THE UNIVERSITY OF SYDNEY

FACULTY OF ENGINEERING

COMP4318

Semester 1, 2023

Assignment 2 - Group: 76

1. 520600843
2. 510409755

Assignment 2 - Group: 76

1. 520600843
2. 510409755

July 6, 2023

Abstract

This study is focused on exploring different approaches to perform sentiment analysis on tweets. Hence the objective is to correctly predict the ”emotional tone” defined as either positive or negative of new unseen tweets. In this work, we compare the effectiveness of different traditional machine learning models such as Naive Bayes and Logistic Regression and the most promising approach derived from artificial intelligence, namely, LSTM. A series of ablation studies have been performed in order to understand the impact of several individual components. The performances have then been recorded making use of different evaluation metrics such as: accuracy, precision, recall and f1 score. Ultimately LSTM proved to be the best performing model.

1 Introduction

This task requires three machine learning algorithms to be implemented in order to predict the sentiment of a selected dataset of tweets as either positive or negative. Sentiment analysis is a NLP task with the goal of understanding the emotional tone of a piece of text in order to classify it as either positive or negative (and occasionally neutral). In today's day and age this task has been growing more interest as it opens up the possibility for businesses to gain valuable insights. For example, Market Research could make use of such automated tools in order to gather consumer opinions and steer product development towards the right direction. Another interesting application involves politics; performing opinion mining in order to better understand the public opinion. This particular field of application requires rigorous ethic scrutiny as it paves the way to debatable practices like those exposed in the Cambridge Analytical scandal. Driven by the growth and relevancy of this task we opted to direct our efforts in the development of a classification pipeline for NLP that would aim to tackle the problem statement:

”Can the sentiment of tweets be accurately predicted by their words with the input being the tweet itself and the output being a label of positive or negative?”

Over time the approach to solving this sort of task has been shifting from traditional Machine Learning techniques to implementing Deep Neural Networks, which as of today represents the state of the art. In our work we implemented three different models namely: Naive Bayes, Logistic Regression and LSTM. We trained the models on a smaller version of the Sentiment140 dataset which is a collection of tweets extracted using Twitter API. Each data sample comprises a tweet and is labeled as positive (“4”) or negative (“0”). The same train and test set has been used to train every model in order to make it possible to compare the performances. Hence, as far as possible,

the cleaning and pre-processing steps are common to all three models. However some differences were inevitable and are further described in section 2.2. Throughout the report we explain what sort of pre-processing we performed and how every model was implemented, presenting the different approaches and techniques that we adopted. We then present our findings in light of the evaluation metrics yielded by our models.

2 EDA and pre-processing

2.1 Premise

In order to produce a fair comparison of the three different methods implemented, the part of the pipeline relative to pre-processing was kept the same for all of the models. This "modus operandi" allowed us to really compare side by side the performances of each model independently from the quality of the underlying data set, which can of course have a big impact on the performances observed. However, the different nature of the three models necessarily required some "ad hoc" pre-processing which was therefore unique to the specific model. In the following section we describe all the operations performed over the original data set.

2.2 Data Cleaning, Pre-processing, EDA

The original dataset comprises 300000 samples and has 5 columns. For our task we are only interested in the columns "text" and "sentiment"; the first one containing the tweet and the second one the labeling of the previous as Positive (= 4) or Negative (= 0). To begin with, we focused on cleaning the tweets from any tags (i.e. @…) and URL that they might contain as well as from sequences of characters other than letters or numbers (i.e. -.-) The reason for this is due to the fact that usernames and URL links do not contribute to the sentiment of a tweet and although emoticons may present some meaning there is often unrelated punctuation used which would not have much predictive power. In order to thin out the vocabulary of words fed to our models we also filtered out stop words. To perform this step we made use of Natural Language Toolkit (nltk) importing "stopwords" from "nltk.corpus"; a collection of commonly occurring words that are considered to carry little meaning (e.g. "the", "a", "of"). Furthermore we tested two different techniques to extract the root of the words; this is necessary so that for instance singular and plural of the same word is not accounted for as two different words. To perform this task we used two functions provided by the "nltk.stem" library, namely "SnowballStemmer" and "WordNetLemmatizer". The first one is a pre-processing technique that extracts the root of a word simply by removing suffixes. As a result, however, the words yielded might not be real words. On the other hand, "WordNetLemmatizer" takes into account the context of a word in order to reduce it to its base form; in this case all the words returned are correct English words. Both approaches were tested on the different models and lemmatization proved to be the best of the two. We then mapped label 4 to Positive and 0 to Negative to make the dataset more easily interpretable. On the processed tweets we then went on with EDA to gather some better understanding of the data. First we checked whether the classes of Positive and Negative tweets were balanced (Figure 1). Indeed having highly unbalanced classes would have negatively influenced the predictive performances of our models. Training on unbalanced classes leads to biased models that as such are better at predicting the majority class.

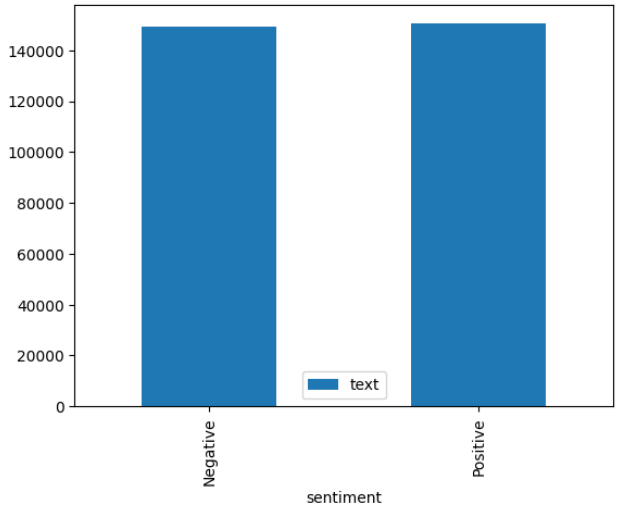
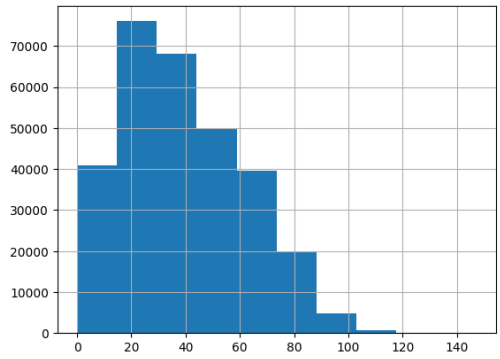
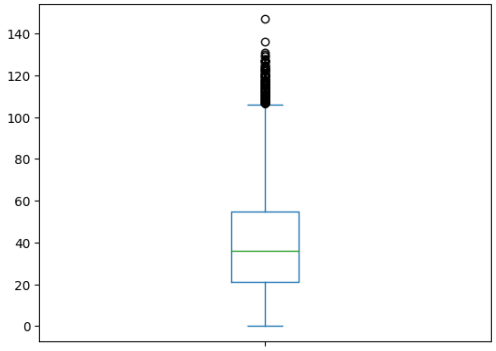


Figure 1: Classes for Negative and Positive Tweets

We then checked the distribution of the tweets' length. Indeed, in order to implement a Neural Network capable of processing the tweets all the inputs, hence the sentences, need to have the same length. Plotting the word length's distribution for the tweets we found them to be skewed right with a mean of 39.3 and a standard deviation of 22.3 (Figure 2(a) and 2(b)). Given the tweets length distribution observed, a score of 77 was chosen as the input length for the sentences fed to the LSTM. In taking this decision padding was preferred over cropping the tweets in order to make them all have the same length. Following this heuristic only a small percentage (5%) of the sentences needed to be cropped, since the cut-off value chosen was 2 standard deviations above the mean. The rationale behind it is that cropping a sentence might cause loss of key information. Moreover, cropping might disrupt the sequential structure of the input. It is obvious how in understanding the meaning of a sentence, words can not be considered singularly but the context must be taken into account. Indeed capturing the sequentiality of the input has been the main drive in the development of Recurrent neural network and LSTM (section 3.4.1 and 3.4.2 for further details). So, given that to our best knowledge, it is rather unpredictable whether the crucial information of a tweet would be stored within the beginning of it or toward the end we decided to trade memory efficiency (using larger input sequences) hoping that it would yield best results. Our tests did prove the correctness of our line of reasoning (results are presented in section 4.3.3).



(a) Histogram for distribution of Tweet's length

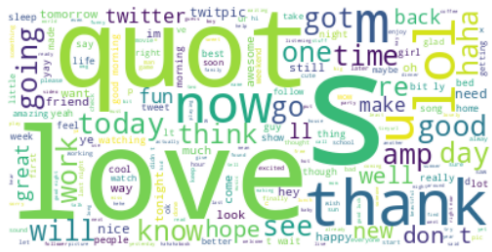


(b) Box plot for distribution of Tweet's length

Figure 2: Tweet's length

No further specific pre-processing was necessary for Naive Bayes and Logistic Regression.

Lastly we used words cloud plots (Figure 3(a) and 3(b))to get a rough understanding of which seemed to be the most common words associated with either positive or negative tweets. We found that some words seemed to be overlapping (e.g. today), whilst other frequent words did not appear to be very relevant even if very frequent (e.g. re, u, m).



(a) Cloud of words for positive tweets

(b) Cloud of words for negative tweets

Figure 3: Word Clouds

3 Models

3.1 Premise

For this investigation a variety of three different models were implemented in order to thoroughly assess how well the sentiments of the given tweets could be predicted. The three techniques were Naive Bayes, Logistic Regression and finally Long Short Term Memory (LSTM) network. The three methods introduced have a range of differences in terms of implementation as well as complexity. This broad coverage of techniques allows for greater confidence that the majority of avenues have been explored and tested.

3.2 Naive Bayes

3.2.1 Overview

The Naive Bayes algorithm is a probabilistic classifier that has been used for classification problems throughout countless machine learning tasks. The reason for this is that the Naive Bayes method has a number of benefits that make it a strong candidate towards challenging classification problems. Firstly, the model is quite simple to implement. Within a text classification task, the number of features or in this case words is linear to the number of parameters necessary to build the Naive Bayes model (Gautam et al. 2023). This allows for the model to be highly scalable for larger datasets which is typical of text corpuses such as a collections of tweets which has high dimensionality(Song et al. 2017). The scalability of the technique also doesn't come with a computational impact as the method doesnt require "expensive iterative approximation"(Gautam et al. 2023) as is the case for many other complex classifying algorithms. The major assumption behind the model, which also gives it it's name, is that the relationship between input variables are conditionally independent (Talbot et al. 2023). When applied to a sentiment analysis task it is clear that the words within that statement are dependent upon one another as together they build additional context and meaning. Take the below tweet from the given dataset as an example:

"I am so hungry, I hate not eating breakfast!"

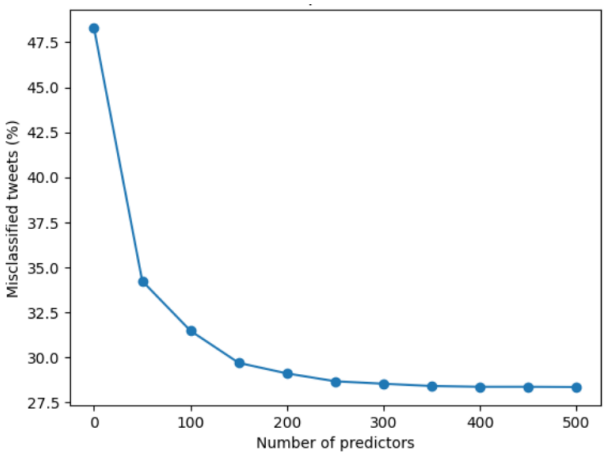
It is clear when reading the tweet in its entirety that it is a negative statement. The second part of the sentence gives further context to the first part and together forms syntax and builds a story. The Naive Bayes model, however, treats all the words in the tweet as having conditional independence. Although as a whole the tweet is clearly negative, the words 'eating' and 'breakfast' by themselves may have positive connotations and therefore appear in more sentences deemed as positive which ultimately affects the posterior probabilities which the model uses to make new predictions. So although this assumption helps with computational performance it can also impact classification accuracy and other techniques that are not reliant on this may perform better as a result of not incurring this limitation (Talbot et al. 2023). As discussed there are both advantages and limitations to the Naive Bayes method. However, it was decided that this method would be used in order to classify the tweets for a couple of reasons. Firstly, this technique has been used in the past for very similar text sentiment classification problems and seen relatively good results. Gautam et al. 2023 used a Naive Bayes approach "to determine the semantic orientation of the opinion words in tweets" and found that they achieved a fairly accurate and reliable result with 34.8% root squared error. Talbot et al. 2023 also used a Naive Bayes method to classify the sentiments of tweets. Tweets were deemed to be either positive, neutral or negative and their final model achieved a 59.26% F1-score. Song et al. 2017 used a Multinomial Naive Bayes algorithm in order to classify the sentiments of tweets after doing some bespoke attribute weighting which resulted in a maximum accuracy of 85.33%. As can be seen this technique is quite popular in similar tasks and although it does not yield outstanding results it is still very well regarded. The second justification for using this technique was that due to the simplicity of the model it would be quite beneficial to act as the baseline model. This would mean that we could use the results of the classifier as a benchmark when testing other approaches that may be somewhat more complex and computationally intensive.

3.2.2 Implementation

The way this Naive Bayes method was implemented was using the BernoulliNB function that is found in the sklearn.naive_bayes module. The reason for this is that the Bernoulli is a simplified version of the Multinomial Naive Bayes method which is often used in text sentiment classification tasks. It is therefore less prone to overfitting and because of this was selected as the preferred approach of applying Naive Bayes. As there are 300,000 tweets containing over 94,000 unique words/tokens after preprocessing and the Naive Bayes method takes the number of features as the number of parameters within the model, the first step is to reduce the number of dimensions for the model. The way this was done was by only including words that appear in at least 500 tweets. Using the CountVectorizer function from the sklearn.feature_extraction.text module the frequency of words was able to be calculated and filtering on words that only appear in at least 500 tweets the total pool of words to only 542. Another step was taken to reduce dimensionality further. This involved building a Bernoulli Naive Bayes with a single parameter one by one for each selected word that appeared in at least 500 tweets. The log loss of each feature was then calculated to see its predictive power on classifying a tweet. The top 10 most predictive features are presented in Figure 4(a)

	Loss NB
sad	0.684496
thanks	0.685916
miss	0.686977
love	0.688392
hate	0.688888
good	0.689118
sick	0.689261
bad	0.689764
thank	0.689846
sorry	0.689933

(a) Top 10 Predictive Words



(b) Misclassification vs Number of Predictors

Figure 4: Naive Bayes Exploration

As can be seen these words inherently have rather strong positive or negative connotations with words like good,bad,love and hate appearing revealing that these words commonly appear when a tweet is classified correctly. It could now be assessed how many of the features would be required to actually help the model predict the sentiment. A function was defined which, after ordering the features by log loss, iteratively built an NB model (with a cross validation of 5 folds) by adding one of the features each build. The misclassification error from this process was then stored and the results are presented visually in Figure 4(b). As can be seen initially, the first couple of hundred features that are added results in significant reduction in misclassification error. However, this flattens out as more and more are added. After 350 features which resulted in a misclassification error of 28.4%, any further parameters that are added do not contribute to reducing the error. As such it was decided that only the top 350 predictive words were required to build the final Naive Bayes model which reduced the dimensionality further.

3.3 Logistic Regression

3.3.1 Overview

The Logistic Regression is another type of popular machine learning algorithm that has been widely used for binary classification tasks such as text sentiment classification. This statistical method uses a logistic (sigmoidal) curve to fit its data to. By fitting this curve, it produces a probability between 0 and 1 that the given features will result in one of the two possible outcomes, which in this case are positive or negative tweets. With a threshold of 0.5, predictions above this level would be assigned a positive class and those below would be assigned a negative class. The following equation gives the probability of the sigmoidal or logistic curve. where p is the probability, b0 and b1 are weights of the parameters and x is the input.

The Logisitic Regression method has a number of benefits that make it worth utilising for this twitter sentiment task. Firstly it is able to be implemented relatively simply due to low number

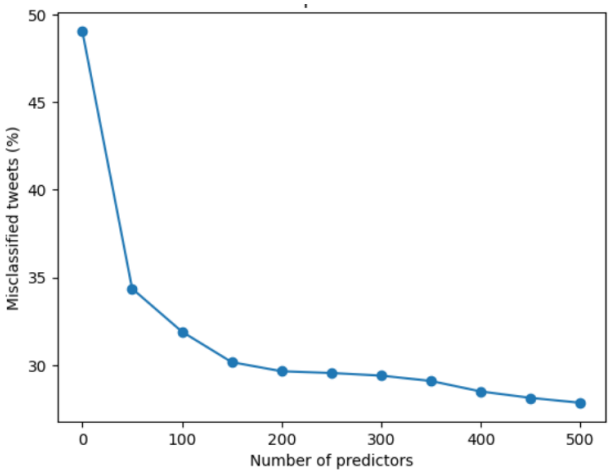
of hyper-parameters. As a result of this it means that it is able to be scaled to larger complex datasets quite easily which makes it a good candidate for a larger number of text based features. This is revealed in a number of previous similar works. Kumar et al. 2016 aimed to classify the sentiment of Amazon product reviews as either positive or negative. They opted to use a Logistic Regression model and found decent performance of 0.59 to 0.74 f1-score depending on the product being reviewed. In a more similar problem to this investigation Poornima and Sathiya Priya 2020 used Logistic Regression to classify the sentiment of 1.6 million tweets. The large dataset meant that the Logistic Regression model was well suited to be able to handle this problem and they found that they achieved an accuracy of 86%. Finally, in other instance of twitter sentiment classification (Rajasekhar V3ISSUE11) had to predict whether a tweet would be positive, neutral or negative. After implementing a Logistic Regression method, their final model achieved an accuracy of 82.6%. These previous works reveal that the Logistic Regression technique does have justification for being incorporated into a piece of work such as this. However, there are some limitations that should be addressed. The Logistic Regression model may not be able to capture more complex relationships or patterns as the assumption behind the model requires there to be a generalised linear relationship between the explanatory features and the target variable in the logit space Kumar et al. 2016. In this way it is similar to the Naive Bayes method as it adds a weight to linearly to the number of parameters and sums them up and then applies a function in this case the sigmoid Kumar et al. 2016. Overall, due to its ability to work well with large datasets and it's simplicity and interpretability it was decided that this method would be used for the classification task.

3.3.2 Implementation

The way in which the Logistic Regression classifier was implemented was using the SGDClassifier function from the sklearn.linear_model module. By setting the loss parameter to 'log_loss', the function uses a logistic regression classifier with stochastic gradient descent learning, which is helpful for computational efficiency for larger datasets (scikit-learn contributors 2023). The overall structure of the code follows the same steps as the Naive Bayes due to the models having similarities as well as for consistency. Firstly, individual classifiers are built with single parameters in order to find out which words have the most predictive power. The log loss of each of these features is calculated and then ordered. The top 10 features are presented in Figure 5(a).

	Loss SGD
sad	0.684676
thanks	0.686238
miss	0.686989
love	0.688427
hate	0.688912
good	0.689215
sick	0.689788
bad	0.689923
thank	0.690100
suck	0.690292

(a) Top 10 Predictive Words



(b) Misclassification vs Number of Predictors

Figure 5: Logistic Regression Exploration

As can be seen once again very strong emotive words come out as the most predictive with considerable overlap to what was seen in the Naive Bayes model. Once in order the features are added one by one and a classifier is built and the misclassification rate is calculated. The results are shown in Figure 5(b). Unlike the Naive Bayes which saw performance level out, it appears that the Logistic Regression method does see continued performance improvements as more features are included, reaching its lowest error rate of 27.9% at 500 features. This meant that the final Logistic Regression model was ready to be built.

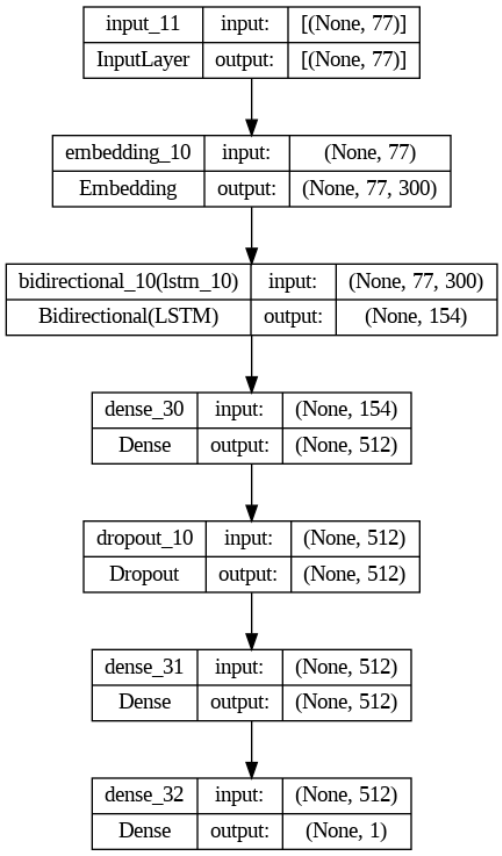
3.4 Bidirectional LSTM

3.4.1 Overview

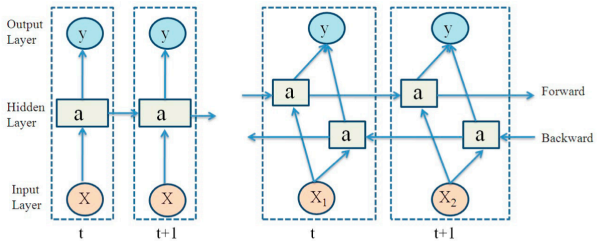
In NLP tasks the context in which a word is used is key. Indeed it is obvious how, to successfully grasp the meaning of a sentence, it is crucial to understand, not only the word itself, but also its relationships with other words within the sentence. In light of this fact there is a specific class of Deep Neural Networks that particularly fit this task given their capability to remember information over long periods of time. RNN indeed are designed so that at any given time a state is a function of the current input and all the previous states. This is made possible thanks to an architecture based on cells that maintain a hidden state. As for the functioning, RNNs receive in input a sequence, let's say a vectorized word from a sentence; the word is then processed and an output as well as an hidden state are produced. The hidden state is then fed back into the NN together with the input of the following timestamp. However RNNs suffer from the problem of vanishing gradient. Indeed RNN are trained with Backpropagation which can cause the gradients to get exponentially smaller preventing the network from learning. A more advanced architecture that successfully deals with this problem is that of the LSTM. This Neural Networks solve the issue of the vanishing gradient thanks to the addition of a memory cell that can selectively remember or forget information fed to it. This cell encompasses 3 gates: input, forget and output gate. The paper (Mahadevaswamy and Swathi 2023) is used as a guideline in the implementation of the Neural Network hereafter discussed.

3.4.2 Architectural Implementation

For the implementation of the NN Keras API was used and a sequential model was defined. The overall architecture is depicted in figure 6(a). The input is fed to an Embedding Layer. This layer is crucial to learn a contextual representation of the input. Indeed it produces a representation of the input in the form of a dense vector that attempts to capture the semantic meaning of the sequential input. At the output of this layer words that appear to be close in meaning will be represented with vectors that are close in the embedding space. As for the specific set up of this layer we experimented different alternatives which are presented in section 4.3.2. After the embedding layer a Bidirectional LSTM is then implemented. For this setting we opted for the bidirectional version of LSTM, which is nothing but two consequential LSTM. This variation allows capturing information both from the past and future context given that the input is processed twice, once forward and once backward as visible from Figure 6(b). Other than this, Bidirectional LSTM function just like regular LSTM with each unit processing one word embedding. Following the LSTM we can find 3 Dense Layers. In particular the last one uses Sigmoid as an activation function in order to produce a probabilistic output that represents the confidence that we have in labeling a certain tweet as positive or negative. Given the binary nature of the classification task 0.5 was used as the cutoff value.



(a) Architecture of the Neural Network



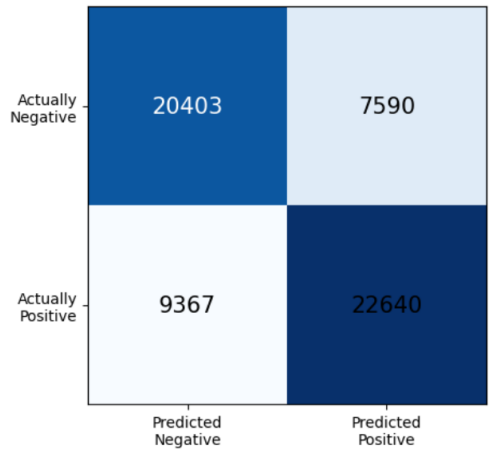
(b) LSTM (left) vs Bidirectional LSTM (right)

Figure 6: Comparison of different LSTM architectures

4 Experiments and Results

4.1 Naive Bayes Model

After establishing the best setup for the Naive Bayes method, it was now time to perform some experiments and fine tune the model in order to yield the best possible results. As discussed the Naive Bayes method is a relatively simple model which means there is only the alpha hyper-parameter to fine tune. Alpha represents the smoothing parameter which helps tackle the problem of zero probability. In order to check what the optimal value of alpha was for this particular problem, the GridSearchCV function was used from the sklearn.model_selection module. This allowed for a 10-fold cross validation to be performed on the Bernoulli Naive Bayes model for every given value of alpha. In this case a parameter grid with 5 values of alpha was chosen (0.001,0.1,1,10,100). The best test set result was using an alpha value of 1. The final Naive Bayes method was then built with an alpha of 1 and the confusion matrix and classification report are shown below in Figure 7(a) and 7(b). As can be seen this resulted in an f1 score of 72%. This represents a fairly good accuracy for a simpler model, with the model predicting positive and negative classes at about the same rate (73% and 71% respectively).



(a) Confusion Matrix

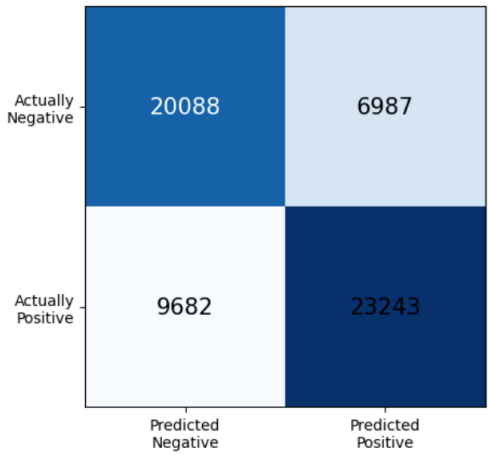
	precision	recall	f1-score	support
Negative	0.73	0.69	0.71	29770
Positive	0.71	0.75	0.73	30230
accuracy			0.72	60000
macro avg	0.72	0.72	0.72	60000
weighted avg	0.72	0.72	0.72	60000

(b) Classification Report for Naive Bayes

Figure 7: Naive Bayes Results

4.2 Logistic Regression Model

After establishing the design for the final Logistic Regression model it was now time to set up a cross validated grid search using the GridSearchCV function for robustness. The only hyperparameter of the Logistic Regression using SGD is alpha which, similar to the Naive Bayes, represents the smoothing parameter. A grid of six values were taken (0.0001,0.001, 0.1, 1, 10, 100) and the best test set accuracy came from alpha set as 0.0001. The confusion matrix and classification report for this final model are shown below in Figure 8(a) and 8(b). This model achieved an f1-score of 72% predicting positive and negative classes at 74% and 71% respectively.



(a) Confusion Matrix

	precision	recall	f1-score	support
Negative	0.74	0.67	0.71	29770
Positive	0.71	0.77	0.74	30230
accuracy			0.72	60000
macro avg	0.72	0.72	0.72	60000
weighted avg	0.72	0.72	0.72	60000

(b) Classification Report for Logistic Regression

Figure 8: Logistic Regression Results

4.3 LSTM Model

Different experiments have been performed in order to identify the best performing set up for the NN. Hyper parameters tuning has been performed splitting the training set in training and validation (with an 80-20% proportion). Doing so allowed us to obtain loss and accuracy scores for the validation set which are the metrics that have ultimately been used in order to take several decision regrading the hyper parameters.

4.3.1 Impact of Simple LSTM vs Bidirectional LSTM

First of different variations of the architecture were tested especially comparing an implementation with LSTM versus its Bidirectional counterpart. The second one proved more promising based on the results yielded.

4.3.2 Impact of Embedding Layer Set up

Three main approaches were tested in the set up of the Embedding Layer. However the input size of the layer was kept fixed and equal to the number of unique words found in the training set. The embedding size as well was set to 300. A common technique to be applied in such a layer is that of transfer learning. Indeed in our implementation we exploited GloVe (taking inspiration from Vidhya 2021), an unsupervised learning algorithm that produces word embedding. In particular we used GloVe.6b.300d which encompasses word vectors with a size of 300 words. As a result we obtained a set of weights that we used to initialize the embedding layer which was then either re-trained, to hopefully better fit our specific data set, or frozen. In comparison to these two techniques we also tested the performances without initialising the embedding layer with any specific set of weights. Ultimately the second approach, hence initialization of the layer and freezing it, was chosen since it yielded better accuracy scores on the validation set as well as lower values for the validation loss. Moreover freezing the embedding layer allowed to drastically reduce the number of training parameters going from over 28M to roughly 500 thousands.

4.3.3 Impact of Input Vector Size

Once the overall architecture was chosen we focused on the input vector size. As explained in section 2.2 that is a key parameter to be decided. Hence, to back up the rational that drove our

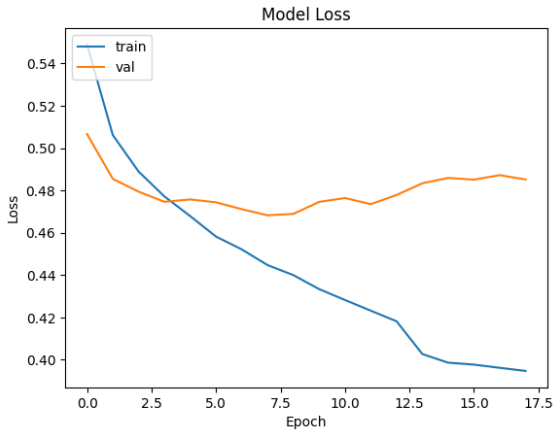
decision for the cut off value we tested 3 different values: 30, 77 and 100. In order to effectively test how changing the input size would effect the performances we kept all the other model's parameters and hyper parameter fixed. However the number of units for the LSTM layer necessarily had to be changed. Indeed every LSTM's unit has to receive one word embedding from the embedding layer. Hence, together with the input size also the Bi-directional LSTM layer was changed in order to encompass 30, 77 or 100 units. As a result an input size of 77 proved to obtain slightly better results compared to the 2 alternatives.

4.3.4 Impact of Batch Size

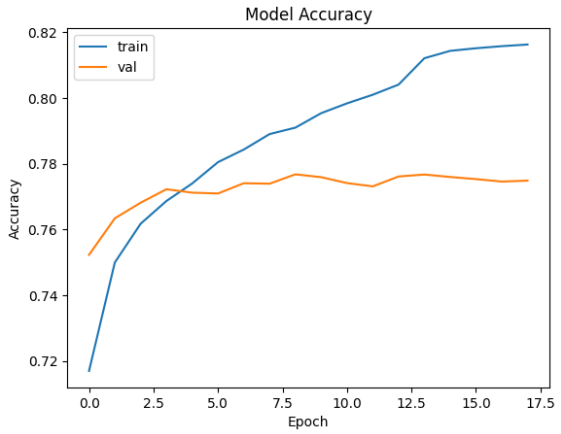
Different values were tested for this hyper parameter: 256, 512 and 1024. 512 was ultimately chosen as it is the one value that better balanced performances and training time. Indeed increasing the batch size gradually decreased the time necessary for each epoch, however it also worsen the loss and accuracy scores.

4.3.5 Impact of Callbacks

In order to optimize the training process we implemented adaptive learning rate as well as early stopping. The first one is a scheduling technique that makes use of the ReduceLROnPlateau model of Keras. This allows us to dynamically adjust the learning rate throughout the training process based on the monitoring of a specified metric, which in our case was chosen to be the validation loss. This call back starts decreasing the learning rate when it seems to stop improving based on the parameter monitored. For the training we started off with a learning rate of 0.1 and set 0.00001 as lower bound. Another module that we exploited in the training phase is EarlyStopping (from Keras). This technique allows to interrupt the training before the last epoch occurs in case a given parameter stops improving. Once again the parameter monitored was the validation loss. In order to have a better understanding of the proceeding of the training we produced the plots of Figure 9(a) and 9(b). In both cases it can be seen how the model gradually improves the training loss and accuracy whilst the validation scores stop improving pretty soon in the training process. That is a clear sign of a turning point at which our NN starts over fitting the input undermining its generalization power. However thanks to "ModelCheckpoint" module of Keras we saved the weights that are computed right at the turning point and used those for the final model.



(a) Comparison of training and validation loss



(b) Comparison of training and validation accuracy

Figure 9: Training and validation performance

4.3.6 Evaluation Metrics

In Figure 10 we report several evaluation metrics.

	precision	recall	f1-score	support
Negative	0.78	0.76	0.77	29770
Positive	0.77	0.79	0.78	30230
accuracy			0.77	60000
macro avg	0.77	0.77	0.77	60000
weighted avg	0.77	0.77	0.77	60000

Figure 10: Evaluation metrics

All of the metrics have been computed over the predictions performed on a test set, which, as such, was never disclosed at training time (indeed for hyper parameter tuning we made use of a validation set). The macro and weighted scores of each metric achieved equal scores thanks to the fact that the training set used was balanced. The support shows that also the test set is balanced in between positive and negative tweets. Furthermore, from the Confusion Matrix in Table 1 it can be seen how the final model is not bias toward any sort of error (type I or type II) as the number of FP and FN roughly balance each other out.

		Actual Class
	Positive	Negative
Predicted Class	22648 (TP)	7122 (FP)
	6446 (FN)	23784 (TN)

Table 1: Confusion Matrix

4.4 Model Comparison and Reflection

The final model outputs are summarised in 2. It is important to assess a model in terms of computational time as well as accuracy. As can be seen the more complex LSTM model has a better accuracy of 77% compared to the other two simpler models. However, the training of this model took significantly more time taking roughly an hour to train as opposed to the 10-13m that the simpler models took. This is important to consider especially if wanting to perform such an analysis on more data. Overall, the three chosen models were appropriate for the given task. They have been used widely before for such sentiment classification tasks and they represented a variety of methods and complexity. It can be seen in the methodology that measures were put in place to improve performance and optimise each of the models implementation through experiments and exploration and hyper-parameter tuning.

Table 2: Model Comparison

Model Type	Accuracy	Time	Complexity
Naive Bayes	0.72	10-12m	Low
Logistic Regression	0.72	10-13m	Low
LSTM	0.77	60m	High

5 Summary and future works

Overall when addressing the problem statement, it can be concluded that the sentiment of tweets can be accurately predicted by the contents of the tweet. The best achieved accuracy was 77% which was achieved by the LSTM method. This was to be expected as the LSTM model has a much higher complexity allowing it to capture both semantic and syntactic relationships that the Naive Bayes and Logistic Regression model are unabale to achieve. However, in saying that it is important to consider that although the two simpler models suffered a 5% lower accuracy, they are much quicker to run. If this task was to be implemented in a business context or applied on more data it is possible that sacrificing some accuracy in favour of a computationally less expensive method would be preferred. The possible reason for the relatively good performance of the simpler models, is that the actual presence of certain words is enough to give an accurate prediction as opposed to having to take in the context and relationships between words.

There are a number of potential improvements and future applications that could be applied to this work. Firstly, a number of other models could be applied to see if better results could be achieved. For example Support Vector Machine or other neural networks such as Convolutional Neural Network or Deep Neural Network. The other improvement could be to assess the given models on other twitter data sets to ensure robustness and good generalisation. In terms of future work there are a number of other tasks that could be implemented. For example the sentiment could be more granular like predicting a neutral class or even a specific emotion such as anger or love. Another piece of future work could be to classify the sentiment of multiple languages so the application could be more broad globally. Finally, one more future work could be to focus on how to deal with language forms such as irony or sarcasm which are quite difficult to isolate from standard language but would be very powerful in helping predict the sentiment.

References

Gautam, Jyoti, Mihir Atrey, Nitima Malsa, Abhishek Balyan, Rabindra Nath Shaw, and Ankush Ghosh (2023). *Twitter Data Sentiment Analysis Using Naive Bayes Classifier and Generation of Heat Map for Analyzing Intensity Geographically*. City: Publisher.

Kumar, Santhosh K.L., Jayanti Desai, and Jharna Majumdar (Dec. 2016). "Opinion mining and sentiment analysis on online customer review." In: *2016 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*. Nitte Meenakshi Institute of Technology. DOI: 10.1109/ICCIC.2016.7919584. URL: https://ieeexplore.ieee.org/document/7919584.

Mahadevaswamy, U.B. and P. Swathi (2023). *Sentiment Analysis using Bidirectional LSTM Network*. Vol. 218, pp. 45–56. DOI: 10.1016/j.procs.2022.12.400. URL: https://doi.org/10.1016/j.procs.2022.12.400.

Poornima, A. and K. Sathiya Priya (2020). "A Comparative Sentiment Analysis Of Sentence Embedding Using Machine Learning Techniques." In: *2020 6th International Conference on Advanced Computing & Communication Systems (ICACCS)*. PSG College of Technology. Coimbatore, India. URL: https://ieeexplore-ieee-org.ezproxy.library.sydney.edu.au/stamp/stamp.jsp?tp=&arnumber=9074312&tag=1.

Rajasekhar, Kinjarapu (V3ISSUE11). "Sentimental Analysis on Instagram Using Machine Learning Techniques." In: *International Journal of Research Publication and Reviews*, V3ISSUE11. URL: https://ijrpr.com/uploads/V3ISSUE11/IJRPR7843.pdf.

scikit-learn contributors (2023). *scikit-learn SGDClassifier*. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html.

Song, Junseok, Kyung Tae Kim, Byungjun Lee, Sangyoung Kim, and Hee Yong Youn (2017). "A novel classification approach based on Naïve Bayes for Twitter sentiment analysis." In: *Journal Name* Volume Number.Issue Number. *Corresponding author: Hee Yong Youn, Page Range.

Talbot, Ruth, Chloe Acheampong, and Richard Wicentowski (2023). "SWASH: A Naive Bayes Classifier for Tweet Sentiment Identification." In: *Proceedings of the Conference Name*. Swarthmore College. Swarthmore, PA, USA.

Vidhya, Analytics (Dec. 2021). "Sentiment Analysis on Tweets with LSTM for Beginners." In: URL: https://www.analyticsvidhya.com/blog/2021/12/sentiment-analysis-on-tweets-with-lstm-for-beginners/.

6 Appendix

6.1 Instructions to run the code

The code is meant to be run sequentially. The first 3 cells of code however should be either commented or modified depending on the environment the code is being run in. The very first cell is only required if running the code in GoogleColab and should otherwise be commented out. The second cell is used to change the current directory to the one where the data set is stored, hence the path is dependant to the local folder's structure and should be modified accordingly (otherwise it will be necessary to use absolute path in cell 3). Cell 3 contains the name of the data set, however if cell 2 command is not run it should be modified so that it contains the absolute path to the data set. For the LSTM please un-comment the cell responsible of downloading GloVe (first cell in the Words embedding section).

7 Experimental Environment Specifications

The pipeline was run on Google Colab Pro with the following Hardware specifications:
83.5 GB RAM
166.8 GB Disk
40 GB GPU
Python version: 3.10.11 (main, Apr 5 2023, 14:15:10) [GCC 9.4.0]
pandas version: 1.5.3
nltk version: 3.8.1
keras version: 2.12.0
numpy version: 1.22.4
seaborn version: 0.12.2
sklearn version: 1.2.2
tensorflow version: 2.12.0