University of Thessaly
Department of Computer Science and Electrical Engineering

ECE514: PSE's and Data Science

Sentiment Analysis

Theiou Vasileios 2685, Vlachos Evgenios 2499, Letsios Alexandros 2556

January 28, 2022
Volos

Contents

List of Figures

CHAPTER 1

Introduction

In this project, which is run entirely on Python in Jupyter Notebook from Anaconda, we use Natural Language Processing (NLP) algorithms in order to let our computer understand the natural language texts. To be more specific, we analyse the data so that we can extract the emotion of the person when he was writting the text. Before beginning analyzing the data sets and the algorithms that we used let us talk a bit about NLP and sentiment analysis.

Due to the huge technology development that is happening in the last years, more and more people tend to use internet and social media as an effort to express their feeling or their opinion about certain topics. Everything that is written online or said carries huge information, which can be used to understand and probably predict the human behaviour. The amount of information that is produced each day is enormous, though, which led to the development of a field known as Natural Language Processing. NLP is basically a field of artificial intelligence that allows machines to read, understand and extract the meaning from human languages. Sentiment analysis in NLP is a field which is aims to analyze the language and extract the emotion, feeling or opinion of a person. It is widely applied to voice of the customer materials such as reviews and survey responses, online and social media, and healthcare materials for applications that range from marketing to customer service to clinical medicine.

The present document describes the data, the preprocessing procedure and the algorithms that were used in an effort to find out whether an opinion has a positive or negative vibe.

CHAPTER 2

Data Analysis

As described before we will use many classification algorithms in order to get the polarity of our texts and to do that we need to train the models of these algorithms and that is why we need datasets with any kind of text that has been evaluated as positive or negative. We need to train the models so that they understand which words and what sentencews give these 2 emotions and try to use the training to predict the sentiment of new texts.

2.1 Datasets used

- The first dataset that was used is called "TripAdvisor Restaurants Info for 31 Euro-Cities" from kaggle and it is about reviews on restaurants worldwide.
- The second source of data was the csv file "e-food-restaurants-reviews-thessaly" which is a dataset for e-food restaurant reviews collected by Dr. Magda Foti. It contains reviews of the e-food restaurants in Thessaly, Greece.

2.2 Pre-processing the Datasets

At this section we will report some of the steps that we used in order to clean the datasets:

1. Remove Duplicates

2. Drop columns that will not help with the prediction of the sentiment. Basically, we only need the reviews and the ratings.

3. Get rid of all the rows that have Nan values on the reviews or ratings column.

4. Take out the reviews that have a neutral rating(rating = 3) because we want to classify only as negative or positive.

2.3 Data Analysis

At the beginning dataset from Trip Advisor contained 125527 reviews and 10 columns, while the one from e-food contains 1040 reviews and 6 columns. From all this information, we were interested in the reviews and ratings columns. After concatenating the 2 datasets and applying pre-process techniques, we end up with 53667 rows and 2 columns. Afterwards, we calculated some metrics so that we can have a better look at the reviews, which are mentioned bellow:

- Average word's length in reviews = 5.44
- Average words per review = 7.99
- Average character's count = 48.72
- Mean stop words count = 2
- Mean stop words rate = 0.2

We, then, checked the number of positive and negative reviews in the dataset. We show that there were 51.453 positive and 23083.

As we can see, the positive reviews are about double than the negative ones, which but be a problem for the algorithms, known as class imbalance problem. This occurs in classification problems where one class dominates the other and it can be dealt by either undersampling or oversampling. Undersampling can be defined as removing some observations of the majority

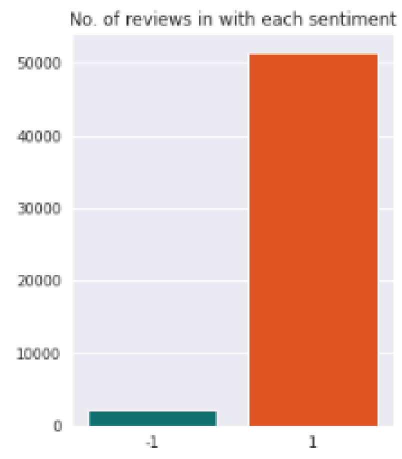


Figure 2.1: Sentiment Diagram

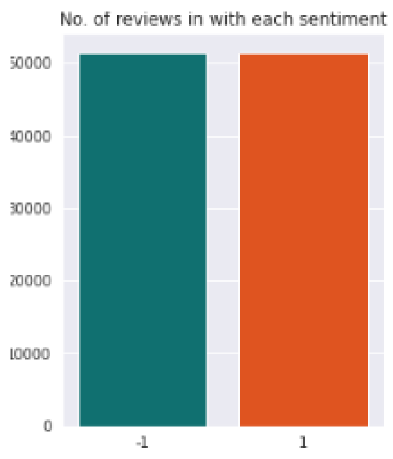


Figure 2.2: Sentiment Diagram

class, while oversampling can be defined as adding more copies to the minority class. In our case, we used oversampling method to balance our data.

Moreover, in the image 2.2 we can see the more frequently used words after of course removing the stopwords. Stopwords are words that occur very often in text and do not describe the content of our topic, such as âtheâ, âaâ, âanâ, âinâ.

Great food
food good
Good food
Amazing
Best
Deliciou
great service
good service
food nice
Nice place
Great place
service
food
great
dinner
Excellent food
Poor service
Perfect
customer service
bad service
Fantastic
Nice food

Figure 2.3: Sentiment Diagram

2.4 Data Preparation

2.4.1 Stopwords Removal

As mentioned before, stopwords are words that appear mreo ften than others and do not affect the polarity of the text. All languages contain stopwords and Natural Language Toolkit(NTLK) provides us with a list of stopwords. It is very important for the algorithms to remove the stopwords from a text as it results in huge text reduction, which will make our algorithms run a lot faster.

2.4.2 Tokenization

Tokenization is a process used in sentiment analysis which given a sentence,a phrace or generally text it into singe words or terms called tokens. In this project, we used the Toktok-Tokenizer.

2.4.3 Stemming

Stemming is the process of mapping words into their root form which leads to further reduce of the text given as input. Also, it extracts the real meaning of a word and makes it easier for the machine to find patterns in the data.

2.4.4 Lemmatization

Lemmatization is similar to stemming but it relies on a lexical knowledge base to obtain the correct base form of words.

2.4.5 Remove Punctuation

This technique is used for further reduction of the input data and it includes the removal of punctuation such as "?", ".", ";", ":", "!",′"′, "<", ">","/", ",", "′", "[", "]".

2.4.6 Vectorization

Vetorization is a procedure used in nlp to map words or phrases from vocabulary to a corresponding vector of real numbers which used to find word predictions, word similarities/semantics. It is obvious that computers are unable to read words because they only work with numbers. So, this techniques let's us transform words to vectors of numbers. There exist many kinds of vectorizers, but in this case we used TF-IDF vectorizer. TF-IDF is a statistical measure that evaluates how relevant a word is to a document in a collection of documents. This is done by multiplying two metrics: how many times a word appears in a document, and the inverse document frequency of the word across a set of documents. It has many uses, most importantly in automated text analysis, and is very useful for scoring words in machine learning algorithms for Natural Language Processing (NLP).

CHAPTER 3

Classification Algorithms

After the transformation of the review texts into vectors and marking the polarity of the texts as -1 for negative and 1 for postive we need to split the data into training and testing parts. In this case, we decided to use the 80% as training and the remaining 20% for testing and evaluation of the models. In the sections bellow, we present the algorithms that we used for classification:

3.1 Logistic Regression

Logistic Regression is a machine learning algorithm used in classification problems. It is used to predict a binary output such as true/false, 0,1 etc.. The probabilities describing the possible outcomes of a single trial are modelled using a S-shaped logistic function.

3.2 KNeighbors Classifier

Neighbours based classification is a type of lazy learning as it does not attempt to construct a general internal model, but simply stores instances of the training data. Classification is computed from a simple majority vote of the k nearest neighbours of each point.

3.3 Random Forest Classifier

Random Forest Classifier is a metaÂestimator that fits a number of decision trees on various subÂsamples of datasets and uses average to improve the predictive accuracy of the model and controls overÂfitting. The subÂsample size is always the same as the original input sample size but the samples are drawn with replacement.

3.4 Gaussian Naive Bayes

NaiveÂBayes algorithm based on Bayesâ theorem with the assumption of independence between every pair of features. NaiveÂBayes classifiers work well in many realÂworld situations such as document classification and spam filtering.

3.5 Support Vector Machine Classifier

Support Vector Machine is a representation of the training data as points in space separated into categories by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

3.6 Long Short-Term Memory (LSTM)

Long shortÂterm memory (LSTM) is an artificial recurrent neural network (RNN) architecture used in the field of deep learning. Unlike standard feedÂforward neural networks, LSTM has feedback connections. It can process not only single data points such as images, but also entire sequences of data such as speech or video.

At this point, we need to note that for the LSTM Classifier instead of using the TF-IDF vectorizer we need to use the "Tokenizer" from Keras implementation which can then convert vectors to sequences that are easier for the model to deal with.

3.7 AutoSklearn Classifier

Auto-Sklearn is an open-source library for performing AutoML in Python. It makes use of the popular Scikit-Learn machine learning library for data transforms and machine learning algorithms and uses a Bayesian Optimization search procedure to efficiently discover a top-performing model pipeline for a given dataset.

3.8 Xgboost

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. It provides a parallel tree boosting that solves many data science problems in a fast and accurate way

3.9 Bert

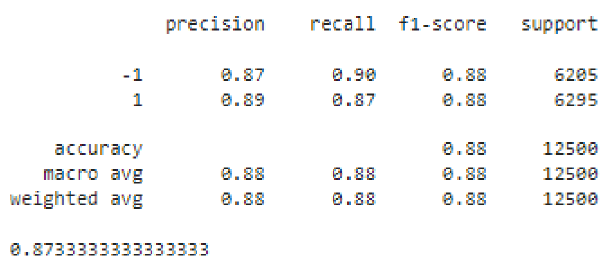
BERT stands for Bidirectional Encoder Representations from Transformers. It is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of NLP tasks.

CHAPTER 4

Models Evaluatuion and Results

We evaluates our model using K-Fold cross validation with n_splits = 5 and then we calculated the mean of the 5 scores that were produced. Moreover, for the most of them we obtained the classification reports, from which we can check precision, recall and f1-score. Low recall indicates that the positive sentiment is not classified well, while high precision means that positive values are coreectly classified as true positive.

4.1 Logistic Regression

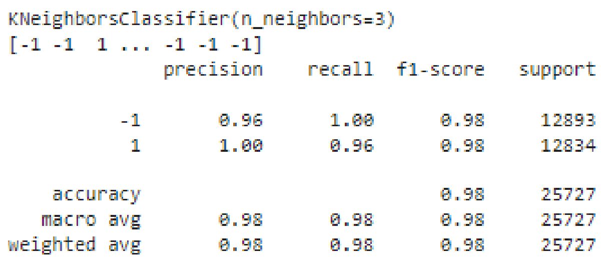


```
              precision    recall  f1-score   support

          -1       0.87      0.90      0.88      6205
           1       0.89      0.87      0.88      6295

    accuracy                           0.88     12500
   macro avg       0.88      0.88      0.88     12500
weighted avg       0.88      0.88      0.88     12500

0.8733333333333333
```

Figure 4.1: Logistic Regression classification report

The logistic regression classifier achieved an accuracy up to 87%. We can observe from the classification report that both precision and recall achieve high scores which indicates that the algorithm does a good job at classifying the classes.

4.2 KNeighbors Classifier

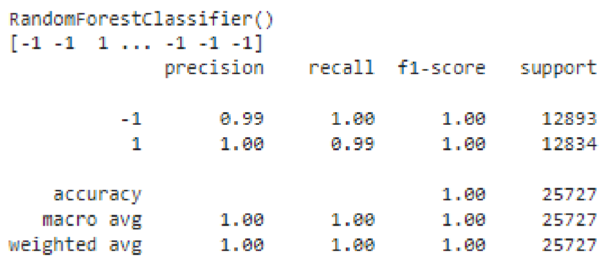


```
KNeighborsClassifier(n_neighbors=3)
[-1 -1  1 ... -1 -1 -1]
              precision    recall  f1-score   support

          -1       0.96      1.00      0.98     12893
           1       1.00      0.96      0.98     12834

    accuracy                           0.98     25727
   macro avg       0.98      0.98      0.98     25727
weighted avg       0.98      0.98      0.98     25727
```

Figure 4.2: KNN classification report

Knn algorithm performed very well, almost perfect, as it achieved accuracy up to 97.7%. Also, precision and recall appear very high, so our model performs very well.

4.3 Random Forest Classifier

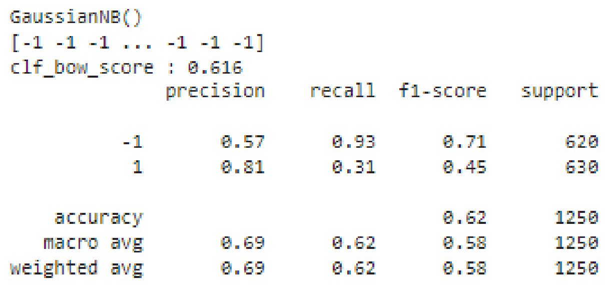


```
RandomForestClassifier()
[-1 -1  1 ... -1 -1 -1]
              precision    recall  f1-score   support

          -1       0.99      1.00      1.00     12893
           1       1.00      0.99      1.00     12834

    accuracy                           1.00     25727
   macro avg       1.00      1.00      1.00     25727
weighted avg       1.00      1.00      1.00     25727
```

Figure 4.3: Random Forest classification report

Random Forest classifier is the algorithm that had the best results. It outperformed the other models and achieved accuracy of 99% while maintaining high precision and recall.

4.4 Gaussian Naive Bayes

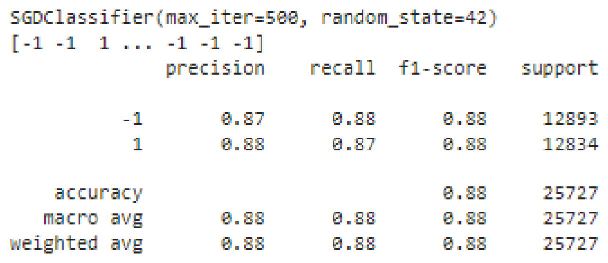


```
GaussianNB()
[-1 -1 -1 ... -1 -1 -1]
clf_bow_score : 0.616
              precision    recall  f1-score   support

          -1       0.57      0.93      0.71       620
           1       0.81      0.31      0.45       630

    accuracy                           0.62      1250
   macro avg       0.69      0.62      0.58      1250
weighted avg       0.69      0.62      0.58      1250
```

Figure 4.4: Gaussian Naive Bayes classification report

Gaussian NB did not perform very well and its accuracy was 61% and had very low precision. This might be caused because we took a sample of the whole dataset because it could not run with all the rows.

4.5 Support Vector Machine Classifier



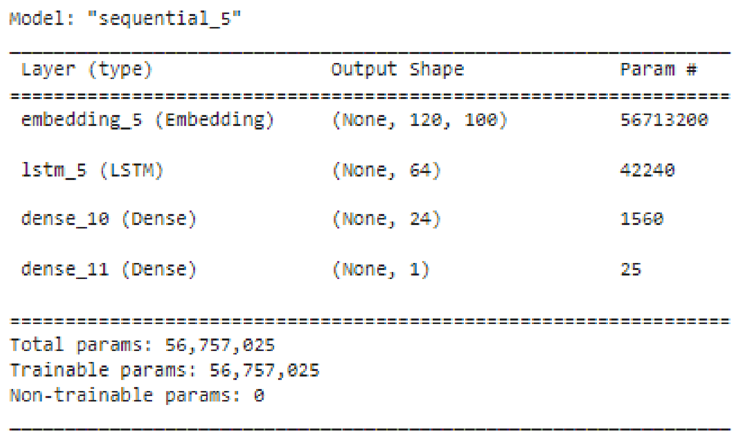
```
SGDClassifier(max_iter=500, random_state=42)
[-1 -1  1 ... -1 -1 -1]
              precision    recall  f1-score   support

          -1       0.87      0.88      0.88     12893
           1       0.88      0.87      0.88     12834

    accuracy                           0.88     25727
   macro avg       0.88      0.88      0.88     25727
weighted avg       0.88      0.88      0.88     25727
```

Figure 4.5: SGD classification report

SGDclassifier's accuracy is 86%, while the precision and recall scored 0.87 and 0.88 each.

4.6 AutoSklearn Classifier

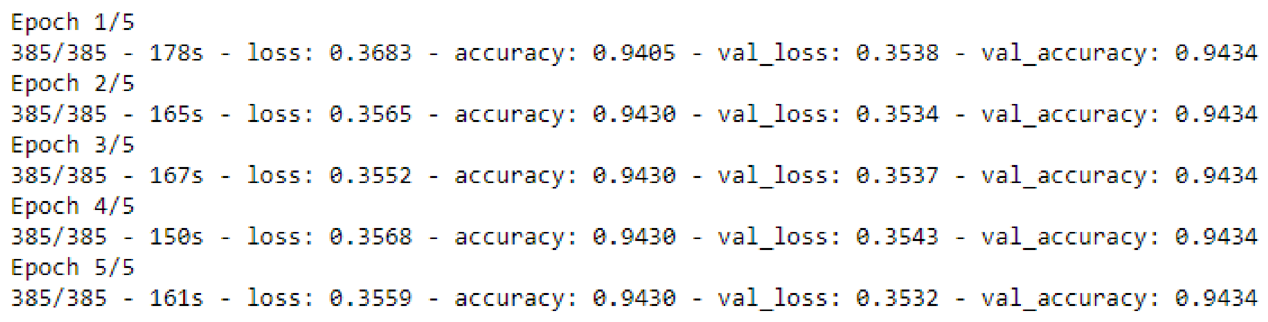
Autosklearn classifier performed 83% accuracy while running for 2 minutes which is controlled by the parameter (time_left_for_this_task = 2*60. We chose this parameter cause otherwise the algorithm took too long to make the fit.



```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_5 (Embedding)     (None, 120, 100)          56713200

 lstm_5 (LSTM)               (None, 64)                42240

 dense_10 (Dense)            (None, 24)                1560

 dense_11 (Dense)            (None, 1)                 25

=================================================================
Total params: 56,757,025
Trainable params: 56,757,025
Non-trainable params: 0
_________________________________________________________________
```

Figure 4.6: LSTM report

4.7 Long Short-Term Memory (LSTM)



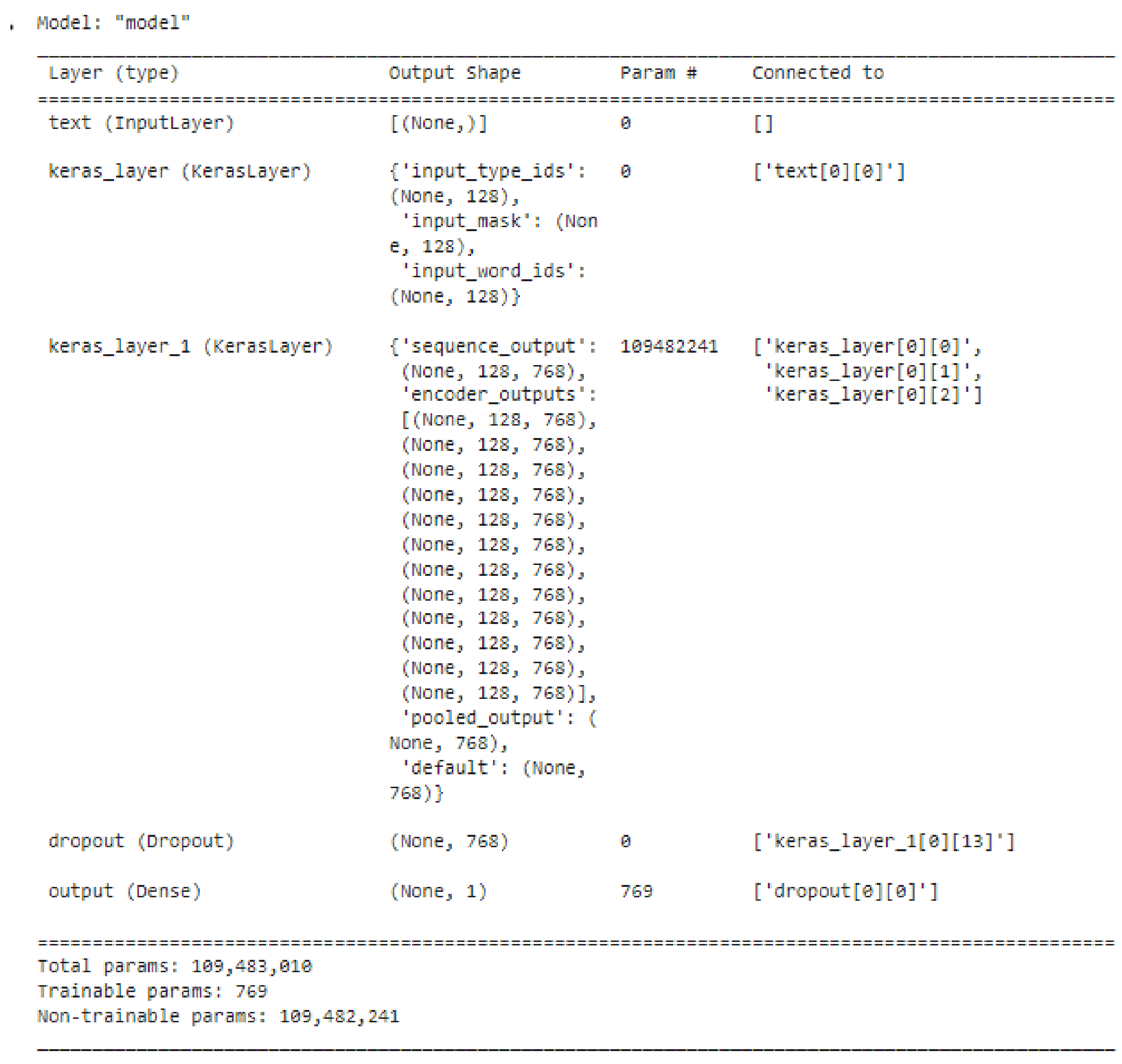
```
Epoch 1/5
385/385 - 178s - loss: 0.3683 - accuracy: 0.9405 - val_loss: 0.3538 - val_accuracy: 0.9434
Epoch 2/5
385/385 - 165s - loss: 0.3565 - accuracy: 0.9430 - val_loss: 0.3534 - val_accuracy: 0.9434
Epoch 3/5
385/385 - 167s - loss: 0.3552 - accuracy: 0.9430 - val_loss: 0.3537 - val_accuracy: 0.9434
Epoch 4/5
385/385 - 150s - loss: 0.3568 - accuracy: 0.9430 - val_loss: 0.3543 - val_accuracy: 0.9434
Epoch 5/5
385/385 - 161s - loss: 0.3559 - accuracy: 0.9430 - val_loss: 0.3532 - val_accuracy: 0.9434
```

Figure 4.7: LSTM epochs

The lstm model achieved an accuracy of 87%. Further information about the fit process, the accuracy and the lost in each epoch are shown in the images above.

4.8 Xgboost

Xgboost's k folds average score was 95%.

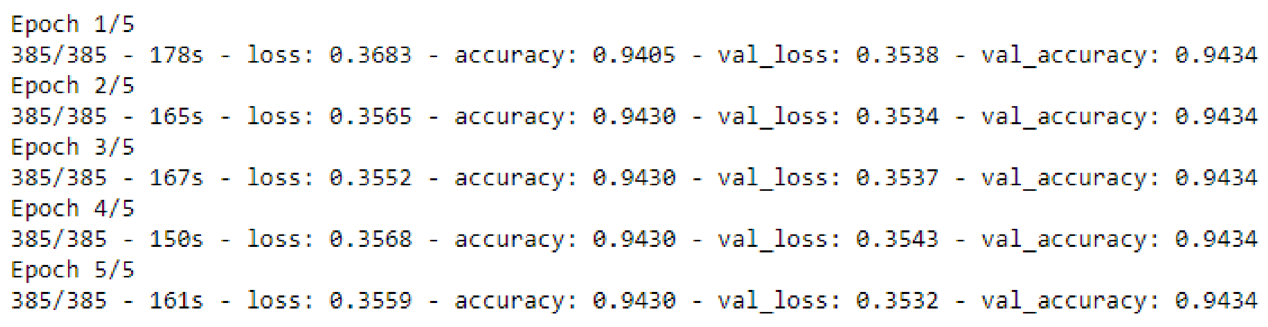


```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 text (InputLayer)              [(None,)]            0           []

 keras_layer (KerasLayer)       {'input_type_ids':   0           ['text[0][0]']
                                (None, 128),
                                 'input_mask': (Non
                                e, 128),
                                 'input_word_ids':
                                (None, 128)}

 keras_layer_1 (KerasLayer)     {'sequence_output':  109482241   ['keras_layer[0][0]',
                                 (None, 128, 768),                'keras_layer[0][1]',
                                 'encoder_outputs':               'keras_layer[0][2]']
                                 [(None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768)],
                                 'pooled_output': (
                                None, 768),
                                 'default': (None,
                                768)}

 dropout (Dropout)              (None, 768)          0           ['keras_layer_1[0][13]']

 output (Dense)                 (None, 1)            769         ['dropout[0][0]']

==================================================================================================
Total params: 109,483,010
Trainable params: 769
Non-trainable params: 109,482,241
__________________________________________________________________________________________________
```

Figure 4.8: Bert report

4.9 Bert

This model's accuracy score was 83%. More information of the build of the model and the metric such as accuracy and loss in each epoch can be shown bellow.



```
Epoch 1/5
385/385 - 178s - loss: 0.3683 - accuracy: 0.9405 - val_loss: 0.3538 - val_accuracy: 0.9434
Epoch 2/5
385/385 - 165s - loss: 0.3565 - accuracy: 0.9430 - val_loss: 0.3534 - val_accuracy: 0.9434
Epoch 3/5
385/385 - 167s - loss: 0.3552 - accuracy: 0.9430 - val_loss: 0.3537 - val_accuracy: 0.9434
Epoch 4/5
385/385 - 150s - loss: 0.3568 - accuracy: 0.9430 - val_loss: 0.3543 - val_accuracy: 0.9434
Epoch 5/5
385/385 - 161s - loss: 0.3559 - accuracy: 0.9430 - val_loss: 0.3532 - val_accuracy: 0.9434
```

Figure 4.9: Bert epochs

CHAPTER 5

Difficulties and Conclusions

5.1 Difficulties

Some of the difficulties we faced were that we had to deal with a huge dataset, which was obtained by concatenating the 2 datasets that are mentioned above(Trip Advisor and e-food). Thus, we had to take a sample of it in order for the algorithms to run and this could result in losing some important information. Also, we can see that because of the huge data volume, the algorithms took very long too finish the fitting. This fact resulted in waiting too much time in order to run experiments and find the best parameters for each model.

5.2 Conclusions

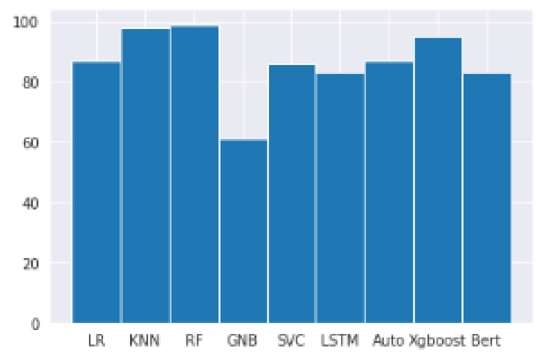


Figure 5.1: Comparative Diagram

We can see from the above diagram that the random forest algorithm outperformed all other models and achieved the best accuracy. Also, it was one of the models that could perform with all the dataset without having to take a sample of it like gaussian naive bayes. Furthermore, when taking into consideration the time that each algorithm needed to run, we observed that generally deep learning models needed more of it, while logistic regression and random forest were the fastest at fitting the data.

Overall, the models performed very good achieving accuracies over 80% except gaussian naive bayes. This fact means that the most of the them are capable of being used in sentiment analysis projects and after performing experiments, we can check which one is better for our current datasets.

5.3 Future Work

As future work, we want to try and collect data from sites on the web using web scrapers, wherever that is possible. This will give us the opportunity to explore and experiment with more data and maybe choose the reviews that have more meaning and less stop words in order to make it easier for the model to understand the sentiment. Also, we might as well try other vectorizers rather than tfidf and check if they improve the performance of the model. Last but not least, after finding the best model we could make an app with graphical user interface(gui) to make sentiment analysis on reviews given by random people. This will not only make it more user friendly but will give the opportunity to maybe sell this app to companies that want only to collect positive and negative reviews in order to improve their product or service.

Bibliography

[1] Hotel Review Sentiment Analysis using Natural Language Processing , *by Alexandros Lykesas*

[2] Mining opinion on COVID 19 based on Twitter Data , *by Athanasios Papanikolaou*