ARTIFICIAL INTELLIGENCE
ENCS3340

Project #2
Tweet Spam Detection

Prepared by:

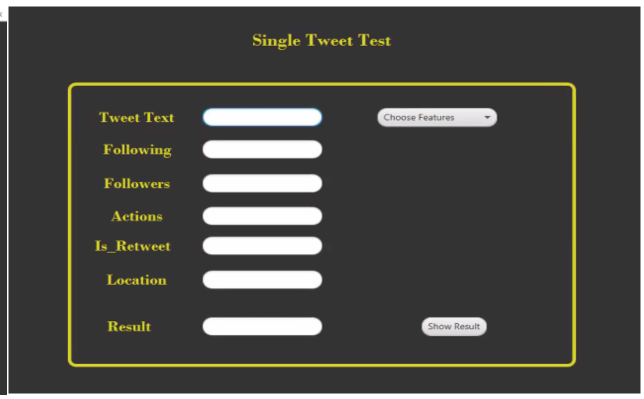
Qutaiba Olayyan 1190760
Ibrahem Duhaidi 1190283

Instructor: Dr.Aziz Qaroush

Section: 2

Abstract

The aims of this project to learn the machine learning libraires in python and use it for detect if the tweet is spam or ham, by building a classifier to detect when a tweet is "Quality" content or "Spam", spam is defined as the tweets that are posted by known fake twitter accounts that are politically motivated, automatically generated content, meaningless content or click bait.



Contents

Figures

Theory

Objective:
The objective is to understand, configure and test some of the machine learning algorithms, by using python libraires for detect if the tweet is spam or quality.
We solved this problem by use Python code, Machine Learning Libraires and JavaFx for show the results.

Introduction:
We built program with different options, firstly you can show the information gain for many features that you selected, secondly show the result of accuracy, precision, recall and f1 scores for test data, thirdly the program can request any new samples Data as doctor file format and give result of it, forth you can request a file that contain just tweets and its classifiers and give result of it by use just text features, finally you can write a single tweet and detect it if spam or quality.

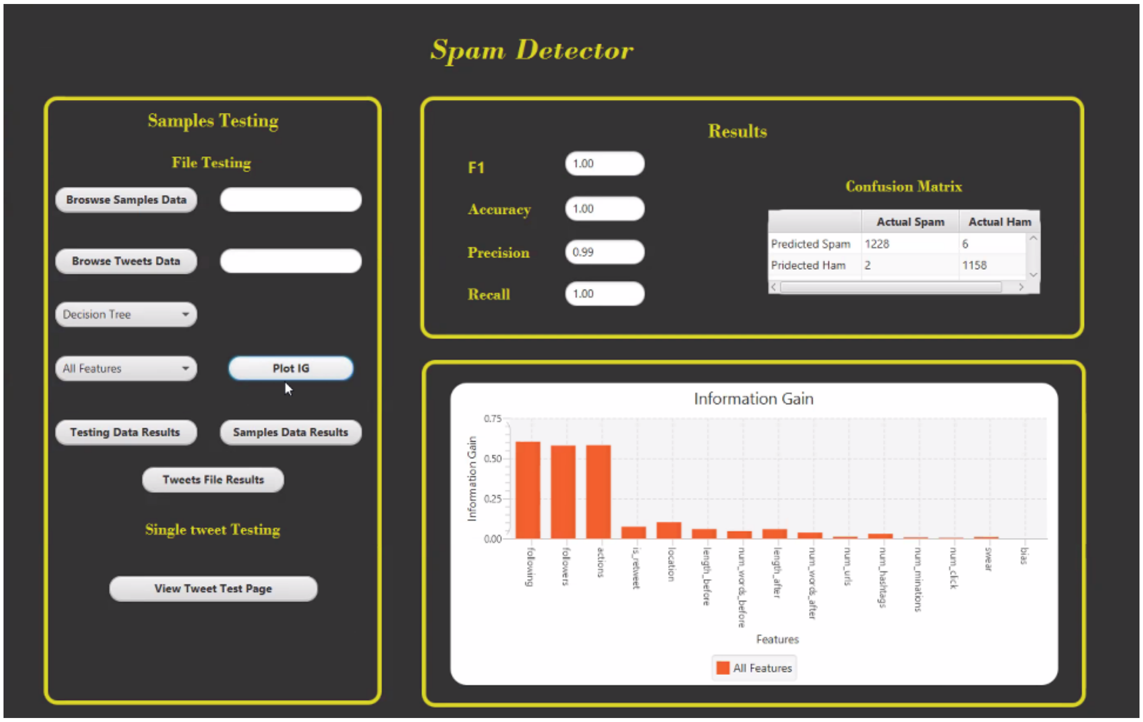


Figure 1: Spam Detector Result

Training Data Form

There is a given Tweet which posted by a user, the following and followers of that user account, number of actions on that tweet, value for detect if that tweet posted more than once and location where the source of that tweet when posted, with all of this information we have Type which detected if that tweet is spam or quality.

Id	Tweet	following	followers	actions	is_retweet	location	Type
10091	It's the eve	0	11500		0	Chicago	Quality
10172	Eren sent a	0	0		0		Quality
7012	I posted a	0	0		0	Scotland, U	Quality
3697	#jan Idiot (	3319	611	294	0	Atlanta, Ga	Spam
10740	Pedophile	4840	1724	1522	0	Blumberg	Spam
9572	EBMUD er	4435	16041	27750	0	UPS	Spam
10792	Big day. #'	0	0	0	0	Toronto, C	Quality
11594	#UPA	0	193000		0	Mumbai	Quality
12594	**MISSIN	39000	46900	47	0	UK	Quality

Figure 2: Data Form

Information Gain

We can define information gain as a measure of how much information a feature provides about a class. Information gain helps to determine the order of attributes in the nodes of a decision tree.

The main node is referred to as the parent node, whereas sub-nodes are known as child nodes. We can use information gain to determine how good the splitting of nodes in a decision tree.

It can help us determine the quality of splitting, as we shall soon see. The calculation of information gain should help us understand this concept better.

$$Gain = E_{parent} - E_{children}$$

The term Gain represents information gain. E_{parent} is the entropy of the parent node and $E_{children}$ is the average entropy of the child nodes.

Accuracy

Accuracy is one metric for evaluating classification models. Informally, accuracy is the fraction of predictions our model got right. Formally, accuracy has the following definition:

Accuracy = Number of correct predictions / Total number of predictions

For binary classification, accuracy can also be calculated in terms of positives and negatives as follows:

Accuracy = (TP+TN) / (TP+TN+FP+FN)

Where TP = True Positives, TN = True Negatives, FP = False Positives, and FN = False Negatives.

Precision

In the simplest terms, Precision is the ratio between the True Positives and all the Positives. For our problem statement, that would be the measure of patients that we correctly identify having a heart disease out of all the patients actually having it. Mathematically:

$$\text{Precision} = \frac{TP}{TP + FP}$$

Recall

The recall is the measure of our model correctly identifying True Positives. Thus, for all the patients who actually have heart disease, recall tells us how many we correctly identified as having a heart disease. Mathematically:

$$\text{Recall} = \frac{TP}{TP + FN}$$

F1

Understanding Accuracy made us realize, we need a tradeoff between Precision and Recall. We first need to decide which is more important for our classification problem.

For example, for our dataset, we can consider that achieving a high recall is more important than getting a high precision – we would like to detect as many heart patients as possible. For some other models, like classifying whether a bank customer is a loan defaulter or not, it is desirable to have a high precision since the bank wouldn't want to lose customers who were denied a loan based on the model's prediction that they would be defaulters.

There are also a lot of situations where both precision and recall are equally important. For example, for our model, if the doctor informs us that the patients who were incorrectly classified as suffering from heart disease are equally important since they could be indicative of some other ailment, then we would aim for not only a high recall but a high precision as well.

In such cases, we use something called F1-score. F1-score is the Harmonic mean of the Precision and Recall:

$$F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

This is easier to work with since now, instead of balancing precision and recall, we can just aim for a good F1-score and that would be indicative of a good Precision and a good Recall value as well.

Confusion Matrix

A Confusion matrix is an N x N matrix used for evaluating the performance of a classification model, where N is the number of target classes. The matrix compares the actual target values with those predicted by the machine learning model. This gives us a holistic view of how well our classification model is performing and what kinds of errors it is making.

For a binary classification problem, we would have a 2 x 2 matrix as shown below with 4 values:

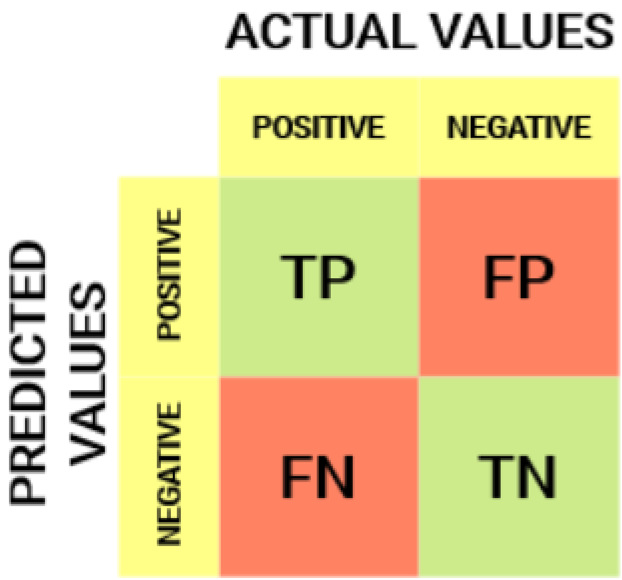


Figure 3: Confusion Matrix

Let's decipher the matrix:

- The target variable has two values: Positive or Negative
- The columns represent the actual values of the target variable
- The rows represent the predicted values of the target variable

True Positive (TP)

- The predicted value matches the actual value
- The actual value was positive and the model predicted a positive value

True Negative (TN)

- The predicted value matches the actual value
- The actual value was negative and the model predicted a negative value

False Positive (FP) – Type 1 error

- The predicted value was falsely predicted
- The actual value was negative but the model predicted a positive value
- Also known as the Type 1 error

False Negative (FN) – Type 2 error

- The predicted value was falsely predicted
- The actual value was positive but the model predicted a negative value
- Also known as the Type 2 error

Decision Tree Algorithm Technique[1]

Decision tree learning is a method commonly used in data mining. The goal is to create a model that predicts the value of a target variable based on several input variables.

A decision tree is a simple representation for classifying examples. For this section, assume that all of the input features have finite discrete domains, and there is a single target feature called the "classification". Each element of the domain of the classification is called a class. A decision tree or a classification tree is a tree in which each internal (non-leaf) node is labeled with an input feature. The arcs coming from a node labeled with an input feature are labeled with each of the possible values of the target feature or the arc leads to a subordinate decision node on a different input feature. Each leaf of the tree is labeled with a class or a probability distribution over the classes, signifying that the data set has been classified by the tree into either a specific class, or into a particular probability distribution (which, if the decision tree is well-constructed, is skewed towards certain subsets of classes).

A tree is built by splitting the source set, constituting the root node of the tree, into subsets which constitute the successor children. The splitting is based on a set of splitting rules based on classification features. This process is repeated on each derived subset in a recursive manner called recursive partitioning. The recursion is completed when the subset at a node has all the same values of the target variable, or when splitting no longer adds value to the predictions. This process of *top-down induction of decision trees* (TDIDT) is an example of a greedy algorithm, and it is by far the most common strategy for learning decision trees from data.

In data mining, decision trees can be described also as the combination of mathematical and computational techniques to aid the description, categorization and generalization of a given set of data.

Data comes in records of the form:

$$(\mathbf{x}, Y) = (x_1, x_2, x_3, \ldots, x_k, Y)$$

The dependent variable, Y, is the target variable that we are trying to understand, classify or generalize. The vector X is composed of the features, x1, x2, x3 etc., that are used for that task.

[1] Reference: https://en.wikipedia.org/wiki/Decision_tree_learning

Naive Bias Algorithm Technique[2]

Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. There is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable.

In many practical applications, parameter estimation for naive Bayes models uses the method of maximum likelihood; in other words, one can work with the naive Bayes model without accepting Bayesian probability or using any Bayesian methods.

An advantage of naive Bayes is that it only requires a small number of training data to estimate the parameters necessary for classification.

$$p(C_k \mid \mathbf{x}) = \frac{p(C_k)\, p(\mathbf{x} \mid C_k)}{p(\mathbf{x})}$$

[2] Reference: https://en.wikipedia.org/wiki/Naive_Bayes_classifier

Neural Network Algorithm Technique[3]

A biological neural network is composed of a groups of chemically connected or functionally associated neurons. A single neuron may be connected to many other neurons and the total number of neurons and connections in a network may be extensive. Connections, called synapses, are usually formed from axons to dendrites, though dendrodendritic synapses and other connections are possible. Apart from the electrical signaling, there are other forms of signaling that arise from neurotransmitter diffusion.

Artificial intelligence, cognitive modeling, and neural networks are information processing paradigms inspired by the way biological neural systems process data. Artificial intelligence and cognitive modeling try to simulate some properties of biological neural networks. In the artificial intelligence field, artificial neural networks have been applied successfully to speech recognition, image analysis and adaptive control, in order to construct software agents (in computer and video games) or autonomous robots.

Historically, digital computers evolved from the von Neumann model, and operate via the execution of explicit instructions via access to memory by a number of processors. On the other hand, the origins of neural networks are based on efforts to model information processing in biological systems. Unlike the von Neumann model, neural network computing does not separate memory and processing.

Neural network theory has served both to better identify how the neurons in the brain function and to provide the basis for efforts to create artificial intelligence.

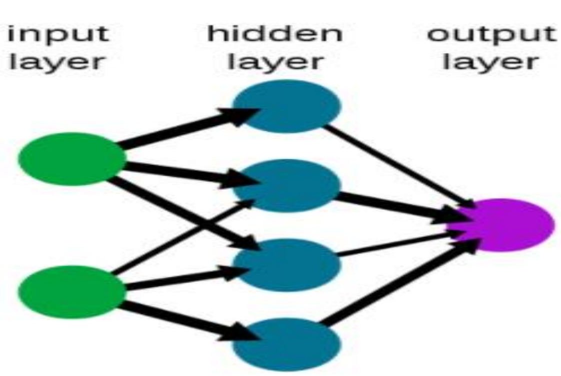


Figure 4: Neural Network Layers

[3] Reference: https://en.wikipedia.org/wiki/Neural_network

Features

Following

The feature for show how many users that account follow which posted that tweet, this helpful for know if that account fake or not when he follows many accounts.

Followers

The feature for show how many users that follow this account which posted that tweet, this helpful for know there is many people follow this account for detect that people not follow fake accounts.

Actions

This feature for know how many like or comments on this tweet, and this helpful for detect if this account fake because of the high number of actions.

Is_retweet

This feature for know if this tweet repeated by same account more than one times, and this helpful for show this account fake because for automatically post this tweet.

Location

This feature for show the location of the user who posted this tweet, and we can make helpful of this feature by comparing the location with the actual cities in the worlds or if there no location then this account fake and we used this type for detect if the tweet spam or ham.

Length_before

This feature for count the number of characters in the tweet before processing, that helpful for detect if the tweet is very long then its spam.

Length_after

This feature for count the number of characters in the tweet after processing, that helpful for detect if the tweet is very long also after processing, then its spam.

Num_of_words_before

This feature for count the number of words in the tweet before processing, that helpful for detect if the tweet has many words, then its spam.

Num_of_words_after

This feature for count the number of words in the tweet after processing, that helpful for detect if the tweet has many words also after processing, then its spam.

Num_urls

This feature for count number of URLs that tweet contain, that helpful for detect if the tweet is spam or not by contain URLs.

Num_of_hashtags

This feature for count number of (#) that tweet contain, that helpful for detect if the tweet is spam or not by contain many of hashtags.

Num_of_mentions

This feature for count number of (@) that tweet contain, that helpful for detect if the tweet is ham or not by contain many of mentions that mean that mentions for real accounts that the tweet refer to.

Num_of_clicks

That feature for show the number of words (click) that tweet contain, because of the URLs that the fake account wants you to click on.

Swear

This feature for detect if the tweet has bad words or not.

Bias

This feature exist by count the number of repeated of words in the ham words and compare it the repeated of words in the spam words, after that make bias to ham or spam.

Implementation by Python Code

This code written by Qutaiba Olayyan.

Libraries

```
import nltk  # for text formatting
from nltk.tokenize import word_tokenize  # for divide the text to words
from nltk.corpus import stopwords  # for get the non important words
from nltk.stem import WordNetLemmatizer  # for get the source of the words
import re  # for regex
import string  # for get the prepositions
import pandas as pa  # for read from the csv files
import matplotlib.pyplot as plt  # for plot the information gain
from sklearn.model_selection import train_test_split  # for split the data to training and testing data
import numpy as np  # for use matrix
from sklearn.feature_selection import SelectKBest  # for select the best features
from sklearn import tree  # for create the decision tree model
from sklearn.neural_network import MLPClassifier  # for create the neural network tree model
from sklearn.naive_bayes import GaussianNB  # for create the naive bias model
from sklearn.naive_bayes import MultinomialNB  # for create the naive bias with multi. features model
from sklearn.feature_extraction.text import CountVectorizer  # for get multi. features from the tweets
from sklearn.preprocessing import LabelEncoder  # for change the classifiers to 0 or 1
from sklearn.metrics import recall_score  # for calculate the recall
from sklearn.metrics import precision_score  # for calculate the precision
from sklearn.metrics import f1_score  # for calculate the f1
from sklearn.metrics import accuracy_score  # for calculate the accuracy
from sklearn.metrics import classification_report, confusion_matrix  # for plot the confusion matrix
from sklearn.feature_selection import mutual_info_classif  # for select the best features
import sys  # for handle the processes from the java interface actions
```

Figure 5: Libraries

Inputs from the user

```
# Input variables from user
selected_doctor_form_file = None  # for get the csv sample tweets file with all features
selected_net_form_file = None  # for get the csv sample tweets file just with the tweets
selected_model = None  # for set the model which selected by the user
selected_features = None  # for set the features which selected by the user
x_train, y_train, x_test, y_test = None, None, None, None
xn, yn, xd, yd = None, None, None, None
text_train_x, text_train_y = None, None
text_model = None
```

Figure 6: Inputs from the user

Decision Tree model

```
# for make Decision Tree Model
def tree_model(x_train, y_train):
    tree_model = tree.DecisionTreeClassifier()
    tree_model.fit(x_train, y_train)
    return tree_model
```

Figure 7: Decision Tree model

Naive Bias model

```
# for make Gaussian Naive Bias Model
def naive_bias_model(x_train, y_train):
    naive_bias_model = GaussianNB()
    naive_bias_model.fit(x_train, y_train)
    return naive_bias_model
```

Figure 8: Naive Bias model

Neural Network model

```
# for make MLP Model
def network_model(x_train, y_train):
    network_model = MLPClassifier()
    network_model.fit(x_train, y_train)
    return network_model
```

Figure 9: Neural Network model

Naive Bias model with multiple features

```
# for create naive bias model with multiple features
def create_naive_bias_mul_features_model_then_test_tweet(tweet):


    text = pa.DataFrame({"Tweet": [tweet]})
    tweets_types_columns = csv_train_file[["Tweet", "Type"]]
    tweets_file = pa.DataFrame.copy(tweets_types_columns)

    le = LabelEncoder()

    # for replace the ham with 0
    # and the spam with 1
    tweets_file['Type'] = le.fit_transform(tweets_file['Type'])

    count_vector = CountVectorizer(analyzer=text_processing)

    training_data = count_vector.fit_transform(tweets_file['Tweet'])
    test_text = count_vector.transform(text)

    naive_bias_text_model = MultinomialNB()
    naive_bias_text_model.fit(training_data, tweets_file['Type'])

    print(int(naive_bias_text_model.predict(test_text)))
```

Figure 10: Naive Bias model with multiple features

Spam and Ham function

After split the data into training and testing and save the source of each word into files with it frequency from the training data, we create function for get the spam words and ham words from the tweets.

```
# for save the spam and ham words in dictionaries
def categorize_words(csv_data):
    spam_words = {}
    ham_words = {}

    for line in csv_data['roots'][csv_data['Type'] == 1]:
        for word in str(line).split(" "):
            if word in spam_words.keys():
                spam_words[word] += 1
            else:
                spam_words[word] = 1

    for line in csv_data['roots'][csv_data['Type'] == 0]:
        for word in str(line).split(" "):
            if word in ham_words.keys():
                ham_words[word] += 1
            else:
                ham_words[word] = 1

    return spam_words, ham_words
```

Figure 11: Split ham and spam words frequencies function

```python
# for read the ham and spam words from the file
def get_ham_spam_words(ham_path, spam_path):
    ham_words = {}
    spam_words = {}

    with open(ham_path, "rb") as f:
        for line in f.readlines():
            if str(line) != '\n':
                line = str(line).split(" ")
                key = line[0][2:]
                value = int(line[1][:-3])
                ham_words[key] = value

    with open(spam_path, "rb") as f:
        for line in f.readlines():
            if str(line) != '\n':
                line = str(line).split(" ")
                key = line[0][2:]
                value = int(line[1][:-3])
                spam_words[key] = value

    return ham_words, spam_words
```

Figure 12: get the spam and ham words frequencies

Swear Words function

For get the swear words from the file.

```
def read_swear_words(file_path):
    words = []
    with open(file_path, "r") as f:
        for line in f.readlines():
            line = str(line)
            if line != "\n":
                words.append(line[:-1])

    return words
```

Figure 13: read the swear words from the file function

```
# for check if the tweet has swear words or not
def contains_profanity(tweet):
    for word in tweet.split(" "):
        if word in swear_words:
            return True
    return False
```

Figure 14: check if the list of words has bad words or not function

The Features used in the program

```
################### Features Labels ##############################
all_features_labels = ["following", "followers", "actions", "is_retweet", "location", "length_before",
                       "num_words_before", "length_after", "num_words_after", "num_urls", "num_hashtags",
                       "num_minations", "num_click", "swear", "bias"]

text_features_labels = ["length_before", "num_words_before", "length_after", "num_words_after", "num_urls",
                        "num_hashtags", "num_minations", "num_click", "swear", "bias"]

best_features_labels = ["following", "followers", "actions"]

best_text_features_labels = ["length_before", "num_words_before", "length_after", "num_words_after"]
##################################################################
```

Figure 15: The features

Clean function

For clean the important words from the list of words.

```
# for clean the words that not important
def clean_text(list_words):
    regex1 = re.compile(r'@[A-Za-z0-9]+|#|@|^http[s]*|[0-9]')
    filtered1 = [i for i in list_words if not regex1.search(i)]

    regex2 = re.compile('[%s]' % re.escape(string.punctuation))
    filtered2 = [i for i in filtered1 if not regex2.search(i)]

    return filtered2
```

Figure 16: Clean the non-important word's function

Root function

For return the roots of the list of words.

```
# for return the tweet words to its roots and delete the duplicate
def get_roots(tweet):
    check_types = nltk.pos_tag(tweet)
    root = WordNetLemmatizer()
    root_words = set()
    for word in check_types:
        type = word[1][0]
        if type == 'N':
            root_words.add(root.lemmatize(word[0], pos="n"))
        elif type == "V":
            root_words.add(root.lemmatize(word[0], pos="v"))
        elif type == "A":
            root_words.add(root.lemmatize(word[0], pos="a"))
        elif type == "S":
            root_words.add(root.lemmatize(word[0], pos="s"))
        else:
            root_words.add(root.lemmatize(word[0], pos="n"))

    return list(root_words)
```

Figure 17: get the root of words function

Create features vector function

```python
# for get the tweet features depend on the text
def preprocessing(tweet):
    tweet = str(tweet)

    # create length_before feature
    length_tweet_before = len(tweet)

    # create num_words_before feature
    num_of_words_before = len(tweet.split(" "))

    # create num_urls feature
    url = re.findall("(http[s]:\/\/)?([\w-]+\.)+([a-z]{2,5})(\/+\w+)?", tweet)
    num_of_urls = len(url)

    # create swear feature
    contain_swear_words = contains_profanity(tweet)
    if contain_swear_words:
        contain_swear_words = 1
    else:
        contain_swear_words = 0

    # for delete the stop words
    stop_words = set(stopwords.words("english"))
    divide_tweet_to_words = list(word_tokenize(str(tweet).lower()))
    remaining_words = [word for word in divide_tweet_to_words if word not in stop_words]
```

Figure 18: preprocessing function part 1

```
# create num_hashtags feature
num_of_hashtags = remaining_words.count("#")

# create num_minations feature
num_of_minations = remaining_words.count("@")

# create num_click feature
num_of_click_word = remaining_words.count("click")

# for clean the tweet
cleaning_tweet = clean_text(remaining_words)

# create num_words_after feature
num_of_words_after = len(cleaning_tweet)

# create length_after feature
length_tweet_after = len("".join(cleaning_tweet))

# for get the roots of the tweet words
tweet_roots = get_roots(cleaning_tweet)
tweet_roots = " ".join(tweet_roots)
```

Figure 19: preprocessing function part 2

We made the bias feature after save the ham and spam words into files with it frequencies, and get the number of frequencies of the word in the spam file and ham file if the word repeated in the ham file more than spam file then the word bias to be ham otherwise spam.

```python
# for make bias feature
def bias_feature(roots, spam_words, ham_words):
    roots = str(roots)
    ham = 0
    spam = 0

    for word in roots.split(" "):
        if word in spam_words.keys():
            spam += spam_words[word]
        if word in ham_words.keys():
            ham += ham_words[word]

    if ham > spam:
        return 0

    return 1
```

Figure 20: create bias feature depend on the spam and ham words frequencies

Information Gain function

```python
# for show the information gain
def show_IG_for_features(x_train, y_train):
    # configure to select all features
    fs = SelectKBest(score_func=mutual_info_classif, k='all')
    # learn relationship from data
    fs.fit(x_train, y_train)

    for i in range(len(fs.scores_)):
        print('Feature[%d]: %s = %f' % (i, fs.feature_names_in_[i], fs.scores_[i]))
```

Figure 21: print Information Gain for features function

Tweet Detection function

For detect if the tweet is spam or ham depend on the selected features and selected model.

```python
# for print if the tweet is ham or spam
def print_model_result(model, features_type, spam_words, ham_words, tweet, following=0, followers=0, actions=0,
                       is_retweet=0, location=""):
    if location == "":
        location = 0
    else:
        location = 1
    lb, la, nwb, nwa, nu, nh, nm, nc, s, r, bias = get_text_features(tweet, spam_words, ham_words)
    features_vector = None
    if features_type == "text":
        features_vector = pa.DataFrame({"length_before": [lb], "num_words_before": [nwb],
                                        "length_after": [la], "num_words_after": [nwa], "num_urls": [nu],
                                        "num_hashtags": [nh], "num_minations": [nm], "num_click": [nc], "swear": [s],
                                        "bias": [bias]})
    elif features_type == "best_text":
        features_vector = pa.DataFrame({"length_before": [lb], "num_words_before": [nwb],
                                        "length_after": [la], "num_words_after": [nwa]})
    elif features_type == "all":
        features_vector = pa.DataFrame({"following": [following], "followers": [followers], "actions": [actions],
                                        "is_retweet": [is_retweet], "location": [location],
                                        "length_before": [lb], "num_words_before": [nwb], "length_after": [la],
                                        "num_words_after": [nwa], "num_urls": [nu], "num_hashtags": [nh],
                                        "num_minations": [nm], "num_click": [nc], "swear": [s], "bias": [bias]
                                        })
    elif features_type == "best_all":
        features_vector = pa.DataFrame({"following": [following], "followers": [followers], "actions": [actions]})
    else:
        print("Wrong features type parameter")
    # print("Tweet: {0}\nResult: {1}".format(tweet, model.predict(features_vector)))
    print(int(model.predict(features_vector)))
```

Figure 22: Tweet Detection function

Score results function

```python
# for print
# the accuracy
# and precision and recall and f1 scores
def print_score(model, x, y_true):
    acc, recall, precision, f1, y_pred = cal_scores(model, x, y_true)

    print("Accuracy: ", np.round(acc, 2))
    print("Recall: ", np.round(recall, 2))
    print("Precision: ", np.round(precision, 2))
    print("F1: ", np.round(f1, 2))
    print("confusion_matrix: \n", confusion_matrix(y_true, y_pred))
    print("classification_report: \n", classification_report(y_true, y_pred))
```

Figure 23: print score result of the model function

Create new training and testing data with features vectors

```python
# the created file:
# split_data_file_inaddition_to_new_features("train.csv", "train_file.csv", "test_file.csv", "ham_words_file", "spam_words_file")
# this function for make all features and split the train and test data into two csv files
def split_data_file_inaddition_to_new_features(file_path, new_train_path, new_test_path, ham_words_path,
                                               spam_words_path, test_size=0.2, random_state=10):


    df = pa.read_csv(file_path)

    df_GF = df[["Tweet", "following", "followers", "actions", "is_retweet", "location", "Type"]]

    le = LabelEncoder()
    df_update = pa.DataFrame.copy(df_GF)
    # for replace the location with 1 if exist and 0 if null
    df_update['location'] = df_update['location'].apply(lambda x: 1 if not pa.isnull(x) else 0)
    # for replace the null with zero
    df_update['actions'] = df_update['actions'].replace(np.nan, 0)
    df_update['following'] = df_update['following'].replace(np.nan, 0)
    df_update['followers'] = df_update['followers'].replace(np.nan, 0)
    df_update['is_retweet'] = df_update['is_retweet'].replace(np.nan, 0)

    # for replace the ham with 0
    # and the spam with 1
    df_update['Type'] = le.fit_transform(df_update['Type'])
```

Figure 24: create features vectors of the data part 1

```
# for get the features vectors for all tweet
length_before_column = []
length_after_column = []
num_words_before_column = []
num_words_after_column = []
num_urls_column = []
num_hashtags_column = []
num_minations_column = []
num_click_column = []
swear_column = []
root_column = []
count = 0
tweet_column = list(df_update["Tweet"])
for tweet in tweet_column:
    lb, la, nwb, nwa, nu, nh, nm, nc, s, r = preprocessing(tweet)
    length_before_column.append(lb)
    length_after_column.append(la)
    num_words_before_column.append(nwb)
    num_words_after_column.append(nwa)
    num_urls_column.append(nu)
    num_hashtags_column.append(nh)
    num_minations_column.append(nm)
    num_click_column.append(nc)
    swear_column.append(s)
    root_column.append(r)
    print(count)
    count += 1
```

Figure 25: create features vectors of the data part 2

```
new_csv = pa.DataFrame({"Type": df_update['Type'], "Tweet": df_update['Tweet'], "roots": root_column,
                        "following": df_update['following'],
                        "followers": df_update['followers'], "actions": df_update['actions'],
                        "is_retweet": df_update['is_retweet'],
                        "location": df_update['location'], "length_before": length_before_column,
                        "length_after": length_after_column, "num_words_before": num_words_before_column,
                        "num_words_after": num_words_after_column, "num_urls": num_urls_column,
                        "num_hashtags": num_hashtags_column,
                        "num_minations": num_minations_column,
                        "num_click": num_click_column, "swear": swear_column})

# for split the data into training and testing csv files
x = new_csv.drop('Type', axis=1)
y = new_csv.Type

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=test_size, random_state=random_state)

print('size of test dataset = {}, size of traing data = {}, percentage = {}%'.format(len(x_test), len(x_train),
                                                                                     len(x_test) * 100 / (
                                                                                             len(x_test) + len(
                                                                                         x_train))))

train_csv_file = pa.DataFrame(x_train)
train_csv_file["Type"] = y_train

test_csv_file = pa.DataFrame(x_test)
test_csv_file["Type"] = y_test
```
Figure 26: create features vectors of the data part 3

```python
# for create the bias feature for all tweets
spam_words, ham_words = categorize_words(train_csv_file)
train_csv_file["bias"] = train_csv_file['roots'].apply(lambda b: bias_feature(b, spam_words, ham_words))
test_csv_file["bias"] = test_csv_file['roots'].apply(lambda b: bias_feature(b, spam_words, ham_words))

# for save the train and test csv files
train_csv_file.to_csv(new_train_path, index=False)
test_csv_file.to_csv(new_test_path, index=False)

# for save the ham words in the file
file_for_save_ham_words = open(ham_words_path, "wb")
for element in ham_words:
    file_for_save_ham_words.write(bytes(element, 'utf-8'))
    file_for_save_ham_words.write(bytes(" ", 'utf-8'))
    file_for_save_ham_words.write(bytes(str(ham_words[element]), 'utf-8'))
    file_for_save_ham_words.write(bytes("\n", 'utf-8'))
file_for_save_ham_words.close()

# for save the spam words in the file
file_for_save_spam_words = open(spam_words_path, "wb")
for element in spam_words:
    file_for_save_spam_words.write(bytes(element, 'utf-8'))
    file_for_save_spam_words.write(bytes(" ", 'utf-8'))
    file_for_save_spam_words.write(bytes(str(spam_words[element]), 'utf-8'))
    file_for_save_spam_words.write(bytes("\n", 'utf-8'))
file_for_save_spam_words.close()
```

Figure 27: create features vectors of the data part 4

Handler function

```python
# for handling the arguments which request from the java code
# and make response for it by print on the console
def handler(args):
    global xd, yd, xn, yn
    global selected_doctor_form_file, selected_net_form_file

    args = list(map(str, args))
    # following=0, followers=0, actions=0,
    #                   is_retweet=0, location=""
    # when the user enter tweet and want to check it if spam or ham
    # you should enter
    # args[2] --> tf: use text features model, nbf: use multi features by naive bias, af: use all features tree model
    # args[3] = tweet
    # args[4,5,6,7,8] : just if you pass "af" --> 4: following, 5: followers, 6: actions, 7: is_retweet, 8: location
    if args[1] == "tweet":
        if args[2] == "tf":
            create_text_model()
            print_model_result(text_model, "text", spams_words, qualities_words, tweet=args[3])
        elif args[2] == "nbf":
            create_naive_bias_mul_features_model_then_test_tweet(args[3])
        elif args[2] == "af":
            create_the_system("tree", "af")
            print_model_result(selected_model, "all", spams_words, qualities_words, tweet=args[3], following=args[4],
                               followers=args[5], actions=args[6], is_retweet=args[7], location=args[8])
```

Figure 28: Handler function part 1

```python
# when the user want to know the information gain for the features
# you should pass
# args[2] = the features_label selected by user (af, at, ba, bt)
elif args[1] == "ig":
    pick_selected_features(args[2])
    x_ig, y_ig = get_x_y(csv_train_file, selected_features)
    print_ig_on_console(x_ig, y_ig)

# when the user want to show the accuracy of the test file
# you should pass
# args[2] = the model selected by user (tree, network, naive_bias)
# args[3] = the features_label selected by user (af, at, ba, bt)
elif args[1] == "atf":
    create_the_system(args[2], args[3])
    print_score_on_console(selected_model, x_test, y_test)

# when the user want to show the accuracy of the sample file
# you should pass
# args[2] = the model selected by user (tree, network, naive_bias)
# args[3] = the features_label selected by user (af, at, ba, bt)
# args[4] = the path of the sample file
elif args[1] == "adf":
    create_the_system(args[2], args[3])
    selected_doctor_form_file = read_from_test_file_with_doctor_features(args[4], spams_words, qualities_words)
    xd, yd = get_x_y(selected_doctor_form_file, selected_features)
    print_score_on_console(selected_model, xd, yd)
```

Figure 29: Handler function part 2

```python
# when the user want to show the accuracy of the net file
# you should pass
# args[2] = the path of the tweets file
# not required to the selected model and features selected by user
elif args[1] == "anf":
    create_text_model()
    selected_net_form_file = read_from_test_file_just_tweets(args[2], spams_words, qualities_words)
    xn, yn = get_x_y(selected_net_form_file, text_features_labels)
    print_score_on_console(text_model, xn, yn)
```

Figure 30: Handler function part 3

Testing by Java Code

The interface code written by Ibrahim Duhaidi.

Testing data result

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	Tweet	roots	following	followers	actions	is_retweet	location	length_bef	length_aft	num_word	num_word	num_urls	num_hash	num_mina	num_click	swear	bias	Type
2	Court orde	school idic	0	0	0	0	1	134	58	21	10	1	0	0	0	0	0	0
3	Jin:	scissor rm	0	184000	1398	0	1	133	54	20	14	0	0	0	0	0	0	0
4	ICYMI: Tre	icymi trey	253	2575	162577	1	1	88	34	10	7	1	0	0	0	0	0	1
5	Original dr	charity sca	0	12200	0	0	1	135	99	20	13	0	1	0	0	0	0	0
6	#GrowingU	clinger bibl	5539	5034	12378	1	1	78	47	12	6	0	1	0	0	0	0	1
7	Eyebrows	eyebrow	0	0	0	0	1	43	8	3	1	1	0	0	0	0	0	0
8	Toxic 'red	tide salmo	0	0	0	0	0	91	47	11	7	1	0	0	0	0	0	0
9	https://t.c	zimbabwe	1931	1118	4694	0	1	123	49	14	7	2	0	0	0	0	0	1
10	We love it	way â€¦ g	0	0	0	0	1	142	39	15	8	2	0	0	0	0	0	0
11	okay lang l	welcome y	0	0	0	1	0	75	41	14	9	0	0	0	0	0	0	0
12	11:11	meet	0	0	0	0	1	63	4	7	1	0	0	5	0	0	0	0
13	Siri Correc	siri jenner	606	409	15124	1	1	86	45	11	8	1	1	0	0	0	0	1
14	Get your t	tack rtz ge	521	876	6452	1	1	95	26	15	6	1	1	1	0	0	0	1
15	13. THE	advice doc	0	14200	0	0	0	131	47	19	9	1	0	0	0	0	0	0
16	Now Playi	parton clo	0	0	0	0	1	122	56	16	11	1	2	0	0	0	0	0
17	#may Over	google ma	2944	708	2787	0	1	109	39	12	7	2	2	0	0	0	0	1
18	About Julia	julian char	0	0	0	0	0	138	60	19	9	1	0	0	0	0	0	0
19	Less than	launch det	0	53000	52	0	0	194	44	17	8	2	1	0	0	0	0	0
20	Franois Fill	allegation	4302	3389	17700	0	1	116	56	11	8	2	0	0	0	0	0	1

Figure 31: Testing data form

Decision Tree model

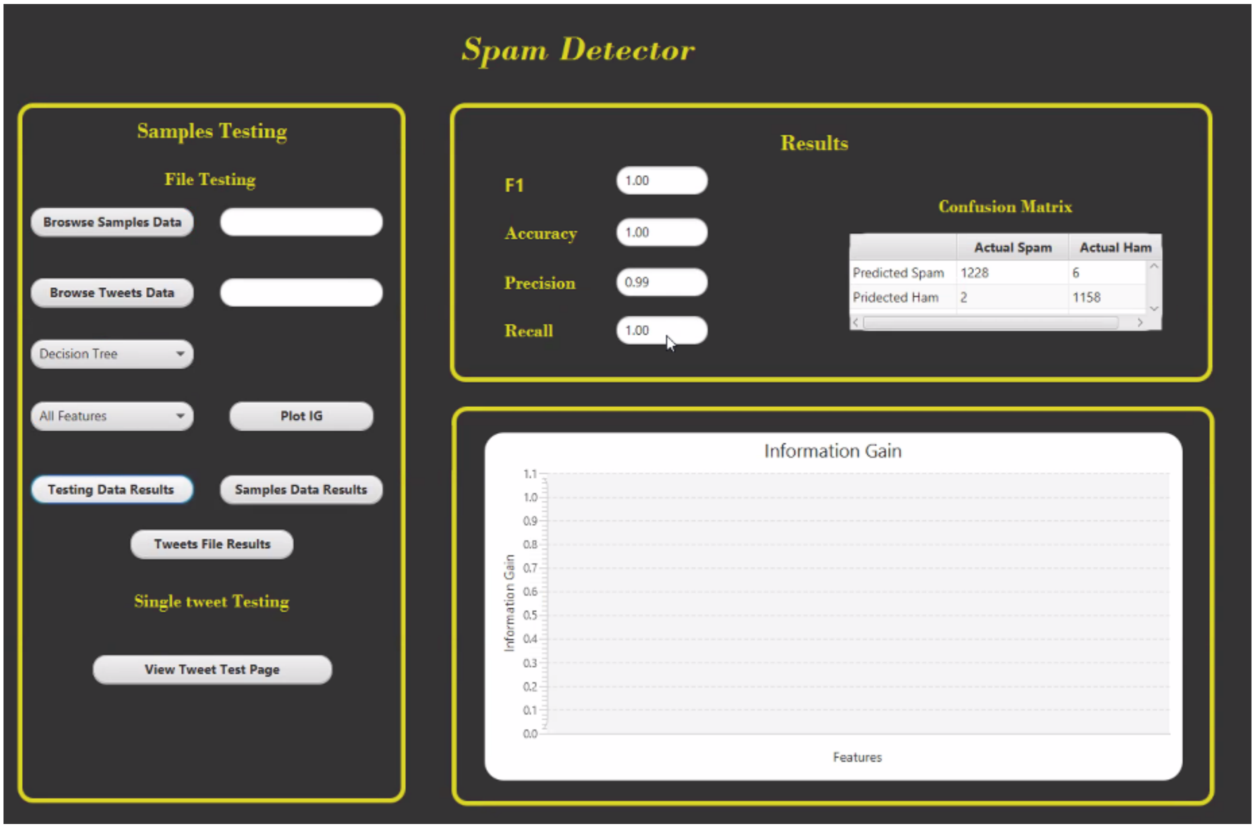


Figure 32: Decision Tree model result

Neural Network model

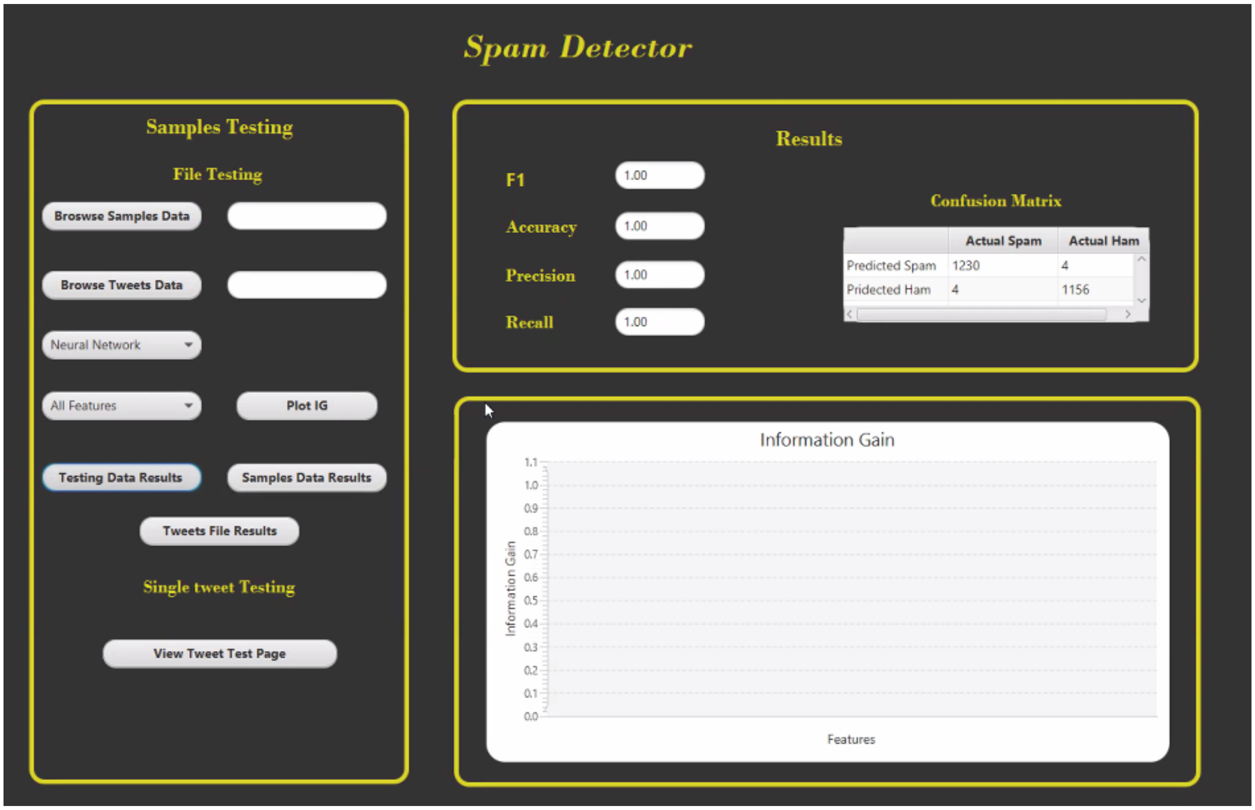


Figure 33: Neural Network model result

Naive Bias model

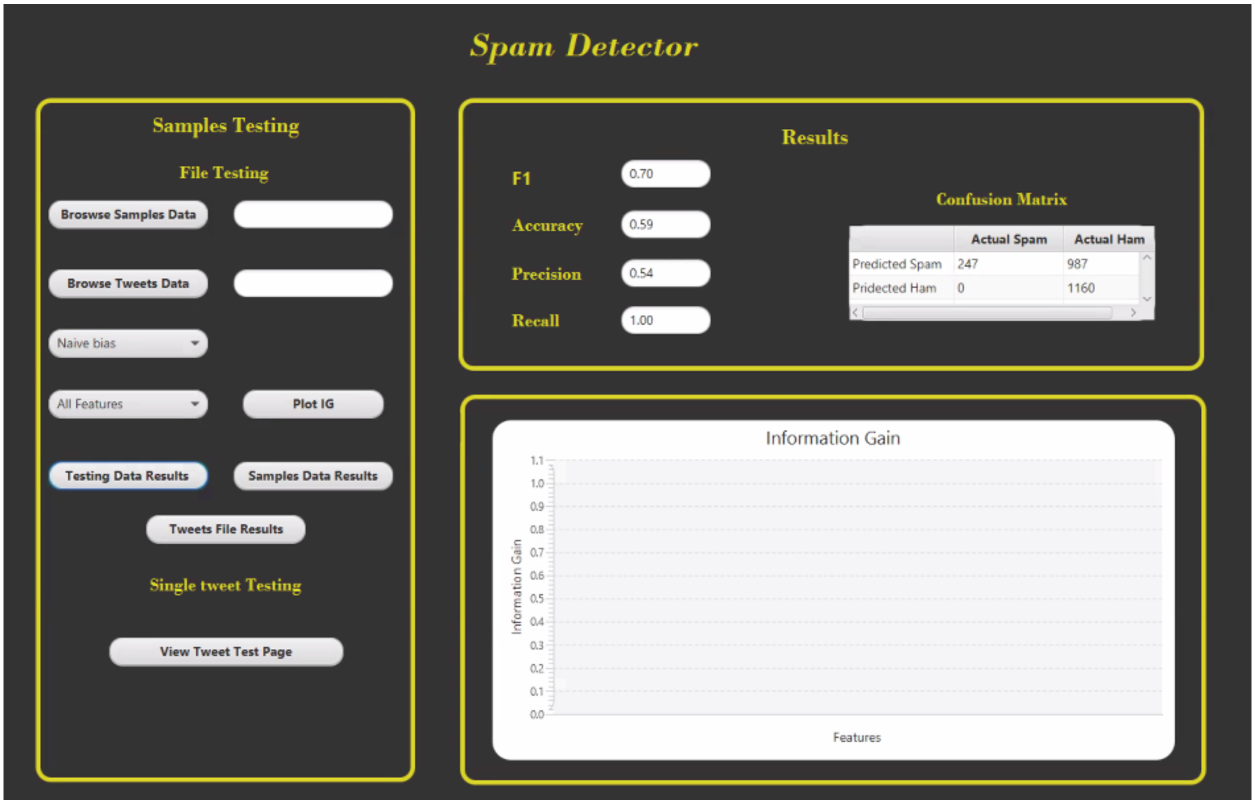


Figure 34: Naive Bias model result

Sample file Result

	A	B	C	D	E	F	G	H
1	Id	Tweet	following	followers	actions	is_retweet	location	Type
2	10091	It's the eve	0	11500		0	Chicago	Quality
3	10172	Eren sent a	0	0		0		Quality
4	7012	I posted a	0	0		0	Scotland, U	Quality
5	3697	#jan Idiot (	3319	611	294	0	Atlanta, Ga	Spam
6	10740	Pedophile	4840	1724	1522	0	Blumberg	Spam
7	9572	EBMUD er	4435	16041	27750	0	UPS	Spam
8	10792	Big day. #'	0	0	0	0	Toronto, C	Quality
9	11594	#UPA	0	193000		0	Mumbai	Quality
10	12594	**MISSIN	39000	46900	47	0	UK	Quality
11	10963	Paraguaya	9025	20165	6331	0	Cape Town	Spam
12	10778	Tagged by	0	0		0	siempre, ro	Quality
13	4174	"WE	0	0		0		Quality
14	5401	#NowPlayi	780	897	4792	1	UK	Spam
15	7636	The Guard	1893	1651	3564	1	Mumbai, M	Spam
16	6908	Terrorists.	7981	12815	13601	1	Austin, Tex	Spam
17	5265	Eastside Iv	0	0	0	0		Quality
18	10433	Boston bo	85	73	434	0	South MY	Spam
19	1643	Lovelyz - B	0	0		0	LovelyzRP	Quality
20	2445	Happy #Tic	0	0	259	0	em	Quality

Figure 35: Sample file form

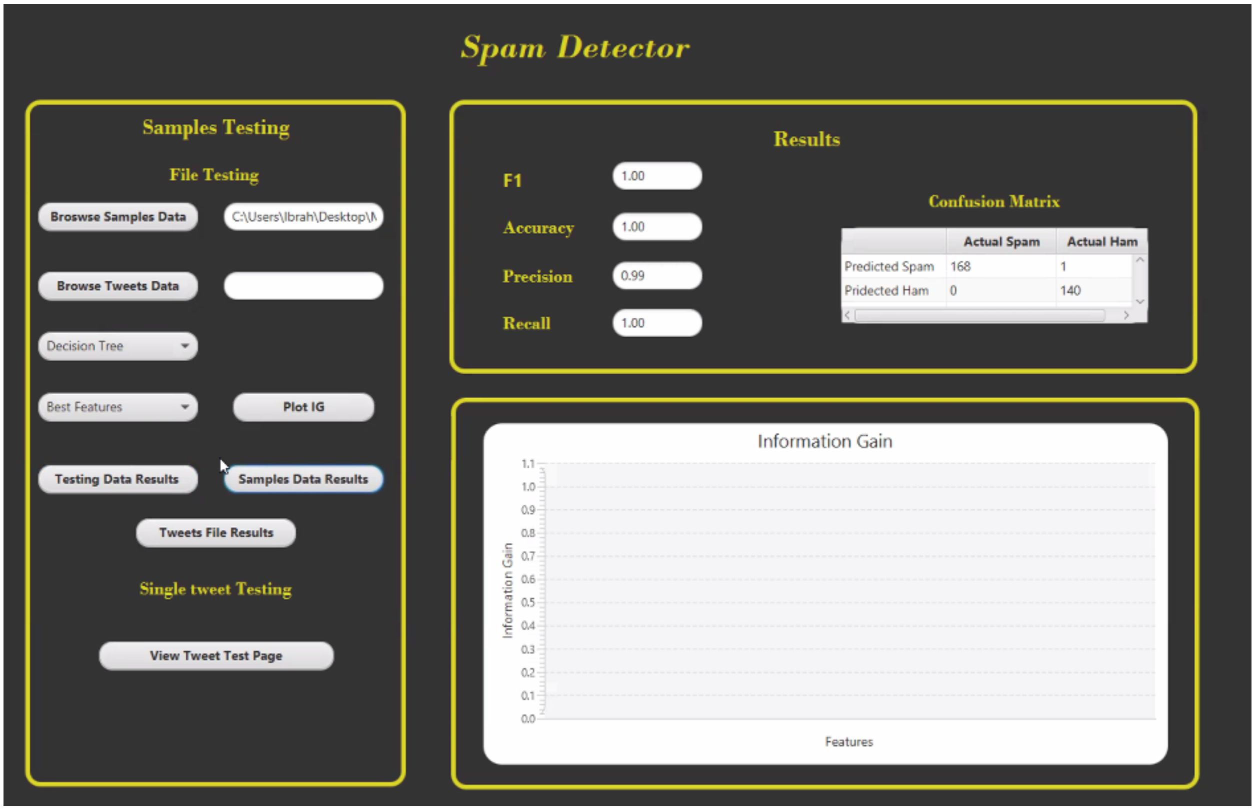


Figure 36: Sample file Result

Tweets file Result

	Type	Tweet
1	Type	Tweet
2	ham	Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...
3	ham	Ok lar... Joking wif u oni...
4	spam	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's
5	ham	U dun say so early hor... U c already then say...
6	ham	Nah I don't think he goes to usf, he lives around here though
7	spam	FreeMsg Hey there darling it's been 3 week's now and no word back! I'd like some fun you up for it still? Tb ok! XxX std chgs to send, â£1.50 to rcv
8	ham	Even my brother is not like to speak with me. They treat me like aids patent.
9	ham	As per your request 'Melle Melle (Oru Minnaminunginte Nurungu Vettam)' has been set as your callertune for all Callers. Press *9 to copy your friends Callertune
10	spam	WINNER!! As a valued network customer you have been selected to receivea â£900 prize reward! To claim call 09061701461. Claim code KL341. Valid 12 hours only.
11	spam	Had your mobile 11 months or more? U R entitled to Update to the latest colour mobiles with camera for Free! Call The Mobile Update Co FREE on 08002986030
12	ham	I'm gonna be home soon and i don't want to talk about this stuff anymore tonight, k? I've cried enough today.
13	spam	SIX chances to win CASH! From 100 to 20,000 pounds txt> CSH11 and send to 87575. Cost 150p/day, 6days, 16+ TsandCs apply Reply HL 4 info
14	spam	URGENT! You have won a 1 week FREE membership in our â£100,000 Prize Jackpot! Txt the word: CLAIM to No: 81010 T&C www.dbuk.net LCCLTD POBOX 4403LDNW1A7RW18
15	ham	I've been searching for the right words to thank you for this breather. I promise i wont take your help for granted and will fulfil my promise. You have been wonderful and a blessing at all times.
16	ham	I HAVE A DATE ON SUNDAY WITH WILL!!
17	spam	XXXMobileMovieClub: To use your credit, click the WAP link in the next txt message or click here>> http://wap. xxxmobilemovieclub.com?n=QJKGIGHJJGCBL
18	ham	Oh k...i'm watching here:)
19	ham	Eh u remember how 2 spell his name... Yes i did. He v naughty make until i v wet.
20	ham	Fine if thatåÕs the way u feel. ThatåÕs the way its gota b

Figure 37: Tweets file form

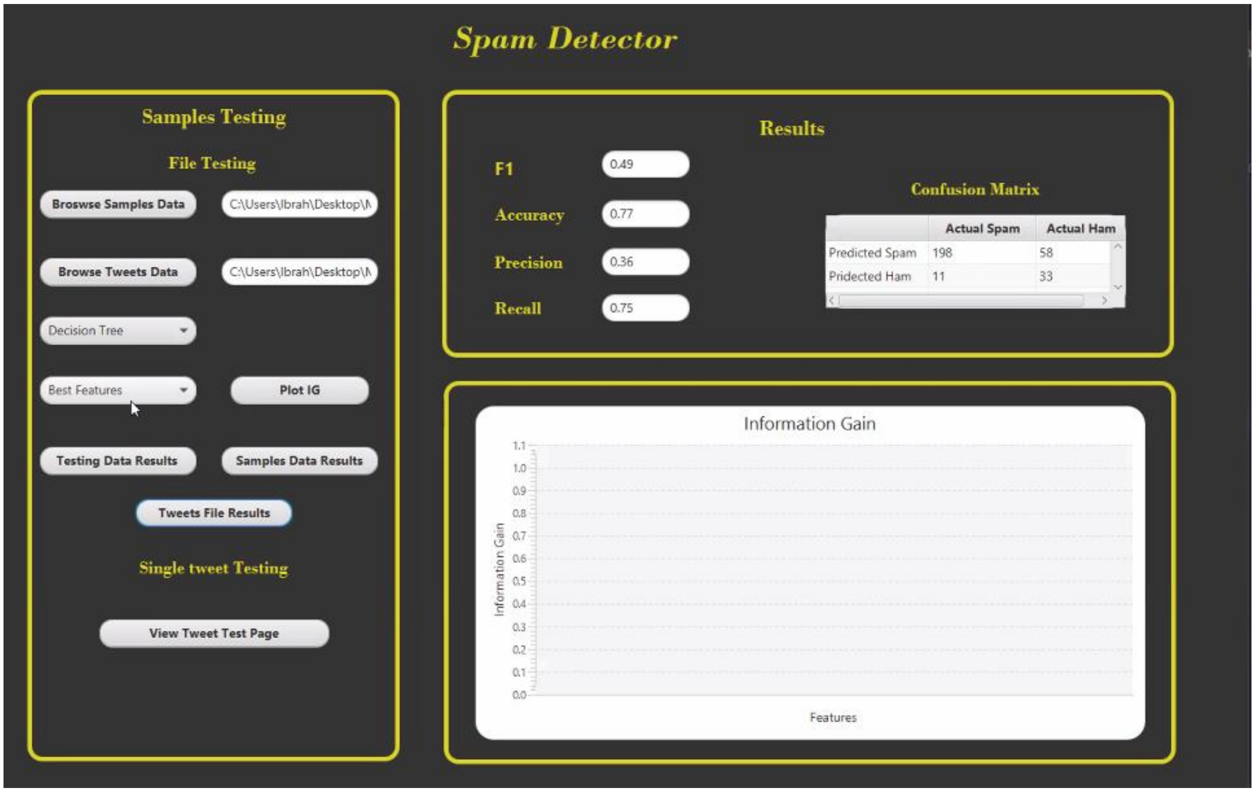


Figure 38: Tweets file Result

Information Gain for all features

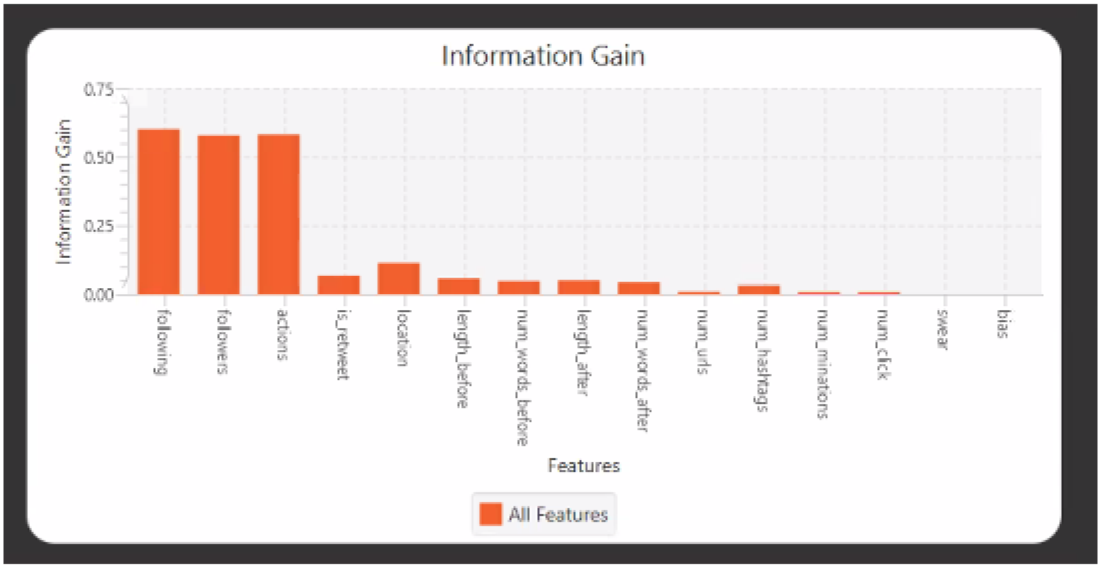


Figure 39: Information Gain for all features

Information Gain for text features

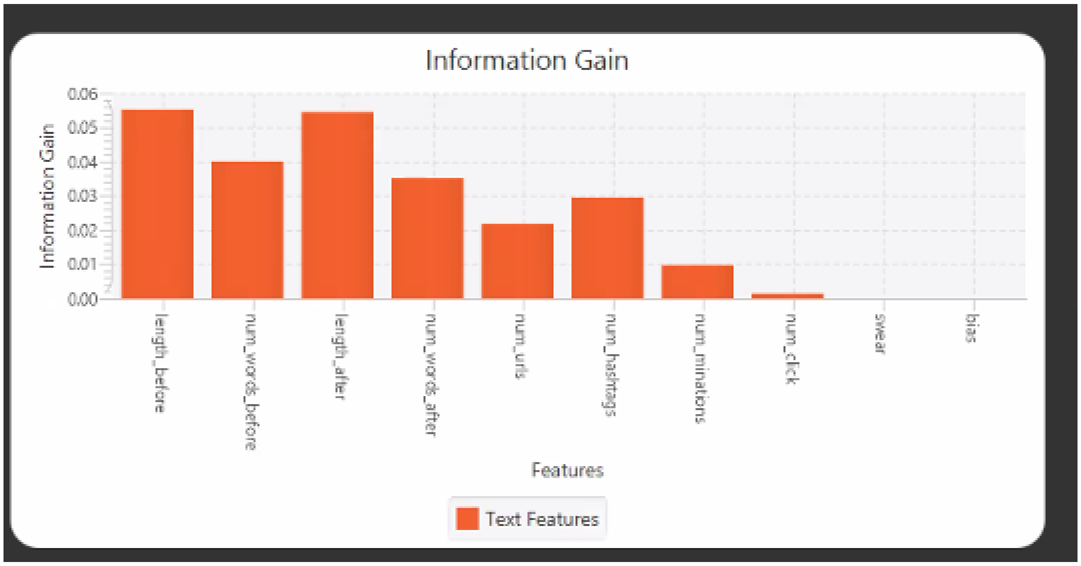


Figure 40: Information Gain for text features

Single Tweet with all features result

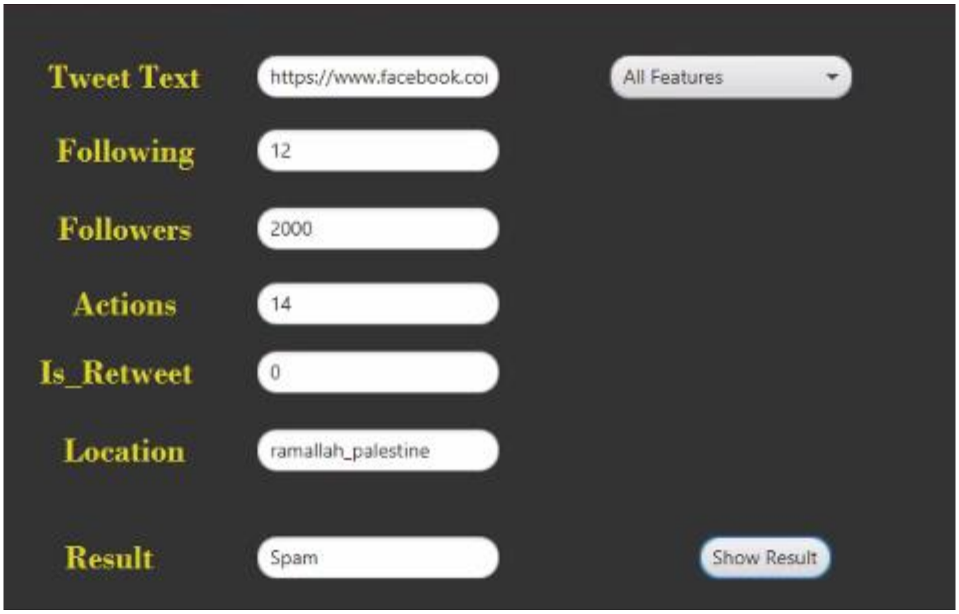


Figure 41: Single Tweet with all features result

Single Tweet with just text features result

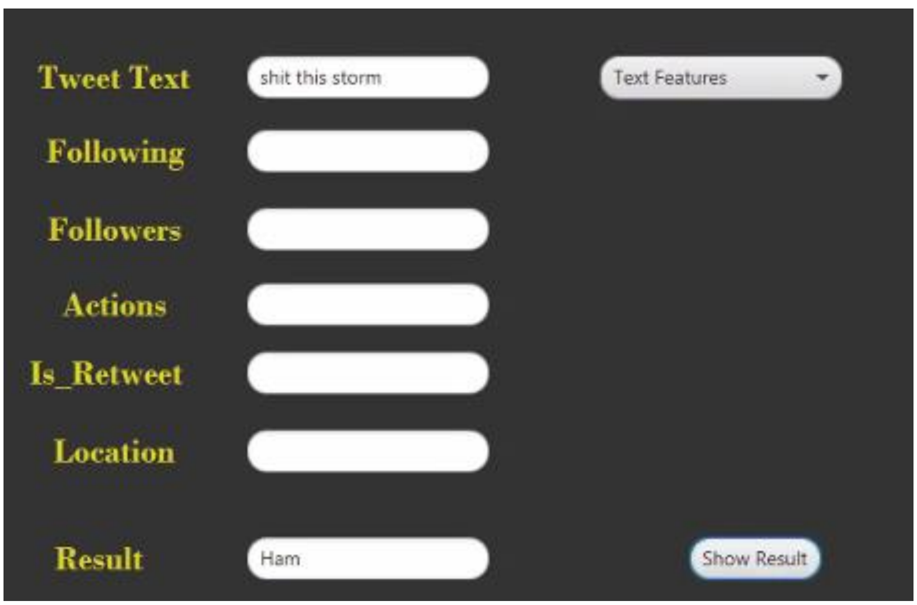


Figure 42: Single Tweet with just text features result

Single Tweet result by naive bias with multiple features

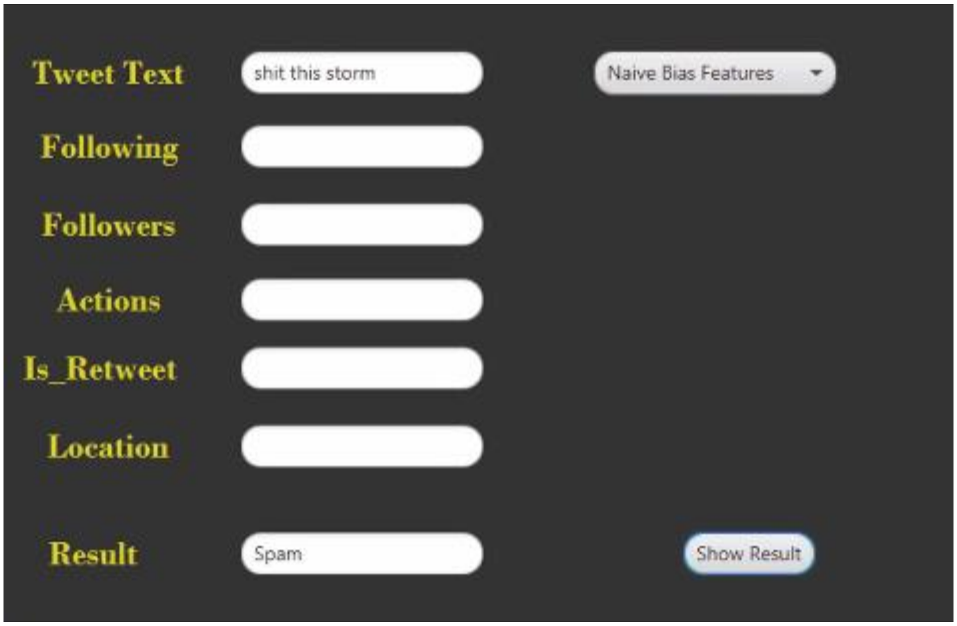


Figure 43: Single Tweet result by naive bias with multiple features

Conclusion

As we see from the testing result, the decision tree is the best model for our problem but the naive bias is the worst one.

For the best features we found that the follower, following and actions have the highest information gain so we made the decision tree just for these features for make the model more general for different tweets.

For let the naïve bias helpful we used it for make multiple of features depend on the words of the tweets and use this model for detect if the single tweet is spam or ham, so the naïve bias is good when there many features and the decision tree is the best when there few features.

Appendix

Phyton Code (Back-End)

Main File

```
#####

import nltk  # for text formatting
from nltk.tokenize import word_tokenize  # for divide the text to words
from nltk.corpus import stopwords  # for get the non important words
from nltk.stem import WordNetLemmatizer  # for get the source of the words
import re  # for regex
import string  # for get the prepositions
import pandas as pa  # for read from the csv files
import matplotlib.pyplot as plt  # for plot the information gain
from sklearn.model_selection import train_test_split  # for split the data to
training and testing data
import numpy as np  # for use matrix
from sklearn.feature_selection import SelectKBest  # for select the best
features
from sklearn import tree  # for create the decision tree model
from sklearn.neural_network import MLPClassifier  # for create the neural
network tree model
from sklearn.naive_bayes import GaussianNB  # for create the naive bias model
from sklearn.naive_bayes import MultinomialNB  # for create the naive bias
with multi. features model
from sklearn.feature_extraction.text import CountVectorizer  # for get multi.
features from the tweets
from sklearn.preprocessing import LabelEncoder  # for change the classifiers
to 0 or 1
from sklearn.metrics import recall_score  # for calculate the recall
from sklearn.metrics import precision_score  # for calculate the precision
from sklearn.metrics import f1_score  # for calculate the f1
from sklearn.metrics import accuracy_score  # for calculate the accuracy
from sklearn.metrics import classification_report, confusion_matrix  # for
plot the confusion matrix
from sklearn.feature_selection import mutual_info_classif  # for select the
best features
import sys  # for handle the processes from the java interface actions

#####


# Input variables from user
selected_doctor_form_file = None  # for get the csv sample tweets file with
all features
selected_net_form_file = None  # for get the csv sample tweets file just with
the tweets
selected_model = None  # for set the model which selected by the user
```

```python
selected_features = None  # for set the features which selected by the user
x_train, y_train, x_test, y_test = None, None, None, None
xn, yn, xd, yd = None, None, None, None
text_train_x, text_train_y = None, None
text_model = None


# for read the ham and spam words from the file
def get_ham_spam_words(ham_path, spam_path):
    ham_words = {}
    spam_words = {}

    with open(ham_path, "rb") as f:
        for line in f.readlines():
            if str(line) != '\n':
                line = str(line).split(" ")
                key = line[0][2:]
                value = int(line[1][:-3])
                ham_words[key] = value

    with open(spam_path, "rb") as f:
        for line in f.readlines():
            if str(line) != '\n':
                line = str(line).split(" ")
                key = line[0][2:]
                value = int(line[1][:-3])
                spam_words[key] = value

    return ham_words, spam_words


# for divide the features and the classifier
def get_x_y(csv_pa, features_labels):
    x = csv_pa[features_labels]
    y = csv_pa.Type
    return x, y


# for make Decision Tree Model
def tree_model(x_train, y_train):
    tree_model = tree.DecisionTreeClassifier()
    tree_model.fit(x_train, y_train)
    return tree_model


# for make MLP Model
def network_model(x_train, y_train):
    network_model = MLPClassifier()
    network_model.fit(x_train, y_train)
    return network_model


# for make Gaussian Naive Bias Model
def naive_bias_model(x_train, y_train):
    naive_bias_model = GaussianNB()
    naive_bias_model.fit(x_train, y_train)
```

```
    return naive_bias_model


def read_swear_words(file_path):
    words = []
    with open(file_path, "r") as f:
        for line in f.readlines():
            line = str(line)
            if line != "\n":
                words.append(line[:-1])

    return words


swear_words = read_swear_words("fucking")


# define the ham and spam words as list
qualities_words, spams_words = get_ham_spam_words("ham_words_file",
"spam_words_file")

################### Features Labels ##############################
all_features_labels = ["following", "followers", "actions", "is_retweet",
"location", "length_before",
                       "num_words_before", "length_after", "num_words_after",
"num_urls", "num_hashtags",
                       "num_minations", "num_click", "swear", "bias"]

text_features_labels = ["length_before", "num_words_before", "length_after",
"num_words_after", "num_urls",
                        "num_hashtags", "num_minations", "num_click",
"swear", "bias"]

best_features_labels = ["following", "followers", "actions"]

best_text_features_labels = ["length_before", "num_words_before",
"length_after", "num_words_after"]
##################################################################

# for read the train and test files
csv_train_file = pa.read_csv("train_file.csv")
csv_test_file = pa.read_csv("test_file.csv")


# for check if the tweet has swear words or not
def contains_profanity(tweet):
    for word in tweet.split(" "):
        if word in swear_words:
            return True
    return False



# for clean the words that not important
def clean_text(list_words):
    regex1 = re.compile(r'@[A-Za-z0-9]+|#|@|^http[s]*|[0-9]')
```

```
    filtered1 = [i for i in list_words if not regex1.search(i)]

    regex2 = re.compile('[%s]' % re.escape(string.punctuation))
    filtered2 = [i for i in filtered1 if not regex2.search(i)]

    return filtered2


# for return the tweet words to its roots and delete the duplicate
def get_roots(tweet):
    check_types = nltk.pos_tag(tweet)
    root = WordNetLemmatizer()
    root_words = set()
    for word in check_types:
        type = word[1][0]
        if type == 'N':
            root_words.add(root.lemmatize(word[0], pos="n"))
        elif type == "V":
            root_words.add(root.lemmatize(word[0], pos="v"))
        elif type == "A":
            root_words.add(root.lemmatize(word[0], pos="a"))
        elif type == "S":
            root_words.add(root.lemmatize(word[0], pos="s"))
        else:
            root_words.add(root.lemmatize(word[0], pos="n"))

    return list(root_words)


# for get the tweet features depend on the text
def preprocessing(tweet):
    tweet = str(tweet)

    # create length_before feature
    length_tweet_before = len(tweet)

    # create num_words_before feature
    num_of_words_before = len(tweet.split(" "))

    # create num_urls feature
    url = re.findall("(http[s]:\/\/)?([\w-]+\.)+([a-z]{2,5})(\/+\w+)?",
tweet)
    num_of_urls = len(url)

    # create swear feature
    contain_swear_words = contains_profanity(tweet)
    if contain_swear_words:
        contain_swear_words = 1
    else:
        contain_swear_words = 0

    # for delete the stop words
    stop_words = set(stopwords.words("english"))
    divide_tweet_to_words = list(word_tokenize(str(tweet).lower()))
    remaining_words = [word for word in divide_tweet_to_words if word not in
stop_words]
```

```
    # create num_hashtags feature
    num_of_hashtags = remaining_words.count("#")

    # create num_minations feature
    num_of_minations = remaining_words.count("@")

    # create num_click feature
    num_of_click_word = remaining_words.count("click")

    # for clean the tweet
    cleaning_tweet = clean_text(remaining_words)

    # create num_words_after feature
    num_of_words_after = len(cleaning_tweet)

    # create length_after feature
    length_tweet_after = len("".join(cleaning_tweet))

    # for get the roots of the tweet words
    tweet_roots = get_roots(cleaning_tweet)
    tweet_roots = " ".join(tweet_roots)

    return length_tweet_before, length_tweet_after, num_of_words_before,
num_of_words_after, num_of_urls, num_of_hashtags, num_of_minations,
contain_swear_words, num_of_click_word, tweet_roots


# for save the spam and ham words in dictionaries
def categorize_words(csv_data):
    spam_words = {}
    ham_words = {}

    for line in csv_data['roots'][csv_data['Type'] == 1]:
        for word in str(line).split(" "):
            if word in spam_words.keys():
                spam_words[word] += 1
            else:
                spam_words[word] = 1

    for line in csv_data['roots'][csv_data['Type'] == 0]:
        for word in str(line).split(" "):
            if word in ham_words.keys():
                ham_words[word] += 1
            else:
                ham_words[word] = 1

    return spam_words, ham_words


# for make bias feature
def bias_feature(roots, spam_words, ham_words):
    roots = str(roots)
    ham = 0
    spam = 0
```

```
    for word in roots.split(" "):
        if word in spam_words.keys():
            spam += spam_words[word]
        if word in ham_words.keys():
            ham += ham_words[word]

    if ham > spam:
        return 0

    return 1


# the created file:
# split_data_file_inaddition_to_new_features("train.csv", "train_file.csv",
"test_file.csv", "ham_words_file", "spam_words_file")
# this function for make all features and split the train and test data into
two csv files
def split_data_file_inaddition_to_new_features(file_path, new_train_path,
new_test_path, ham_words_path,
                                               spam_words_path,
test_size=0.2, random_state=10):


    df = pa.read_csv(file_path)

    df_GF = df[["Tweet", "following", "followers", "actions", "is_retweet",
"location", "Type"]]

    le = LabelEncoder()
    df_update = pa.DataFrame.copy(df_GF)
    # for replace the location with 1 if exist and 0 if null
    df_update['location'] = df_update['location'].apply(lambda x: 1 if not
pa.isnull(x) else 0)
    # for replace the null with zero
    df_update['actions'] = df_update['actions'].replace(np.nan, 0)
    df_update['following'] = df_update['following'].replace(np.nan, 0)
    df_update['followers'] = df_update['followers'].replace(np.nan, 0)
    df_update['is_retweet'] = df_update['is_retweet'].replace(np.nan, 0)

    # for replace the ham with 0
    # and the spam with 1
    df_update['Type'] = le.fit_transform(df_update['Type'])

    # for get the features vectors for all tweet
    length_before_column = []
    length_after_column = []
    num_words_before_column = []
    num_words_after_column = []
    num_urls_column = []
    num_hashtags_column = []
    num_minations_column = []
    num_click_column = []
    swear_column = []
    root_column = []
    count = 0
    tweet_column = list(df_update["Tweet"])
```

```
    for tweet in tweet_column:
        lb, la, nwb, nwa, nu, nh, nm, nc, s, r = preprocessing(tweet)
        length_before_column.append(lb)
        length_after_column.append(la)
        num_words_before_column.append(nwb)
        num_words_after_column.append(nwa)
        num_urls_column.append(nu)
        num_hashtags_column.append(nh)
        num_minations_column.append(nm)
        num_click_column.append(nc)
        swear_column.append(s)
        root_column.append(r)
        print(count)
        count += 1

    print("features extraction done")

    new_csv = pa.DataFrame({"Type": df_update['Type'], "Tweet":
df_update['Tweet'], "roots": root_column,
                            "following": df_update['following'],
                            "followers": df_update['followers'], "actions":
df_update['actions'],
                            "is_retweet": df_update['is_retweet'],
                            "location": df_update['location'],
"length_before": length_before_column,
                            "length_after": length_after_column,
"num_words_before": num_words_before_column,
                            "num_words_after": num_words_after_column,
"num_urls": num_urls_column,
                            "num_hashtags": num_hashtags_column,
                            "num_minations": num_minations_column,
                            "num_click": num_click_column, "swear":
swear_column})

    # for split the data into training and testing csv files
    x = new_csv.drop('Type', axis=1)
    y = new_csv.Type

    x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=test_size, random_state=random_state)

    print('size of test dataset = {}, size of traing data = {}, percentage =
{}%'.format(len(x_test), len(x_train),

len(x_test) * 100 / (

len(x_test) + len(

x_train))))

    train_csv_file = pa.DataFrame(x_train)
    train_csv_file["Type"] = y_train

    test_csv_file = pa.DataFrame(x_test)
    test_csv_file["Type"] = y_test
```

```
    # for create the bias feature for all tweets
    spam_words, ham_words = categorize_words(train_csv_file)
    train_csv_file["bias"] = train_csv_file['roots'].apply(lambda b:
bias_feature(b, spam_words, ham_words))
    test_csv_file["bias"] = test_csv_file['roots'].apply(lambda b:
bias_feature(b, spam_words, ham_words))

    # for save the train and test csv files
    train_csv_file.to_csv(new_train_path, index=False)
    test_csv_file.to_csv(new_test_path, index=False)

    # for save the ham words in the file
    file_for_save_ham_words = open(ham_words_path, "wb")
    for element in ham_words:
        file_for_save_ham_words.write(bytes(element, 'utf-8'))
        file_for_save_ham_words.write(bytes(" ", 'utf-8'))
        file_for_save_ham_words.write(bytes(str(ham_words[element]), 'utf-
8'))
        file_for_save_ham_words.write(bytes("\n", 'utf-8'))
    file_for_save_ham_words.close()

    # for save the spam words in the file
    file_for_save_spam_words = open(spam_words_path, "wb")
    for element in spam_words:
        file_for_save_spam_words.write(bytes(element, 'utf-8'))
        file_for_save_spam_words.write(bytes(" ", 'utf-8'))
        file_for_save_spam_words.write(bytes(str(spam_words[element]), 'utf-
8'))
        file_for_save_spam_words.write(bytes("\n", 'utf-8'))
    file_for_save_spam_words.close()


# for show the information gain
def show_IG_for_features(x_train, y_train):
    # configure to select all features
    fs = SelectKBest(score_func=mutual_info_classif, k='all')
    # learn relationship from data
    fs.fit(x_train, y_train)

    for i in range(len(fs.scores_)):
        print('Feature[%d]: %s = %f' % (i, fs.feature_names_in_[i],
fs.scores_[i]))


# for print the information gain on the console
def print_ig_on_console(x_train, y_train):
    fs = SelectKBest(score_func=mutual_info_classif, k='all')
    # learn relationship from data
    fs.fit(x_train, y_train)

    for i in range(len(fs.scores_)):
        print('%s %f' % (fs.feature_names_in_[i], fs.scores_[i]))


# for plot all features information gain
def plot_information_gain(csv_file, x_train, y_train):
```

```
    importances = mutual_info_classif(x_train, y_train)
    feat_importances = pa.Series(importances,
csv_file.columns[2:len(csv_file.columns) - 1])
    feat_importances.plot(kind='barh', color='teal')
    plt.show()


# for get the text features from the tweet
def get_text_features(tweet, spam_words, ham_words):
    lb, la, nwb, nwa, nu, nh, nm, nc, s, r = preprocessing(tweet)
    bias = bias_feature(r, spam_words, ham_words)
    return lb, la, nwb, nwa, nu, nh, nm, nc, s, r, bias


# for print if the tweet is ham or spam
def print_model_result(model, features_type, spam_words, ham_words, tweet,
following=0, followers=0, actions=0,
                       is_retweet=0, location=""):
    if location == "":
        location = 0
    else:
        location = 1
    lb, la, nwb, nwa, nu, nh, nm, nc, s, r, bias = get_text_features(tweet,
spam_words, ham_words)
    features_vector = None
    if features_type == "text":
        features_vector = pa.DataFrame({"length_before": [lb],
"num_words_before": [nwb],
                                        "length_after": [la],
"num_words_after": [nwa], "num_urls": [nu],
                                        "num_hashtags": [nh],
"num_minations": [nm], "num_click": [nc], "swear": [s],
                                        "bias": [bias]})
    elif features_type == "best_text":
        features_vector = pa.DataFrame({"length_before": [lb],
"num_words_before": [nwb],
                                        "length_after": [la],
"num_words_after": [nwa]})
    elif features_type == "all":
        features_vector = pa.DataFrame({"following": [following],
"followers": [followers], "actions": [actions],
                                        "is_retweet": [is_retweet],
"location": [location],
                                        "length_before": [lb],
"num_words_before": [nwb], "length_after": [la],
                                        "num_words_after": [nwa], "num_urls":
[nu], "num_hashtags": [nh],
                                        "num_minations": [nm], "num_click":
[nc], "swear": [s], "bias": [bias]
                                        })
    elif features_type == "best_all":
        features_vector = pa.DataFrame({"following": [following],
"followers": [followers], "actions": [actions]})
    else:
```

```
        print("Wrong features type parameter")
    # print("Tweet: {0}\nResult: {1}".format(tweet,
model.predict(features_vector)))
    print(int(model.predict(features_vector)))


# first column should be Tweet
# second column should be following
# third column should be followers
# forth column should be actions
# fifth column should be is_retweet
# sixth column should be location
# seventh column should be Type
def read_from_test_file_with_doctor_features(test_file_path, spam_words,
ham_words):


    df = pa.read_csv(test_file_path)

    df_GF = df[["Tweet", "following", "followers", "actions", "is_retweet",
"location", "Type"]]

    le = LabelEncoder()
    df_update = pa.DataFrame.copy(df_GF)

    df_update['location'] = df_update['location'].apply(lambda x: 1 if not
pa.isnull(x) else 0)
    df_update['actions'] = df_update['actions'].replace(np.nan, 0)
    df_update['following'] = df_update['following'].replace(np.nan, 0)
    df_update['followers'] = df_update['followers'].replace(np.nan, 0)
    df_update['is_retweet'] = df_update['is_retweet'].replace(np.nan, 0)

    df_update['Type'] = le.fit_transform(df_update['Type'])

    length_before_column = []
    length_after_column = []
    num_words_before_column = []
    num_words_after_column = []
    num_urls_column = []
    num_hashtags_column = []
    num_minations_column = []
    num_click_column = []
    swear_column = []
    root_column = []

    count = 0

    tweet_column = list(df_update["Tweet"])
    for tweet in tweet_column:
        lb, la, nwb, nwa, nu, nh, nm, nc, s, r = preprocessing(tweet)
        length_before_column.append(lb)
        length_after_column.append(la)
        num_words_before_column.append(nwb)
        num_words_after_column.append(nwa)
        num_urls_column.append(nu)
        num_hashtags_column.append(nh)
```

```
        num_minations_column.append(nm)
        num_click_column.append(nc)
        swear_column.append(s)
        root_column.append(r)
        # print(count)
        count += 1

    # print("features extraction done")

    new_csv = pa.DataFrame({"Type": df_update['Type'], "Tweet":
df_update['Tweet'], "roots": root_column,
                            "following": df_update['following'],
                            "followers": df_update['followers'], "actions":
df_update['actions'],
                            "is_retweet": df_update['is_retweet'],
                            "location": df_update['location'],
"length_before": length_before_column,
                            "length_after": length_after_column,
"num_words_before": num_words_before_column,
                            "num_words_after": num_words_after_column,
"num_urls": num_urls_column,
                            "num_hashtags": num_hashtags_column,
                            "num_minations": num_minations_column,
                            "num_click": num_click_column, "swear":
swear_column})

    new_csv["bias"] = new_csv['roots'].apply(lambda b: bias_feature(b,
spam_words, ham_words))
    new_csv["bias"] = new_csv['roots'].apply(lambda b: bias_feature(b,
spam_words, ham_words))

    return new_csv


# first column should be Tweet
# second column should be Type
def read_from_test_file_just_tweets(test_file_path, spam_words, ham_words):

    le = LabelEncoder()

    df = pa.read_csv(test_file_path)

    df_GF = df[["Tweet", "Type"]]
    df_update = pa.DataFrame.copy(df_GF)
    df_update['Type'] = le.fit_transform(df_update['Type'])

    length_before_column = []
    length_after_column = []
    num_words_before_column = []
    num_words_after_column = []
    num_urls_column = []
    num_hashtags_column = []
    num_minations_column = []
    num_click_column = []
    swear_column = []
    root_column = []
```

```
    count = 0

    tweet_column = list(df_update["Tweet"])
    for tweet in tweet_column:
        lb, la, nwb, nwa, nu, nh, nm, nc, s, r = preprocessing(tweet)
        length_before_column.append(lb)
        length_after_column.append(la)
        num_words_before_column.append(nwb)
        num_words_after_column.append(nwa)
        num_urls_column.append(nu)
        num_hashtags_column.append(nh)
        num_minations_column.append(nm)
        num_click_column.append(nc)
        swear_column.append(s)
        root_column.append(r)
        # print(count)
        count += 1

    # print("features extraction done")

    new_csv = pa.DataFrame({"Type": df_update['Type'], "Tweet":
df_update['Tweet'], "roots": root_column,
                            "length_before": length_before_column,
                            "length_after": length_after_column,
"num_words_before": num_words_before_column,
                            "num_words_after": num_words_after_column,
"num_urls": num_urls_column,
                            "num_hashtags": num_hashtags_column,
                            "num_minations": num_minations_column,
                            "num_click": num_click_column, "swear":
swear_column})

    new_csv["bias"] = new_csv['roots'].apply(lambda b: bias_feature(b,
spam_words, ham_words))
    new_csv["bias"] = new_csv['roots'].apply(lambda b: bias_feature(b,
spam_words, ham_words))

    return new_csv


# for print the number of hams ans spams tweets in the csv file
def print_spams_qualities_info(csv_file_path):
    data = pa.read_csv(csv_file_path)

    spam = data[data.Type == "Spam"]
    No_spam = spam.shape[0]
    quality = data[data.Type == "Quality"]
    No_quality = quality.shape[0]
    Per_spam = No_spam / (No_spam + No_quality)

    print(data.info())
    print('spam = {}, quality = {} , Percentage of spam = {}
%'.format(No_spam, No_quality, Per_spam * 100))
```

```
# for print the mean for all features the the csv files group by type
def print_mean_group_by_type(csv_file_path, features_names):
    data = pa.read_csv(csv_file_path)
    for f in features_names:
        print(data.groupby('Type').mean()[f])


# for get the the classification model result
# and calculate the accuracy
# and precision and recall and f1 scores
def cal_scores(model, x, y_true):


    y_pred = model.predict(x)

    acc = accuracy_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)

    return acc, recall, precision, f1, y_pred


# for print
# the accuracy
# and precision and recall and f1 scores
def print_score(model, x, y_true):
    acc, recall, precision, f1, y_pred = cal_scores(model, x, y_true)

    print("Accuracy: ", np.round(acc, 2))
    print("Recall: ", np.round(recall, 2))
    print("Precision: ", np.round(precision, 2))
    print("F1: ", np.round(f1, 2))
    print("confusion_matrix: \n", confusion_matrix(y_true, y_pred))
    print("classification_report: \n", classification_report(y_true, y_pred))


# for print
# the accuracy
# and precision and recall and f1 scores
# on the console
def print_score_on_console(model, x, y_true):


    acc, recall, precision, f1, y_pred = cal_scores(model, x, y_true)

    print("accuracy %.2f" % acc)
    print("recall %.2f" % recall)
    print("precision %.2f" % precision)
    print("f1 %.2f" % f1)
    print("tp ", list(confusion_matrix(y_true, y_pred))[0][0])
    print("fp ", list(confusion_matrix(y_true, y_pred))[0][1])
    print("fn ", list(confusion_matrix(y_true, y_pred))[1][0])
    print("tn ", list(confusion_matrix(y_true, y_pred))[1][1])

# for create just text spam model
```

```
def create_text_model():
    global text_train_x, text_train_y, text_model
    # for make text_features model
    text_train_x, text_train_y = get_x_y(csv_train_file,
text_features_labels)
    text_model = tree_model(text_train_x, text_train_y)


# for read the selected features from the user and choose it for the model
def pick_selected_features(sel_features):
    global selected_features

    if sel_features == "af":
        selected_features = all_features_labels
    elif sel_features == "at":
        selected_features = text_features_labels
    elif sel_features == "ba":
        selected_features = best_features_labels
    elif sel_features == "bt":
        selected_features = best_text_features_labels


# for read the selected model from the user and choose it
def pick_selected_model(sel_model, x_train, y_train):
    global selected_model

    if sel_model == "tree":
        selected_model = tree_model(x_train, y_train)
    elif sel_model == "network":
        selected_model = network_model(x_train, y_train)
    elif sel_model == "naive_bias":
        selected_model = naive_bias_model(x_train, y_train)

# create system based on which selected from user
def create_the_system(model, sel_features):
    global selected_doctor_form_file, selected_net_form_file, selected_model,
selected_features
    global x_train, y_train, x_test, y_test
    global xn, yn, xd, yd

    pick_selected_features(sel_features)

    x_train, y_train = get_x_y(csv_train_file, selected_features)
    x_test, y_test = get_x_y(csv_test_file, selected_features)

    pick_selected_model(model, x_train, y_train)




# for cleaning the text from punctuations and stopwords
def text_processing(text):


    text = [char for char in text if char not in string.punctuation]
    text = ''.join(text)
```

```
    text = [word for word in text.split() if word.lower() not in
stopwords.words("english")]

    return text


# for create naive bias model with multiple features
def create_naive_bias_mul_features_model_then_test_tweet(tweet):


    text = pa.DataFrame({"Tweet": [tweet]})
    tweets_types_columns = csv_train_file[["Tweet", "Type"]]
    tweets_file = pa.DataFrame.copy(tweets_types_columns)

    le = LabelEncoder()

    # for replace the ham with 0
    # and the spam with 1
    tweets_file['Type'] = le.fit_transform(tweets_file['Type'])

    count_vector = CountVectorizer(analyzer=text_processing)

    training_data = count_vector.fit_transform(tweets_file['Tweet'])
    test_text = count_vector.transform(text)

    naive_bias_text_model = MultinomialNB()
    naive_bias_text_model.fit(training_data, tweets_file['Type'])

    print(int(naive_bias_text_model.predict(test_text)))

# for handling the arguments which request from the java code
# and make response for it by print on the console
def handler(args):
    global xd, yd, xn, yn
    global selected_doctor_form_file, selected_net_form_file

    args = list(map(str, args))
    # following=0, followers=0, actions=0,
    #                   is_retweet=0, location=""
    # when the user enter tweet and want to check it if spam or ham
    # you should enter
    # args[2] --> tf: use text features model, nbf: use multi features by
naive bias, af: use all features tree model
    # args[3] = tweet
    # args[4,5,6,7,8] : just if you pass "af" --> 4: following, 5: followers,
6: actions, 7: is_retweet, 8: location
    if args[1] == "tweet":
        if args[2] == "tf":
            create_text_model()
            print_model_result(text_model, "text", spams_words,
qualities_words, tweet=args[3])
        elif args[2] == "nbf":
            create_naive_bias_mul_features_model_then_test_tweet(args[3])
        elif args[2] == "af":
            create_the_system("tree", "af")
            print_model_result(selected_model, "all", spams_words,
```

```
qualities_words, tweet=args[3], following=args[4],
                                   followers=args[5], actions=args[6],
is_retweet=args[7], location=args[8])

    # when the user want to know the information gain for the features
    # you should pass
    # args[2] = the features_label selected by user (af, at, ba, bt)
    elif args[1] == "ig":
        pick_selected_features(args[2])
        x_ig, y_ig = get_x_y(csv_train_file, selected_features)
        print_ig_on_console(x_ig, y_ig)

    # when the user want to show the accuracy of the test file
    # you should pass
    # args[2] = the model selected by user (tree, network, naive_bias)
    # args[3] = the features_label selected by user (af, at, ba, bt)
    elif args[1] == "atf":
        create_the_system(args[2], args[3])
        print_score_on_console(selected_model, x_test, y_test)

    # when the user want to show the accuracy of the sample file
    # you should pass
    # args[2] = the model selected by user (tree, network, naive_bias)
    # args[3] = the features_label selected by user (af, at, ba, bt)
    # args[4] = the path of the sample file
    elif args[1] == "adf":
        create_the_system(args[2], args[3])
        selected_doctor_form_file =
read_from_test_file_with_doctor_features(args[4], spams_words,
qualities_words)
        xd, yd = get_x_y(selected_doctor_form_file, selected_features)
        print_score_on_console(selected_model, xd, yd)

    # when the user want to show the accuracy of the net file
    # you should pass
    # args[2] = the path of the tweets file
    # not required to the selected model and features selected by user
    elif args[1] == "anf":
        create_text_model()
        selected_net_form_file = read_from_test_file_just_tweets(args[2],
spams_words, qualities_words)
        xn, yn = get_x_y(selected_net_form_file, text_features_labels)
        print_score_on_console(text_model, xn, yn)

handler(sys.argv)
```

Java Code (Front-End)

Main Class

```java
package sample;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class Main extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception{
        Parent root = FXMLLoader.load(getClass().getResource("sample.fxml"));
        primaryStage.setTitle("Spam Detector");
        primaryStage.setScene(new Scene(root));
        primaryStage.show();
    }


    public static void main(String[] args) {
        launch(args);
    }
}
```

Controller Class

```
package sample;

import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.chart.BarChart;
import javafx.scene.chart.XYChart;
import javafx.scene.control.*;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.TextField;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.stage.FileChooser;
import javafx.stage.Stage;

import java.io.BufferedReader;
import java.io.File;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.URL;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.MissingFormatArgumentException;
import java.util.ResourceBundle;


public class Controller implements Initializable {

    HashMap<String,Boolean>hashMap = new HashMap<>() ;//hasp map tp prevent
the user from plotting a chart that already plotted
    String model = "" ;//for the chosen model
    String feature ="" ; //for the chosen feature
    boolean model_selected =false ;//to check if the user select model or not
    boolean feature_selected=false ;//to check if the user select a feature
or not
    ArrayList <String>results = new ArrayList<>() ;//to store the result
string
    ObservableList<table> list = FXCollections.observableArrayList();//for
the table

    @FXML
    private ComboBox chosefeature;

    @FXML
    private TextField resulttext;
    @FXML
    private TextField acurecy;
    @FXML
```

```
    private TextField tweettext;
    @FXML
    private TableColumn<table, String> actualham;

    @FXML
    private TableColumn<table, String> actualspam;
    @FXML
    private TableColumn<table, String> ccc;
    @FXML
    private TableView<table> table;
    @FXML
    private TextField precsion;

    @FXML
    private TextField recall;


    @FXML
    void chsefeatureaction(ActionEvent event) {
        //actions in combobox for features
        if
(chosefeature.getSelectionModel().getSelectedItem().toString().equals("All
Features")){
            feature ="af" ;
        }
        else if
(chosefeature.getSelectionModel().getSelectedItem().toString().equals("Best
Features")){
            feature="ba" ;
        }
        else
if(chosefeature.getSelectionModel().getSelectedItem().toString().equals("Text
Features")){
            feature="at" ;
        }
        else{
            feature ="bt" ;
        }
        feature_selected =true ;
    }

        @FXML
        private TextField browsesamplesdatatext;

        @FXML
        private TextField browsetweetdatattext;

        @FXML
        void brosesampledata(ActionEvent event) {
            //browse sample data button
            FileChooser fileChooserShares = new FileChooser();
            fileChooserShares.setTitle("Select project file ");
            fileChooserShares.getExtensionFilters().addAll(
                    new FileChooser.ExtensionFilter("Text Files", "*.csv"));

            File selectedFile = fileChooserShares.showOpenDialog(null);
```

```
            if (String.valueOf(selectedFile).equals("null")) {
                return;
            } else {
                browsesamplesdatatext.setText(selectedFile.toString());
            }
        }

        @FXML
        void browsetweetdata(ActionEvent event) {
            //browse tweet data button
            FileChooser fileChooserShares = new FileChooser();
            fileChooserShares.setTitle("Select project file ");
            fileChooserShares.getExtensionFilters().addAll(
                    new FileChooser.ExtensionFilter("Text Files", "*.csv"));

            File selectedFile = fileChooserShares.showOpenDialog(null);
            if (String.valueOf(selectedFile).equals("null")) {
                return;
            } else {
                browsetweetdatattext.setText(selectedFile.toString());
            }
        }



    @FXML
    void viewtweettestpage(ActionEvent event) throws IOException {
            //to view the page to test a single tweet
        Parent root = FXMLLoader.load(getClass().getResource("tweet.fxml"));
        Stage stage3 = new Stage();
        stage3.setScene(new Scene(root));
        stage3.setTitle("Single Tweet Test");
        stage3.showAndWait();
    }


    @FXML
    private ComboBox chosemodel;

    @Override
    public void initialize(URL url, ResourceBundle resourceBundle) {
        //this function run when the program start and set the items in both
combo box and set all the feature as not plotted
        //set the items in both combobox
        ObservableList<String> models =
FXCollections.observableArrayList("Decision Tree" ,"Naive bias","Neural
Network") ;
        chosemodel.setItems(models);
        ObservableList<String> Typefeature =
FXCollections.observableArrayList("All Features" ,"Best Features","Text
Features" ,"Best Text Features") ;
        //set all the features as not plotted
        chosefeature.setItems(Typefeature);
        hashMap.put("All Features",false) ;
        hashMap.put("Best Features",false);
        hashMap.put("Text Features",false);
```

```
        hashMap.put("Best Text Features",false) ;

    }

    @FXML
    void chosemodelaction(ActionEvent event) {
        //for chosen item from the model combobox
        if
(chosemodel.getSelectionModel().getSelectedItem().toString().equals("Decision
Tree")){
            model = "tree" ;
        }
        else if
(chosemodel.getSelectionModel().getSelectedItem().toString().equals("Naive
bias")){
            model ="naive_bias" ;
        }
        else {
            model = "network" ;
        }
        model_selected = true ;
    }


    @FXML
    void TestingdatatAction(ActionEvent event) throws IOException,
InterruptedException {
        try{
            //check if the user chose the model and feature
            if (!checkvalid() )
                throw new MissingFormatArgumentException("please choose the
model and features");
            list.clear();
            results.clear();
           //creat the process builder and set the arguments and the path for
executable file on it
            ProcessBuilder builder = new
ProcessBuilder("C:\\Users\\Ibrah\\Desktop\\Machinelearning
Script\\dist\\main.exe","atf",model,feature) ;
           //run the process builder
            Process p = builder.start() ;
            p.waitFor() ;
            //read the data from python script console it by buffer reader
using process input stream
            BufferedReader b = new BufferedReader(new
InputStreamReader(p.getInputStream()));
            ////read the errors from python script console it by buffer
reader using process error stream
            BufferedReader b1 = new BufferedReader(new
InputStreamReader(p.getErrorStream()));
            String line = null;
            String line1 = null;
            //read the data line by line
            while ((line = b.readLine()) !=null){
                results.add(line) ;
            }
```

```
            //read the errors line by line
            while ((line1 = b1.readLine()) !=null){
                System.out.println(line1);
            }

            String[]res = new String[3] ;

            for (int i = 0 ; i < 3 ; i++){
                String s= results.get(i) ;
                String []s1 = s.split(" ") ;
                res[i] = s1[1] ;
            }
            //split the string and get the results
            acurecy.setText(res[0]);
            recall.setText(res[1]);
            precsion.setText(res[2]);
            String []conf = new String[4] ;
            for (int i = 4,j=0 ; i < 8 ;i++,j++){
                String s= results.get(i) ;
                String []s1 = s.split(" ") ;
                conf[j] = s1[2] ;
            }
            //print the confusion matrix in the table
            list.add(new table("Predicted Spam" ,conf[0],conf[1])) ;
            list.add(new table("Pridected Ham" ,conf[2],conf[3])) ;
            ccc.setCellValueFactory(new PropertyValueFactory<>("c1"));
            actualspam.setCellValueFactory(new PropertyValueFactory<>("c2"));
            actualham.setCellValueFactory(new PropertyValueFactory<>("c3"));
            table.setItems(list);
        }
        //show the thrown exception using alert
        catch(MissingFormatArgumentException e){
            Alert alertCreat = new Alert(Alert.AlertType.ERROR);
            alertCreat.setTitle("Error");
            alertCreat.setHeaderText(null);
            alertCreat.setContentText(e.getMessage());
            alertCreat.showAndWait();
        }

    }
    @FXML
    void samplesdaataresults(ActionEvent event) throws IOException,
InterruptedException {
        try{
            //check that the user choose the feature and model and browse the
file
            if (!checkvalid() || browsesamplesdatatext.getText().equals(""))
                throw new MissingFormatArgumentException("please browse file
for sample data or choose model and feature");
            results.clear();
            list.clear();

            ProcessBuilder builder = new
ProcessBuilder("C:\\Users\\Ibrah\\Desktop\\Machinelearning
Script\\dist\\main.exe","adf",model,feature,browsesamplesdatatext.getText())
;
```

```
            Process p = builder.start() ;
            p.waitFor() ;
            BufferedReader b = new BufferedReader(new
InputStreamReader(p.getInputStream()));
            BufferedReader b1 = new BufferedReader(new
InputStreamReader(p.getErrorStream()));
            String line = null;
            String line1 = null;

            while ((line = b.readLine()) !=null){
                results.add(line) ;
            }
            while ((line1 = b1.readLine()) !=null){
                System.out.println(line1);
            }


            String[]res = new String[3] ;

            for (int i = 0 ; i < 3 ; i++){
                String s= results.get(i) ;
                String []s1 = s.split(" ") ;
                res[i] = s1[1] ;
            }
            acurecy.setText(res[0]);
            recall.setText(res[1]);
            precsion.setText(res[2]);
            String []conf = new String[4] ;
            for (int i = 4,j=0 ; i < 8 ;i ++,j++){
                String s= results.get(i) ;
                String []s1 = s.split(" ") ;
                conf[j] = s1[2] ;
            }
            list.add(new table("Predicted Spam" ,conf[0],conf[1])) ;
            list.add(new table("Pridected Ham" ,conf[2],conf[3])) ;
            ccc.setCellValueFactory(new PropertyValueFactory<>("c1"));
            actualspam.setCellValueFactory(new PropertyValueFactory<>("c2"));
            actualham.setCellValueFactory(new PropertyValueFactory<>("c3"));
            table.setItems(list);
        }
        catch (MissingFormatArgumentException e){
            Alert alertCreat = new Alert(Alert.AlertType.ERROR);
            alertCreat.setTitle("Error");
            alertCreat.setHeaderText(null);
            alertCreat.setContentText(e.getMessage());
            alertCreat.showAndWait();
        }

    }
    boolean checkvalid(){
        if (model_selected && feature_selected)
            return true ;
        return false ;
    }
    @FXML
    void tweetsfileresults(ActionEvent event) {
```

```
        try{
            if (browsetweetdatattext.getText().equals(""))
                throw new MissingFormatArgumentException("please browse file
for sample data or choose model and feature");
            results.clear();
            list.clear();
            ProcessBuilder builder = new
ProcessBuilder("C:\\Users\\Ibrah\\Desktop\\Machinelearning
Script\\dist\\main.exe","anf",browsetweetdatattext.getText()) ;
            Process p = builder.start() ;
            p.waitFor() ;
            BufferedReader b = new BufferedReader(new
InputStreamReader(p.getInputStream()));
            BufferedReader b1 = new BufferedReader(new
InputStreamReader(p.getErrorStream()));
            String line = null;
            String line1 = null;

            while ((line = b.readLine()) !=null){
               results.add(line) ;
            }
            while ((line1 = b1.readLine()) !=null){
                System.out.println(line1);
            }


            String[]res = new String[3] ;

            for (int i = 0 ; i < 3 ; i++){
                String s= results.get(i) ;
                String []s1 = s.split(" ") ;
                res[i] = s1[1] ;
            }
            acurecy.setText(res[0]);
            recall.setText(res[1]);
            precsion.setText(res[2]);
            String []conf = new String[4] ;
            for (int i = 4,j=0 ; i < 8 ;i ++,j++){
                String s= results.get(i) ;
                String []s1 = s.split(" ") ;
                conf[j] = s1[2] ;
            }
            list.add(new table("Predicted Spam" ,conf[0],conf[1])) ;
            list.add(new table("Pridected Ham" ,conf[2],conf[3])) ;
            ccc.setCellValueFactory(new PropertyValueFactory<>("c1"));
            actualspam.setCellValueFactory(new PropertyValueFactory<>("c2"));
            actualham.setCellValueFactory(new PropertyValueFactory<>("c3"));
            table.setItems(list);
        }
        catch (MissingFormatArgumentException | IOException |
InterruptedException e){
            Alert alertCreat = new Alert(Alert.AlertType.ERROR);
            alertCreat.setTitle("Error");
            alertCreat.setHeaderText(null);
            alertCreat.setContentText(e.getMessage());
            alertCreat.showAndWait();
```

```
        }
    }
    @FXML
    private BarChart<String, Double> barchart;
    @FXML
    void plotig(ActionEvent event) {
        try{
            if (!feature_selected)
                throw new MissingFormatArgumentException("please choose the
features");
            else if
(hashMap.get(chosefeature.getSelectionModel().getSelectedItem().toString()))
                throw new MissingFormatArgumentException("the chart is
already plotted") ;
            results.clear();

            ProcessBuilder builder = new
ProcessBuilder("C:\\Users\\Ibrah\\Desktop\\Machinelearning
Script\\dist\\main.exe"
                    ,"ig",feature) ;
            Process p = builder.start() ;
            p.waitFor() ;
            BufferedReader b = new BufferedReader(new
InputStreamReader(p.getInputStream()));
            BufferedReader b1 = new BufferedReader(new
InputStreamReader(p.getErrorStream()));
            String line = null;
            String line1 = null;

            while ((line = b.readLine()) !=null){
                results.add(line) ;
            }
            while ((line1 = b1.readLine()) !=null){
                System.out.println(line1);
            }



            XYChart.Series<String,Double> series = new XYChart.Series<>() ;


series.setName(chosefeature.getSelectionModel().getSelectedItem().toString())
;

hashMap.put(chosefeature.getSelectionModel().getSelectedItem().toString(),tru
e);
            for (int i =0 ; i < results.size();i++){
                String[]s = results.get(i).split(" ");
                double ig = Double.parseDouble(s[1]) ;
                series.getData().add(new XYChart.Data<>(s[0],ig));
            }
            barchart.getData().add(series) ;
        }
        catch (MissingFormatArgumentException | IOException |
InterruptedException e){
            Alert alertCreat = new Alert(Alert.AlertType.ERROR);
```

```
            alertCreat.setTitle("Error");
            alertCreat.setHeaderText(null);
            alertCreat.setContentText(e.getMessage());
            alertCreat.showAndWait();
        }
    }

}
```

Table Class

```
package sample;

public class table {
    String c1 ;
    String c2 ;
    String c3 ;

    public table(String c1, String c2, String c3) {
        this.c1 = c1;
        this.c2 = c2;
        this.c3 = c3;
    }

    public String getC1() {
        return c1;
    }

    public void setC1(String c1) {
        this.c1 = c1;
    }

    public String getC2() {
        return c2;
    }

    public void setC2(String c2) {
        this.c2 = c2;
    }

    public String getC3() {
        return c3;
    }

    public void setC3(String c3) {
        this.c3 = c3;
    }
}
```

Tweet Class

```
package sample;

import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.Alert;
import javafx.scene.control.ComboBox;
import javafx.scene.control.TextField;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.URL;
import java.util.MissingFormatArgumentException;
import java.util.ResourceBundle;

public class Tweet implements Initializable {



        String feature_selected ="" ;
        boolean fs =false ;

        @FXML
        private ComboBox Features;

        @FXML
        private TextField actions;

        @FXML
        private TextField followers;

        @FXML
        private TextField following;

        @FXML
        private TextField isretweet;

        @FXML
        private TextField location;

        @FXML
        private TextField result;

        @FXML
        private TextField tweettext;

        @FXML
        void FeaturesActions(ActionEvent event) {
            if
(Features.getSelectionModel().getSelectedItem().toString().equals("All
```

```
Features"))
                feature_selected = "af" ;
            else if
(Features.getSelectionModel().getSelectedItem().toString().equals("Text
Features"))
               feature_selected="tf" ;
            else
                feature_selected="nbf" ;
            fs = true ;
        }

        @FXML
        void resultaction(ActionEvent event) {
            try
            {
                if (!fs)
                    throw new MissingFormatArgumentException("you should
choose the feature") ;
                else if
(Features.getSelectionModel().getSelectedItem().toString().equals("All
Features"))
                {
                    if (tweettext.getText().equals("") ||
followers.getText().equals("") ||
                    following.getText().equals("") ||
isretweet.getText().equals("")||
                    location.getText().equals("") ||
actions.getText().equals(""))
                        throw new MissingFormatArgumentException("you should
fill all the fields") ;
                    else if (!isretweet.getText().equals("0") &&
!isretweet.getText().equals("1"))
                        throw new MissingFormatArgumentException("is retweet
should be 0 or 1 only") ;
                    else if (!check_is_digits())
                        throw new MissingFormatArgumentException("follower
and following and actions should be only digits") ;
                    else {
                        ProcessBuilder builder =new
ProcessBuilder("C:\\Users\\Ibrah\\Desktop\\Machinelearning
Script\\dist\\main.exe",

"tweet",feature_selected,tweettext.getText(),following.getText(),

followers.getText(),actions.getText(),isretweet.getText(),location.getText())
;

                        Process p = builder.start() ;
                        p.waitFor() ;
                        BufferedReader b = new BufferedReader(new
InputStreamReader(p.getInputStream()));
                        BufferedReader b1  = new BufferedReader(new
InputStreamReader(p.getErrorStream()));
                        String line = null;
                        String line1 = null;
```

```
                        String s ="" ;
                        while ((line = b.readLine()) !=null){
                            s+=line ;
                        }
                        if (s.equals("0"))
                            result.setText("Ham");
                        else
                            result.setText("Spam");
                        while ((line1 = b1.readLine()) !=null){
                            System.out.println(line1);
                        }
                    }
                }
                else if
(Features.getSelectionModel().getSelectedItem().toString().equals("Text
Features")){
                    if (tweettext.getText().equals(""))
                        throw new MissingFormatArgumentException("you should
enter the tweet text") ;
                    ProcessBuilder builder =new
ProcessBuilder("C:\\Users\\Ibrah\\Desktop\\Machinelearning
Script\\dist\\main.exe"
                            ,"tweet",feature_selected,tweettext.getText());

                    Process p = builder.start() ;
                    p.waitFor() ;
                    BufferedReader b  = new BufferedReader(new
InputStreamReader(p.getInputStream()));
                    BufferedReader b1  = new BufferedReader(new
InputStreamReader(p.getErrorStream()));
                    String line = null;
                    String line1 = null;
                    String s ="" ;
                    while ((line = b.readLine()) !=null){
                        s+=line ;
                    }
                    if (s.equals("0"))
                        result.setText("Ham");
                    else
                        result.setText("Spam");
                    while ((line1 = b1.readLine()) !=null){
                        System.out.println(line1);
                    }
                }
                else if
(Features.getSelectionModel().getSelectedItem().toString().equals("Naive Bias
Features")){

                    if (tweettext.getText().equals(""))
                        throw new MissingFormatArgumentException("you should
enter the tweet text") ;
                    ProcessBuilder builder =new
ProcessBuilder("C:\\Users\\Ibrah\\Desktop\\Machinelearning
Script\\dist\\main.exe"
                            ,"tweet",feature_selected,tweettext.getText());
```

```
                    Process p = builder.start() ;
                    p.waitFor() ;
                    BufferedReader b = new BufferedReader(new
InputStreamReader(p.getInputStream()));
                    BufferedReader b1  = new BufferedReader(new
InputStreamReader(p.getErrorStream()));
                    String line =null;
                    String line1 = null;
                    String s ="";
                    while ((line = b.readLine()) !=null){
                            s+=line ;
                    }
                    if (s.equals("0"))
                        result.setText("Ham");
                    else
                        result.setText("Spam");
                    while ((line1 = b1.readLine()) !=null){
                        System.out.println(line1);
                    }

                }
            }
            catch (MissingFormatArgumentException | IOException |
InterruptedException e){
                Alert alertCreat = new Alert(Alert.AlertType.ERROR);
                alertCreat.setTitle("Error");
                alertCreat.setHeaderText(null);
                alertCreat.setContentText(e.getMessage());
                alertCreat.showAndWait();
            }
        }

    @Override
    public void initialize(URL url, ResourceBundle resourceBundle) {
        ObservableList<String> Typefeature =
FXCollections.observableArrayList("All Features" ,"Text Features" ,"Naive
Bias Features") ;
        Features.setItems(Typefeature);
    }
    boolean check_is_digits(){
        if (!following.getText().matches("[0-9]+") &&
!followers.getText().matches("[0-9]+")
        && actions.getText().matches("[0-9]+"))
            return false ;
        return true ;
    }
}
```