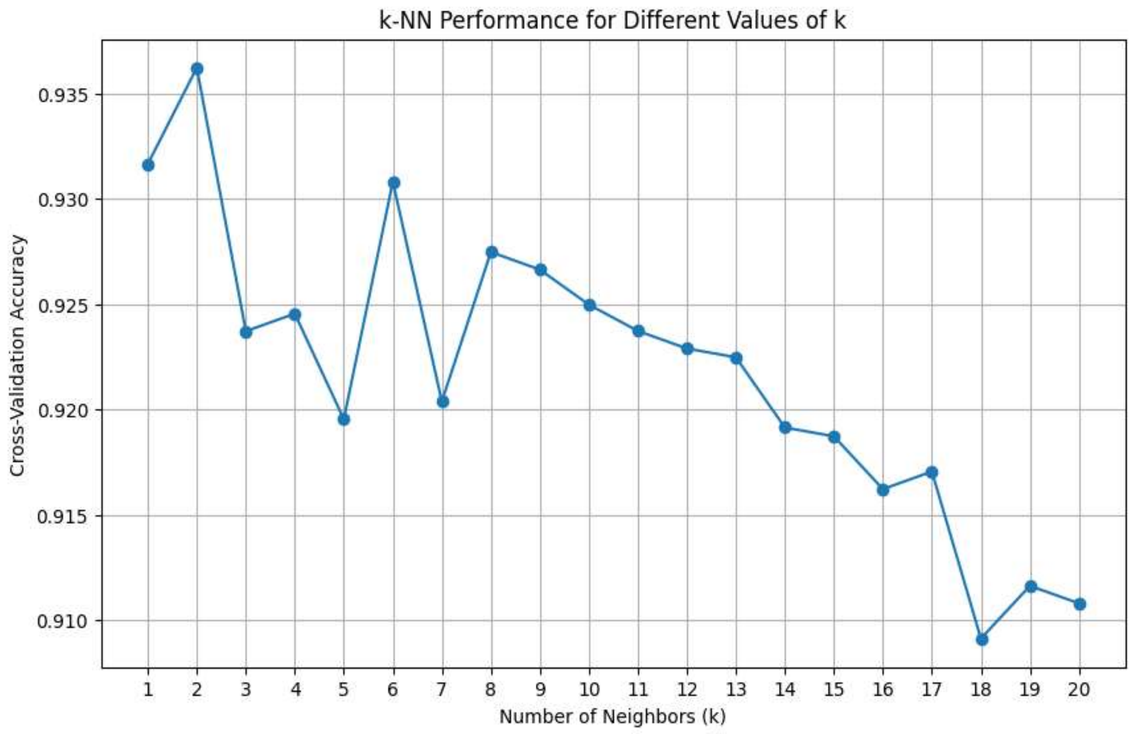
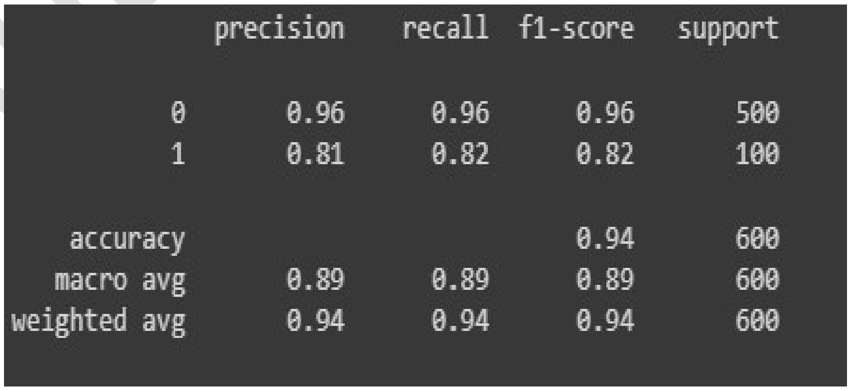


Figure 5: Cross Validation accuracy vs different value of k

- In above figure we can see that optima value of k is 2. In the subsequent steps, we creates a k-NN classifier with `n_neighbors=2` and perform cross-validation using the `cross_val_score` function. The classifier is then fitted to the training data, and predictions are made on the test data `X_test`. The accuracy of the k-NN model is computed using the `accuracy_score` function and stored in the variable `accuracy_knn`. This accuracy score represents how well the model performs on the test data with the chosen value of k. The accuracy of model is 94%.


```
              precision    recall  f1-score   support

           0       0.96      0.96      0.96       500
           1       0.81      0.82      0.82       100

    accuracy                           0.94       600
   macro avg       0.89      0.89      0.89       600
weighted avg       0.94      0.94      0.94       600
```

Figure 6: Classification report of KNN

- The code also generates a classification report using the `classification_report` function, providing detailed performance metrics such as precision, recall, F1-

score, and support for each class in the test data. This report gives a comprehensive overview of the model's predictive capabilities for each class.

- Furthermore, the confusion matrix is computed using the `confusion_matrix` function and displayed as a heatmap using seaborn's `heatmap` function. The confusion matrix presents a visual representation of the model's predictions, showing the counts of true positive, true negative, false positive, and false negative instances.

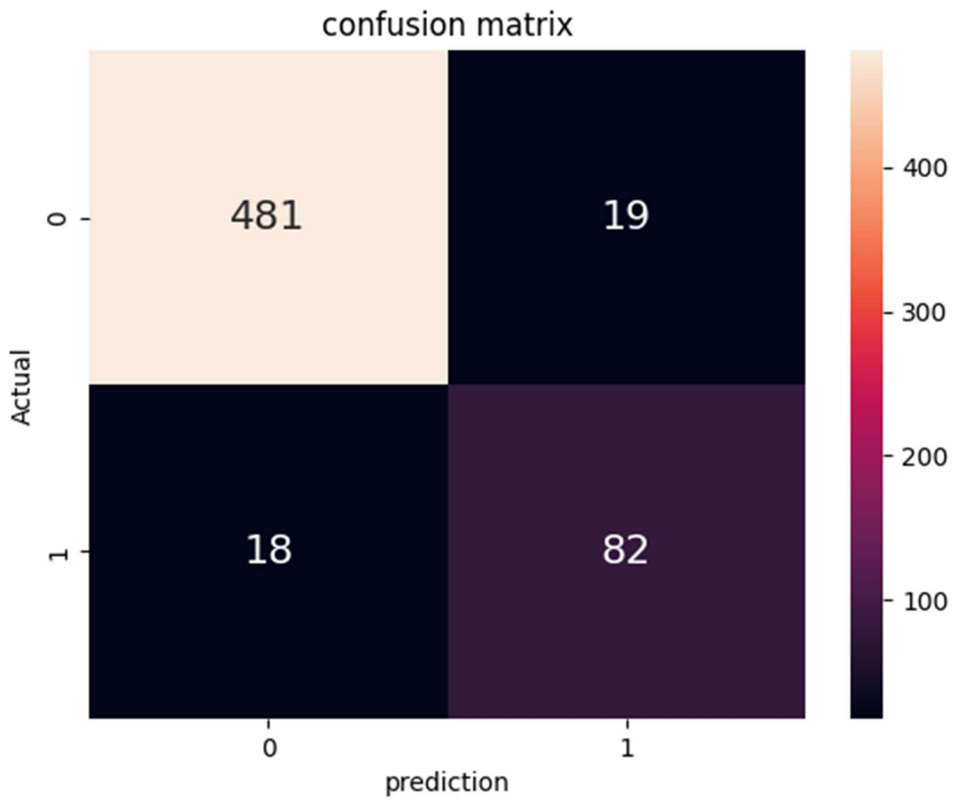


Figure 7: Confusion matrix of KNN

- Finally, the Receiver Operating Characteristic (ROC) curve is plotted using the `RocCurveDisplay.from_estimator` function to visualize the model's performance across different classification thresholds. The ROC curve is a valuable tool for assessing the trade-off between true positive rate and false positive rate for various decision thresholds.

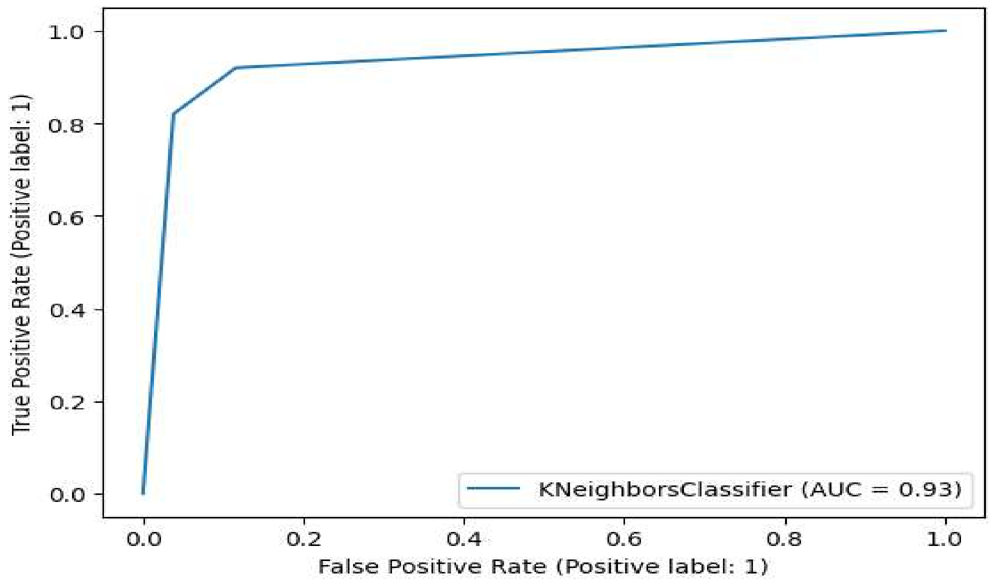
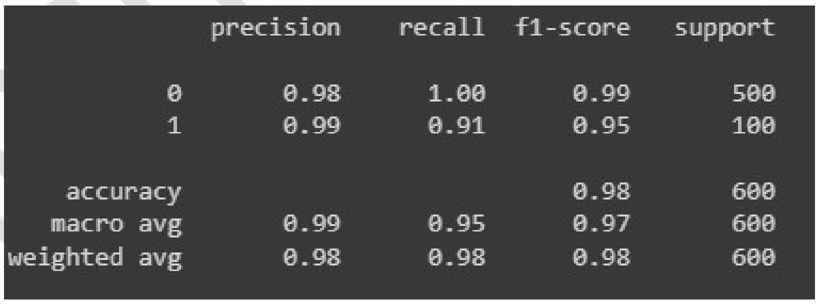


Figure 8: ROC curve of KNN

-

ii. **Random Forest**: A Random Forest classifier is trained on the training data, and its accuracy is evaluated on the test set.

A Random Forest classifier is applied to a binary classification task. The classifier is initialized with the parameter n_estimators=120, which indicates the number of decision trees to be used in the Random Forest ensemble. The n_estimators parameter controls the number of trees in the forest and generally, a higher number of estimators can lead to better performance, but it also increases computation time.



```
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       500
           1       0.99      0.91      0.95       100

    accuracy                           0.98       600
   macro avg       0.99      0.95      0.97       600
weighted avg       0.98      0.98      0.98       600
```

Figure 9: Classification report of Random Forest

The classifier is first trained using the training data X_train and their corresponding labels y_train. Subsequently, predictions are made on the test data X_test, and the accuracy of the classifier is calculated by comparing the predicted labels y_pred with the true labels y_test. The obtained accuracy score is printed and stored in a data structure called acc_score_mat with the key "Random_Forest" for future reference.

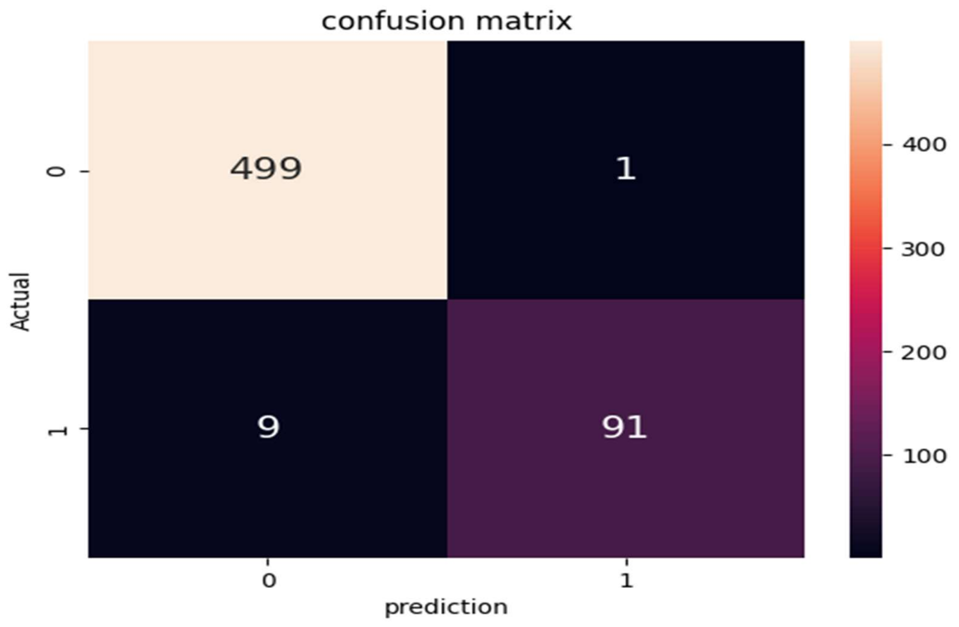


Figure 10: Confusion matrix of Random Forest

Additionally, a classification report is generated, displaying precision, recall, F1-score, and support for each class. The confusion matrix, representing the counts of true positive, true negative, false positive, and false negative predictions, is computed and visualized using a heatmap with annotations. Furthermore, the ROC curve for the Random Forest classifier is plotted using plot_roc_curve and displayed.

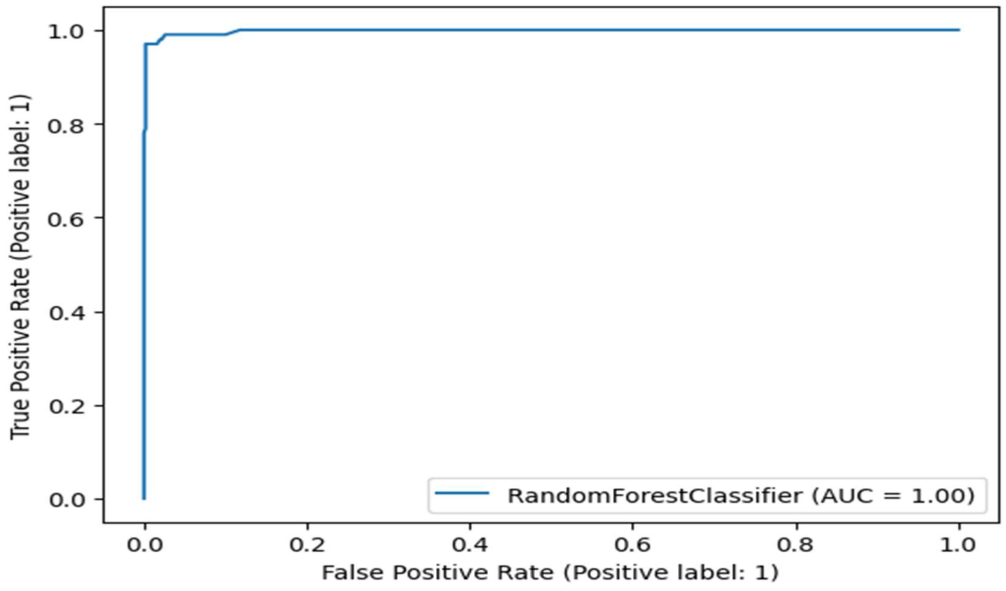
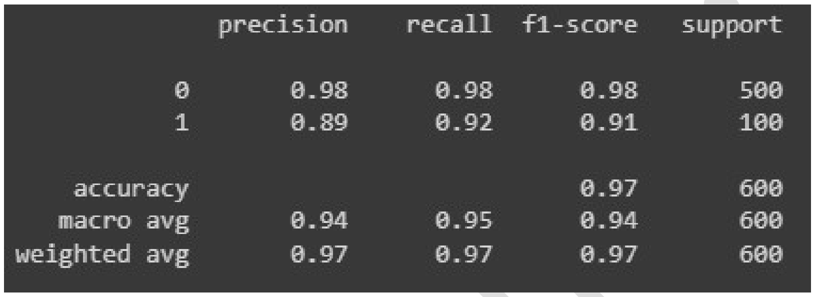


Figure 11: ROC curve of Random Forest

iii. **Decision Tree**: A Decision Tree classifier is trained on the training data, and its accuracy is evaluated on the test set.

The classifier is trained on the training data X_train and their corresponding labels y_train. After training, predictions are made on the test data X_test, and the accuracy of the Decision Tree classifier is calculated by comparing the predicted labels y_pred2 with the true labels y_test. The obtained accuracy score is printed and stored in a data structure named acc_score_mat with the key "Decision Tree" for future reference.



```
              precision    recall  f1-score   support

           0       0.98      0.98      0.98       500
           1       0.89      0.92      0.91       100

    accuracy                           0.97       600
   macro avg       0.94      0.95      0.94       600
weighted avg       0.97      0.97      0.97       600
```

Figure 12: Classification report of DT

Additionally, a classification report is generated, displaying precision, recall, F1-score, and support for each class. The confusion matrix, which indicates the counts of true positive, true negative, false positive, and false negative predictions, is computed and visualized using a heatmap with annotations. Finally, the ROC curve for the Decision Tree classifier is plotted using plot_roc_curve and displayed.

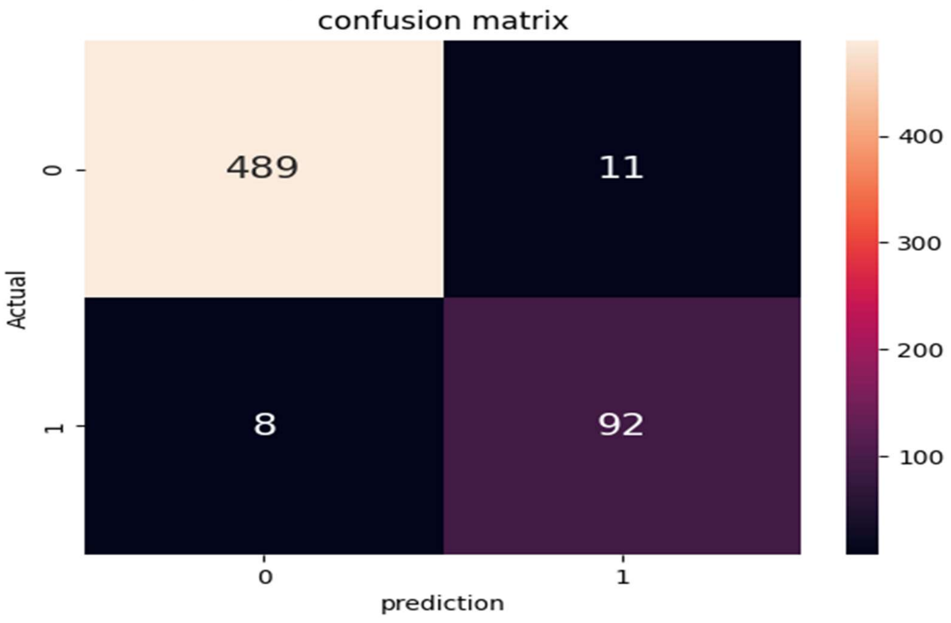


Figure 13: Confusion matrix of DT

This code provides a comprehensive analysis of the Decision Tree classifier's performance, offering accuracy, classification report metrics, confusion matrix, and the ROC curve to evaluate the classifier's effectiveness in the binary classification task.

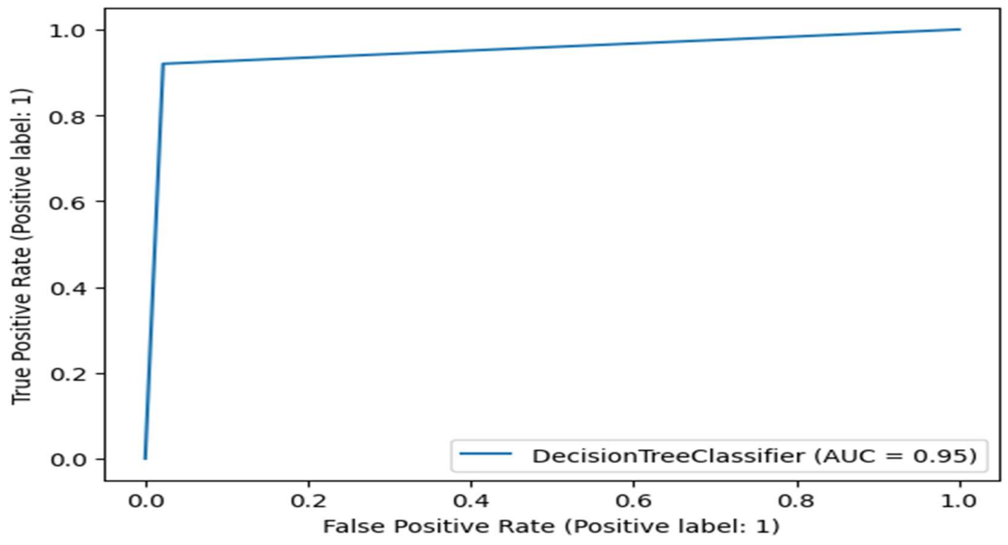
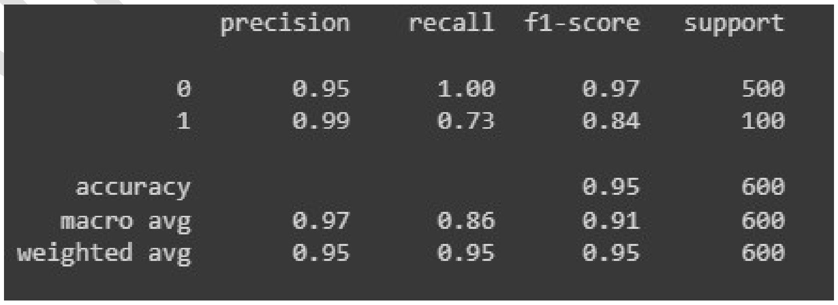


Figure 14: ROC curve of DT

iv. **Support Vector Machine (SVM):** An SVM classifier is trained on the training data, and its accuracy is evaluated on the test set.

The classifier is trained on the training data X_train and their corresponding labels y_train using the svc.fit() method. After training, predictions are made on the test data X_test, and the accuracy of the SVM classifier is calculated by comparing the predicted labels y_pred4 with the true labels y_test. The obtained accuracy score is printed and saved in a data structure called acc_score_mat with the key "SVC" for future reference.


```
              precision    recall  f1-score   support

           0       0.95      1.00      0.97       500
           1       0.99      0.73      0.84       100

    accuracy                           0.95       600
   macro avg       0.97      0.86      0.91       600
weighted avg       0.95      0.95      0.95       600
```

Figure 15: Classification report of SVM

Additionally, a classification report is generated, presenting precision, recall, F1-score, and support for each class. The confusion matrix is computed, indicating the counts of true positive, true negative, false positive, and false negative predictions, and it is visualized as a heatmap with annotations using the seaborn library. Labels and a title are added to the heatmap to improve readability.

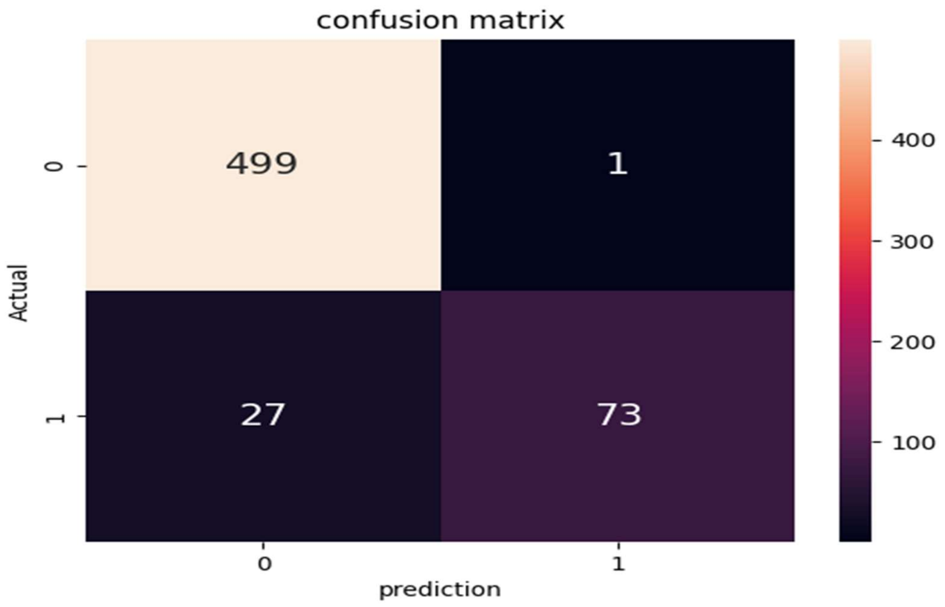


Figure 16: Confusion matrix of SVM

Furthermore, the Receiver Operating Characteristic (ROC) curve for the SVM classifier is plotted using plot_roc_curve and displayed. The ROC curve helps assess the classifier's performance by illustrating the trade-off between true positive rate (sensitivity) and false positive rate (1-specificity) for different classification thresholds.

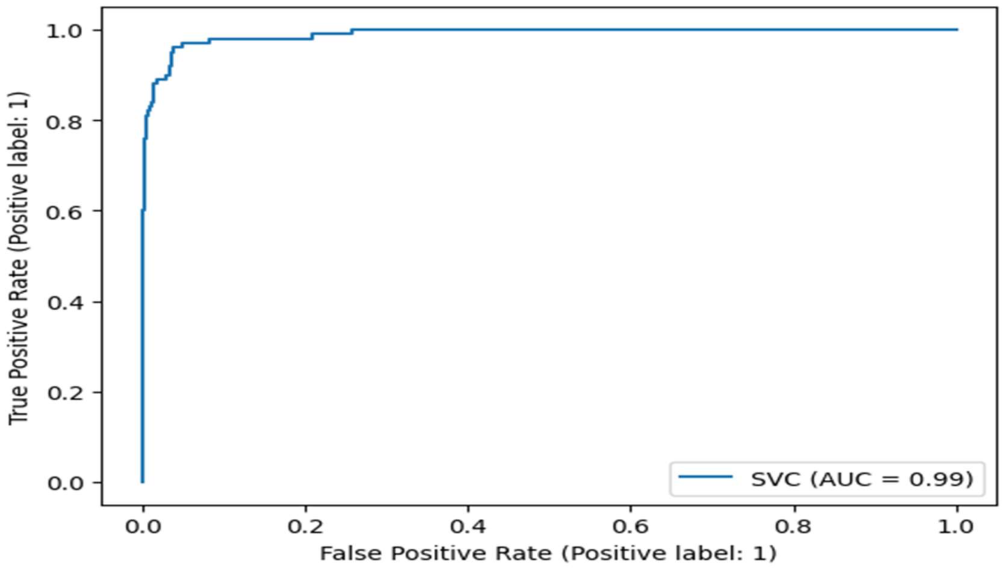


Figure 17: ROC curve of SVM

4. Results:

The performance evaluation of four different algorithms on the binary classification task is presented below. The accuracy scores achieved by each algorithm on the test dataset are as follows:

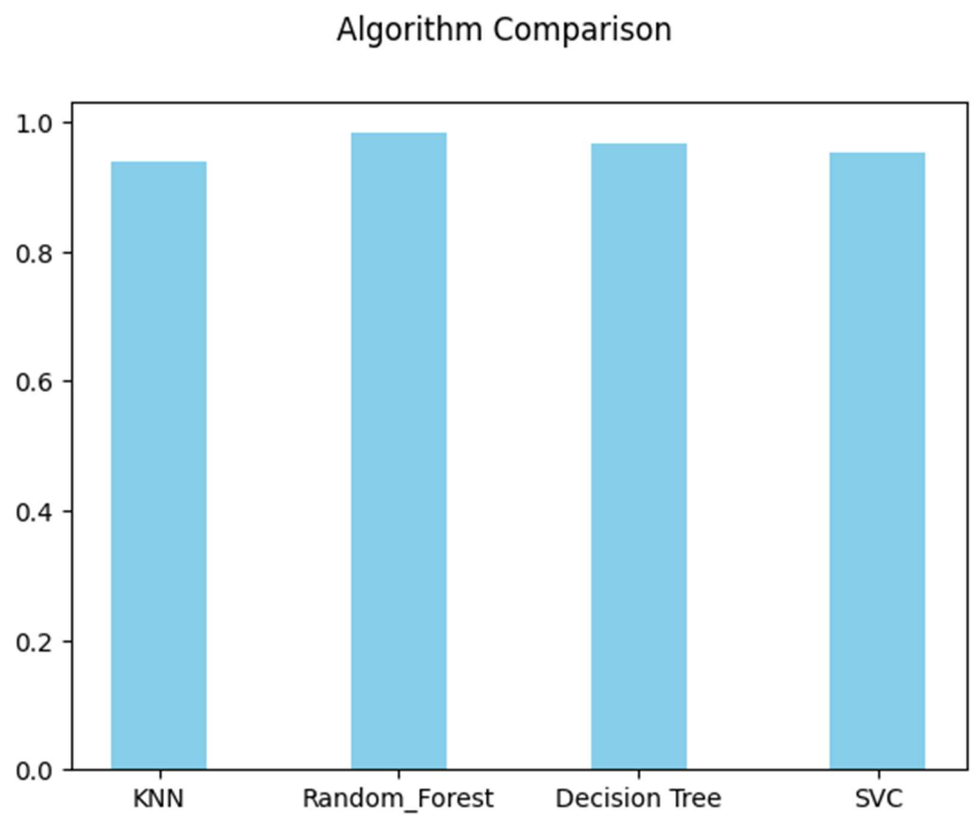


Figure 18: Comparison of Accuracy

The Random Forest algorithm demonstrated the highest accuracy of 0.983, followed closely by the Decision Tree algorithm with an accuracy of 0.968. K-Nearest Neighbors and SVC also achieved respectable accuracy scores of 0.938 and 0.953, respectively. These results suggest that all four algorithms showed promising performance in the binary classification task, with Random Forest standing out as the most accurate among them.

5. Discussion:

During the assignment, several insights and observations were made. The k-NN model's performance was visualized for different values of k, enabling the selection of the best k value. Each algorithm's accuracy, precision, recall, and F1-score were compared to understand their effectiveness. Challenges encountered, if any, during data preprocessing or model implementation were discussed, along with potential ways to improve the results.

Source code: https://github.com/anshshankar/Celebal-task/tree/main/Task-1

Video demonstration:
https://drive.google.com/drive/folders/1AMZ7AHSAf6Z2pP4tZKkpguxWMmYSg_kN?usp=sharing