ASSIGNMENT 2

GROUP 144

Aryan Bhatia (abha6244, 490352056)

Mudit Malhotra (mmal4153, 490583269)

TABLE OF CONTENTS

INDIVIDUAL CONTRIBUTION **2**

ABSTRACT **2**

INTRODUCTION **2**

PREVIOUS WORK **3**

METHODOLOGY **4**

CLASSIFIERS 4

Naïve Bayes *4*

Logistic Regression *5*

Convolutional Neural Network (CNN) *5*

PRE-PROCESSING 6

Data Cleaning *6*

Feature Extraction using TF-IDF FOr naïve bayes and logistic regression *6*

Feature extraction using tokenizer and sequence padding for cnn *6*

EXPERIMENT **7**

IMPROVING MODEL PERFORMANCE USING PARALLELIZATION 8

CNN (BEST ALGO) 8

NAÏVE BAYES V/S LOGISTIC REGRESSION 11

DISCUSSION **12**

CONCLUSION AND FUTURE WORK **13**

REFERENCES **13**

APPENDIX **15**

SYSTEM SPECIFICATIONS 15

LIBRARY REQUIREMENTS 15

INSTRUCTIONS TO RUN CODE 15

INDIVIDUAL CONTRIBUTION

Aryan Bhatia (abha6244, 490352056):

- Naïve Bayes
- Logistic Regression
- Parallelization

Mudit Malhotra (mmal4153, 490583269):

- Convolutional Neural Network (CNN)

Data cleaning and pre-processing was a combined effort by both team members.

ABSTRACT

Twitter is a popular application for microblogging where users share their views and opinions on different topics in form of tweets. Each tweet can have a maximum of 280 characters and a user's sentiments in a tweet can be categorised in binary as *'positive'* and *'negative'* [2]. This type of data can be highly beneficial to understand the opinion of people on various topics.

For this assignment, we are using the *Sentiment140* dataset provided in the assignment specification document. Our aim is to classify users' sentiments by performing supervised learning on the data using the following classifiers/deep learning models:

- Naïve Bayes
- Logistic Regression
- Convolutional Neural Network (CNN)

The performance of the models created using these classifiers is evaluated by calculating the prediction accuracy, precision, recall and F1 score on test set & visualising the performance using ROC.

INTRODUCTION

Sentiment analysis is the computational identification of opinions expressed in a text form and categorising it as '*positive'* and '*negative'*. Twitter is a large source of short texts (tweets) containing users' opinions towards a particular topic/product and our aim is to perform accurate sentiment analysis on such data through the means of machine learning.

Performing sentiment analysis on twitter data is of great use for companies/brands to understand their customers' emotions towards their marketing campaigns, products, services etc. Furthermore, analysing emotions of people on twitter can be of significant help for governments to gain insights about public opinion towards new policies [3]. The ability to quickly analyse public's sentiments on social media applications like twitter helps such entities in quickly enhancing their strategies and plans.

Information about the Sentiment140 dataset

The data provided for this assignment consist of 1,600,000 tweets and 6 features:

- *target*: the polarity of the tweet (*0* = negative, *4* = positive)
- *ids*: The id of the tweet (*2087*)
- *date*: the date of the tweet (Sat May 16 23:58:44 UTC 2009)
- *flag*: The query (*lyx*). If there is no query, then this value is NO_QUERY.
- *user*: the user that tweeted (*robotickilldozr*)
- *text*: the text of the tweet (*Lyx is cool*)

Out of these features, *target* and *text* are used to train the models. Data pre-processing is performed on the *text* data as it is unstructured and can be difficult to use for machine learning.

Overview of the methods

Naïve Bayes is a simple, probabilistic classifier and provides a reasonable accuracy while being highly efficient. For sentiment analysis, it utilizes conditional probability to classify words into their respective categories.

Logistic Regression uses sigmoid function which outputs values between -1 and 1 and therefore enables binary classification for sentiment analysis. Furthermore, it is easy to implement and efficient to train.

CNN, from its name, is a neural network that has one or more convolutional layers. It also consists of pooling layers to extract high level features. For sentiment analysis, it helps in identifying relevant patterns in text, irrespective of their position in the sentence since it is a translation invariant.

PREVIOUS WORK

Natural Language Processing and Sentiment Analysis are topics that have grasped the interest of many Data Scientists and Engineers since the 1950s. Twitter Sentiment Analysis is a very popular subset of this field and many researchers have dabbled with it previously to release papers that brough forwards effective solutions. Since we are dealing with the Sentiment140 dataset, we looked at previous studies utilising the same dataset as our reference point.

The first research paper we would like to refer is "Real Time Sentiment Analysis of Tweets using Naïve Bayes" [4]. The authors implemented the Naïve Bayes algorithm using NLTK and utilised SentiWordNet along with it to improve its accuracy. They train on the entire dataset of 1.6 million tweets. The final accuracy they managed to achieve was 58.4% on the most recent 100 tweets. This is where the difference lies between their implementation and ours. The system we possessed didn't have the computation power to train a model using 1.6 million tweets and over 1000 features. Therefore, we had to train using 100,000 tweets and instead of testing it on the most recent 100 tweets, we tested using a dataset of 42000 tweets from the Sentiment140 dataset itself making it a 70-30 train-test split (stratified). Additionally, while most of the data-cleaning steps performed in this study were like the ones we performed, we decided to remove emoticons while they decided to classify them as positive or negative. The reason we strongly believe that our approach is better here is because, in recent times, the use of emoticons has completely transformed. Different generations use emojis differently and it is easy to get multiple meanings from the same

emoticon. For example, in Figure 1, the model proposed in the study would fail miserably since it would classify the tweet as negative based on emoticons even though it is a positive message. Therefore, it is better to go by the words themselves.

Figure 1: Wrong Emoticon

The second research paper, "A Novel Machine Learning Approach for Classification of Emotion and Polarity in Sentiment140 Dataset" [5], also utilises Naïve Bayes and implements similar data cleaning and pre-processing steps, however, again, the model was trained on the entire Sentiment140 dataset and then tested on 2000 tweets from the same dataset. The accuracy they achieved in this manner was 84.2%.

The metrics reported by these papers cannot be treated as a direct point of reference for our work since the choice of training and testing data is significantly different. Additionally, these papers only focused on Naïve Bayes, but we also implemented Logistic Regression and Convolutional Neural Network. Since papers published on Sentiment140 using the other two models were scarce, we had to look beyond research papers at some articles that followed a similar approach to us on the same dataset. Based on these articles we understood that our goal should be to achieve an accuracy of over 70% on all three models if we are performing our experiments on a 70-30 train test split (stratified) [6]. Furthermore, one the best performing models in the past on this dataset has been Naïve Bayes.

METHODOLOGY

CLASSIFIERS

NAÏVE BAYES

One of the most prominent examples of Probabilistic Classifiers. The idea behind Naïve Bayes is to predict the probability that a given example belongs to a particular class. It is a supervised learning algorithm based on Bayes Theorem.

Formula for Bayes Theorem[1]:

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)}$$

Where:

- H, E represent events
- P(H|E) is the probability of the occurrence of H given that E has occurred
- P(H) and P(E) are independent probabilities of event H and E

Why is it Naïve and why was it still chosen?
The reason this algorithm is usually not considered effective and rather "Naïve" is because it works with the assumption that the attribute values are conditionally independent of each other for each class value[1]. This is unrealistic in a real-world scenario.

However, the problem at hand is a **supervised**, **binary** classification problem. Naïve Bayes is one of the most used algorithms for such scenarios. Additionally, despite the "naïve" assumptions, this algorithm is still the best in terms of the time taken to train the model compared to other binary classifiers[4].

LOGISTIC REGRESSION

Logistic regression is also a supervised learning algorithm that gives the probability that a given datapoint belongs to a particular class. Unlike Naïve Bayes, it uses the sigmoid function for the probability calculation.

Formula for sigmoid function:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Why was it chosen?
Once again, the problem at hand is a **supervised**, **binary** classification problem. Logistic Regression is one of the most used algorithms for such scenarios. Theoretically, this should give better results compared to Naïve Bayes because it doesn't make any "naïve" assumptions.

CONVOLUTIONAL NEURAL NETWORK (CNN)

CNN is a type of neural network consisting of 1 or more convolutional layers. Convolutional layer in a neural network helps in detecting edges, corners and multiple textures in an image and enables the model to detect more complex features. For text data provided in the chosen dataset, a word embedding layer needs to be created, followed by 1-dimensional convolutional layer/s for a CNN to be trained for sentiment analysis [7]. Word embedding represents the density of the word vector. This enables the embedding layer to map semantically similar words. A pooling layer in the neural network helps in reducing the number of parameters and computation time in the network which further helps with overfitting and is therefore an integral part of this neural network when dealing with a large text data for sentiment analysis[1].

Why was it chosen?
Performing accurate sentiment analysis on a text data from a tweet is highly complex as similar words used in different patterns can depict different sentiments. CNN being translation invariant, can detect patterns effectively irrespective of the position of words in a sentence. Each filter in the convolutional layer detects a particular feature. In the case of sentiment analysis, it enables to differentiate between positive ('happy', 'excited') and negative ('sad', 'angry') words. One of the key components of sentiment analysis is to detect the presence/absence of key phrases which can be present anywhere in the sentence [8]. CNN can be an effective model for this as it is effective at extracting positional-invariant and local features from the data[1].

PRE-PROCESSING

DATA CLEANING

As the text data is unstructured, it consisted of a lot of noise. Therefore, following data cleaning steps were performed to extract the relevant information from the text data:

- Dropping duplicates
- Converting the text to lower case
- Removing at sign for user mentions
- Removing URLs
- Removing punctuations
- Removing numbers
- Removing emojis
- Removing stop words
- Removing 1 letter words
- Lemmatization

FEATURE EXTRACTION USING TF-IDF FOR NAÏVE BAYES AND LOGISTIC REGRESSION

After performing data cleaning on the text field of each tweet, features were extracted using the TF-IDF (Term Frequency – Inverse Document Frequency) algorithm [9]. This algorithm extracts the most relevant words in a document or a set of documents (in this case, tweets). These words are then used as features to train the model. The relevancy of a word is calculated using two measures:

Term Frequency: The frequency of the word in a particular tweet as it helps the algorithm decide how relevant that word is in determining the sentiment of that tweet.

Inverse Document Frequency: An offset term that represents the overall frequency of the word in the dataset of tweets. For example, a word like "that" would be very common in the dataset, so if it occurred frequently in a particular tweet, it shouldn't have a high weightage in determining its sentiment.

We chose the 1000 most relevant words using this method. This number was decided based on our system limitations, as whenever we set the number of features to be greater than 1000, the kernel used to crash.

FEATURE EXTRACTION USING TOKENIZER AND SEQUENCE PADDING FOR CNN

The cleaned text data is further processed to be able to go through a CNN model. This is done by:

Tokenizing: Using Keras' inbuilt tokenizer API, the sentences from text column are split into words and a dictionary of all unique words found is created along with uniquely assigned integers. Maximum number of words here is set to be 10000. Each sentence is converted into an array of integers which represents all the unique words present in it. *292527* unique tokens were found after

Sequence Padding: The sequence of arrays generated from tokenizing can be of different sizes. Therefore, to make the 2-D array uniform, the arrays representing each sentence are filled with zeroes to the left to

equalize the size of all the arrays. Since the maximum sequence length of a sentence in our data is less than 50, we set the maximum sequence length as 50 so that no data is lost.

EXPERIMENT

The problem at hand is a classification task, therefore, we used the following metrics to evaluate the performance of our models:

- Accuracy

$$accuracy = \frac{Correctly\ Identified\ Samples}{Total\ Samples}$$

- Precision

$$precision = \frac{TP}{TP + FP}$$

- Recall

$$recall = \frac{TP}{TP + FN}$$

- F1 Score: This is a metric that combines Precision and Recall making future comparisons easier

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

- Confusion Matrix
  The confusion matrix is a matrix that summarises the TP, FP, TN, FN. (Figure 2) [10]

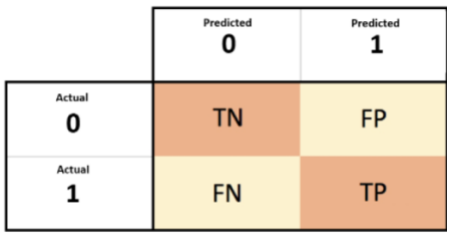


Figure 2: Confusion Matrix[7]

- Area Under Curve for ROC Curve
  Used to visualise the performance of a classifier. An ROC Curve or Receiver Operating Characteristics Curve is the plot of the precision v/s recall. The area under this curve is a value between 0 and 1.

Key:
TP -> True Positives
TN -> True Negatives

FP -> False Positives
FN -> False Negatives

IMPROVING MODEL PERFORMANCE USING PARALLELIZATION

In our code you will see the use of python's `joblib` library for the purpose of **Multi-threading** while training models. We have also implemented **Multi-Processing** during parameter tuning by setting the value `n_jobs` parameter of `GridSearchCV` as 1. This instructs GridSearch to use all the cores of the system while performing cross validation to find the best parameters for the classifier.

Multi-threading v/s Multi-processing

- **Multi-Threading** is the ability of a **processor** to execute multiple threads concurrently where each thread runs a process.
- **Multi-processing** is the ability of a **system** (CPU) to run multiple processors (cores) concurrently where each processor can run one or more threads.

CNN (BEST ALGO)

After data cleaning and performing pre-processing, a training-validation-testing split of (80-10-10 split) is performed on the subset of data containing 1 million observations chosen randomly. The data subset is taken to reduce the computational time as neural networks are highly complex and take a significant time to train.

To prepare the embedding layer, firstly, an embedding index dictionary is created using GloVe word embeddings taken from *'glove.6B.300d.txt'* [11]. It consists of words as keys and their corresponding embedding list as values. This is followed by creating an embedding matrix in which each row corresponds to the index of the word which is taken from embedding index dictionary. The dimension of the matrix is 1000x300, where the 300 columns represent the GloVe word embeddings.

The neural network model is created with the following steps in order:

1. **Input Layer:** Input layer takes an array of length 50, which is the sequence length set during the data pre-processing.
2. **Embedding Layer:** The layer will have an input length of 50, which is the sequence length set during data pre-processing. The output vector dimension will also be 50. Since embeddings are generated from external source, we set the training attribute as *False*. We pass the embedding matrix to the weights attribute.
3. **1D Convolutional Layer 1:** With 64 filters, kernel size 3 and relu activation
4. **1D Convolutional Layer 2:** With 32 filters, kernel size 3 and relu activation
5. **1D Convolutional Layer 3:** With 16 filters, kernel size 3 and relu activation
6. **1D Global Max Pooling Layer:** Helps in extracting low level features from the convolutional layer
7. **Flattening Layer:** To transform the multidimensional matrix of features from previous layer into a vector that can be fed into a fully connected neural network classifier.
8. **Dropout Layer:** Nullifies contribution of 20% of the neurons towards the next layer and leaves the others unmodified.

9. **Dense Layer:** Creates a fully connected dense layer of 180 units with relu activation
10. **Dropout Layer:** Nullifies contribution of 20% of the neurons towards the next layer and leaves the others unmodified.
11. **Dense Layer:** Creates a fully connected dense layer of 2 units
12. **Activation Layer:** Outputs the final probabilities of the units to classify as either of the 2 classes. Sigmoid activation is used as it provides values between 0 and 1 and works well with categorical cross-entropy loss, which is used to train this model.

The model is compiled with *categorical cross-entropy loss* and *adam* optimiser. Categorical cross-entropy is a loss function commonly used for multi-class classification tasks [12]. Adam in an excellent algorithm which can handle sparse gradients on noisy problems [13].

More Information about Design Choice

- Using 3 consecutive convolutional layers enables an effective extraction of high-level features [14].
- Using dropout layers as a part of fully connected layer helps in reducing over-fitting as the 1st batch of training samples can influence the learning in a disproportionately high manner [14].

The model architecture is given below:

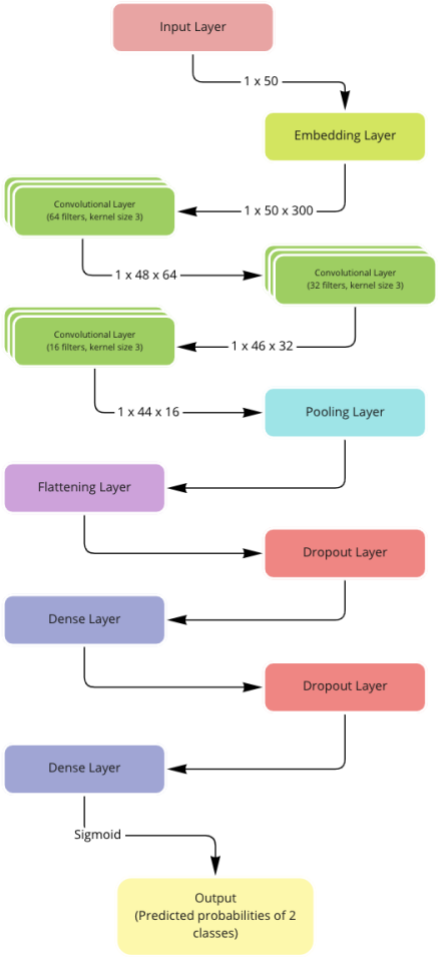


Figure 3: CNN Model Architecture

The model is then trained on the training set with validation set to test the accuracy after each epoch. The training is performed with batch size = 1024 and number of epochs = 10. A large batch size was taken because it allows larger learning rate and faster convergence. Furthermore, it significantly reduced the training time as training a model with data consisting of 800000 observations can be very time consuming.

Performance of the model is as follows:

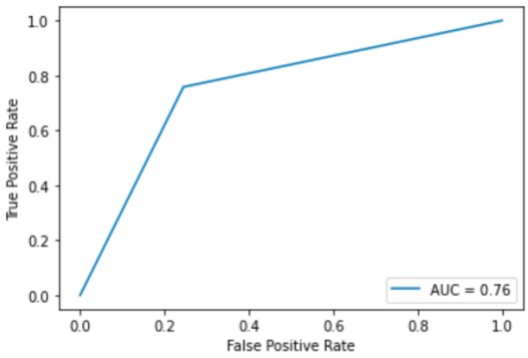


Figure 4: CNN ROC Curve

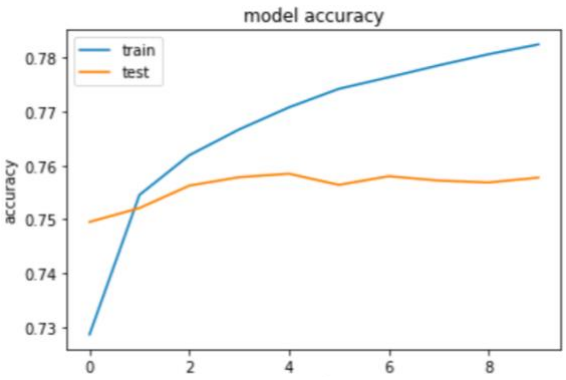


Figure 6: Model Accuracy v/s Epochs CNN

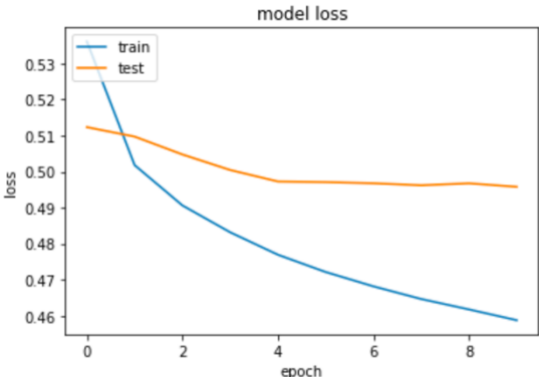


Figure 5: Model Loss v/s Epochs CNN

Looking at the ROC curve, we can see the curve is closer to the true positive rate. The area under the ROC curve is higher than 0.7 which is used to determine the usefulness of a test.

Looking at the model accuracy graph, From the 1st epoch to 2nd epoch, the accuracy of the model increased significantly for training and testing data set. For each subsequent epoch the accuracy was stable between 75% and 76% for validation dataset and increased for the training dataset which shoes that the model began to overfit according to the training data set.

Looking at the model loss graph, the loss on training set decreases significantly after each epoch. However, for the validation set, it almost remains around the 0.5 mark after the 2nd epoch which shows that the model is generalising well for the unseen data.

Lastly, the model accuracy is evaluated on the testing data set and turns out to be *75.665%* which aligns well with all the other accuracies on the validation datasets during training. This shows that the model provides consistent results.

NAÏVE BAYES V/S LOGISTIC REGRESSION

The first step before training Naïve Bayes and Logistic Regression models was to split the pre-processed data into a training and test split. Since we did not have the computing power to perform parameter tuning on the entire dataset with 1000 features, we created a training set of 100,000 tweets and a test set of 42,000 tweets (70-30 split). This split was performed in a stratified manner i.e.

The library we used to implement these two algorithms was scikit-learn (sklearn). Out of the various implementations of Naïve Bayes available on sklearn, we decided to go for Multinomial Naïve Bayes and that suited our requirements the most since according to their docs:

"The multinomial Naive Bayes classifier is suitable for classification with discrete features (e.g., word counts for text classification). The multinomial distribution normally requires integer feature counts. However, in practice, fractional counts such as tf-idf may also work."

The performance of the default classifiers without any hyperparameter tuning were as follows:

Table 1: Naïve Bayes v/s LogisticRegression (Default)

Classifier	Accuracy	Precision	Recall	F1 Score
Naïve Bayes	72.05 %	72.81 %	70.51 %	71.64 %
LogisticRegression	73.28 %	71.65 %	77.18 %	74.31 %

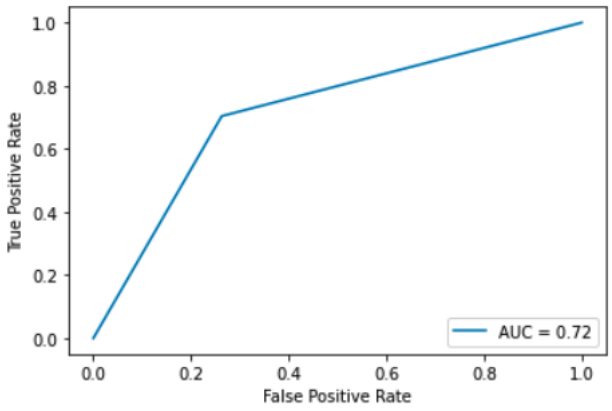


Figure 8: Naive Bayes ROC Curve

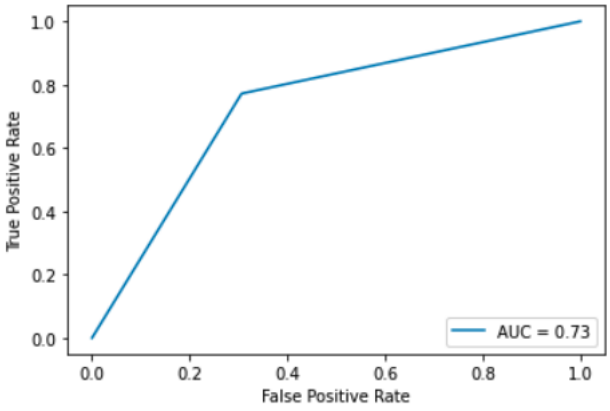


Figure 7: Logistic Regression ROC Curve

Next, we performed hyperparameter tuning using GridSearchCV to find the most ideal parameter values for both these classifiers. The following were the best parameters:

Naïve Bayes
alpha = 20 (GridSearchCV), alpha = 3 (Manual Tuning)

Logistic Regression
C = 1.0, penalty = l1, solver = saga

The performance of the classifiers after hyperparameter tuning (using the best parameters) were as follows:

Table 2: Naïve Bayes v/s LogisticRegression (Hyperparameter Tuned)

Classifier	Accuracy	Precision	Recall	F1 Score
Naïve Bayes (GridSearch)	72.04 %	72.86 %	70.38 %	71.6 %
Naïve Bayes (Manual Tuning)	72.08 %	72.85 %	70.55 %	71.69 %
LogisticRegression	73.34 %	71.58 %	77.55 %	74.45 %

DISCUSSION

Note: The green colour in Table 2 signifies the superior performance of one classifier over the other for that metric. Example, the recall for Logistic Regression is higher than Naïve Bayes.

The performances of the two classifiers are similar because both Naïve Bayes and Logistic Regression are supervised, binary and probabilistic classifiers. The default classifiers performed as per our initial expectations in terms of performance. While the performance of Logistic Regression is better than Naïve Bayes (Table 1 and 2), Naïve Bayes performs significantly better when it comes to model training times. While Naïve Bayes takes 0.2 seconds to fit the data, Logistic Regression takes around 8 seconds. This is because Naïve Bayes is a much scalable and simpler algorithm to implement making it significantly faster.

Note: The explanation for the lower performance for Naïve Bayes can be linked to the “naïve” assumption discussed in the methodology section.

As seen in Table 1 and 2, GridSearchCV is reducing the performance of our model compared to default. This is because, GridSearchCV only considers the training set and performs cross validation on it to find the best model. GridSearchCV is the right approach considering that it doesn't take the test data into account at all so it gives the best parameters (based on training data) that would perform the best on a blind dataset. In most cases this approach gives us a model that performs better on the test dataset as well but that wasn’t the case here. However, since we wanted to find the model that would perform the best on the test set, we wrote a simple manual script that gave a different set of parameters. As expected, it performed better on the test set.

Bringing CNN into the picture, it performed the best with a significant margin. This can be because of its translation invariant nature and its effectiveness in feature extraction. In terms of computational time, CNN model takes a significantly larger time to train due to its complex nature and tuning of weights by inputting the data multiple times in the network. Due to this reason, parameter tuning using cross validation could not be performed on the CNN model.

CONCLUSION AND FUTURE WORK

By analysing the performance of the 3 models for sentiment analysis we can see that Naïve Bayes and Logistic Regression models are similar in terms of accuracy, but Logistic Regression model performs much better when it comes to Recall score and model training time. The computational time is significantly low for these models compared to CNN. Both can be said to perform well in terms of accuracy, given the low computational time. Convolutional Neural Network on the other hand is relatively complex in nature and provides the best accuracy. However, due to its complex nature there is a downside of high computational time when training it. Therefore, any of the 3 models can be used depending on their use case. For example, if quick predictions are required by an organisation, then Naïve Bayes/ Logistic Regression models are the most optimal choice. However, if more focus is given on the accuracy, for instance government seeking public's opinion on new policies, then CNN will be a better choice.

Sentiment analysis is still at its early stages as it is only used for binary classification. In future, we can delve deeper into the topic by going beyond the concept of positive/negative and have a much greater and comprehensive understanding of users' feelings from the tweet. Due to this reason, sentiment analysis is becoming more important for many organisations as the data underlying the interactions between them and their customers is growing rapidly and becoming more complex. There are other datasets which provide a range of other emotions. One such dataset is SMILE with over 3000 tweets and emotions including happiness, anger, outrage, sadness and more [15]. However due to such a small size of the dataset, it is not possible to build an effective model as sentiment analysis becomes more complicated as more emotions are added into the picture.

Furthermore, more complex models can be created to achieve better performance for analysing users' sentiments for the *Sentiment140* especially when it comes to neural networks. There is high scope of improvement by adding attention layers, bidirectional LSTM layers, running multiple layers in parallel etc. An effective combination of such layers can be highly beneficial for enhancing the model performance.

REFERENCES

1. COMP5318 Lecture Slides

2. K. Gligorić, A. Anderson and R. West, "Adoption of Twitter's New Length Limit: Is 280 the New 140?", *arXiv.org*, 2022. Available: https://arxiv.org/abs/2009.0766

3. E. International Journal of Scientific Research in Computer Science, "A Literature Review on Sentiment Analysis", *Academia.edu*, 2022. Available: https://www.academia.edu/44778278/A_Literature_Review_on_Sentiment_Analysis.

4. Ortner, Alex. "Top 10 Binary Classification Algorithms [A Beginner's Guide]." *Medium*, Thinkport Technology Blog, 28 May. 2020, https://medium.com/thinkport/top-10-binary-classification-algorithms-a-beginners-guide-feeacbd7a3e2.

5. A. Goel, J. Gautam and S. Kumar, "Real time sentiment analysis of tweets using Naive Bayes," 2016 2nd International Conference on Next Generation Computing Technologies (NGCT), 2016, pp. 257-261, doi: 10.1109/NGCT.2016.7877424.

6. Behera, Rabi & Dash, Sujata & Roy, Manan. (2015). “A Novel Machine Learning Approach for Classification of Emotion and Polarity in Sentiment140 Dataset”.

7. “12 Twitter Sentiment Analysis Algorithms Compared.” *AI Perspectives*, 15 June 2021, https://www.aiperspectives.com/twitter-sentiment-analysis/.

8. "Research on Text Classification Based on CNN and LSTM", *Ieeexplore.ieee.org*, 2022. Available: https://ieeexplore.ieee.org/document/8873454

9. Stecanella, Bruno. “Understanding TF-ID: A Simple Introduction.” MonkeyLearn Blog, 11 May 2019, https://monkeylearn.com/blog/what-is-tf-idf/.

10. “Confusion Matrix.” *Packt Subscription*, https://subscription.packtpub.com/confusion_matrix

11. "Sentiment Analysis and Opinion Mining | Synthesis Lectures on Human Language Technologies", *Morganclaypool.com*, 2022. Available: https://www.morganclaypool.com/doi/abs/10.2200/S00416ED1V01Y201204HLT016.

12. Pennington, "GloVe: Global Vectors for Word Representation", *Nlp.stanford.edu*, 2022. Available: https://nlp.stanford.edu/projects/glove/.

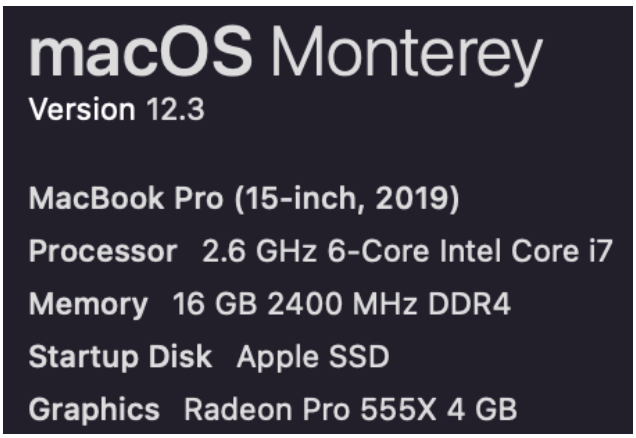
13. "Understanding Categorical Cross-Entropy Loss, Binary Cross-Entropy Loss, Softmax Loss, Logistic Loss, Focal Loss and all those confusing names", *Gombru.github.io*, 2022. Available: https://gombru.github.io/2018/05/23/cross_entropy_loss/.

14. Brownlee, "Gentle Introduction to the Adam Optimization Algorithm for Deep Learning", *Machine Learning Mastery*, 2022. Available: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/.

15."SMILE Twitter Emotion dataset", *figshare*, 2022. Available: https://figshare.com/articles/dataset/smile_annotations_final_csv/3187909.

APPENDIX

SYSTEM SPECIFICATIONS



LIBRARY REQUIREMENTS

Make sure all the libraries that have been mentioned are installed on your system before running the code. These can be installed by using:

pip install <library-name>

If you are using Anaconda (recommended) then the commands to install the libraries will vary. These would have to be found on https://anaconda.org/

Refer to final page for list.

INSTRUCTIONS TO RUN CODE

1. Download the Sentiment140 Dataset: https://www.kaggle.com/datasets/kazanova/sentiment140
2. Save the csv while as *"tweets.csv"* in the same directory as the Jupyter Notebooks
3. Download the Stanford Global Vectors for Word Representation. Save the 'glove.6B.300d.txt' file in the same directory as the Jupyter Notebooks
4. Click *Run All* on the notebook title "group144_data_cleaning" as that goes through the csv file and performs the cleaning and pre-processing of data
5. The "group144_other_algorithms" notebook contains the implementation of Naïve Bayes and Logistic Regression. The "group144_best_algorithm" notebook contains the implementation of CNN. Click *Run All* on the notebook you wish to run.

Note: We performed parameter tuning using GridSearchCV which means the code could take several hours to run depending on the system

Name	Version	Build Channel
appnope	0.1.2	py37hecd8cb5_1001
backcall	0.2.0	pyhd3eb1b0_0
blas	1.0	mkl
bottleneck	1.3.4	py37h67323c0_0
brotlipy	0.7.0	py37h9ed2024_1003
ca-certificates	2022.4.26	hecd8cb5_0
catalogue	2.0.7	py37hecd8cb5_0
certifi	2021.10.8	py37hecd8cb5_2
cffi	1.15.0	py37hc55c11b_1
charset-normalizer	2.0.4	pyhd3eb1b0_0
clean-text	0.3.0	pyhd8ed1ab_0 conda-forge
click	8.0.4	py37hecd8cb5_0
colorama	0.4.4	pyhd3eb1b0_0
cryptography	37.0.1	py37hf6deb26_0
cycler	0.11.0	pyhd3eb1b0_0
cymem	2.0.6	py37he9d5cce_0
cython	0.29.28	py37he9d5cce_0
cython-blis	0.7.7	py37h67323c0_0
debugpy	1.5.1	py37he9d5cce_0
decorator	5.1.1	pyhd3eb1b0_0
emoji	1.7.0	pyhd8ed1ab_0 conda-forge
en-core-web-sm	3.3.0	pypi_0 pypi
entrypoints	0.4	py37hecd8cb5_0
freetype	2.11.0	hd8bbffd_0
ftfy	5.8	py_0
gensim	4.1.2	py37he9d5cce_0 anaconda
idna	3.3	pyhd3eb1b0_0
importlib-metadata	4.11.3	py37hecd8cb5_0
intel-openmp	2021.4.0	hecd8cb5_3538
ipykernel	6.9.1	py37hecd8cb5_0
ipython	7.31.1	py37hecd8cb5_0
jedi	0.18.1	py37hecd8cb5_1
jinja2	3.0.3	pyhd3eb1b0_0
joblib	1.1.0	pyhd3eb1b0_0 anaconda
jupyter_client	7.2.2	py37hecd8cb5_0
jupyter_core	4.10.0	py37hecd8cb5_0
kiwisolver	1.3.2	py37he9d5cce_0
langcodes	3.3.0	pyhd3eb1b0_0
libcxx	12.0.0	h2f01273_0
libffi	3.3	hb1e8313_2
libgfortran	3.0.1	h93005f0_2
libpng	1.6.37	ha441bb4_0
libsodium	1.0.18	h1de35cc_0
llvm-openmp	12.0.0	h0dcd299_1
markupsafe	2.0.1	py37h9ed2024_0
matplotlib	3.2.2	1 conda-forge
matplotlib-base	3.2.2	py37h5670ca0_0
matplotlib-inline	0.1.2	pyhd3eb1b0_2
mkl	2021.4.0	hecd8cb5_637
mkl-service	2.4.0	py37h9ed2024_0
mkl_fft	1.3.1	py37h4ab4a9b_0
mkl_random	1.2.2	py37hb2f4e1b_0
murmurhash	1.0.7	py37he9d5cce_0
ncurses	6.3	hca72f7f_2
nest-asyncio	1.5.5	py37hecd8cb5_0
numexpr	2.8.1	py37h2e5f0a9_0
numpy	1.21.5	py37h2e5f0a9_2
numpy-base	1.21.5	py37h3b1a694_2
openssl	1.1.1o	hca72f7f_0
packaging	21.3	pyhd3eb1b0_0
pandas	1.3.5	py37h743cdd8_0
parso	0.8.3	pyhd3eb1b0_0
pathy	0.6.1	py37hecd8cb5_0
pexpect	4.8.0	pyhd3eb1b0_3
pickleshare	0.7.5	pyhd3eb1b0_1003
pip	21.2.2	py37hecd8cb5_0