Faculty of Engineering & Technology

Electrical & Computer Engineering Department

ENCS 3340-Artifical Intelligence

Project Two- Automatic tweet spam detection

Prepared by: Fatima shrateh-1181550

Jehan Alshafie- 1181641 & Hedaya Mustafa-1182126

Instructor: Dr. Aziz Qaroush.

Section: 1

Date: 14-2-2022

Abstract:

Twitter is a social media which most people use in their day to day life. Twitter has grown to be very famous and is currently has 396.5 million users. Twitter allows users to "follow" another user’s account that are of interest to them. Compared to various other social media platform, the connection between users is bi-directional rather than unidirectional connection; this means that a particular user may not be following one of his followers. A user can "like" or "retweet (RT)" a tweet which implies sharing that tweet to his "followers", while retweet is when a tweet made by one user is shared and used by another user.

This project retrieved 11969 twitter account information, to classify if the tweet “spam” or “Quality”. A python code was written to run machine learning algorithms like decision tree, Naïve Bayes, Neural Network and Random forest to find out which algorithm is the suitable for the identification of spam or quality on. Moreover, the python program identified many features that can be used to identify if a tweet is a spam or not.

Contents

Abstract: 2
Introduction: 4
Related Work: 5
Proposed Work (Stages): 8
Experiments & Results: 11
Conclusion and Future Work: 22
References: 23
Appendix: 24

Introduction:

- **machine learning**

Machine learning (ML) is the study of computer algorithms that can improve automatically through experience and by the use of data. It is seen as a part of artificial intelligence. Machine learning algorithms build a model based on sample data, known as training data, in order to make predictions or decisions without being explicitly programmed to do so.

- **Naïve Bayes Classification**:

Naïve Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems. It is mainly used in text classification that includes a high-dimensional training dataset. Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions. is a probabilistic classifier, which means it predicts on the basis of the probability of an object.

- **Decision tree Classification**:

Decision tree analysis is a predictive modelling tool that can be applied across many areas. Decision trees can be constructed by an algorithmic approach that can split the dataset in different ways based on different conditions. Decisions tress are the most powerful algorithms that falls under the category of supervised algorithms.

- **Neural Network Classification**:

The neural network is a general circuit of neurons that can work on any number of inputs and are usually suitable for dealing with nonlinear datasets. Neural networks are more flexible and can be used with both regression and classification problems. So in a situation in which we are supposed to train a model and check if we are on the right track or not, hence we repeatedly forward propagate and back propagate to get the highest accuracy, this whole procedure is working on a neural network!

- **Random Forest Classification**:

Random forest is a technique used in modeling predictions and behavior analysis and is built on decision trees. It contains many decision trees representing a distinct instance of the classification of data input into the random forest. The random forest technique considers the instances individually, taking the one with the majority of votes as the selected prediction.

Related Work:

Spam detection has been studied for a long time. The existing work mainly focuses on email spam detection and Web spam detection. We take these three papers that have target the same subject which is Twitter Spam Detection:

- **Don't Follow Me: Spam Detection in Twitter-Alex Hai Wang- College of Information Sciences and Technology, The Pennsylvania State University, Dunmore, PA 18512, USA.**

In this paper, the author studies the spam behaviors in a popular online social networking site, Twitter. He formalized the problem, and proposed a directed social graph model. He defined in the paper the "follower" and "friend" relationships. Based on the spam policy of Twitter, novel content-based and graph-based features are proposed to facilitate spam detection.

Traditional classification algorithms are applied to detect suspicious behaviors of spam accounts. A Web crawler using Twitter API methods is also developed to collect real data set from public available information on Twitter. Finally, he analyzed the data set and evaluated the performance of the detection system. The results show that among the graph-based features, the proposed reputation feature has the best performance of detecting abnormal behaviors. No many spam accounts follow large amount of users. Also some spammers have many followers. For the content-based features, most spam accounts have multiple duplicate tweets. This was an important feature to detect spam. The author realized that not all spam account posts multiple duplicate tweets and some legitimate users also post duplicate tweets. So he can not only rely on this feature. The results also show that almost all spam tweets contain links and reply sign "@". Finally, several popular classification algorithms are studied and evaluated. The results show that the Bayesian classifier has a better overall performance with the highest F score. The learned classifier is applied to large amount of data and achieve an 89% precision.

- **Kabakus, A. T., & Kara, R. (2017). A survey of spam detection methods on twitter. International Journal of Advanced Computer Science and Application.**

In this paper, the features of Twitter spam detection are presented with discussing their effectiveness in detecting spam. Then, the proposed works in literature are categorized by the authors into four: Account-based, tweet-based, graph-based, and hybrid spam detection methods which use a combination of others.

Methods based on account-based features analyze account by using features related with accounts which some of them can be manipulated by spammers such as the number of following, the number of tweets sent by the account, the number of lists created by the account, the number of moments created by the account which is a brand new feature and the number of mentions the account received, the number of likes received by the tweets of account, and the number of retweets received by the tweets of account. Similarly, the number of followers, the ratio between the number of followers over the number of following, the ratio of the number of tweets liked by others, the ratio of the number of tweets retweeted also can be slightly manipulated by using a group of bots. Bots use various tools to do automated tasks such as following a user, sending a tweet. Some works investigate a number of last tweets of an account in order to reveal if the account posts spam tweets whose contents are almost identical to the tweets recently posted which is useful to detect spam distributed by bots, a set of accounts under the command and control (C&C) infrastructure.

Account-based features are lightweight enough to be used detecting real-time spam which requires instant analysis. The number of lists the user is a member of can be considered a useful metric to detect spammers since it is an obvious sign of the user's impact on others but it is open to manipulation by creating fake lists and adding the fake accounts which are under the C&C infrastructure into these lists. Account-based features are lightweight enough to be used detecting real-time spam which requires instant analysis but they can be easily manipulated by spammers.

Tweet-based spam detection methods use parts of a tweet such as mentions, hashtags, the number of likes the tweet received, the number of retweets the tweet received, the content of tweet, lexical analysis of the tweet, the URL of the tweet, the location of the tweet, the post date of the tweet. Since the most common way to spread spam is sharing via a malicious URL, URLs of tweets are needed to be inspected. Therefore, almost all Twitter spam detection methods inspect URLs of tweets. The traditional ways to filter spam are based on IP blacklisting, domain and URL blacklisting. Since spammers tend to use shortened URLs, traditional URL or IP blacklisting methods are not able to filter malicious URLs in Twitter. Also, Grier et al. show that methods based on blacklisting are too slow to protect users since there is a delay before the malicious URLs are included in the database.

Similar to account-based features, tweet-based features are lightweight enough to be used detecting real-time spam which requires instant analysis. Graph-based spam detection methods use features of relationships between the sender and the mentions of a tweet such as connectivity and distance to analyze how these accounts are connected each other and to measure strengths of their connections in order to reveal the possibility of a spam connection.

Graph-based features are hard to be manipulated, unlike account-based and tweet-based features. However, extracting of these features require in-depth analysis on the huge and complex Twitter graph which is time and resource intensive. Therefore, unlike account-based and tweet-based features, graph-based features are not lightweight enough for real-time spam detection. Another limitation of the graph based approaches is that they assume that tweets come from friends are benign regardless of their content which is not valid when attackers steal the accounts of legitimate users for their malicious aims.

In this paper, the features of Twitter spam detection and proposed approaches in the literature are discussed with considering their advantages and disadvantages.

- **Susana Boniphace Mazikua, A. R. Rahiman, Abdullah Mohammed, Mohd Taufik Abdullah. A Novel Framework for Identifying Twitter Spam Data Using Machine Learning Algorithms.**

This paper proposed a novel framework for identifying twitter spam data using machine learning algorithms to address the feature selection issue. The proposed methods were categorized into three sections: pre-processing, feature selection, and classification.

The experimental results on a series of multilevel datasets demonstrated that their novel framework could effectively identify the best feature subset and minimize larger dataset size errors. Furthermore, they compared with the existing methods of twitter spam detection versions such as Naïve Bayes, SVM, KNN to show the importance of feature selection results. In classification, there experiments' results show the identification of spam and non-spam for real-time running classification jobs. A novel framework method seems to have the best of all performance accuracy and running time. Existing methods showed a long time for training data, overlapping, and performance measures are lower, and some features were dropped during the selection period.

The contribution of this research can reduce dimension from tweets datasets, identify tweet spam in real-time, increase speed up in detecting spam, select the best features for learning model and enhancing efficiency accuracy tasks.

In fact, we read a lot of related works and papers while we worked on this interesting project. in this section we only summarized three of them, but the knowledge we take from the published researches was really huge and we were very happy to read and learn how the researchers thought and made their mark in spam detection field.

Proposed Work (Stages):

We walked through the following steps to build our program:

1. **Read the file**

At first The program reads _using panda library_ the training dataset which is a csv file that called "train" which contains the following columns: Id, Tweet, Following, followers, actions, is_retweet, location, type, where the Id is a unique integer identifying the user, type is either spam or quality.

2. **Preprocessing data.**

Tweets scraped from twitter generally result in a noisy dataset. This is due to the casual nature of people's usage of social media. Tweets have certain special characteristics such as retweets, user mentions, etc. which have to be suitably extracted. Therefore, tweets have to be normalized to create a dataset which can be easily learned by various classifiers. We have applied an extensive number of pre-processing steps to standardize the dataset and reduce its size. We first do some general pre-processing on tweets which is as follows:

- Convert to lower case.
- Remove the URLs from the column tweet.
- Remove the punctuation.
- Replace two dots or more with space.
- Convert more than two letters repetitions to two letters, for example:
  Happppppy → Happy.
- Replace multiple spaces with a single space.
- Remove stop words from tweets.

Also we used a function called "isnull (). sum () "to check if there was an empty cell which returns the number of them for each columns, therefore we computed the mean for both actions and follower's columns to fill them in the empty cells.

The following table shows the number of empty cells for each column:

Id	Tweet	following	followers	Is_retweet	actions	Location	Type
0	0	145	16	1	2773	1651	0

3. Feature extraction process

The feature extraction is a critical step; we processed a larger dataset that was completed from the previous step. Those tweet data were transformed into a new dataset by reduced unwanted features. We worked on the data to make the dataset suitable for further processing. Then we dropped the columns we won't use. Those were the features we extracted:

- **Tweet**

After we cleaned the tweet, we used CountVectorizer to Convert the tweet to a matrix of token counts. After that we used getnnz function which gave us Number of stored values, including explicit zeros. Finally, we stored it in a column called "Tweet".

- **Actions**

As we mentioned in the preprocessing part, we used a function called "isnull (). sum () "to check if there was an empty cell which returns the number of them, therefore we computed the mean for actions column to fill them in the empty cells. Then we stored it in column called "actions"

- **Followers**

Here we did the same as in the previous feature then we stored it in column called "followers".

4. Training and Testing the classifier.

At first we split our data into train and test data as following:

```
x_train, x_test, y_train, y_test = train_test_split(features,twitterData['Type'], test_size=0.20,
random_state=0)
```

then we called the specific library for the required classifier from scikit learn library, then we fit the classifier to the training set.

After that we predict the test set result. Finally, we got the accuracy, report and confusion matrix: Accuracy score means how accurate our model is. Now, to find accuracy we need classification report and confusion matrix. The matrix is a 2X2 matrix which tells about correct and wrong predictions as the form of positive and negative. From here we can say that the accuracy will be the addition of all the truly positive and truly negative predictions divided by the addition of all the numbers in the matrix but first let us understand the matrix and how it works. The matrix has four columns as shown below:

Matrix = [truely_positive   falsely_negative

falsely_positive   truely_negative]

accuracy = (truely_positive+truely_negative) / (truely_positive+truely_negative + falsely_positive + falsely_negative)

while,

truely_positive = case was positive and the model predicted it positive.

truely_negative = case was positive and the model predicted it negative.

falsely_negative = case was negative but the model predicted it positive.

falsely_positive = case was positive but the model predicted it negative.

Experiments & Results:

We preferred to make a useful user interface to display our program easily. When we run the tweet spam detection program the below interface will be shown, it contains four choices, each choice represents a classifier: first button for naïve Bayes classifier, the second for decision tree classifier, then the third one for random forest classifier, and the last represents the neural network classifier. The user could choose one of them, then another window will be opened.

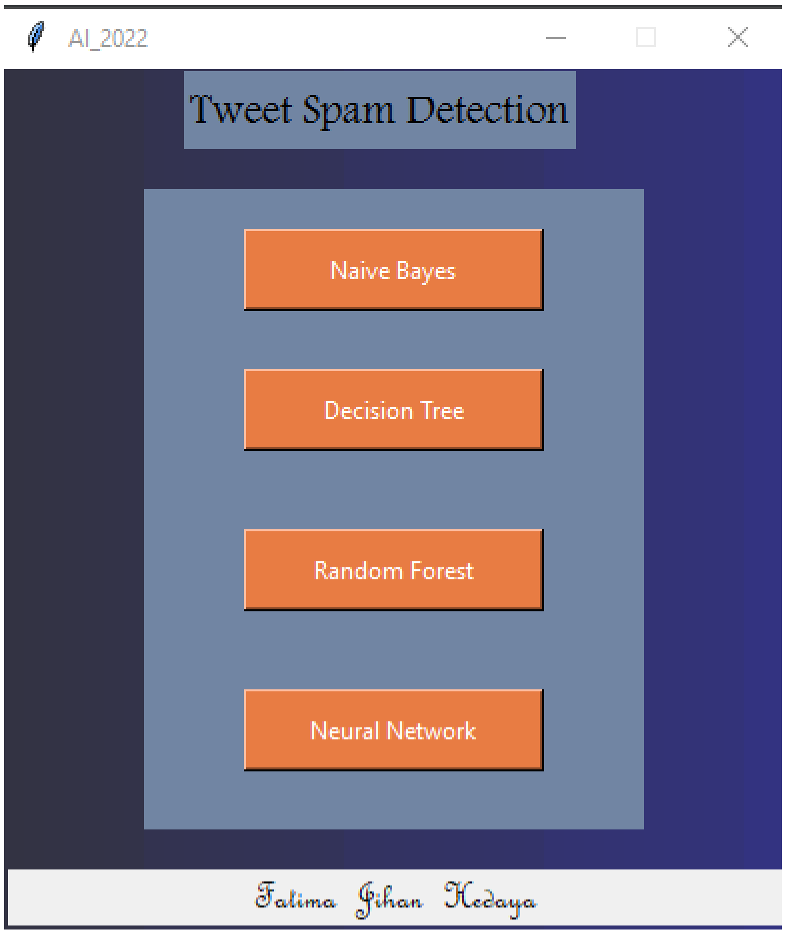


FIGURE 1: MAIN INTERFACE

In the beginning, the window in the middle of figure 2 below shows to display the train and test separated.

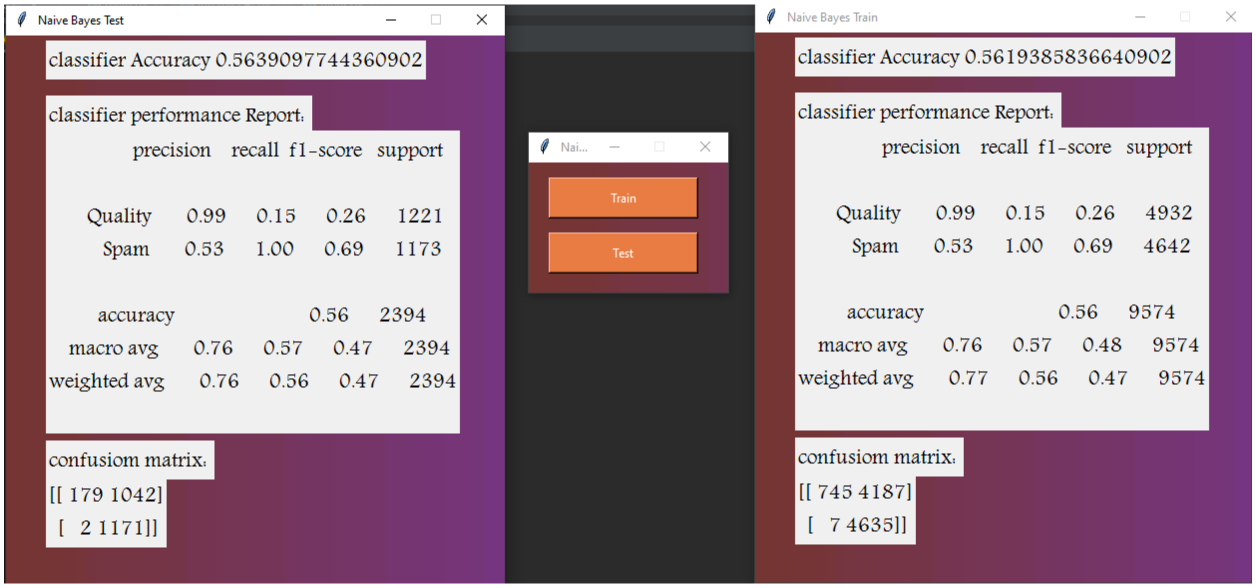
The train window in the right side shows the result we got. the train means that we want to teach the model with our classification to make it able to take the decision if the tweet spam or not, so first we got the accuracy around 0.56 which was not very good accuracy so we searched for the reason since the accuracy is one of the most important performance evaluation metrics for a classification model, what we got to is that This is not a big error for Naive Bayes, this is extremely simple classifier and we should not expect it to be strong, we thought that if we have more data it will be better but we reached that more data probably won't help. our multinomial estimators were probably already very good; simply Naive assumptions are the problem.

then we display the classifier performance report, it displays our model's precision, recall, F1 score and support. It provides a better understanding of the overall performance of our trained model. Precision is the ratio of true positives to the sum of true and false positives. Recall is the ratio of true positives to the sum of true positives and false negatives. The F1 is the weighted harmonic mean of precision and recall. The closer the value of the F1 score is to 1.0, the better the expected performance of the model is but in our case it wasn't as good as we except. And support is the number of actual occurrences of the class in the dataset.

Finally, we display the confusion matrix which tells about correct and wrong predictions as the form of positive and negative. The matrix contains two rows and two columns that represent the following: The number of truly positive samples here equals to 745, the number of the falsely negative equals to 4187, the number of falsely positive samples equals to 7, and the truly negative 4635.

The test window in the left side tells the result of testing the model we trained before. So that the model will decide by itself if the tweet is spam or quality.

We determine 80% of the data for the training and the other 20% for testing. So in the training part we have two classes: Quality and spam, quality was 4932 and the spam was 4642. The testing part has also two classes, the Quality has 1221 and the spam 1173. The sum of these numbers equal 11968 which was the data we have in the excel sheet.



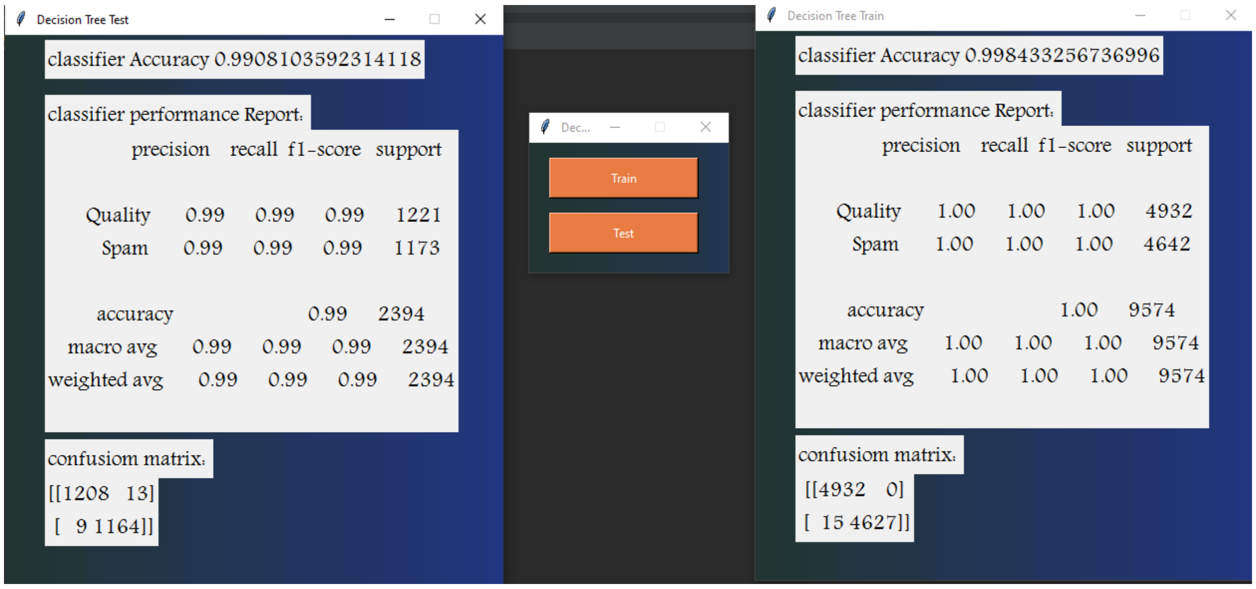
```
classifier Accuracy 0.5639097744360902

classifier performance Report:
              precision    recall  f1-score   support

     Quality       0.99      0.15      0.26      1221
        Spam       0.53      1.00      0.69      1173

    accuracy                           0.56      2394
   macro avg       0.76      0.57      0.47      2394
weighted avg       0.76      0.56      0.47      2394

confusiom matrix:
[[ 179 1042]
 [   2 1171]]
```

```
classifier Accuracy 0.5619385836640902

classifier performance Report:
              precision    recall  f1-score   support

     Quality       0.99      0.15      0.26      4932
        Spam       0.53      1.00      0.69      4642

    accuracy                           0.56      9574
   macro avg       0.76      0.57      0.48      9574
weighted avg       0.77      0.56      0.47      9574

confusiom matrix:
[[ 745 4187]
 [   7 4635]]
```

FIGURE 2: NAIVE BAYES TRAINING & TESTING

As we told in the first classifier we were upset from the accuracy of the naïve Bayes, so we started in the decision tree with hope to get a better accuracy and that what happened. So at first we did as the previous, we build the same interface with another colors, the accuracy we got after run the train for 80% of the data, which equals 9574 sample. Accuracy was around 0.998 which was almost perfect. Then the classifier performance report showed that the result was perfect since the score equal to 1.0. here The matrix contains two rows and two columns that represent the following: The number of truly positive samples here equals to 4932, the number of the falsely negative equals to 0, the number of falsely positive samples equals to 15, and the truly negative 4627.

Then we ran the test with 20% of the data we had, which equals 2394 sample. Accuracy was around 0.9908 which was very good but less than the training. So the score was 0.99 not exactly 1.0 but still satisfying. The matrix in the test contains two rows and two columns that represent the following: The number of truly positive samples here equals to 1208, the number of the falsely negative equals to 13, the number of falsely positive samples equals to 9, and the truly negative 1164.



```
classifier Accuracy 0.9908103592314118

classifier performance Report:
              precision    recall  f1-score   support

     Quality       0.99      0.99      0.99      1221
        Spam       0.99      0.99      0.99      1173

    accuracy                           0.99      2394
   macro avg       0.99      0.99      0.99      2394
weighted avg       0.99      0.99      0.99      2394

confusiom matrix:
[[1208   13]
 [   9 1164]]
```

```
classifier Accuracy 0.9984332567369966

classifier performance Report:
              precision    recall  f1-score   support

     Quality       1.00      1.00      1.00      4932
        Spam       1.00      1.00      1.00      4642

    accuracy                           1.00      9574
   macro avg       1.00      1.00      1.00      9574
weighted avg       1.00      1.00      1.00      9574

confusiom matrix:
 [[4932    0]
 [  15 4627]]
```

FIGURE 3: DECISION TREE TRAINING & TESTING

Also we plot the decision tree as shown in figure 4 below, we used graphviz library to implement this tree of decision tree classifier, to track a case on the tree we started from the head, which had the action less than or equal 0.5, if this match the case (true) we will turn to the left side of the tree, otherwise (false) we will turn to the right side of the tree until we get to the last node in the branch.

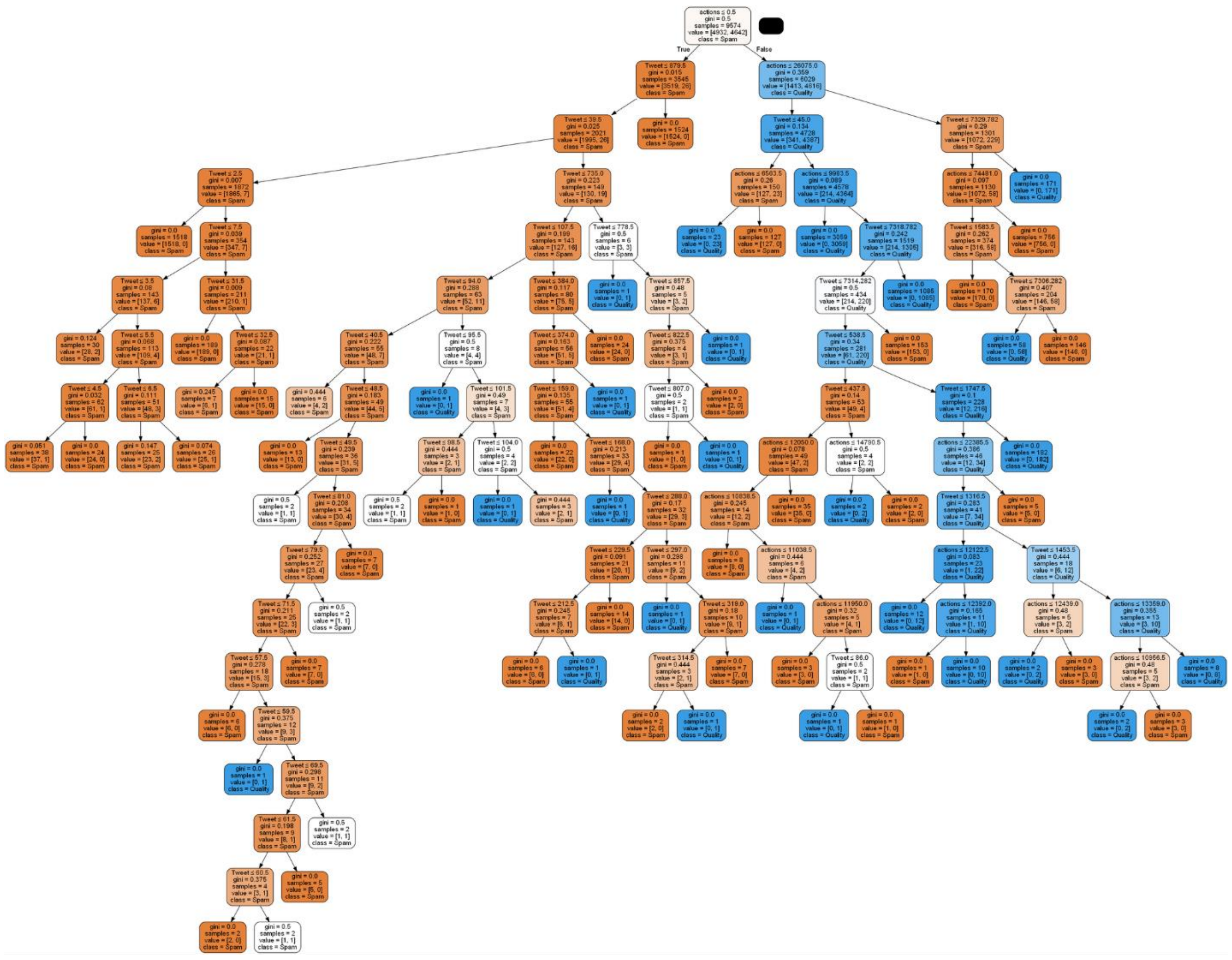
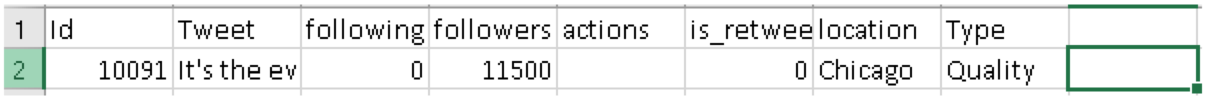


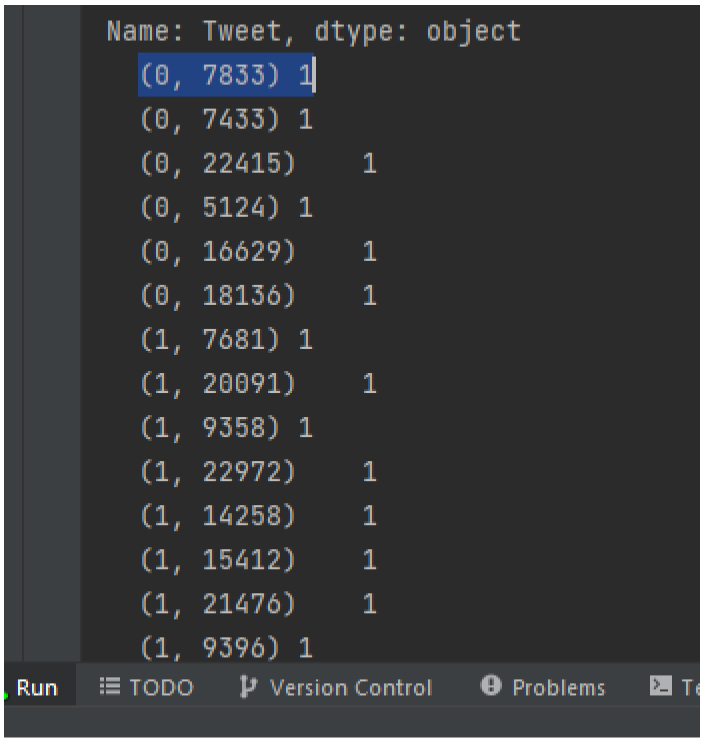
FIGURE 4: DECISION TREE

Here, we applied the following case from excel sheet on the tree. At first we started from the head where the action less than or equal 0.5, as in our case: the action was an empty cell, and as we filled the null cells with the mean which was equal to 7314, since its more than 0.5 so we turned to the right side where the action in the current node will be less than or equal 26075.0 which matched our action case, then we turned to the left side to find that the current node had less than or equal 45, so we print the tweet token in our program to find that the tweet token 7833 as shown in figure 6, since the token more than 45 we turned to the right side where the action at that node less than or equal 9983.5, then we reached to the least node at left side, which was the quality class.



1	Id	Tweet	following	followers	actions	is_retwee	location	Type	
2	10091	It's the ev	0	11500		0	Chicago	Quality	

FIGURE 5: CASE TRACKED ON THE TREE



```
Name: Tweet, dtype: object
  (0, 7833) 1
  (0, 7433) 1
  (0, 22415)	1
  (0, 5124) 1
  (0, 16629)	1
  (0, 18136)	1
  (1, 7681) 1
  (1, 20091)	1
  (1, 9358) 1
  (1, 22972)	1
  (1, 14258)	1
  (1, 15412)	1
  (1, 21476)	1
  (1, 9396) 1
```

FIGURE 6: TWEET TOKEN

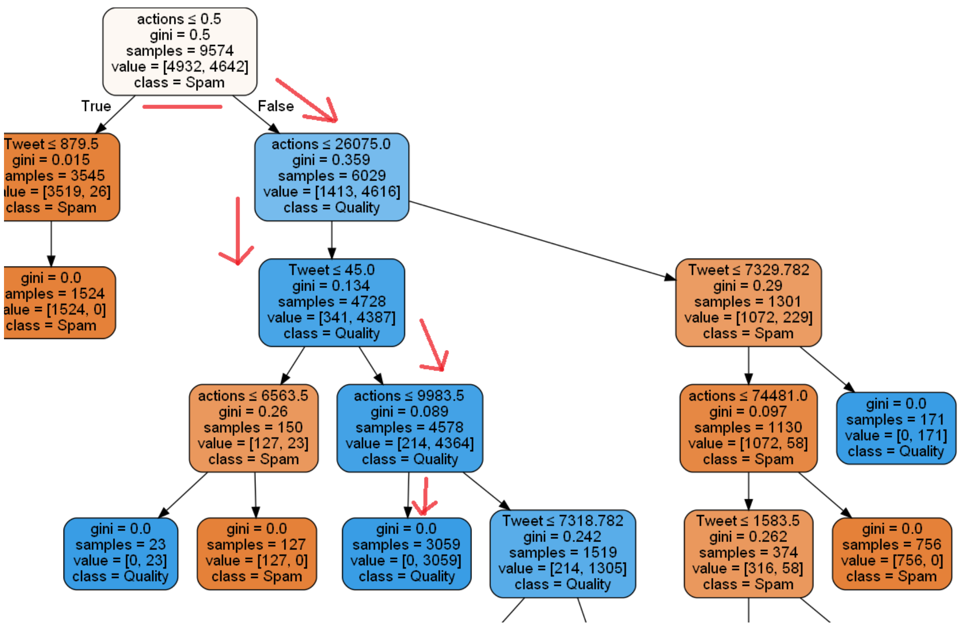
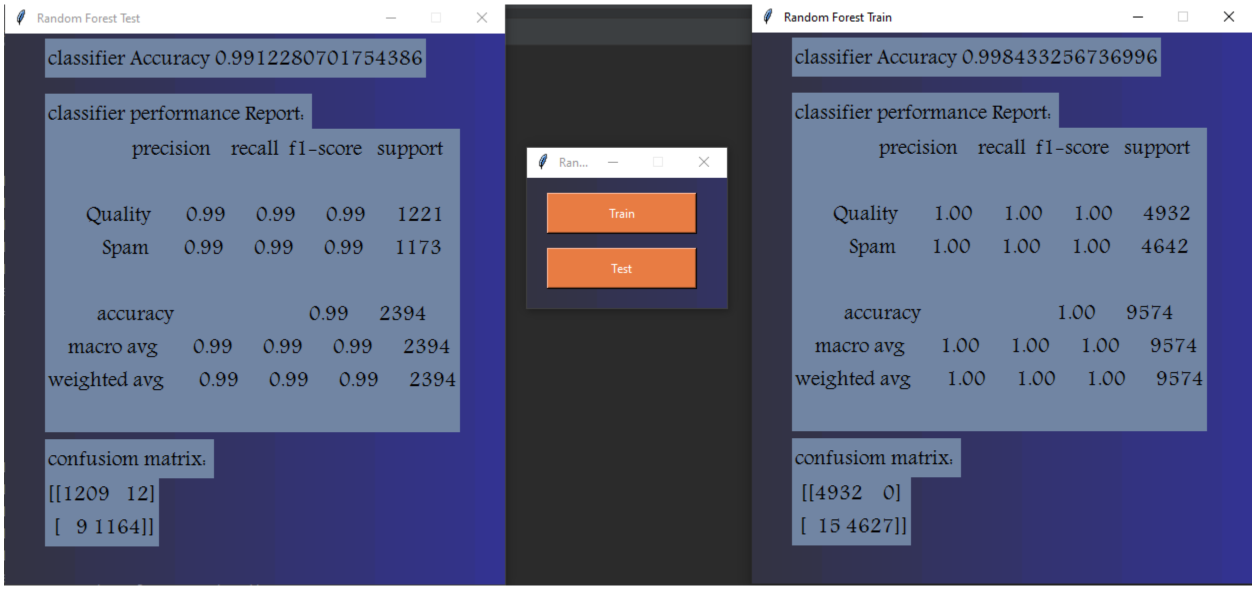


FIGURE 7: TRACKING THE CASE ON THE TREE

Then we moved to the third classifier which called Random Forest. So at first we did as the previous, the accuracy we got after run the train for 80% of the data, which equals 9574 sample. Accuracy was around 0.998 which was near to the decision tree classifier train's result. Then the classifier performance report showed that also the result was perfect since the score equal to 1.0. here The matrix contains two rows and two columns that represent the following: The number of truly positive samples here equals to 4932, the number of the falsely negative equals to 0, the number of falsely positive samples equals to 15, and the truly negative 4627.

Then we ran the test with 20% of the data we had, which equals 2394 sample. Accuracy was around 0.9912 which was very good but less than the training. So the score was 0.99 not exactly 1.0 but still satisfying. The matrix in the test contains two rows and two columns that represent the following: The number of truly positive samples here equals to 1208, the number of the falsely negative equals to 12, the number of falsely positive samples equals to 9, and the truly negative 1164.



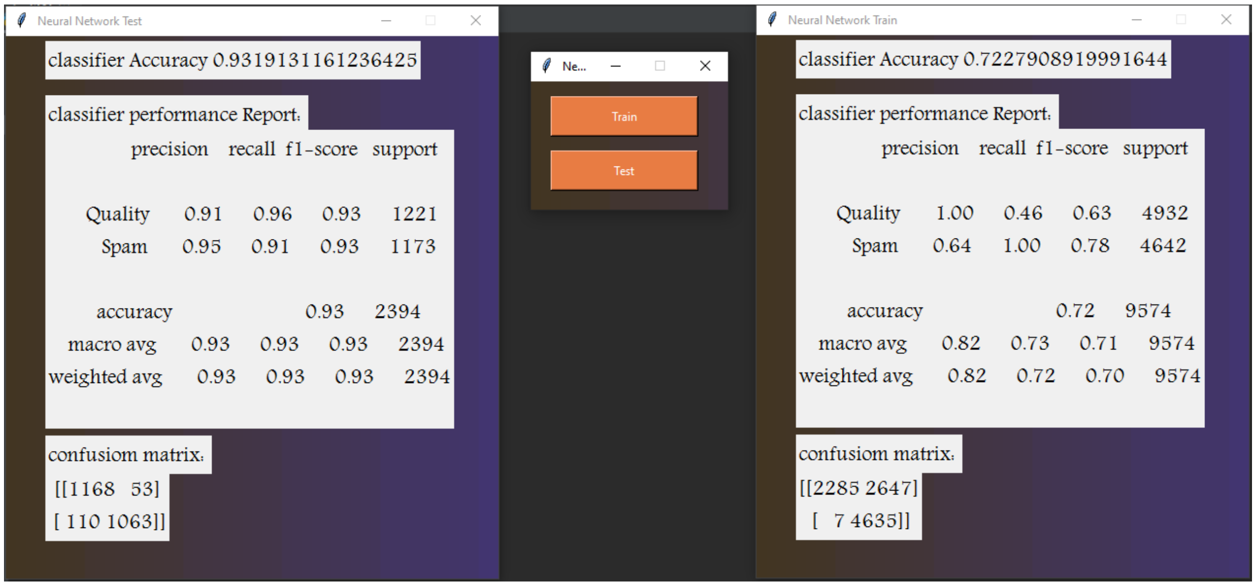
```
classifier Accuracy 0.9912280701754386

classifier performance Report:
              precision    recall  f1-score   support

     Quality       0.99      0.99      0.99      1221
        Spam       0.99      0.99      0.99      1173

    accuracy                           0.99      2394
   macro avg       0.99      0.99      0.99      2394
weighted avg       0.99      0.99      0.99      2394

confusiom matrix:
[[1209   12]
 [   9 1164]]
```

```
classifier Accuracy 0.998433256736996

classifier performance Report:
              precision    recall  f1-score   support

     Quality       1.00      1.00      1.00      4932
        Spam       1.00      1.00      1.00      4642

    accuracy                           1.00      9574
   macro avg       1.00      1.00      1.00      9574
weighted avg       1.00      1.00      1.00      9574

confusiom matrix:
 [[4932    0]
 [  15 4627]]
```

FIGURE 8: RANDOM FOREST TRAINING & TESTING

Finally, we arrived to the forth and the last: The Neural Network Classifier. So at first we did as the previous, the accuracy we got after run the train for 80% of the data, which equals 9574 sample. Accuracy was around 0.722 which was better than the naïve but less than the decision tree and the random forest. Then the classifier performance report showed that the result was good since the score near to 1.0. here The matrix contains two rows and two columns that represent the following: The number of truly positive samples here equals to 2285, the number of the falsely negative equals to 2647, the number of falsely positive samples equals to 1, and the truly negative 4635.

Then we ran the test with 20% of the data we had, which equals 2394 sample. Accuracy was around 0.93 which was much better than the training. So the score was between 0.91 and 0.93 which was near to 1.0. The matrix in the test contains two rows and two columns that represent the following: The number of truly positive samples here equals to 1168, the number of the falsely negative equals to 53, the number of falsely positive samples equals to 110, and the truly negative 1063.



```
classifier Accuracy 0.9319131161236425

classifier performance Report:
              precision    recall  f1-score   support

     Quality       0.91      0.96      0.93      1221
        Spam       0.95      0.91      0.93      1173

    accuracy                           0.93      2394
   macro avg       0.93      0.93      0.93      2394
weighted avg       0.93      0.93      0.93      2394

confusiom matrix:
 [[1168   53]
 [ 110 1063]]
```

```
classifier Accuracy 0.7227908919991644

classifier performance Report:
              precision    recall  f1-score   support

     Quality       1.00      0.46      0.63      4932
        Spam       0.64      1.00      0.78      4642

    accuracy                           0.72      9574
   macro avg       0.82      0.73      0.71      9574
weighted avg       0.82      0.72      0.70      9574

confusiom matrix:
[[2285 2647]
 [   7 4635]]
```

FIGURE 9: NEURAL NETWORK TRAINING & TESTING

After we tried the above classifiers we searched about overfitting and we started to apply our work on it, Overfitting is a common explanation for the poor performance of a predictive model. So we plot overfitting graphs on both decision tree and random forest as shown in the figures below.

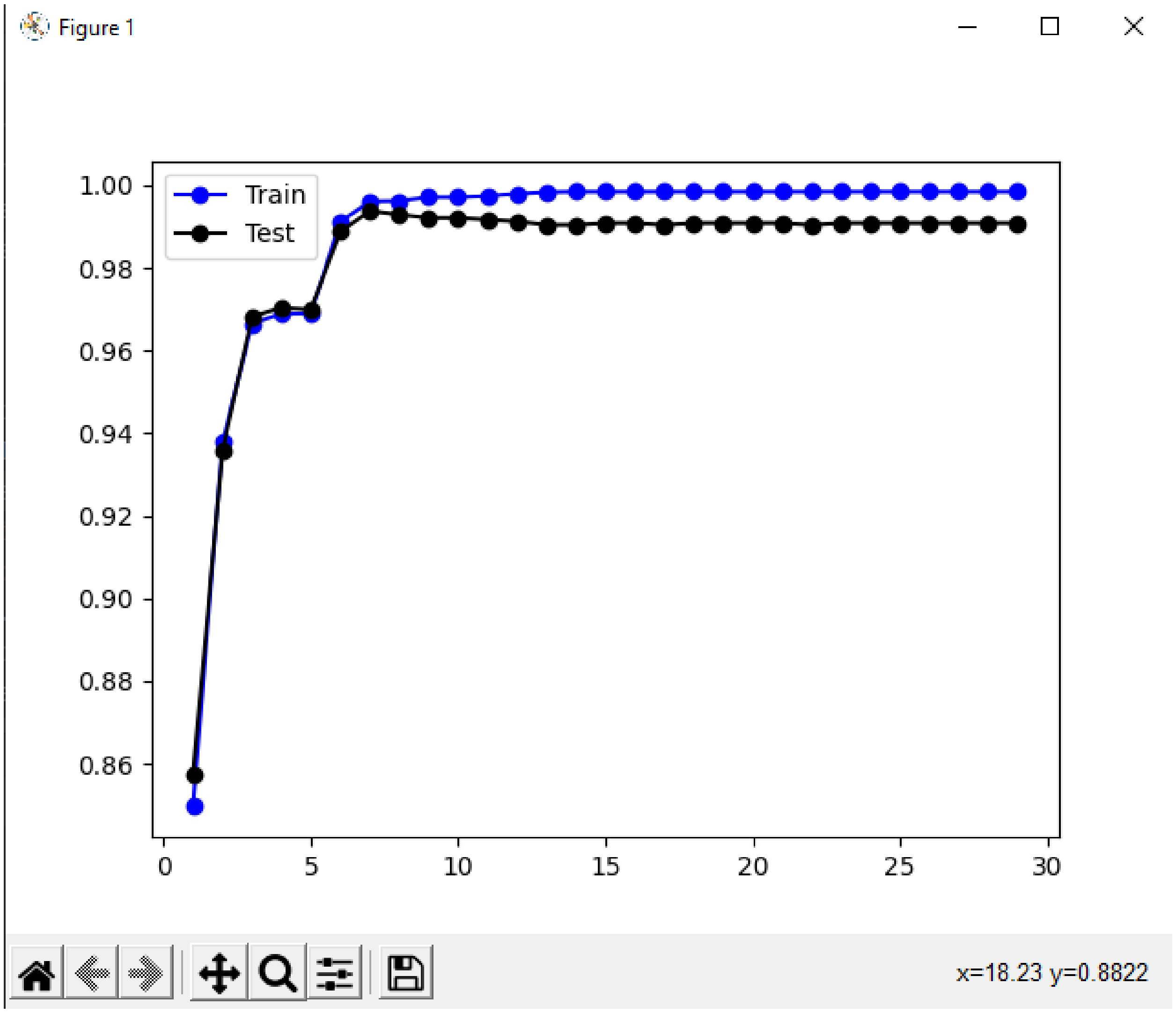


FIGURE 10: DECISION TREE

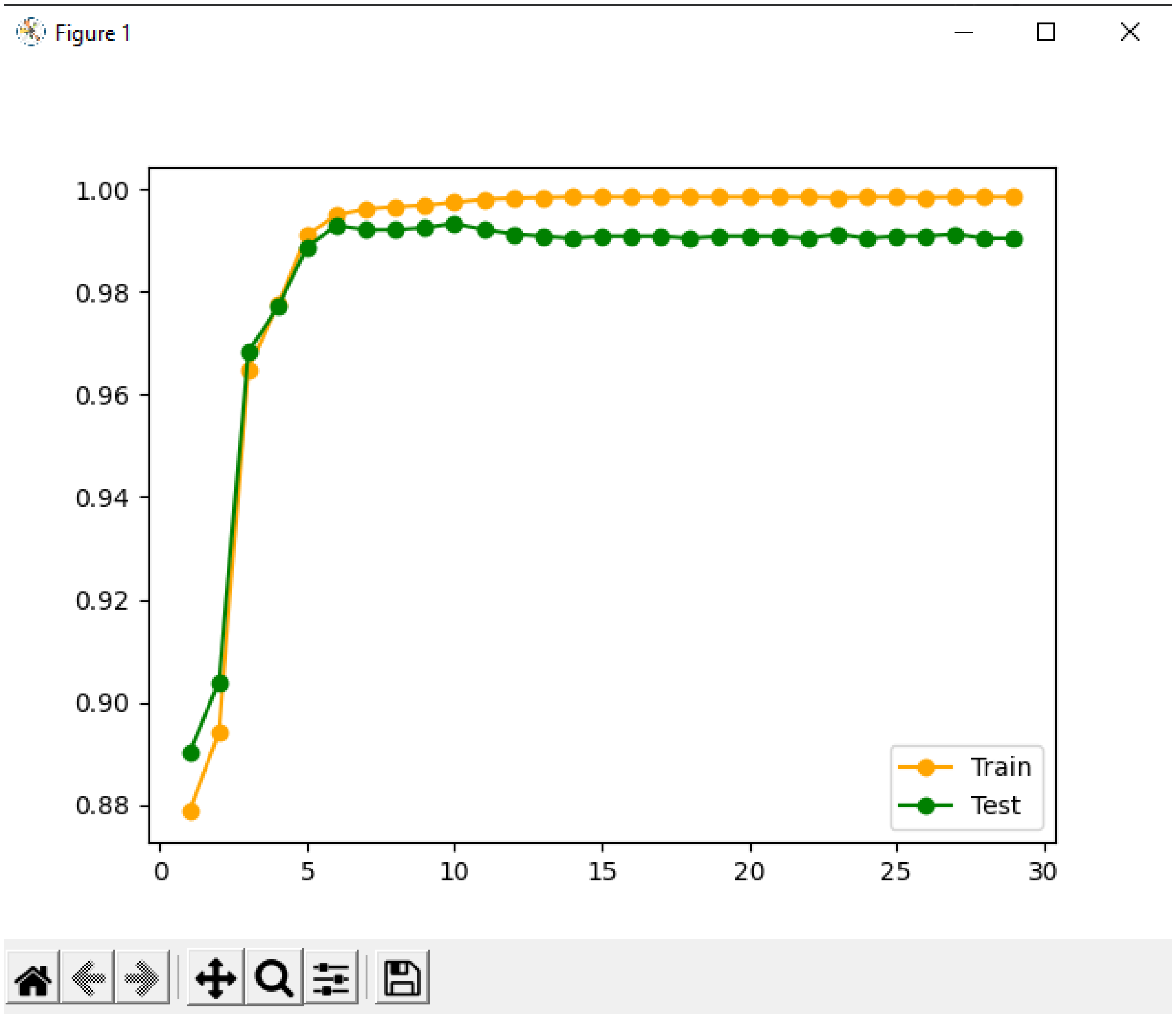


FIGURE 11: RANDOM FOREST

Conclusion and Future Work:

In this project we have learned a lot about classifiers (naïve Bayes, random forest, decision tree and neural network), and how to think and formalize a real world problem in terms of Artificial Intelligence field. Also, we have succeeded in implementing a program that automatically detect if the tweet is spam or not. As a future work, this project motivated us to learn more about machine learning, and other techniques in AI field, since AI is the basic unit of our life in all fields in these days, and it can make our life easier.

This project was huge and special in this semester, we learned a lot by reading many research papers, for this reason a lot of ideas were generated that could improve our project in the future, such that:

- we can identify the sentiment in the tweet whether it's positive or negative by the emojis, symbols and some words or sentences. Which lead us to recognize if the tweet is spam or not.
- Instead of using only Multinomial in the naïve Bayes algorithm we can use the Gaussian method.
- We can add more algorithms like: K-Nearest Neighbors and support vector machines.
- We can collect the dataset of tweets in different languages.
- We can edit this program to work with other social media platforms like Facebook.
- More features can be added that can make Twitter spam detection more valuable for users.

References:

- Don’t Follow Me: Spam Detection in Twitter-Alex Hai Wang- College of Information Sciences and Technology, The Pennsylvania State University, Dunmore, PA 18512, USA.
- Kabakus, A. T., & Kara, R. (2017). A survey of spam detection methods on twitter. International Journal of Advanced Computer Science and Application.
- Susana Boniphace Mazikua, A. R. Rahiman, Abdullah Mohammed, Mohd Taufik Abdullah. A Novel Framework for Identifying Twitter Spam Data Using Machine Learning Algorithms.
- Unsupervised collective-base d framework for dynamic retraining of supervised real-time spam tweets detection model. Mahdi Washha, Aziz Qaroush, Manel Mezghani, Florence Sedes.
- Wu, T., Wen, S., Xiang, Y., & Zhou, W. (2017). Twitter spam detection: Survey of new approaches and comparative study. Computers and Security, 76. doi: 10.1016/ j. cose.2017.11.013.
- **https://stackoverflow.com/**
- **https://www.geeksforgeeks.org/**
- **https://www.codecademy.com/learn/paths/machine-learning**
- **https://machinelearningmastery.com/overfitting-machine-learning-models/**

Appendix:

```
Fatima shrateh-1181550 Jihan Alshafie- 1181641 & Hedaya Mustafa-1182126
# Automatic tweet spam detection Project
#-----------------------LIBRARIES---------------------------
import os
import tkinter
import tkinter as tk
from tkinter import *
import numpy as np
import pandas as pd
import nltk
from nltk.corpus import stopwords
import string
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn import metrics
from sklearn.ensemble import RandomForestClassifier
import sklearn.neural_network
import matplotlib.pyplot as plt
import sklearn.model_selection
import re
import sys
from nltk.stem.porter import PorterStemmer
import numpy as geek
from sklearn.datasets import load_iris
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from matplotlib import pyplot

#################################################################
from sklearn.tree import export_graphviz
import graphviz
from IPython.display import Image
import pydotplus
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn import tree
from six import StringIO
from IPython.display import Image
import pydotplus

####################################################################
# Welcome window interface
welcome = Tk()
# login window settings
welcome.geometry('390x430')
welcome.title(' AI_2022 ')
welcome.resizable(0, 0)
```

```
#---------------------READ FILE-------------------------
twitterData = pd.read_csv('train.csv')
print(twitterData.isnull().sum())                    # find the null cells in
all columns
#---------------------------------------CLEAN DATA-----------------------------
---------
# Function to clean Tweet coulmn by removing punctuation , stop words , etc.
...
def clean_data(Tweet):

    tweetlower = Tweet.lower() # Convert to lower case
    #this method from library re to remove the URLs from the column tweet
    CleanData = re.sub(r'((www\.[\S]+)|(https?:\/\/\S+))', '',tweetlower)
    #here we removed the punctuation from tweets
    withoutpunctuation = [char for char in CleanData if char not in
string.punctuation]
    withoutpunctuation = ''.join(withoutpunctuation) #replace the punctuation
with space

    replace2dots = re.sub(r'\.{2,}','', withoutpunctuation)#Replace 2+ dots
with space
    without_quotetion = replace2dots.strip(' "\'')# Strip space, " and ' from
tweet

    # Convert more than 2 letter repetitions to 2 letter
    # Happppppy --> Happy
    withoutrepetitions = re.sub(r'(.)\1+', r'\1\1', without_quotetion)

    # Replace multiple spaces with a single space
    multiple_spaces = re.sub(r'\s+', ' ', withoutrepetitions)
    #remove stopwords from tweets
    without_stop_words = [word for word in multiple_spaces.split() if
word.lower() not in stopwords.words('english')]

    return without_stop_words
#-----------------------------------------------------------------------------
--------------
print(twitterData['Tweet'].head().apply(clean_data))
#convert each tweet to tokens and count them and represent these tokens as
vectors
tweet_token =
CountVectorizer(analyzer=clean_data).fit_transform(twitterData['Tweet'])
print(tweet_token)
print("============================")
twitterData['actions'] =
twitterData.actions.fillna(twitterData.actions.mean())
print(twitterData['actions'])
twitterData['followers'] =
twitterData.followers.fillna(twitterData.followers.mean())
#-----------------------------------------------------------------------------
-------
#Drop the coulmns that we don't need
features = twitterData.drop(columns = ['Id','following', 'is_retweet',
'location','Type'])
#we give 20% for testing, 80% for training
x_train, x_test, y_train, y_test =
train_test_split(features,twitterData['Type'], test_size=0.20,
random_state=0)

#---------------------------------------------------------------------------
--------------
#function for the interface of naive bayes to let user choose train data or
test data
def test_train_naiveBayes():
    tt1 = Tk()
    # login window settings
    tt1.geometry('200x130')
    tt1.title(' Naive Bayes ')
    tt1.resizable(0, 0)
    gradienttttt(tt1)

    test1 = Button(tt1, width=20, height=2, text="Train", bg="#E87C43",
fg="white", command=naiveBayes_Train)
    test1.pack()
    test1.place(x=20, y=15)

    test2 = Button(tt1, width=20, height=2, text="Test",bg = "#E87C43",fg =
"white", command=naiveBayes_Test)
    test2.pack()
    test2.place(x=20, y=70)

#function to train the datat using naive bayes classifier and show it in an
interface as an accuracy and report
def naiveBayes_Train():
    nb = Tk()
    # login window settings
    nb.geometry('500x550')
    nb.title(' Naive Bayes Train')
    nb.resizable(0, 0)
    gradienttttt(nb)

    # use ready clasifier from naive bayes
    naive_bayes = MultinomialNB().fit(x_train, y_train)
    #print(naive_bayes.predict_proba(x_test[:5]))


    predicted_values1 = naive_bayes.predict(x_train)  # predicted value (x)
    print(predicted_values1)
    #print the accuracy of Train data
    accuracyy = Label(nb, text=f'classifier Accuracy
{accuracy_score(y_train,predicted_values1)}')
    o = ('Andalus', 16)
    accuracyy.config(font=o)
    accuracyy.place(x=40, y=5)
    #print performance report of train data
    report = Label(nb, text='classifier performance Report: ')
    o = ('Andalus', 16)
    report.config(font=o)
    report.place(x=40, y=60)

    metricss = Label(nb, text=f"{classification_report(y_train,
predicted_values1)}")
    o = ('Andalus', 16)
    metricss.config(font=o)
    metricss.place(x=40, y=95)
    # print the confusion matrix
    confmat = Label(nb, text="confusion matrix: ")
    o = ('Andalus', 16)
    confmat.config(font=o)
    confmat.place(x=40, y=405)

    conf = Label(nb, text=f"{confusion_matrix(y_train,predicted_values1)}")
    o = ('Andalus', 16)
    conf.config(font=o)
    conf.place(x=40, y=440)

# Function to test the datat using naive bayes classifier
def naiveBayes_Test():
    nb1 = Tk()
    # login window settings
    nb1.geometry('500x550')
    nb1.title(' Naive Bayes Test ')
    nb1.resizable(0, 0)
    gradienttttt(nb1)

    # use ready clasifier from naive bayes
    naive_bayes = MultinomialNB().fit(x_train, y_train)
    predicted_values1 = naive_bayes.predict(x_test)  # predicted value (x)
    print(f"predict values{predicted_values1}")
    # print accuracy for test data
    accuracyy = Label(nb1, text=f'classifier Accuracy {accuracy_score(y_test,
predicted_values1)}')
    o = ('Andalus', 16)
    accuracyy.config(font=o)
    accuracyy.place(x=40, y=5)
    # print report for test data
    report = Label(nb1, text='classifier performance Report: ')
    o = ('Andalus', 16)
    report.config(font=o)
    report.place(x=40, y=60)

    metricss = Label(nb1, text=f"{classification_report(y_test,
predicted_values1)}")
    o = ('Andalus', 16)
    metricss.config(font=o)
    metricss.place(x=40, y=95)
    # print confusion matrix for test data
    confmat = Label(nb1, text="confusion matrix: ")
    o = ('Andalus', 16)
    confmat.config(font=o)
    confmat.place(x=40, y=405)

    conf = Label(nb1, text=f"{confusion_matrix(y_test,predicted_values1)}")
    o = ('Andalus', 16)
    conf.config(font=o)
    conf.place(x=40, y=440)
# ----------------------------------------------------------------------------
-----------------------------------------
# function to let the user choose from an interface tran data or test data of
Decision tree
def test_train_DecisionTree():
    tt2 = Tk()
    # login window settings
    tt2.geometry('200x130')
    tt2.title(' Decision Tree ')
    tt2.resizable(0, 0)
    gradienttt(tt2)

    test1 = Button(tt2, width=20, height=2, text="Train", bg="#E87C43",
fg="white", command=DecisionTree_Train)
    test1.pack()
    test1.place(x=20, y=15)

    test2 = Button(tt2, width=20, height=2, text="Test",bg = "#E87C43",fg =
"white", command=DecisionTree_Test)
    test2.pack()
    test2.place(x=20, y=70)
# evaluate decision tree performance on train and test sets with different
tree depths
# function to test the data using decision tree and plot the tree graph
def DecisionTree_Test():
    dt = Tk()
    # login window settings
    dt.geometry('500x550')
    dt.title(' Decision Tree Test')
    dt.resizable(0, 0)
    gradienttt(dt)    #color the background of the interface

    # define lists to collect scores to find the fittness of the classifier
    train_scores, test_scores = list(), list()
    # define the tree depths to evaluate
    values = [i for i in range(1, 30)]
    # evaluate a decision tree for each depth
    for i in values:

        # use Decision Tree Classifier
        decision_tree = DecisionTreeClassifier(max_depth=i)
        decision_tree.fit(x_train, y_train)
        predicted_values2 = decision_tree.predict(x_test)
        test_acc = accuracy_score(y_test, predicted_values2)
        test_scores.append(test_acc)

        predicted_values22 = decision_tree.predict(x_train)
        train_acc = accuracy_score(y_train, predicted_values22)
        train_scores.append(train_acc)

        print('>%d, train: %.3f, test: %.3f' % (i, train_acc, test_acc))
    # plot train and test scores vs tree depth
    pyplot.plot(values, train_scores, '-o',color="blue", label='Train')
    pyplot.plot(values, test_scores, '-o',color="black", label='Test')
    pyplot.legend()
    pyplot.show()
    #  the following features will be used in the decision tree
    feature_col = ["followers", "actions", "Tweet"]
    dot_data = StringIO()
    # plot the tree based on the class name (spam or quality)
    export_graphviz(decision_tree, out_file=dot_data,
                    filled=True, rounded=True,
                    special_characters=True, feature_names=feature_col,
class_names=['Spam', 'Quality'])
    graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
    # make the tree as an image in png
    graph.write_png('decision_tree_Test.png')
    Image(graph.create_png())
    # print the accuracy of the test data
    accuracyy = Label(dt, text=f'classifier Accuracy
{str(metrics.accuracy_score(y_test, predicted_values2))}')
    o = ('Andalus', 16)
    accuracyy.config(font=o)
    accuracyy.place(x=40, y=5)

    # print the report of the test data
    report = Label(dt, text='classifier performance Report: ')
    o = ('Andalus', 16)
    report.config(font=o)
    report.place(x=40, y=60)

    metricss = Label(dt, text=f"{metrics.classification_report(y_test,
predicted_values2)}")
    o = ('Andalus', 16)
    metricss.config(font=o)
    metricss.place(x=40, y=95)

    # print the confusion matrix of the test data
    confmat = Label(dt, text="confusion matrix: ")
    o = ('Andalus', 16)
    confmat.config(font=o)
    confmat.place(x=40, y=405)

    conf = Label(dt, text=f"{metrics.confusion_matrix(y_test,
predicted_values2)}")
    o = ('Andalus', 16)
    conf.config(font=o)
    conf.place(x=40, y=440)
    # ------------------------------------------------------------------
 # function to train the data using decision tree
def DecisionTree_Train():
    dtt = Tk()
    # login window settings
    dtt.geometry('500x550')
    dtt.title(' Decision Tree Train')
    dtt.resizable(0, 0)
    gradienttt(dtt)

    # use Decision Tree Classifier
    decision_tree = DecisionTreeClassifier()
    decision_tree.fit(x_train, y_train)
    predicted_values2 = decision_tree.predict(x_train)

    #print the accuracy
    accuracyy = Label(dtt, text=f'classifier Accuracy
{str(metrics.accuracy_score(y_train, predicted_values2))}')
    o = ('Andalus', 16)
    accuracyy.config(font=o)
    accuracyy.place(x=40, y=5)
    #print the report
    report = Label(dtt, text='classifier performance Report: ')
    o = ('Andalus', 16)
    report.config(font=o)
    report.place(x=40, y=60)

    metricss = Label(dtt, text=f"{metrics.classification_report(y_train,
predicted_values2)}")
    o = ('Andalus', 16)
    metricss.config(font=o)
    metricss.place(x=40, y=95)
    #print the confusion matrix
    confmat = Label(dtt, text="confusion matrix: ")
    o = ('Andalus', 16)
    confmat.config(font=o)
    confmat.place(x=40, y=405)

    conf = Label(dtt, text=f"{metrics.confusion_matrix(y_train,
predicted_values2)}")
    o = ('Andalus', 16)
    conf.config(font=o)
    conf.place(x=40, y=440)
    # -------------------------------------------------------------------
# show the interface of the random forest
def test_train_RandomForest():
    tt3 = Tk()
    # login window settings
    tt3.geometry('200x130')
    tt3.title(' Random Forest ')
    tt3.resizable(0, 0)
    gradient(tt3)

    test1 = Button(tt3, width=20, height=2, text="Train", bg="#E87C43",
fg="white", command=RandomForest_Train)
    test1.pack()
    test1.place(x=20, y=15)

    test2 = Button(tt3, width=20, height=2, text="Test",bg = "#E87C43",fg =
"white", command=RandomForest_Test)
    test2.pack()
    test2.place(x=20, y=70)
 # function to test the data using Random forest
def RandomForest_Test():
    rf = Tk()
    # login window settings
    rf.geometry('500x550')
    rf.title(' Random Forest Test ')
    rf.resizable(0, 0)
    gradient(rf)    # color the background of the random forest interface

    # define lists to collect scores
    train_scores, test_scores = list(), list()
    # define the tree depths to evaluate
    values = [i for i in range(1, 30)]
    # evaluate a decision tree for each depth
    for i in values:
        # use Random Forest Classifier
        random_forest = RandomForestClassifier(max_depth=i)
        random_forest.fit(x_train, y_train)
        predicted_values3 = random_forest.predict(x_test)
        test_acc = accuracy_score(y_test, predicted_values3)
        test_scores.append(test_acc)

        predicted_values33 = random_forest.predict(x_train)
        train_acc = accuracy_score(y_train, predicted_values33)
        train_scores.append(train_acc)

        print('>%d, train: %.3f, test: %.3f' % (i, train_acc, test_acc))
        # plot of train and test scores vs tree depth
    pyplot.plot(values, train_scores, '-o', color="orange", label='Train')
    pyplot.plot(values, test_scores, '-o', color="green", label='Test')
    pyplot.legend()
    pyplot.show()
    #print the accuarcy
    accuracyy = Label(rf, bg="#7185A3",text=f'classifier Accuracy
{str(metrics.accuracy_score(y_test, predicted_values3))}')
    o = ('Andalus', 16)
    accuracyy.config(font=o)
    accuracyy.place(x=40, y=5)
    #print the report
    report = Label(rf, bg="#7185A3",text='classifier performance Report: ')
    o = ('Andalus', 16)
    report.config(font=o)
    report.place(x=40, y=60)

    metricss = Label(rf, bg="#7185A3",
text=f"{metrics.classification_report(y_test, predicted_values3)}")
    o = ('Andalus', 16)
    metricss.config(font=o)
    metricss.place(x=40, y=95)
    #print the confusion matrix
    confmat = Label(rf, bg="#7185A3", text="confusion matrix: ")
    o = ('Andalus', 16)
    confmat.config(font=o)
    confmat.place(x=40, y=405)

    conf = Label(rf, bg="#7185A3", text=f"{metrics.confusion_matrix(y_test,
predicted_values3)}")
    o = ('Andalus', 16)
    conf.config(font=o)
    conf.place(x=40, y=440)
    # ----------------------------------------------------------------
 # function to train  the data using Random forest
def RandomForest_Train():
    rft = Tk()
    # login window settings
    rft.geometry('500x550')
    rft.title(' Random Forest Train ')
    rft.resizable(0, 0)
    gradient(rft)

    # use Random Forest Classifier
    random_forest = RandomForestClassifier()
    random_forest.fit(x_train, y_train)
    predicted_values3 = random_forest.predict(x_train)
    #print the accuarcy
    accuracyy = Label(rft, bg="#7185A3",text=f'classifier Accuracy
{str(metrics.accuracy_score(y_train, predicted_values3))}')
    o = ('Andalus', 16)
    accuracyy.config(font=o)
    accuracyy.place(x=40, y=5)
    #print the report
    report = Label(rft, bg="#7185A3", text='classifier performance Report: ')
    o = ('Andalus', 16)
    report.config(font=o)
    report.place(x=40, y=60)

    metricss = Label(rft, bg="#7185A3",
text=f"{metrics.classification_report(y_train, predicted_values3)}")
    o = ('Andalus', 16)
    metricss.config(font=o)
    metricss.place(x=40, y=95)
    #print the confusion matrix
    confmat = Label(rft, bg="#7185A3", text="confusion matrix: ")
    o = ('Andalus', 16)
    confmat.config(font=o)
    confmat.place(x=40, y=405)

    conf = Label(rft, bg="#7185A3", text=f"{metrics.confusion_matrix(y_train,
predicted_values3)}")
    o = ('Andalus', 16)
    conf.config(font=o)
    conf.place(x=40, y=440)
    # ---------------------------------------------------------------
# show the interface of Nueral network
def test_train_NeuralNetwork():
    tt4 = Tk()
    # login window settings
    tt4.geometry('200x130')
    tt4.title(' Neural Network ')
    tt4.resizable(0, 0)
    gradientt(tt4)


    test1 = Button(tt4, width=20, height=2, text="Train", bg="#E87C43",
fg="white", command=NeuralNetwork_Train)
    test1.pack()
    test1.place(x=20, y=15)

    test2 = Button(tt4, width=20, height=2, text="Test",bg = "#E87C43",fg =
"white", command=NeuralNetwork_Test)
    test2.pack()
    test2.place(x=20, y=70)

 # function to train  the data using Nueral network
def NeuralNetwork_Train():
    nn = Tk()
    # login window settings
    nn.geometry('500x550')
    nn.title(' Neural Network Train ')
    nn.resizable(0, 0)
    gradientt(nn)

    # use nural network classifier
    nural_network = sklearn.neural_network.MLPClassifier()
    # Train the model on the whole data set
    nural_network.fit(x_train, y_train)
    # Evaluate on training data
    predicted_values4 = nural_network.predict(x_train)

    accuracyy = Label(nn,text=f'classifier Accuracy
{sklearn.metrics.accuracy_score(y_train, predicted_values4)}')
    o = ('Andalus', 16)
    accuracyy.config(font=o)
    accuracyy.place(x=40, y=5)

    report = Label(nn, text='classifier performance Report: ')
    o = ('Andalus', 16)
    report.config(font=o)
    report.place(x=40, y=60)

    metricss = Label(nn,
text=f"{sklearn.metrics.classification_report(y_train, predicted_values4)}")
    o = ('Andalus', 16)
    metricss.config(font=o)
    metricss.place(x=40, y=95)

    confmat = Label(nn, text="confusion matrix: ")
    o = ('Andalus', 16)
    confmat.config(font=o)
    confmat.place(x=40, y=405)

    conf = Label(nn, text=f"{sklearn.metrics.confusion_matrix(y_train,
predicted_values4)}")
    o = ('Andalus', 16)
    conf.config(font=o)
    conf.place(x=40, y=440)

 # function to test  the data using nueral network
def NeuralNetwork_Test():
    nn = Tk()
    # login window settings
    nn.geometry('500x550')
    nn.title(' Neural Network Test ')
    nn.resizable(0, 0)
    gradientt(nn)

    # use nural network classifier
    nural_network = sklearn.neural_network.MLPClassifier()
    # Train the model on the whole data set
    nural_network.fit(x_train, y_train)
    # Evaluate on training data
    predicted_values4 = nural_network.predict(x_test)
# print the accuarcy
    accuracyy = Label(nn,text=f'classifier Accuracy
{sklearn.metrics.accuracy_score(y_test, predicted_values4)}')
    o = ('Andalus', 16)
    accuracyy.config(font=o)
    accuracyy.place(x=40, y=5)

    report = Label(nn, text='classifier performance Report: ')
    o = ('Andalus', 16)
    report.config(font=o)
    report.place(x=40, y=60)

    # print the report
    metricss = Label(nn,
text=f"{sklearn.metrics.classification_report(y_test, predicted_values4)}")
    o = ('Andalus', 16)
    metricss.config(font=o)
    metricss.place(x=40, y=95)

    # print the confusion matrix
    confmat = Label(nn, text="confusion matrix: ")
    o = ('Andalus', 16)
    confmat.config(font=o)
    confmat.place(x=40, y=405)

    conf = Label(nn, text=f"{sklearn.metrics.confusion_matrix(y_test,
predicted_values4)}")
    o = ('Andalus', 16)
    conf.config(font=o)
    conf.place(x=40, y=440)
# function to color the background
def gradient(x):
    # Making gradient frame
    j = 0
    r = 10
    for i in range(100):
        c = str(333333 + r)
        Frame(x, width=15, height=550, bg="#" + c).place(x=j, y=0)
        j = j + 10
        r = r + 1

gradient(welcome)  # set the gradient color for the login window

# function to color the background in another colors
def gradientt(t):
    # Making gradient frame
    j = 0
    r = 10
    for i in range(100):
        c = str(423512 + r)
        Frame(t, width=15, height=550, bg="#" + c).place(x=j, y=0)
        j = j + 10
        r = r + 1
# function to color the background in another colors
def gradienttt(t):
    # Making gradient frame
    j = 0
    r = 10
    for i in range(100):
        c = str(223522 + r)
        Frame(t, width=15, height=550, bg="#" + c).place(x=j, y=0)
        j = j + 10
        r = r + 1

# function to color the background in another colors
def gradienttttt(t):
    # Making gradient frame
    j = 0
    r = 10
    for i in range(100):
        c = str(753522 + r)
        Frame(t, width=15, height=550, bg="#" + c).place(x=j, y=0)
        j = j + 10
        r = r + 1


note = Label(welcome,text='                        Fatima  Jihan  Hedaya
')
o = ('French Script MT', 14)
note.config(font=o)
note.place(x=2, y=400)

note1 = Label(welcome,bg="#7185A3",text='Tweet Spam Detection')
o = ('Andalus', 16)
note1.config(font=o)
note1.place(x=90, y=1)

Frame(welcome, width=250,bg = "#7185A3",height=320).place(x=70, y=60)  # set
a white background abov the grdient one

button1 = Button(welcome, width=20, height=2, text="Naive Bayes",bg =
"#E87C43",fg = "white", command=test_train_naiveBayes)
button1.pack()
button1.place(x=120, y=80)

button2 = Button(welcome, width=20, height=2, text="Decision Tree",bg =
"#E87C43",fg = "white", command=test_train_DecisionTree)
button2.pack()
button2.place(x=120, y=150)

button3 = Button(welcome, width=20, height=2, text="Random Forest",bg =
"#E87C43",fg = "white", command=test_train_RandomForest)
button3.pack()
button3.place(x=120, y=230)

button4 = Button(welcome, width=20, height=2, text="Neural Network",bg =
"#E87C43",fg = "white", command=test_train_NeuralNetwork)
button4.pack()
button4.place(x=120, y=310)

welcome.mainloop()
```