FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

ARTIFICIAL INTELLIGENCE

ENCS3340

Project Report

Prepared by:

#Raneen Mahmoud #1181375

#Nemat Mimi #1181766

Section:

1

Instructor:

Dr. Aziz Qaroush

19/02/2022

First semester 2021\2022

Introduction

It is a common practice for people to read and submit tweets for a variety of reasons. One of the most obvious reasons of tweets is to express one's viewpoint and convey ideas; moreover, it helps to connect with others who have similar interests; and, without a doubt, one of its most essential goals is to promote a certain topic and remark on various occurrences.

Spamming tweets this term refers to "illegal" activities that attempt to mislead readers or automated opinion mining and sentiment analysis systems by providing undeserving positive opinions for some events, people, or entities in order to promote them and/or false negative opinions for other events, people, or entities in order to harm their reputations. Spam tweets can take numerous forms, such as tweets with bogus hashtags, tweets with vulgar or racist phrases or meanings, tweets with URL links, and so on.

We believe that tweets on the Web are utilized for decision-making by organizations and enterprises, the influence of tweets on individuals are enormous too. Spamming tweets will become more common, as well as more complex. Detecting spam tweets or opinions will become increasingly difficult and difficult.

Table of Contents

Table of Figures

Theory

Twitter is a well-known social media website that provides a social network of message-publishing users. To keep users aware of the most popular topics on Twitter, Twitter provides a list of the most popular themes at any given time called "Trending Topics. To keep Twitter spam-free, it is vital to identify and separate spammers from real users. This popularity attracts spammers who use Twitter for malicious purposes such as phishing legitimate users or spreading malicious software and advertisements via URLs shared in tweets; aggressively following and unfollowing legitimate users; hijacking trending topics to attract their attention; and spreading pornography.

Twitter has 313 million monthly active users who tweet 500 million times every day, or 350,000 tweets per minute. spammers use links in their tweets to: spread advertising to generate sales; propagate pornography; share malicious links that direct users to malicious software; hijack trending topics for their own purposes; post unsolicited messages to legitimate users to attract their attention; and phish legitimate users.

How Twitter Deals with Spam

In order to create a spam-free environment, Twitter employs both manual and automatic services to combat with spammers. The manual method is that Twitter allows users to report spammers via the spammers' profile pages. These manual procedures are time-consuming and would not be sufficient to catch all spammers given the billions of users.

By following the observation of spammers' actions within the area of account-based features, tweet-based features, and graph-based features, the following facts are observed:

- → Since spammers tend to follow too many legitimate accounts in order to attract attention, the number of followers is expected to be high compared to legitimate users.
- → Spammers' tweets mostly contain links and hashtags to attract the attention of legitimate users.
- → Spammers tend to use links to direct legitimate users to their malicious purposes.
- → Since spammers' tweets are unsolicited, the number of likes and retweets their tweets have received is much lower compared to legitimate users.

→ The connectivity between a spammer and a legitimate user is more robust than the connectivity between two legitimate users.

→ The distance between a spammer and a legitimate user is greater than the distance between two legitimate users.

→ Graph-based features provide the most robust performance to detect spam and spammers since they are hard to manipulate and not user-controlled.

Many efforts have been made to develop spam detection techniques on Twitter in the last decade. In this section, we explain the state of the art in three categories: syntax analysis, feature analysis, and blacklist techniques.

Detection based on Syntax Analysis:

Syntax-based detection methods analyze tweets at character or word level.

There are some works focusing on inspecting shortened URLs inside tweets. Shortened URLs can be generated by shortener services by inputting a long URL, which are used by spammers to hide their malicious URLs.

Lee and Kim have proposed an innovative Twitter malicious URL detecting system according to the relevance of suspicious URLs, though their method could not be used for dynamic redirections. Developed a dataset of click traffic feature to determine shortened URLs to be spam or not spam.

Moreover, there are some works considering tweet content or text.

Extracted tweet content for the input of classifier by employing a deep learning neural network model to learn syntactic contexts of embedded words and label information.

Detection based on Feature Analysis:

In this field, there are also many methods that select account and/or tweet features as training data for the input of machine learning based classifiers. Account features can be the age of user account, the number of followings, and the number of followers. Tweet features include the ratio of tweets which contain URLs, the average number of hashtags in a tweet and etc. Of course, there are many methods to select the best features to detect data. But, with this kind of detection there is a problem

where Twitter data features can be easily extracted with the support of statistical methods, it is difficult to avoid feature fabrication in the data collection processes.

Blacklist Techniques:

Blacklist techniques are commonly deployed in web filtering services such as Twitter spam detection, with the functionality of blocking malicious websites according to their information analysis like user feedback and website crawling. However, this method has to rely on manual labelling which is too time-consuming.

In a nutshell, current spam detection methods on Twitter are still not sufficient to detect spamming activities quickly and accurately in terms of Recall, Precision and F1-measure.

Now, I'd like to describe a revolutionary framework of Tweet spam detection approaches.

Detection Framework:

A. Data collecting:

There are 1,600,000 tweets in the Stanford Twitter sentiment corpus collection. We extracted all 36 characteristics by evaluating the content of tweet IDs, and We retrieved all 36 features by analyzing the content of tweet IDs. The data set algorithm assists social media platforms in detecting spammers before they create havoc.

B. Pre-processing

Data pre-processing can be categorized into data cleaning, data integration, data reduction. NLP purpose is to recognize, read, and analyze analysis to interpret human languages.

Data pre-processing Steps:

→ Original Data
→ Tokenization
→ Stop words removal
→ Stemming
→ POS Tagging

C. Features scaling

Data preprocessing is divided into four categories: data cleansing, data integration, data reduction, and data. The natural language processing pipeline's Java JDK, OpenNLP, and Ling-Pipe libraries were used. NLP's goal is to recognize, read, and analyze data in order to interpret human languages. Here normalization and standardization were applied to make the dataset suitable for further processing. However, extracting features requires in-depth analysis on the huge and complex Twitter graph which is time and resource intensive.

And that’s done by do:

1) Normalization

The technique's goal is to convert raw feature data into positive or negative values. Data from transformed tweets is classified into distinct classes based on the existence of a specific phrase.

2) Standardization

Standardization is a form of unification procedure that takes into account raw qualities as well as value scattering; for example, we use the mean value and standard deviation of a particular tweet feature.

D. Feature extraction

Feature extraction is a difficult concept that involves the transformation of raw data into the inputs required by a Machine Learning system. The model is the motor, but it need fuel to function. The information in the data must be represented by features in a format that best fits the needs of the algorithm that will solve the problem.

E. Classification

— Proposed Algorithm

The Random Forest is an ensemble learning and regression method appropriate for classification problems. It provides an efficient mechanism for calculating the approximate value of missing information. Overfitting is a critical problem that can worsen the results. The Random Forest algorithm generates enough trees to help the classifier not over-fit the model.

— Classification using Random Forest Algorithm

F. Existing system algorithm.

At the end using machine learning methods, a novel methodology for evaluating Twitter spam data was used. and the Experiments show that our unique framework effectively identifies the best feature subset while minimizing bigger dataset size errors. and the studies intend to increase the performance measurements by incorporating new features that stream spam tweets.

Improvements:

There are many methods to get results with higher accuracy and precision. One of them is called stacking classifier.

Stacking or Stacked Generalization is an ensemble machine learning algorithm. It uses a meta-learning algorithm to learn how to best combine the predictions from two or more base machine learning algorithms.

The benefit of stacking is that it can harness the capabilities of a range of well-performing models on a classification or regression task and make predictions that have better performance than any single model in the ensemble.

Problem formalization

In this project, we were provided a raw dataset and requested to train a Tweet spam detection system, as well as test the system with new "never seen before" data to determine the system's efficiency.

At the begin, we check to see if our dataset was balanced; the total number of samples was 11952, with 6137 for Quality and 5815 for spam, as represented graphically in the figure below, thus we conclude that the dataset was almost balanced and appropriate.

```
import matplotlib.pyplot as plt
plt.pie(dataset['Type'].value_counts(), labels=['Quality','Spam'],autopct="%0.2f")
plt.show()
```

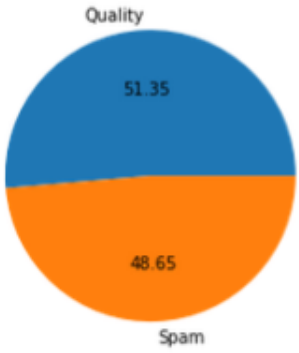

Figure 1 Spam to Quality percentage

We began the construction of our system by selecting the features we believed would be most useful in building an effective spam detection system. Reading and looking for publications about the Automatic Tweet Spam Detection system led to the selection of features. We were able to select the following features:

1-Tweets.

2- Is retweet.

First

The column "Tweet" in the input dataset was chosen as one of the primary structures of a spam tweet. Spammers have a language psychology in which they utilize extremely often used terms such as words with obscene or racist phrases or meanings, and they also have a tendency to

generalize their tweets using unrelated words, or to enhance the length of the tweets, they may repeat some words or characters. The tweet contact was first analyzed, and then we employed stop words, which are highly common terms in the language that do not add up to the value of the tweet. We then utilized heated words to change everything to lower case, and we had our first feature.

Then we run our system and display the results (accuracy and precision) for each model.

The Result for one Future:

```
accuracy_scores = []
precision_scores = []
for name,model in models.items():
    current_accuracy,current_precision = train_classifier(model, X_train,y_train,X_test,y_test)
    print("For ",name)
    print("Accuracy - ",current_accuracy)
    print("Precision - ",current_precision)
    accuracy_scores.append(current_accuracy)
    precision_scores.append(current_precision)
```

```
For  Naive Baise
Accuracy -  0.8380187416331994
Precision -  0.879416282642089
For  Decision Tress
Accuracy -  0.786479250334672
Precision -  0.7995689655172413
For  Gradient Boosting Decision Tree
Accuracy -  0.7764390896921017
Precision -  0.8551236749116607
For  Nuaral Network
Accuracy -  0.7710843373493976
Precision -  0.7717241379310344
```

Figure 2 Accuracy and precision using tweet content only

As shown in figure above, the findings reveal that the outcomes are adequate and decent, but we resorted to thinking about how we may increase our accuracy and precision, which was by adding another feature.

Second

We began looking for additional features that might assist us enhance our system. We tried using following, followers, and action, and it gave us good accuracy and precision, but when we tried to test our system with new data, it always resulted in Spam, which means that our system was overfitting, which means that the model picked up on noise or random fluctuations in the training data and learned them as concepts. The issue is that these notions do not apply to fresh data, limiting the models' capacity to generalize.

When we look at location as a feature, we notice that it is unreasonable to pick it because the classification of tweets based on location is unconnected. It is nonsensical to categorize tweets based on their location.

We discovered that the best one in terms of outcomes and logicality was (is retweet) which means re-posting of a Tweet, since it generated an improvement in accuracy and precision without overfitting, after several efforts and attempting all the accessible and logical features.

The Result:

```
accuracy_scores = []
precision_scores = []
for name,model in models.items():
    current_accuracy,current_precision = train_classifier(model, X_train,y_train,X_test,y_test)
    print("For ",name)
    print("Accuracy - ",current_accuracy)
    print("Precision - ",current_precision)
    accuracy_scores.append(current_accuracy)
    precision_scores.append(current_precision)
```

```
For  Naive Baise
Accuracy -  0.8728246318607764
Precision -  0.9020618556701031
For  Decision Tress
Accuracy -  0.8410307898259706
Precision -  0.8532955350815025
For  Gradient Boosting Decision Tree
Accuracy -  0.8360107095046854
Precision -  0.8472418670438473
For  Nuaral Network
Accuracy -  0.8319946452476573
Precision -  0.8335625859697386
```

Figure 3 Accuracy and precision using tweet content and is retweet features

Proposed work

Our project was started by reading the data file (train.csv) and that's done by using pd. read_csv () as shown in figure below.

```
# Reading data from the CSV file
dataset = pd.read_csv('/content/train.csv')
dataset.head()
```

Figure 4 reading csv file

After that the data was arranged and that's because of nonvalue data so that's solved by filling the missing data by the most appropriate way, there were different ways of center in a numerical data set (Mean, median, and mode):

Mean: The "average" number; found by adding all data points and dividing by the number of data points.

Median: The middle number; found by ordering all data points and picking out the one in the middle (or if there are two middle numbers, taking the mean of those two numbers).

Mode: The most frequent number—that is, the number that occurs the highest number of times.

In our project median was used to fill the missing data in following and actions future, for followers the missing data was dropped This was due to their small number of total data, and for location the empty cells were filled with the previous value, in addition to that's duplicated data was checked and there was one duplicated data and it was removed. And all of that was done as following.

```
[5]  # remove duplicates
     dataset = dataset.drop_duplicates(keep='first')

[6]  #number of missing values in each parameters
     dataset.isnull().sum()

     Id               0
     Tweet            0
     following      144
     followers       15
     actions       2772
     is_retweet       0
     location         0
     Type             0
     dtype: int64

[7]  dataset['following'].fillna(dataset['following'].median(),inplace=True)

[8]  dataset['actions'].fillna(dataset['actions'].median(),inplace=True)

[9]  dataset = dataset.dropna(how='any')

[10] dataset.isnull().sum()

     Id            0
     Tweet         0
     following     0
     followers     0
     actions       0
     is_retweet    0
     location      0
     Type          0
     dtype: int64
```

Figure 5 Removing duplicates and fill missing values

After that, to study tweets in a clear and comprehensive way, spam and Quality words were found, this was in order to clarify the words that led to each of the previous classifications (spam and Quality).

```
spam_words= ' '.join(list(dataset[dataset['Type']=='Spam']['Tweet']))
spam_wc= WordCloud(width=3000,height=3000).generate(spam_words)
plt.figure(figsize=(10,10))
plt.imshow(spam_wc)
plt.show()
```

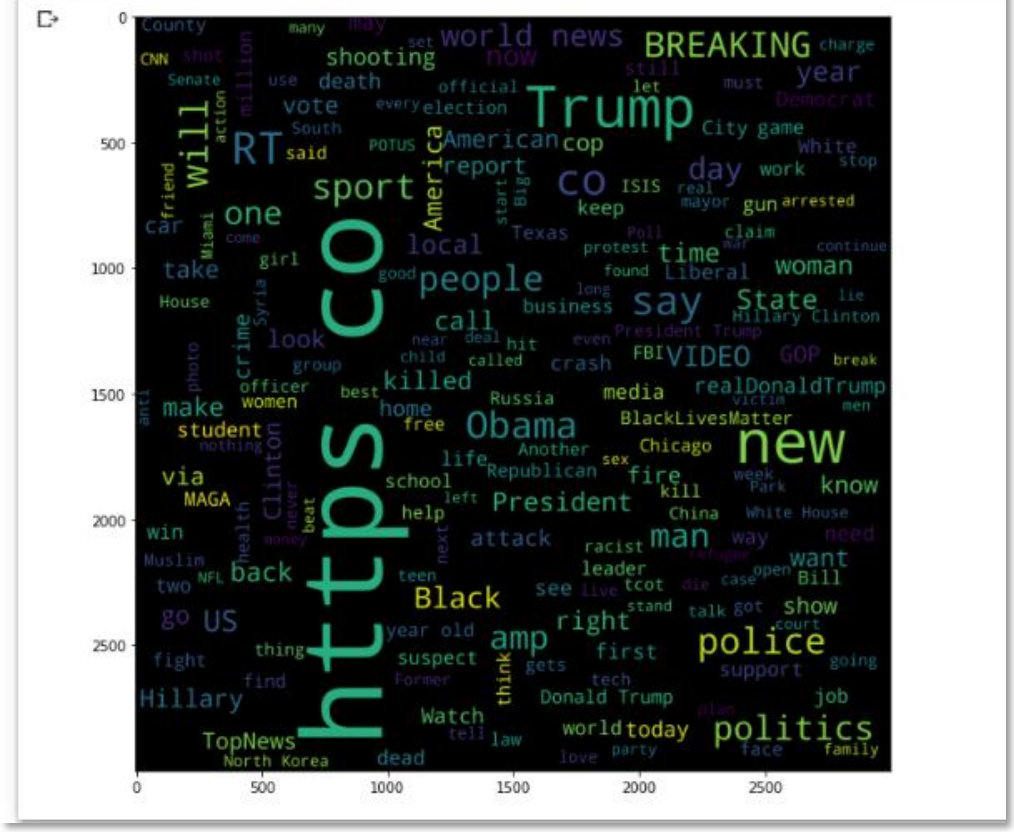

Figure 6 Spam words

As shown in the above figure the words that led to the classification of tweets into spam were found. in the graph, when the word size increase that's means that the word has a greater impact in this classification like (https, police…. etc.)

This interpretation also applies to Quality classification.

Before staring the different models there was a need to preprocess the data so it becomes understandable to the system. First, all the letters of the tweets were converted into lower case so that the same letters would be treated the same. Second, the most used words such as (a, an, the…etc.) were removed using the word tokenize function from the nltk library because we don't

want this kind of words to be classified as spam or not. Finally, the stem function was used so that all the verbs would be converted into the inventive form, for example (going → go) and so on.

```
def transform_tweet(tweet):
  #convert the tweet to lower case
    tweet = tweet.lower()
  #remove the most used word(a,an,the etc...)
    tweet = nltk.word_tokenize(tweet)
    stop_words = True
    words = []
    for i in tweet:
      # returns True if all characters in the string are alphanumeric (either alphabets or numbers)
        if i.isalnum():
            words.append(i)
    tweet = words[:]
    words.clear()
    for i in tweet:
      #check the char is in english
        if i not in stopwords.words('english'):
            words.append(i)
    tweet = words[:]
    words.clear()
    for i in tweet:
      #return the word to its infinitive
        words.append(ps.stem(i))
    return " ".join(words)
```

Figure 7 Preprocessing the data

Then, the frequency of each word was calculated using the count vectorizer function so that the spam words can be known. And, the full data was divided into 25% test and 75% train where the tweet and is retweet features were taken as inputs and the type feature as the output or the class.

```
    cv = CountVectorizer()
    tfidf = TfidfVectorizer(max_features=4002)
    X = tfidf.fit_transform(dataset['Transformed_tweet']).toarray()
    X.shape
    z= np.column_stack((X,dataset[["is_retweet"]].values))
    y = dataset['Type'].values
    #divide data with 25% test and 75 train
    X_train,X_test,y_train,y_test = train_test_split(z,y,test_size=0.25,random_state=2)
```

Figure 8 Calculating the frequency

After these stages, there was a need to find the perfect model to classify the data. We started with the naïve bayes algorithm which is dived into three types: mnb, bnb and gnb. We called these algorithms from their libraries, insert the data and calculate the accuracy for each type as it is shown below in the figures.

```
[21] #Types of Naive Baise algorithms
     gnb = GaussianNB()
     mnb = MultinomialNB()
     bnb = BernoulliNB()
     #takes the training data as arguments
     gnb.fit(X_train,y_train)
     #to predict the Types of the data values
     y_predict1 = gnb.predict(X_test)
     #print accuracy and matrix and the percision
     print(accuracy_score(y_test,y_predict1))
     print(confusion_matrix(y_test,y_predict1))
     print(precision_score(y_test,y_predict1))

     0.7516733601070951
     [[1297  219]
      [ 523  949]]
     0.8125
```

Figure 9 GNB accuracy and precision

```
[22] #takes the training data as arguments
     mnb.fit(X_train,y_train)
     #to predict the Types of the data values
     y_predict2 = mnb.predict(X_test)
     #print accuracy and matrix and the percision
     print(accuracy_score(y_test,y_predict2))
     print(confusion_matrix(y_test,y_predict2))
     print(precision_score(y_test,y_predict2))

     0.8681392235609103
     [[1297  219]
      [ 175 1297]]
     0.8555408970976254
```

Figure 10 MNB accuracy and precision

```
[23] #takes the training data as arguments
     bnb.fit(X_train,y_train)
     #to predict the Types of the data values
     y_predict3 = bnb.predict(X_test)
     #print accuracy and matrix and the percision
     print(accuracy_score(y_test,y_predict3))
     print(confusion_matrix(y_test,y_predict3))
     print(precision_score(y_test,y_predict3))

     0.8728246318607764
     [[1383  133]
      [ 247 1225]]
     0.9020618556701031
```

Figure 11 BNB accuracy and precision

It can be noticed that, bnb type got the highest accuracy and precision since the feature vectors in this classifier represent the frequencies with which certain events have been generated by a multinomial distribution.

In the same way other models were defined as following to check their accuracy on the same data.

```
[25] #we choose three algorithms (bnb(Naive Bais), dicition tree,Gradient Boosting Decision Tree,Nuaral Networks)
     from sklearn.neural_network import MLPClassifier
     nn = MLPClassifier(solver='adam', alpha=0.001,hidden_layer_sizes=(7, 100), random_state=1)
     bnb =  BernoulliNB()
     dtc = DecisionTreeClassifier(max_depth=8963)
     gbdt = GradientBoostingClassifier(n_estimators=100,random_state=5)

[26] #the models
     models = {
         'Naive Baise': bnb,
         'Decision Tress': dtc,
         'Gradient Boosting Decision Tree':gbdt,
         'Nuaral Network':nn,
     }
```

Figure 12 Defining different models

```
#calculate the accuracy and percision for each model
def train_classifier(model,X_train,y_train,X_test,y_test):
    model.fit(X_train,y_train)
    y_predict = model.predict(X_test)
    accuracy = accuracy_score(y_test,y_predict)
    precision = precision_score(y_test,y_predict)
    return accuracy,precision
```

Figure 13 Calculating accuracy and precision function

The pervious function was used for computing the accuracy and precision for all the models employed in this system, with the goal of selecting the most accurate model, which would result in an accurate and precise system.

The above function was simulated and the following results were got.

	Algorithm	Accuracy	Precision
0	Naive Baise	0.872825	0.902062
1	Decision Tress	0.836345	0.849822
2	Gradient Boosting Decision Tree	0.836011	0.847242
3	Nuaral Network	0.831995	0.833563

Figure 14 Results for different classifiers

As it can be seen from the above table Naïve Bayes got the first place on accuracy and precision since it is a generative model that focuses on the distribution of the dataset where Decision trees and Neural network are discriminative model that makes predictions based on conditional probability. Generally, generative models were effective in this project since it has a large data set. Not that difference is between Decision trees and neural network since they are the same type of model but as it can be noticed from the previous results decision trees are a little bit better than neural network because there was a big set of categorical values in the training data.

Moreover, Naïve Baise was the fastest algorithm while studying and checking the data where decision trees and neural network took a lot of time since they have many levels or layers to cross.

```
# Applying stacking
estimators=[('dtc', dtc), ('bnb', bnb), ('gbdt', gbdt)]
final_estimator= RandomForestClassifier()
clf = StackingClassifier(estimators=estimators, final_estimator=final_estimator)
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
print("Accuracy",accuracy_score(y_test,y_pred))
print("Precision",precision_score(y_test,y_pred))
```

```
Accuracy 0.8801874163319946
Precision 0.878396739130434 8
```

Figure 15 Applying stacking classifier

As an improvement step, stacking classifier was used. This classifier lets us use the predictions of multiple classifiers (level- one classifiers) as new features to learn from while training the meta-classifier.

As a final step, vectorizer and model files were downloaded, so our program can be tested using a viewable interface. Notice that, bnb naïve baise was chosen as a model since it has the highest accuracy and precision as it was explained previously and TFIDF (the product of the term frequency and the inverse of tweets with term frequency) was used as a vectorizer because that can give us a measure of how frequent the word is in a tweet multiplied by how unique the word is.

```
[ ]  pickle.dump(tfidf,open('vectorizer.pkl','wb'))
     pickle.dump(bnb,open('model.pkl','wb'))
```

Figure 16 Downloading Model and Vectorizer files

Testing:

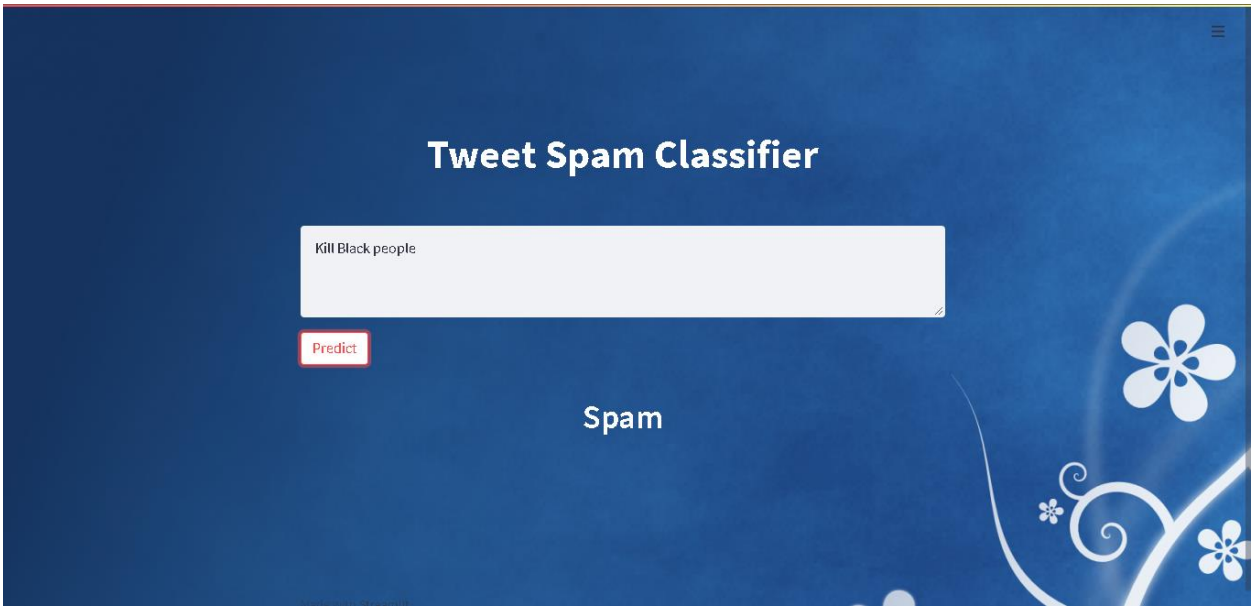

Figure 17 Testing 1 (spam)

As it can be seen, this tweet was classified as Spam since it has spam words such as ‘kill ‘and ‘black’. And this kind of words were defined as spam ones on the training data that the system studied.

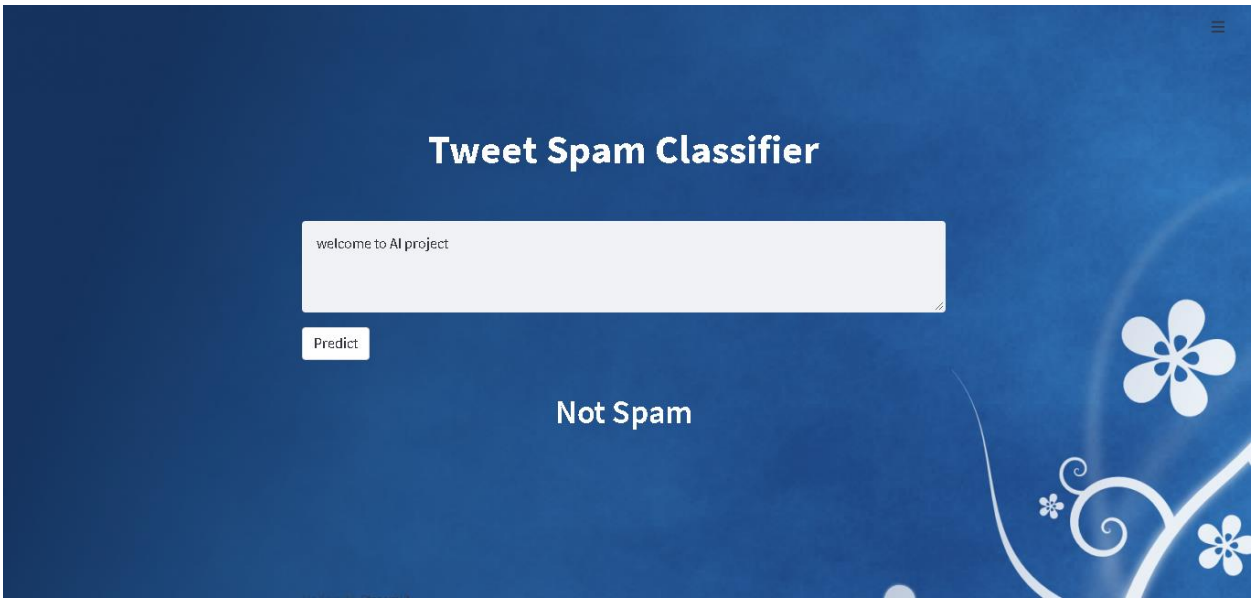

Figure 18 Testing 2 (Not Spam)

Since this tweet has not any kind of spam words it was classified as Not Spam.

Conclusion

By implementing this project many goals were achieved such as:

- Getting familiar with different classifiers such as Naïve bayes, Decision trees and Neural network.
- Learning how to calculate accuracy and precision when using a specific classifier.

It was found that generative models such as Naïve bayes works better with large data set. And Decision trees are better than neural network since they got higher accuracy and precision. Moreover, Naïve bayes classifiers were the fastest while processing the data.

A problem was faced at the beginning of this project because of the missing values in different features. This struggle was solved by filling these gaps with the median values but this method somehow reduced the accuracy of the system.

At the end, absolutely there is no perfect detection system since there is some problems on the training data like spam data that is clarified as not.

References:

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

  Accessed on 10/01/2022 at 3:00 pm

- 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.

  Accessed on 10/01/2022 at 3:30 pm

- Tingmin Wu, Shigang Liu, Jun Zhang and Yang Xiang School of Information Technology Deakin University 221 Burwood Hwy,Burwood, VIC 3125, Australia

  Accessed on 10/01/2022 at 4:00 pm