Sentiment Analysis on Amazon Reviews Data using Machine Learning and Deep Learning Techniques

Emmanouil Lykos
University of Piraeus
NCSR Demokritos
manolislykos97@hotmail.gr

June 23, 2022

Abstract

Sentiment Analysis is the task of determining the polarity of various texts. This task can have various real-world applications like identifying the trend of a product for example. In this paper, we try to address this task with three approaches, where we evaluate thoroughly each one of them. The first approach, we use Traditional Machine Learning and Feature Extraction techniques. Afterwards, we use Long Short Term Memory Networks(LSTMs) which is a Deep Learning approach that can do inference from sequential data like text. Finally, we use Transfer Learning by using BERT model. Afterwards, we compare these approaches in terms of accuracy, where we see that the Transfer Learning one performs better. Last but not least, we summarize the paper and we provide directions for future work.

Keywords: Natural Language Processing, Machine Learning, Deep Learning, Sentiment Analysis, Transfer Learning

1 Introduction

With the accessibility of 4.62 people in social media and 500 million tweets posted every day, people and organizations have many data to handle which provide them with the possibility to use them for many different tasks by taking advantage of the reasons that these platforms are created. One of those reasons, is to discuss what happens in the world like sports events, government decisions and celebrity gossip. However, organizations and governments, for example, need to know if the people liked or not the new product or legislation. This task falls on the field of Sentiment Analysis.

Sentiment Analysis is the use of Natural Language Processing in order to classify the given corpus valence as positive, negative or neutral taking into consideration the content of the instance, which is in text form. Sentiment Analysis can have many real-world applications, as I mentioned before. In order to do that, many techniques can be applied. Initially, organizations can provide feedback forms by grading each feature to a scale and just get the attitude by taking an average. However, those have two major drawbacks that are fixed by taking relevant opinions from social media. Initially, those forms are not accessible like social media, thus the various analysts will have probably a small number of samples. Afterwards, from just a number we cannot extract the same information from a text, so we cannot have insights on what or why people like or dislike us. Thus, it is pretty clear that we need to use Data Mining techniques that can handle Big Data in order to extract the sentiment of each opinion.

In this report we will present and evaluate the performance of three techniques-along with their drawbacks-where one technique is the evolution of another. Initially, we will present the traditional Naive Bayes and SVM Machine Learning models(because they are commonly used in Natural Language Processing tasks) and evaluate them on this task along with the preparation needed to do to the data and the features that we

can extract from them. Afterwards, we will present a Deep Learning approach that can handle sequential data like text, and more specifically Long Short Term Memory(LSTM) networks. Finally, we will present the use of Transfer Learning in this task by using a pretrained BERT Transformer network. The structure of the rest of the report is the following. In Sections 2,3 we will present the traditional Machine Learning, the LSTM and Transfer Learning approaches, respectively. In Section 5, we will present the experimental evaluation of those techniques and reason on why some techniques are better than others. Finally, in Section 6 we will state our conclusions and provide directions for future work.

2 Sentiment Analysis via Traditional Machine Learning using Naive Bayes and Support Vector Machines(SVM)

2.1 Text Preprocessing Steps

Initially, before feeding the data into the various models, we need to do some preprocessing on the initial dataset in order to provide them to the different models in a form that it is easier for them to generalize, thus, they will have better predictive performance. The preprocessing was done according to the following steps:

1. Lowercasing of each text instance because we should eliminate any confusion for the model that it will be created due to different casing of the words.

2. Replacing of the more than one consequent whitespaces with only one, in order to not have unnecessary tokens.

3. Tokenization of each text instance and then we remove its stopwords. The tokenization is done with NLTK's *word_tokenize* because we saw that the resulting classifiers have better performance by using this tokenization method. The stopwords removal is done in order to remove the words that do not provide any particular information by themselves.

4. Finally, we stem each token, i.e. convert each word to its base or root form. This is done in order to make our classifiers to be able to distinguish that same words but on a different grammatical role are the same. Otherwise, for instance, the classifiers will not recognize that run and running are in fact the same word.

2.2 Feature Engineering & Preprocessing

After the preprocessing of the corpus, we need to convert each text instance to a vector in order to feed them to the classifiers, and then apply some more preprocessing, if needed. We consider three types of feature extraction techniques for our model. The first Feature Extraction Technique is called Bag of Words(BoW). BoW is a pretty simple technique where each coordinate of the resulting vector represents a single word that is present in the dataset and its value, for a given instance, is the number of times that this word occurs in that instance. The second technique is the term frequency–inverse document frequency(TF-IDF). This technique works in a similar manner as BoW but the $tf - idf$ value is calculated as follows:

$$tf\text{-}idf(t,d) = tf(t,d)idf(t) = tf(t,d)(log\frac{1+n}{1+df(t)} + 1)$$

where, $tf(t,d)$ is the number of times that term t appears in document(in our case, in instance) d and $df(t)$ is the number of instances that contain the term t. Last but not least, we used Word2Vec SkipGram model, as shown in Figure 1, in order to extract contextual vector representations(i.e. embeddings) for each word present in the whole dataset. Then, we extract the feature vector of each document by taking the mean vector that is extracted from the word embeddings of each word(token), present in the current instance.

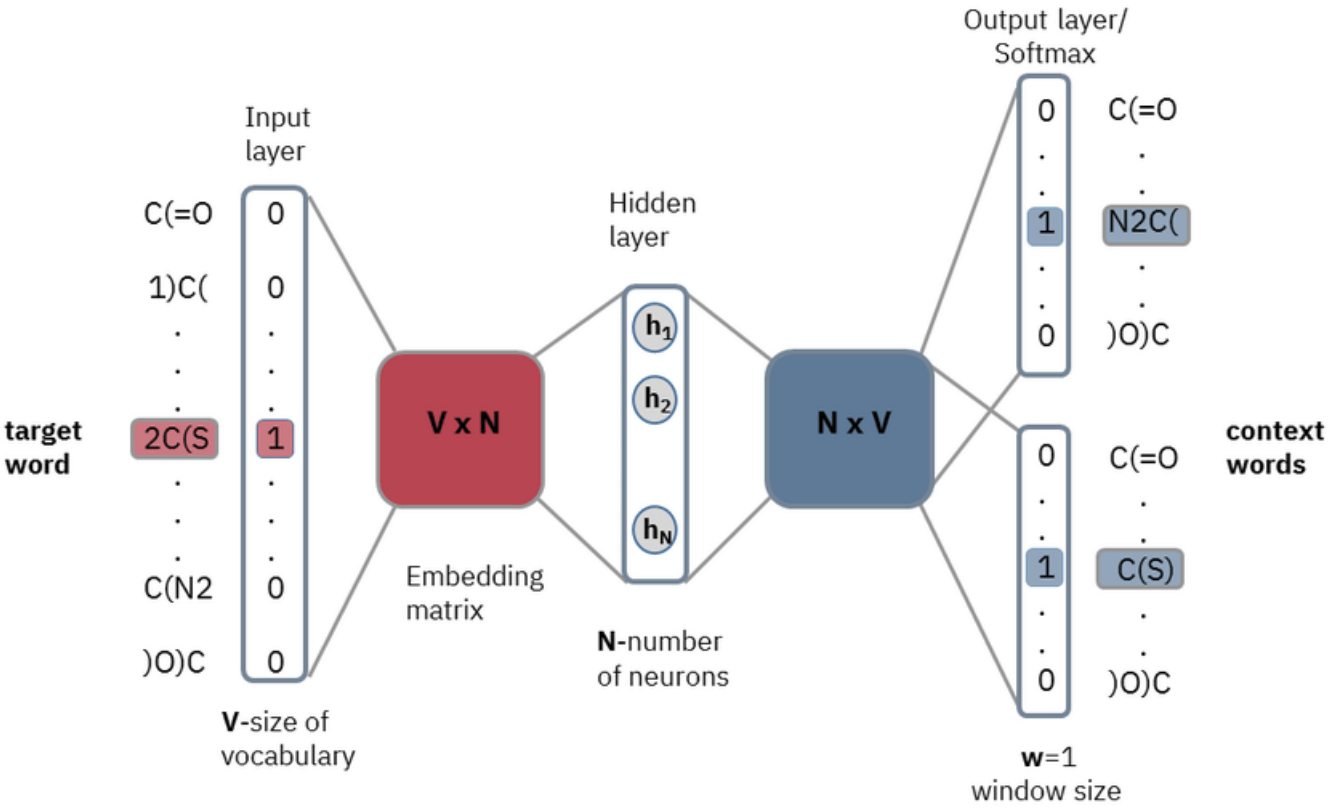

Figure 1: Word2Vec Skip-Gram Model

In terms of preprocessing at this point, the only thing that we do is to remove the constant features in the dataset, i.e. remove features that have virtually the same value in all instances and then we normalize the dataset. Finally in some algorithms, if needed, we apply dimensionality reduction to the instances using Principal Components Analysis(PCA) algorithm.

2.3 Naive Bayes Algorithm

Naive Bayes is a machine learning algorithm that is based its predictions on conditional probabilities and more specifically, on Bayes Theorem. This algorithm does not need to do any training steps, thus, all the calculations are done when we want to predict a given instance. Let's assume that we have an instance x with n features and we want to predict its class. In Naive Bayes, we want to find the probability that the given instance belongs to class y using the Bayes theorem, thus we want to determine $P(y|X = x)$ using the following formula:

$$P(y|X = x) = \frac{P(X = x|y)P(y)}{P(X = x)} = \frac{P(X_1 = x_1, X_2 = x_2, ..., X_n = x_n|y)P(y)}{P(X = x)}$$

and the predicted class is the one with the given probability. However, when finding the probabilities we do two tweaks in our method. Firstly, because the value $P(X = x)$ is not easy to be calculated and also its the same for every class y, we do not calculate it at all. Secondly, because $P(X_1 = x_1, X_2, = x_2, ..., X_n = x_n|y)$ is also difficult to be calculated, we do the naive assumption that the value of each feature is independent from the other ones(that's why this algorithm is called "naive"), hence the aforementioned formula is changed to:

$$P(y|X = x) = P(X_1 = x_1|y)P(X_2 = x_2|y)...P(X_n = x_n|y)P(y)$$

where each probability can be calculated with the following formula:

$$P(X_i = x_i|y) = \frac{N(X_i = x_i, y)}{N(y)}$$

where, $N(X_i = x_i, y)$ is the number of instances in the dataset that belong to class y and feature i has value equal to x_i, and $N(y)$ is the number of instances in the dataset that belong to class y. Although this algorithm seems to work good for NLP tasks and also in case where we do not have many training data, it has the drawback that it does not work well when we do not have categorical data.

2.4 Support Vector Machines

Support Vector Machines is a famous Machine Learning algorithm due to its high predictive performance in a variety of problems including Sentiment Analysis. The high-level description on how this method works is that it tries to find the decision hyperplane that maximizes its distance from its nearest instances of both classes. An example of what the current algorithm tries to achieve, is shown in Figure 2.

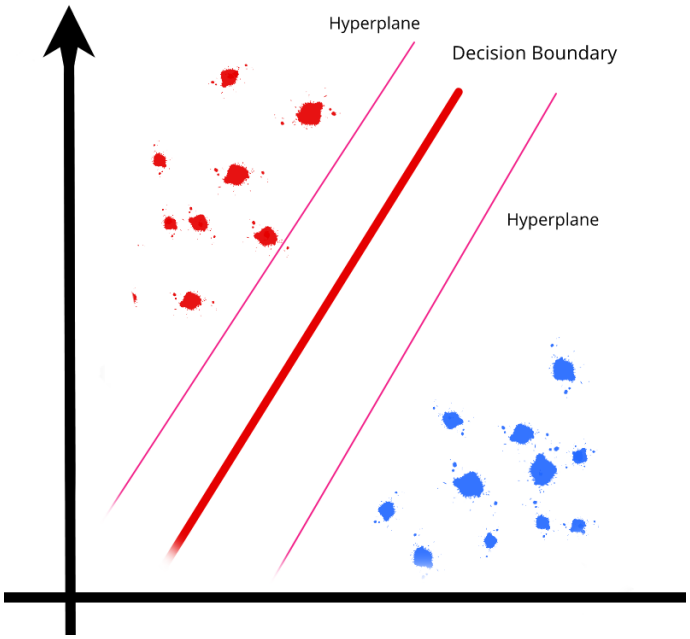

Figure 2: SVM Example Result

As we can see for the above figure, it is clear that in order for SVM to work, the data should be linearly separable. However, in most times, this is not the case. So, this is the point where kernel trick comes into the game. The role of kernel trick is to move the instances to spaces with greater dimensionality in order for the instances to be linearly separable. The prediction algorithm given a test instance just checks in what side of the decision boundary the given instance falls. Although this algorithm seems to have the best predictive performance its pretty inefficient in terms of efficiency because it takes too much time for training due to the fact that the model should do the kernel trick and then solve a linear problem, where the algorithms for them have exponential time complexity, in order to find the best decision boundary, given the training data. On the other hand, it is pretty fast when it comes to do predictions because it just applies the given instance to the decision boundary and checks at what side this instance falls.

3 Sentiment Analysis using Neural Networks

3.1 Long Short-Term Memory Networks

The drawback with the aforementioned traditional Machine Learning Techniques, is that when creating the vectors for each instance, we lose any information about the structure of the given review, because the

features are actually either histograms of some words or just an average of the representations of each word present in the instance. One approach to do this, is to use Feed-forward Neural Networks that will take a sequence of words of fixed length as inputs where each one will be pad or cut depending its number of words and it will return a probability for each possible label. However, this approach has the drawback that we need to tune also the fixed length of each sequence and either way we will lose information for instances of length greater than the maximum length. Also, the current token won't affect the next one and this is where Sequential Neural Networks seems to be useful.

Sequential Neural Networks are networks designed to handle sequential data like timeseries, text etc. The first approach are the Recurrent Neural Networks(RNNs), however, we will not deal with it on this project because it does not do good job due to the vanishing gradient problem, therefore we will deal with Long Short-Term Memory Networks(LSTMs) as their approach is very similar will only difference on how they represent its "memory" that its acquired from previous tokens in our case. Starting from bottom to top, we will initially specify how an LSTM cell works. How an LSTM cell works is specified on Figure 3.

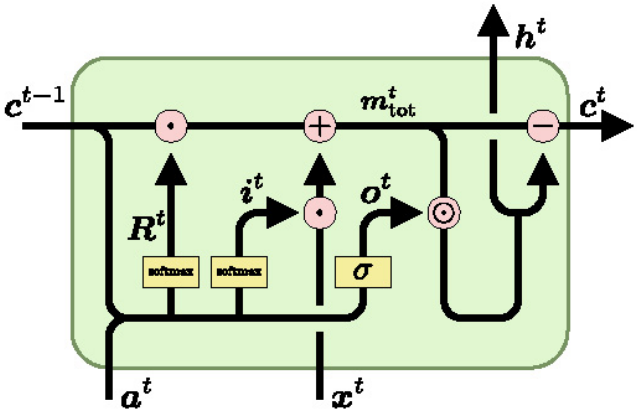

Figure 3: LSTM Cell

In high level, we can see that LSTM has some gates that help him return the output but also control on what it needs to remember and what to forget. The gates are the input, output and forget gates. Also, LSTMs, when invoked, they return the output and the hidden and cell state. Now, the architecture of our LSTM classifier is specified on Figure 3. It is pretty intuitive on why we consider the output of the last token.

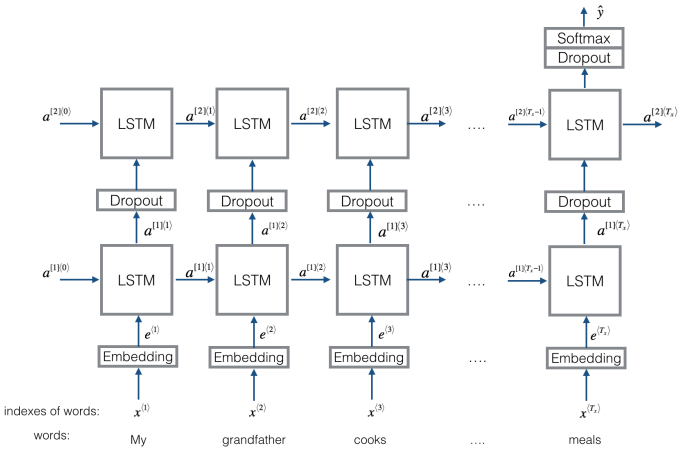

Figure 4: LSTM Sentiment Classifier

Keep in mind that in our implementation we use a single LSTM layer. Also, we do not use dropout. Last but not least, in terms of preprocessing we apply the one on Machine Learning approach but we do not do stemming and we do not remove the stopwords, as this approach gives us better results. Then, we add two special tokens denoting the start and end of each sentence and we replace each token with a number that denotes its "index" in the embedding vector.

3.2 BERT

Although LSTMs provide a solution when we want to handle sequential data, they still have some drawbacks. Initially, LSTMs cannot combine information that might be related but they have some distance in the sequence. Furthermore, in order to inference a point in a given sequence, it needs the previous outputs from previous points. This means that we cannot use parallel computing for such models. Instead, we can use Transformers. Transformers are used mostly for Seq2Seq architecture and rely on attention layers, where the computations that are done in these layers can be paralleled, however we can use the pretrained models also for sequence classification. In this project we used BERT in order to improve our sentiment classifier.

BERT is a model that uses only the Encoder layers of the Transformers and was created in order to provide contextualized embeddings in order to excel on different tasks. The pretrained models that we will use in our classifier were trained for masked language modeling tasks, meaning that they excel on finding missing words in a document. However, just like all pretrained models, we can use its pretrained layers in order to extract contextualized embeddings of each token. Generally, BERT works as follows. Initially, it tokenizes with its own way the given sentence, fixes the size of the sentence according to some specified maximum length, finds the positional encodings in order for the input to contain the information on what position each token is, masks or replaces some tokens from the input sentences and finally adds two special tokens that denote the start and end of the sentence, respectively. Then, the input sequence is passed to a series of Transformer encoder layers which extract the contextualized embeddings of each token. For classification tasks, we only consider the representation of the last token, which is passed later to a Feed-forward Neural Network, because its representation is also the representation of the whole sequence. An illustration of our BERT approach is shown in Figure 5.

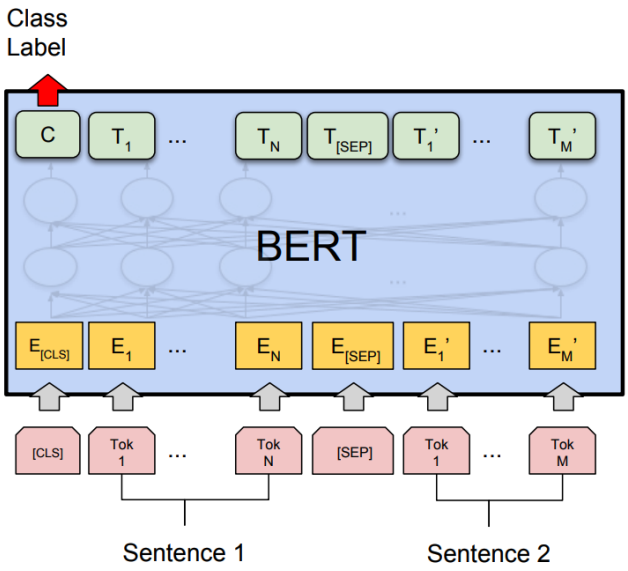

Figure 5: BERT Sentiment Classifier

4 Experimental Evaluation

According to our methodology we need to test how Naive Bayes, SVMs, LSTMs and BERT perform on Sentiment Analysis Task. In order to do that, we will use the IMDB Reviews Dataset on the aforementioned models, and compare their performance in terms of accuracy and F_1-Score with the best performing models, according to papersWithCode, too. This Dataset contains 25000 reviews that will be used for training and 25000 reviews that will be used for testing along with their ratings. Ratings greater or equal that 4 are considered negative and ratings greater or equal than 7 are considered positive. Neutral polarity is not present on this dataset. The models are not trained to predict the rating, but if the review has positive or negative valence.

Initially, using traditional Machine Learning, we trained Naive Bayes and Support Vector Machines models with different values in regularization parameters C and evaluated them on test set. Also, each model was trained with different features, i.e. Bag of Words, TF-IDF and Word2Vec with vector size equal to 300. The train and test accuracies and F_1-Scores of each case is displayed on the following tables:

Model	**Bag Of Words**		**TF-IDF**		**Word2Vec**	
	train	**test**	**train**	**test**	**train**	**test**
Naive Bayes	0.7520	0.6840	0.8032	0.7876	0.7540	0.7467
SVM with $C = 1.0$	0.9072	0.8520	0.9088	0.8336	0.9111	0.8660
SVM with $C = 10.0$	0.9618	0.8400	0.9874	0.8313	0.9817	0.8486
SVM with $C = 50.0$	0.9893	0.8144	0.9996	0.7851	0.9992	0.8291

Table 1: Train and Test Accuracies of Machine Learning Sentiment Classifier in IMDB Dataset

Model	**Bag Of Words**		**TF-IDF**		**Word2Vec**	
	train	**test**	**train**	**test**	**train**	**test**
Naive Bayes	0.7112	0.6144	0.8029	0.7834	0.7457	0.7357
SVM with $C = 1.0$	0.9083	0.8539	0.9097	0.8348	0.9117	0.8661
SVM with $C = 10.0$	0.9620	0.8412	0.9874	0.8018	0.9816	0.8469
SVM with $C = 50.0$	0.9893	0.8152	0.9996	0.7849	0.9992	0.8271

Table 2: Train and Test F_1-Scores of Machine Learning Sentiment Classifier in IMDB Dataset

From Tables 1 and 2, it is evident that SVMs outperform by a wide margin the Naive Bayes algorithm. Also, we can see that Word2Vec features work better in SVMs as the other ones, probably because Word2Vec can be applied better to vector space than the other 2 feature types. Also, the best value of the regularization parameter C is equal to 1.0. Consequently if we had to choose among those models we would choose the SVM with $C = 1.0$ using Word2Vec Features.

Afterwards, we evaluated the performance of the LSTM model. In order to do that, we need to tune the following hyperparameters:

1. The maximum number of tokens that we would keep in a sentence. This is mostly use to restrict the size of the sequences in order to make our model more efficient and also not have the case that a review has too much tokens and force the other ones to have huge padding, thus, the model will need more resources to train.

2. The vector size of the embeddings.

3. The dimensionality of LSTM's hidden layer.

Then, we run a Grid Search using a shell script where maximum numbers of tokens can be in (125 250 500 1000 2000), embedding size can be in (50 150 300 500 1000) and hidden layer size can be in (64 128

256 512 1024) and those experiments were run for 30 epochs(they were enough), batch size equal to 256 using Adam optimizer with learning rate equal to 0.001 and with 1 LSTM layer. Also, we kept for testing the best model in terms of Validation Loss through training and not the one from last epoch. Because we conducted 125 experiments we cannot present all the results but we can tell that we got the best results with max tokens to be equal to 500, hidden state size equal to 512 and embedding size equal to 500, where I got accuracy equal to 0.8650, which seems to be slightly worse than the best SVM model, however note that I did not used pretrained embeddings and on the Bag of Words and TF-IDF case, the LSTM model performs better. Note that some experiments were not run successfully due to resource limitations because we run the experiments on our local GPU.

Last but not least, we evaluated the performance of BERT classifier. In order to do that, we need to tune the following hyperparameters:

1. We need to choose the pretrained model. The chosen model will be froze due to lack of data and also due to resource limitations.
2. The number of max tokens that we will get from each input sequence in order to pass to the model.
3. The batch size.

Then, we run a Grid Search using a shell script where maximum numbers of tokens can be in (64 128 256 512), batch size can be in (8 16 32 64) and those experiments were run for 5 epochs(they were enough), using Adam optimizer with learning rate equal to 0.00001 and using the following pretrained models(which we froze):

1. *bert-small* which uses 4 encoding layers and produces embeddings of size 512.
2. *bert-medium* which uses 8 encoding layers and produces embeddings of size 512.
3. *bert-base* which uses 12 encoding layers and produces embeddings of size 768.

. Also, we kept for testing the best model in terms of Validation Loss through training and not the one from last epoch. Because we conducted 48 experiments we cannot present all the results but we can tell that we got the best results with *bert-medium* model with batch size equal to 8 and maximum tokens equal to 256 because that model yield 89.5% accuracy, around 3-4% better than the rest of the presented approaches. Note that some experiments were not run successfully due to resource limitations because we run the experiments on our local GPU. In summary, comparing the results of all classifiers we can see that BERT model is the best one which is expected because Transformer models are the most powerful than simple Machine Learning and LSTMs. Also, BERT needs less epochs to converge than LSTMs and also less feature engineering because there are libraries to do it for us.

In comparison with state of the art models, we can see from PapersWithCode leaderboard on this dataset on Sentiment Analysis Task, that our best approach ranks 27th, and only 3 per cent worse that the 20th one, thus we believe that our approach is pretty decent in comparison with existing ones. However, the best one uses an XL-Transformer and achieves 96.21% accuracy.

5 Conclusion & Further Work

In this project, we present three approaches in order to address the Sentiment Analysis task. Initially, we presenting a Machine Learning approach that was based on Naive Bayes and Support Vector Machines along with the preprocessing and Feature Extraction that we done. This approach gave us pretty decent accuracy of 86%. Afterwards, we focused on Deep Learning approaches, where we created an LSTM and a BERT model that yielded 86% and 89.5% accuracy respectively. In summary, we came to the conclusion that BERT is the best approach that also stands decently against existing approaches, but there is plenty room for improvement.

In terms of further work, we can initially combine the BERT approach with Traditional Machine Learning methods or LSTM, where the BERT model can work as a Feature Extractor and the SVM or LSTM as the

classifiers. Also, we can create ensembles from the aforementioned approaches where, for example, we can train those approaches and then combine their predictions to train another classifier like Logistic Regression that combines those and then make a final prediction. Lastly, we can create more powerful neural networks like XL-Transformers in order to achieve higher results like the ones displayed on the leaderboard.