Faculty of Engineering and Technology

Department of Electrical and Computer Engineering

ENCS 434

Artificial Intelligence

Project 2

Spam Review Detection

Prepared By:

Name: Baraa Atta. ID: 1180445

Name: Ahmad Hamad. ID: 1180060

Instructor: Mr. Aziz Qaroush

Section: 1

Date: 29/1/2021

Table of Contents

List of Figures:

Introduction:

The Spam Review Detection is a well-known problem that requires taking a set of previous reviews and use them to specify the next reviews whether they are spam or not using a pre-defined machine learning model.

First of all, the formalization of the problem starts by determining the feature vectors to be used to build the knowledge space of the problem. In fact, in problems like this we can find a lot of feature vectors but the real challenge is to choose the minimum number of them with the highest accuracy to reduce the time needed in training and testing as much as possible.

Feature extraction:

Regarding to the feature vectors we saw that the most logical ones to take are the following:

1- review content: this is the most important feature vector because it contains the content of the review, but the problem now is that the training method takes numeric values only, so we used a library called bag of words which take all the words from all the reviews with skipping the ones like (a, an, the ….) then it assigns how many times each word is repeated in each review. Like this we transformed the string content to numeric one.

2- Rating: which gives a high indication about the impression of the reviewer and it is already a numeric value so no need to convert to anything.

3- usfulCount: which can indicate also the whole impression of the users to that review and It is also a numeric data so no need for conversion.

4- First count: which will indicate how speed is the user to write a review like for examples some users give a bad review very early to Mislead the other visitors.

5- Review count: which can help the model to know the most frequent users writing reviews and then determine their impression.

6- Date: we extracted from it if the review was updated or not, the day, month and year at which the review was published.

The used model:

After determining the required feature vectors, we need the chose a model in order to train it on the data.

To be honest we tried different types of models like the naïve bias and the decision tree but the one that gives us the highest accuracy and F1-SCORE was the neural networks.

The neural network was built by specifying firstly the number of layers and the number pf neurons for each layer and lastly the number of eBooks to train the model on.

The number of lyres we chose was 3 which are the input layer with 6 neurons because we chose a 6-feature vector, a hidden lyre with 10 neurons (the number was got experimentally) and finally the output layer with one neuron because we have only a one output. Also, it is worth to say that the activation function used in all the layers was the sigmoid although the output layer should be a unit step function, but because python doesn’t contain it, we chose to use a sigmoid and approximate the output to one if it is more than 0.5 and zero otherwise.

Dealing with missing values:

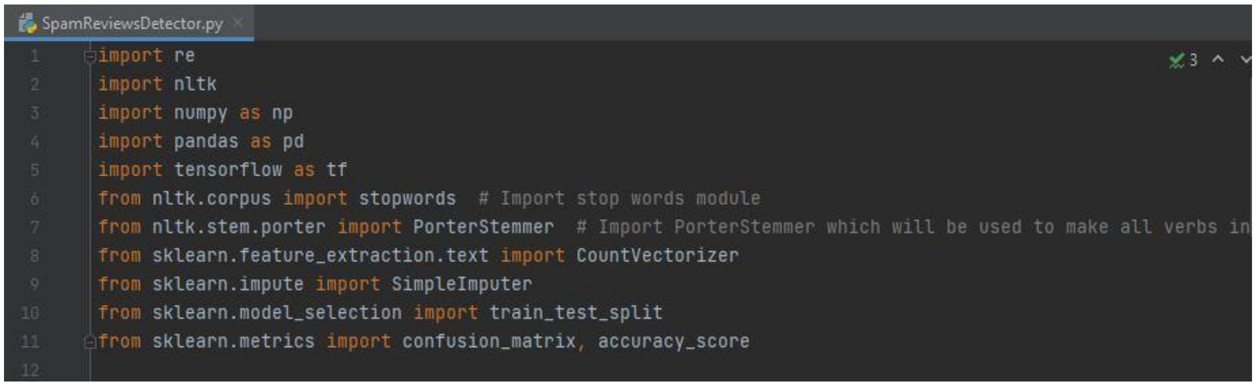
When scanning the data realized that some filed were missing which can cause problems during the training process, so we decided to replace the missing values with the most frequent one in that columns in order to reduce the error resulting error due to them as much as possible.

Taking a set of the data and determine the ratio for training and testing:

We must know that the data given was very huge and our laptops were unable to deal with them all, so we take a sample of 50000 rows of them and take 80% of them as training set and the remaining 20% as a testing set.

Code Explanation:

The first section shown at **Figure 1** shows the modules that are used in this project.

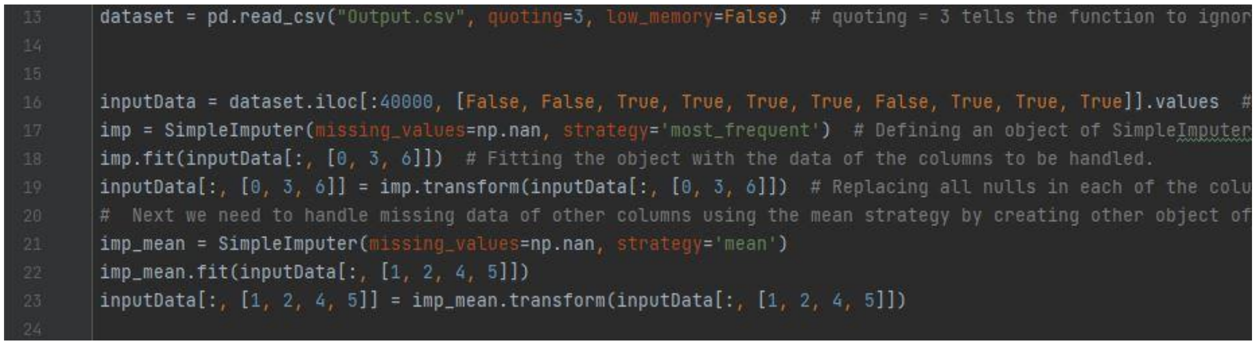


```
import re
import nltk
import numpy as np
import pandas as pd
import tensorflow as tf
from nltk.corpus import stopwords  # Import stop words module
from nltk.stem.porter import PorterStemmer  # Import PorterStemmer which will be used to make all verbs in
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score

```

Figure 1: Code Part 1

We used **re** to help us in removing all special characters and numbers in contents of reviews, **NumPy** for dealing with arrays, **pandas** for reading the CSV file, **TensorFlow** to use the Artificial Neural Network, **nltk** to get the stop words module and the PorterStemmer class that will be used to make all verbs in the present tense to avoid repeating the same verb more than once and **scikit learn** to use the bag of words model for the NLP part, deal with missing data, splitting the dataset to training and testing sets and compute the confusion matrix.

Second part in **Figure 2** shows the code used to read the dataset and dealing with missing data in it.



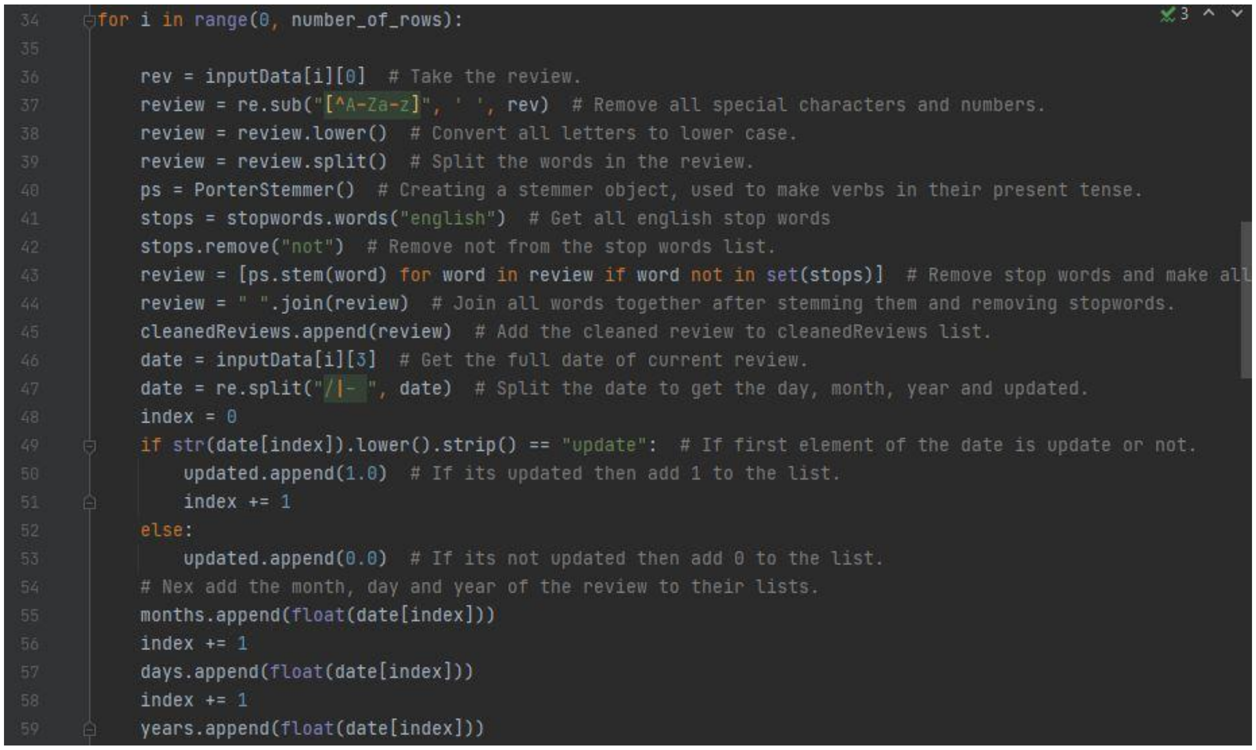
```
dataset = pd.read_csv("Output.csv", quoting=3, low_memory=False)  # quoting = 3 tells the function to ignor


inputData = dataset.iloc[:40000, [False, False, True, True, True, True, False, True, True, True]].values  #
imp = SimpleImputer(missing_values=np.nan, strategy='most_frequent')  # Defining an object of SimpleImputer
imp.fit(inputData[:, [0, 3, 6]])  # Fitting the object with the data of the columns to be handled.
inputData[:, [0, 3, 6]] = imp.transform(inputData[:, [0, 3, 6]])  # Replacing all nulls in each of the colu
#  Next we need to handle missing data of other columns using the mean strategy by creating other object of
imp_mean = SimpleImputer(missing_values=np.nan, strategy='mean')
imp_mean.fit(inputData[:, [1, 2, 4, 5]])
inputData[:, [1, 2, 4, 5]] = imp_mean.transform(inputData[:, [1, 2, 4, 5]])

```

Figure 2: Code Part 2

First line reads the entire dataset, the quoting parameter is set to 3 to ignore any quotations in the dataset. Next, we extracted what features we need from the dataset using **iloc** that is used with data frames and returns an array, the features are reviewContent, rating, usefulCount, date, firstCount, reviewCount and filtered. Then, we use SimpleImputer object to deal with missing values in the reviewContent, date and filtered using the most frequent strategy which simply replaces any missing value with the most frequent element in its column (feature).

After that, another SimpleImputer object is used to replace the missing values in the remaining features with the mean of each feature.

Part 3 of the code shown in **Figure 3** shows extracting features from texts (reviewContent and date).

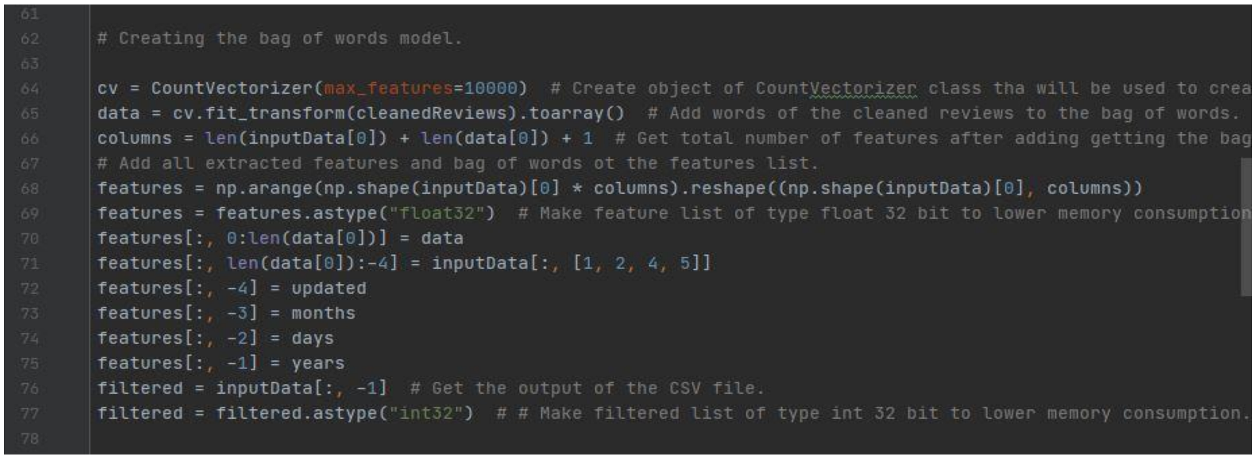


```python
for i in range(0, number_of_rows):

    rev = inputData[i][0]  # Take the review.
    review = re.sub("[^A-Za-z]", ' ', rev)  # Remove all special characters and numbers.
    review = review.lower()  # Convert all letters to lower case.
    review = review.split()  # Split the words in the review.
    ps = PorterStemmer()  # Creating a stemmer object, used to make verbs in their present tense.
    stops = stopwords.words("english")  # Get all english stop words
    stops.remove("not")  # Remove not from the stop words list.
    review = [ps.stem(word) for word in review if word not in set(stops)]  # Remove stop words and make all
    review = " ".join(review)  # Join all words together after stemming them and removing stopwords.
    cleanedReviews.append(review)  # Add the cleaned review to cleanedReviews list.
    date = inputData[i][3]  # Get the full date of current review.
    date = re.split("/|- ", date)  # Split the date to get the day, month, year and updated.
    index = 0
    if str(date[index]).lower().strip() == "update":  # If first element of the date is update or not.
        updated.append(1.0)  # If its updated then add 1 to the list.
        index += 1
    else:
        updated.append(0.0)  # If its not updated then add 0 to the list.
    # Nex add the month, day and year of the review to their lists.
    months.append(float(date[index]))
    index += 1
    days.append(float(date[index]))
    index += 1
    years.append(float(date[index]))
```

Figure 3: Code Part 3

Here, we loop over the entire read dataset to deal with texts in it, first the review is stored in a variable **rev** then all special characters are removed from the review, it's converted to lowercase letters and splitted to get all words in it. Then, a PorterStemmer object is created and the English stop words are stored in a list called stops, not is removed from stops as it may be helpful in the review, then a loop in all words is done it excludes stop words from the review and reforms all verbs to their present tense, after that all remaining words are joined together with a space between them to get the cleaned review and push it to a list called cleanedReviews. Finally, the date is splitted and checked if there is a word update in it then current review marked as updated otherwise its marked not updated and the day, month and the year are extracted and pushed to their suitable lists.

Next part shown in **Figure 4** shows the creation of the bag of words model and concatenating all features together in a single 2D array as well as creating a list for the output filtered.



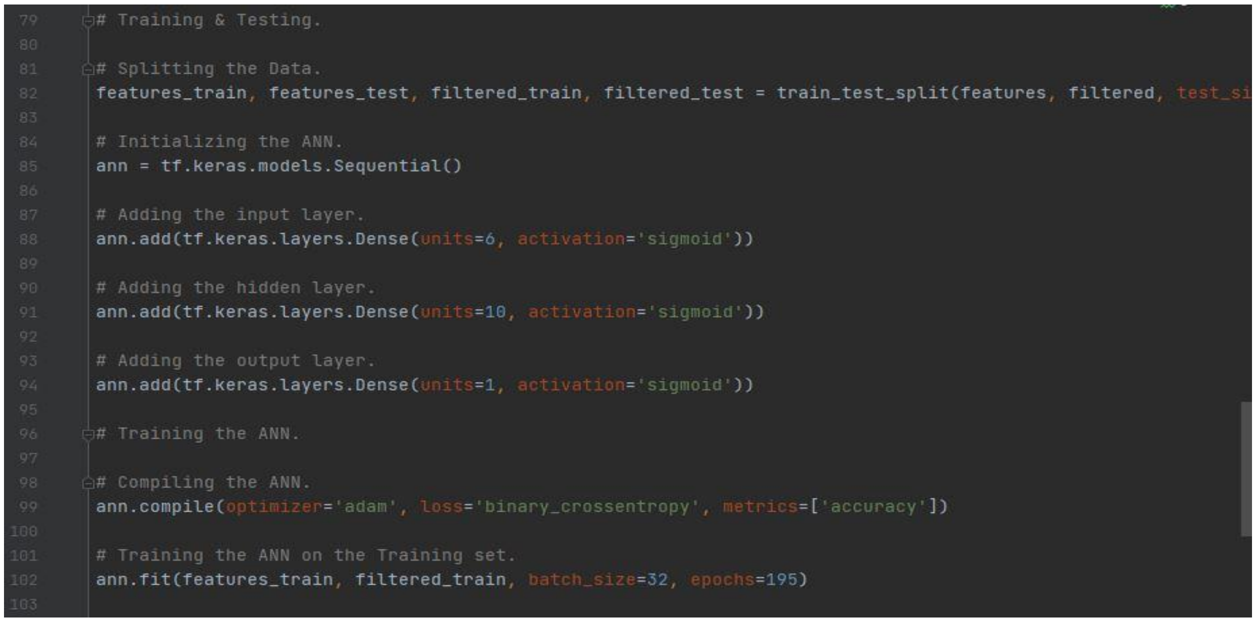
```python

# Creating the bag of words model.

cv = CountVectorizer(max_features=10000)  # Create object of CountVectorizer class tha will be used to crea
data = cv.fit_transform(cleanedReviews).toarray()  # Add words of the cleaned reviews to the bag of words.
columns = len(inputData[0]) + len(data[0]) + 1  # Get total number of features after adding getting the bag
# Add all extracted features and bag of words ot the features list.
features = np.arange(np.shape(inputData)[0] * columns).reshape((np.shape(inputData)[0], columns))
features = features.astype("float32")  # Make feature list of type float 32 bit to lower memory consumption
features[:, 0:len(data[0])] = data
features[:, len(data[0]):-4] = inputData[:, [1, 2, 4, 5]]
features[:, -4] = updated
features[:, -3] = months
features[:, -2] = days
features[:, -1] = years
filtered = inputData[:, -1]  # Get the output of the CSV file.
filtered = filtered.astype("int32")  # # Make filtered list of type int 32 bit to lower memory consumption.

```

Figure 4: Code Part 4

The bag of words model is created using an object of the CounterVectorizer class, the maximum number of words is specified to 10000 words, so the bag of words will take 9999 unique words and the last place is designated for any other words that isn't in the 9999 ones.

Then the maximum number of features (columns) in the features array is calculated by adding the number of unique words in the bag of words model with the number of read features plus 1. Then a 2D array is created with number of columns equal the total number of features and number of rows equal to the total number of read feature vectors, then the type of this array is made float 32 bits so save memory and all features are copied to it. Finally, the filtered list is made and the filtered output is copies to it and its type is converted it int 32 bit to same memory.

Next part of the code shown in **Figure 5** shows splitting the dataset int training and testing sets and training a neural network model.



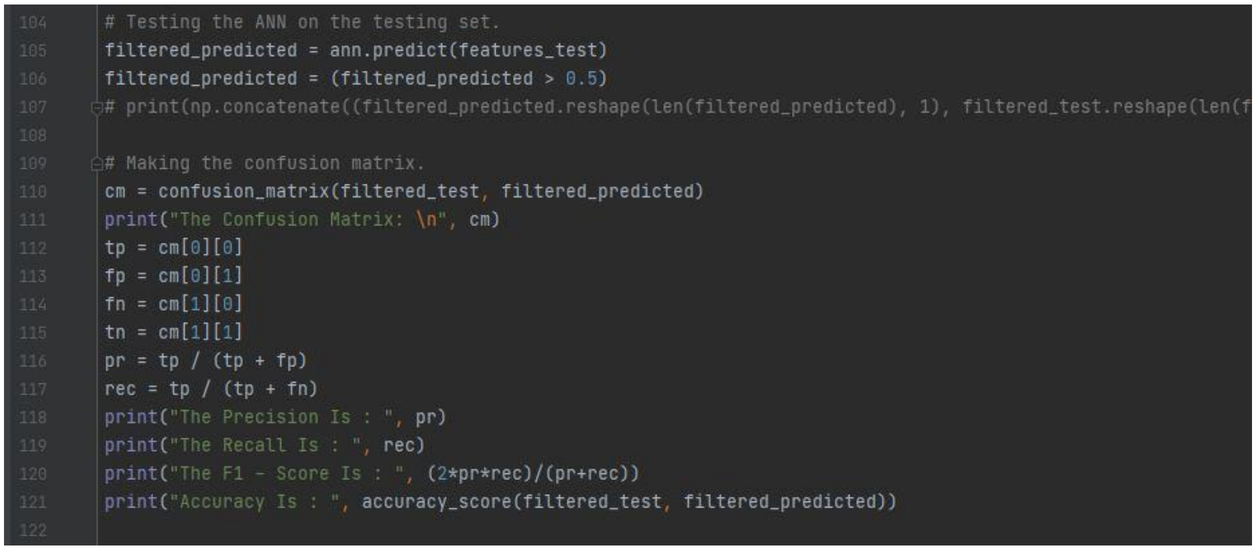
```python
# Training & Testing.

# Splitting the Data.
features_train, features_test, filtered_train, filtered_test = train_test_split(features, filtered, test_si

# Initializing the ANN.
ann = tf.keras.models.Sequential()

# Adding the input layer.
ann.add(tf.keras.layers.Dense(units=6, activation='sigmoid'))

# Adding the hidden layer.
ann.add(tf.keras.layers.Dense(units=10, activation='sigmoid'))

# Adding the output layer.
ann.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))

# Training the ANN.

# Compiling the ANN.
ann.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Training the ANN on the Training set.
ann.fit(features_train, filtered_train, batch_size=32, epochs=195)

```

Figure 5: Code Part 5

Line 82 splits the dataset into training and testing sets (both the features and the filtered output), size of the training set is 80% and the testing set is 20% of the total dataset.

Then an ANN is created and the input layer is added with 6 neurons and its activation function is **sigmoid**, after this a hidden layer is added with 10 neurons and the same activation function as the input layer and at last the output layer is added with a single neuron with the **sigmoid** activation function. Then comes compiling the neural network with optimizer **adam (stochastic gradient descent)** for updating the weights, a loss function that predicts binary outcomes which is binary_crossentropy as we just need 1 or 0 as output and an accuracy metrics. Then, the ANN is trained on the training set using 195 epochs and a batch size of 32 which means comparing 32 predictions with 32 real results.

Last part of the code shown in **Figure 6** shows testing the model and printing the confusion matrix as well as calculations of precision, recall, F1 score and the accuracy.



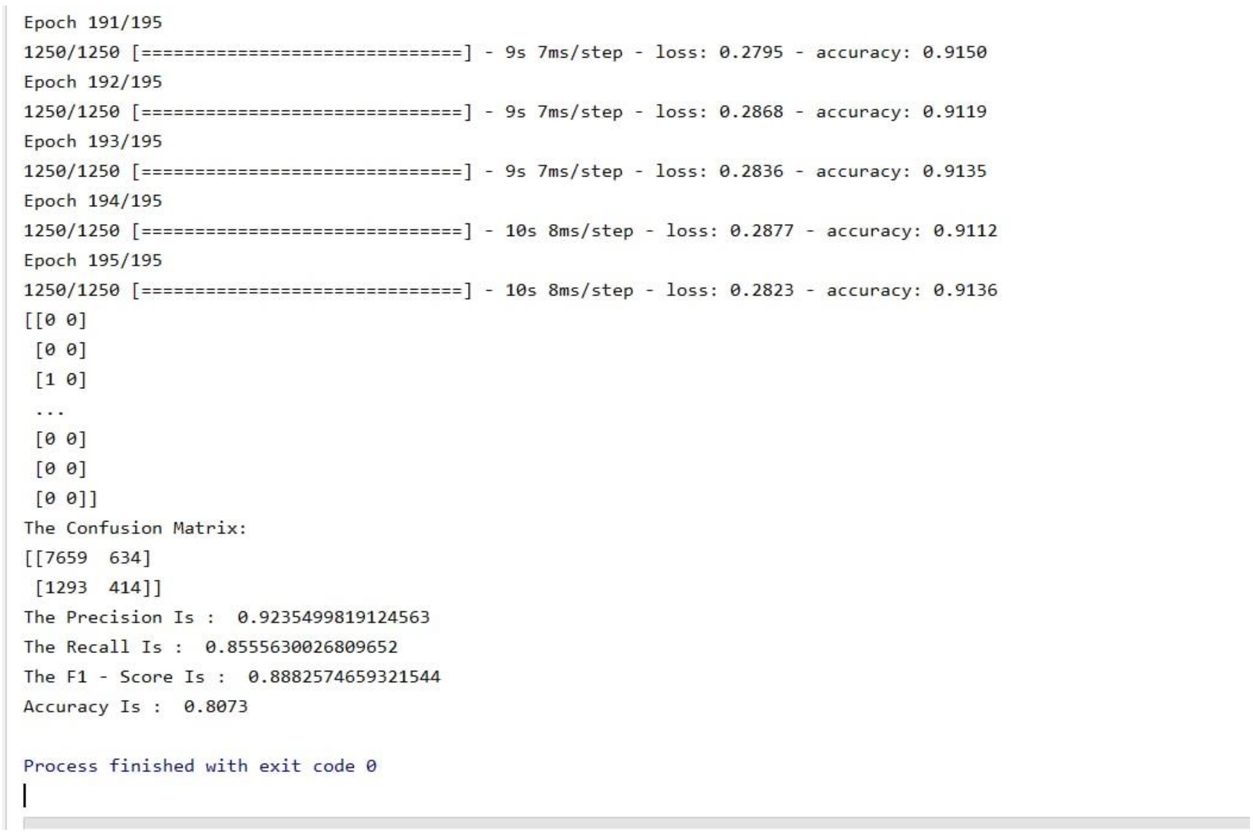
```
# Testing the ANN on the testing set.
filtered_predicted = ann.predict(features_test)
filtered_predicted = (filtered_predicted > 0.5)
# print(np.concatenate((filtered_predicted.reshape(len(filtered_predicted), 1), filtered_test.reshape(len(f

# Making the confusion matrix.
cm = confusion_matrix(filtered_test, filtered_predicted)
print("The Confusion Matrix: \n", cm)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
pr = tp / (tp + fp)
rec = tp / (tp + fn)
print("The Precision Is : ", pr)
print("The Recall Is : ", rec)
print("The F1 - Score Is : ", (2*pr*rec)/(pr+rec))
print("Accuracy Is : ", accuracy_score(filtered_test, filtered_predicted))

```

Figure 6: Code Part 6

At first the model is tested in the test set and then the final results of the sigmoid function are rounded up to 1 if they are more than 0.5, otherwise they are rounded to 0. Finally, the confusion matrix is created and the precision, recall, F1 score and accuracy are calculated and printed.

Simulation:

A simulation is shown in **Figure 7**.



```
Epoch 191/195
1250/1250 [==============================] - 9s 7ms/step - loss: 0.2795 - accuracy: 0.9150
Epoch 192/195
1250/1250 [==============================] - 9s 7ms/step - loss: 0.2868 - accuracy: 0.9119
Epoch 193/195
1250/1250 [==============================] - 9s 7ms/step - loss: 0.2836 - accuracy: 0.9135
Epoch 194/195
1250/1250 [==============================] - 10s 8ms/step - loss: 0.2877 - accuracy: 0.9112
Epoch 195/195
1250/1250 [==============================] - 10s 8ms/step - loss: 0.2823 - accuracy: 0.9136
[[0 0]
 [0 0]
 [1 0]
 ...
 [0 0]
 [0 0]
 [0 0]]
The Confusion Matrix:
[[7659  634]
 [1293  414]]
The Precision Is :  0.9235499819124563
The Recall Is :  0.8555630026809652
The F1 - Score Is :  0.8882574659321544
Accuracy Is :  0.8073

Process finished with exit code 0
```

Figure 7: Simulation

Note: to run the program a RAM more than 10 GB is needed and it takes about 30 minutes.

Conclusion:

At the end we can say that the project was a very interesting one because it takes us from the theoretical part of the ML algorithms to a real-life application requiring algorithms like this.

We’ve learned a lot about using machine learning algorithms in real life and how to compare between them to choose the best one the fits our needs. Also, we’ve learned about natural language processing (NLP) and how to extract features from texts and make a bag of words model.