Abstract

This study presents a comprehensive exploration of sentiment analysis on Twitter data using machine learning techniques. The primary objective was to build effective models for sentiment prediction that could handle the nuances and complexities of human language expressed in tweets. After the extensive preprocessing, three machine learning models are established including Logistic Regression, Support Vector Machines (SVM) and Long Short-Term Memory (LSTM). Each of them underwent a hyperparameter tuning process to optimise their performance. The models are then evaluated using accuracy and confusion matrix. The results showed that all three classifiers performed similarly in terms of accuracy, with scores 0.75, 0.76 and 0.74 respectively.

This study contributes to ongoing research in sentiment analysis, especially in the context of social media, by providing insights into the performance of different machine learning techniques on a large, balanced dataset. It also sheds light on the trade-offs between computational complexity and accuracy in sentiment analysis.

Introduction

Sentiment analysis is the computational study of people's opinions, sentiments, evaluations, and emotions expressed in written language. It has become an increasingly important task in the era of social media and vast amounts of text data available. Real-world applications of sentiment analysis span various industries, including market analysis, healthcare or political analysis. It has been used to understand and analyse customer reviews (Gräbner et al.), identify potential adverse drug reactions (Korkontzelos et al.) and examine how students perceived the learning experience during the COVID pandemic (Mostafa).

Twitter is a social media platform, where users can exchange messages ("New user FAQ"). A tweet contains up to 280 characters ("Counting characters | Docs"), and is posted by users to share opinions, world-views, feelings and every-day commentary. The platform has millions of users worldwide and offers a rich source of data for such analysis. This report presents an in-depth exploration of Twitter sentiment analysis using a balanced dataset obtained from Kaggle (KAZANOVA).

The dataset comprises 300,000 tweets which are evenly balanced between positive and negative sentiments. Each tweet in the dataset contains six attributes: sentiment (labelled as '0' for negative and '4' for positive), tweet ID, date, query, username, and the tweet text itself. The aim of the report is to construct and analyse classification models for the sentiment of the tweets, using the text feature in the dataset.

Previous approaches to classify the sentiment of the data have utilised Support Vector Classifier, Naive Bayes and Maximum Entropy ("Twitter sentiment classification using distant supervision"), as well as convolutional Neural Networks (CNN) and Long Short Term Memory (Gandhi et al.).

Informed by these previous works and considering the computational constraints of a laptop environment, this report employs three well-established machine learning algorithms for the sentiment prediction task: Support Vector Machines (SVM), Logistic Regression, and Long Short-Term Memory (LSTM). These methods were chosen due to their prevalence in similar studies, their suitability for this task, and their feasibility given the computational resources. Each algorithm underwent a hyperparameter tuning process, with a grid search implemented for SVM and Logistic Regression, and manual tuning performed for LSTM.

The performance of all 3 models was similar, with accuracy 0.75 for Logistic Regression, 0.76 for SVM and 0.74 for LSTM.

Methodology

Preprocessing

To prepare the raw data for analysis, a comprehensive preprocessing pipeline was applied. Firstly, to assist with the subsequent model building process, the sentiment attribute was transformed from ‘0’ and ‘4’ to '0' and '1', representing negative and positive sentiment.

We utilised regular expressions to clean the formatting of the text. Usernames, URLs, and hashtags were removed, as they typically have no meaningful sentiment information. The tweet text was further cleaned by removing punctuation, special characters, digits, and words containing non-ASCII characters which simplifies the data and also increases the analysis speed of the models.

The formatting of the tweets includes a lot of typos or intentional misspelling of the words, for example usage of ‘wooooork’ instead of ‘work’ for emphasis of the message. As English does not have words that have 3 letters or more in a row, we normalised the text by limiting those repeated characters to 2. We formatted text with repeated segments (eg. ‘haha’ and ‘xox’ into ‘haha’ and ‘xo’) to help standardise the text and reduce the number of tokens for the same expressions, thus improving the model’s learning efficiency.

Subsequently, the tweet text was tokenized, breaking it down into individual words (“Tokenization”). This was followed by the removal of commonly used words which are also known as stopwords (e.g., "what", "is", "that"). Stop words generally do not contain significant sentiment and are used throughout the text. This step further reduces the data dimensionality, allowing the model to focus on words that likely carry more sentiment information.

Finally, words were reduced to their base or lemma form through the process of lemmatization. For example, the word "running" would be transformed into "run". We chose lemmatization over stemming as it produces linguistically valid word bases and produces better precision (Balakrishnan and Lloyd-Yemoh).

Stemming is a faster approach, however it simply chops off the ends of words based on common prefixes or suffixes, which can result in non-meaningful words. While stemming may be preferred when speed optimization is crucial, it may compromise the quality of the analysis (Pramana et al.). To achieve a more accurate analysis, lemmatization is chosen over stemming as the traits of taking into consideration the morphological analysis of words which is a more suitable choice for our analysis.

We transformed the data into a matrix using the Term Frequency Inverse Document Frequency algorithm (TF-IDF). The vectorizer gives the term frequency adjusted by the inverse document frequency, measuring the importance of the word.

We split up the data into a training and testing set, with a 75:25 ratio. In order to be able to perform the hyperparameter tuning, we obtained a subset of the training data (1%). We utilised that to run the grid search algorithm.

Models

Logistic Regression

Logistic regression is a supervised classification model used to predict the binary sentiment class (positive or negative) based on text of the tweet. One of the advantages of this classifier is that it's easy to interpret and efficient to train, which makes it well suited for the sentiment analysis task with a large dataset. The classifier uses the sigmoidal curve to obtain the probability for class membership for class 1 (positive), and subtracts it from 1 in order to obtain the probability to class 0 (negative). The function for the regression can be defined as $p(z) = \frac{1}{1 + e^{(-z)}}$, where $z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n$, with β_n being the coefficient for feature x_n. It utilises the maximum likelihood method to estimate the parameters for the coefficients that best fit the data, derived from assumed probability distribution of the dependent variable (Czepiel 3).

With the high dimensional vectors as input space, the result can be overfit to the data (Travers), be too specific to the examples it was trained on and not good for generalisation of the trends for the unseen cases. In order to prevent that there are regularisation techniques, including L1 and L2 regularisation. They focus on keeping low respectively the absolute values of regression weights $|\beta|$ or total squared value of the regression weights β^2 (Travers).

Support Vector Machine

Support Vector Machine is another model we utilised for the sentiment classification problem. It computes the decision boundary between the classes as the maximum margin hyperplane for the data transformed to a high-dimensional feature space; an optimal hyperplane to maximise the distance between itself and the closest examples from the different classes.

The model was introduced in 1995 (Cortes and Vapnik), where its high generalisation ability was shown. The classifier is well suited for sentiment classification, as it possesses the ability to be trained on a large number of features (Phienthrakul et al.).

It uses a kernel function to map the dot product of the original features into the higher dimensions and avoid having to do the computations in the new dimensional space. The performance of the classifier relies on the kernel function used (Phienthrakul et al.). The Radial Basis Function (*RBF*) kernel is used as a default one in the SVC model creation. It computes the similarity of the closeness of two points (Sreenivasa), with the equation for the kernel function: $K(x_i, x_j) = e^{-\gamma\left|x_i^T x_j\right|^2}, \gamma > 0$. The Linear kernel has function $K(x_i, x_j) = x_i^T x_j$, polynomial kernel $K(x_i, x_j) = (x_i^T x_j + coef\_0)^d$, Sigmoid $K(x_i, x_j) = tanh(\gamma . x_i^T x_j + coef\_0)^d$ (Wijaya and Karyawati 165).

We will therefore try out the different combinations of a kernel function with values for *gamma* and *C*, corresponding to the influence of a single training example and maximising function’s margin (“RBF SVM parameters”). It is important to note that with the higher the value of *C,* the bigger the tradeoff in the classification of the training samples.

LSTM

For the task of sentiment analysis, one of the chosen methods is Long Short-Term Memory (LSTM), a type of recurrent neural network (RNN) specifically designed for learning and retaining information over long sequences, making it well-suited for text data like tweets.

RNNs are designed to identify patterns in sequential data, such as text, genomes, handwriting, or time series (Debasish Kalita). The model consists of three layers: input, hidden and output as shown in Fig. 1. In an RNN, the input layer receives input sequences over discrete time steps, in our case tokens from a sentence. The hidden layers in an RNN have self-connections, which means they take not only the output from the previous layer but also their own output from the previous time step as input. The self-connections in the hidden layers are associated with weights, which are adjusted during training. Activation functions, such as the hyperbolic tangent function (tanh: $g(z) = (e^z - e^{-z})/(e^z + e^{-z})$) or rectified linear unit (ReLU: $g(z) = max(0, z)$) (Afshine Amidi and Shervine Amidi), are applied to the hidden layers to determine neuron activation. The hidden state, influenced by the input, previous state, and bias, defines the new hidden state.

The output layer of an RNN generates the final output using a sigmoid function. In the context of sentiment analysis, the output is a single value at the end of the sequence, representing the sentiment classification.

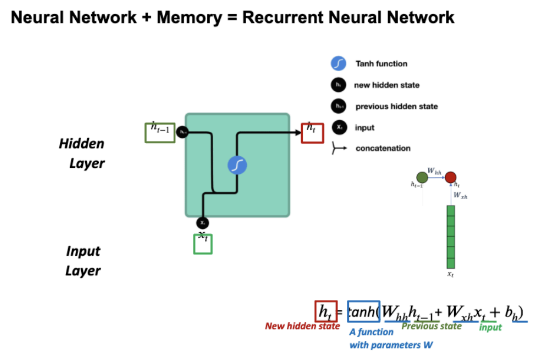


Figure. 1.: RNN network structure (*Jonathan*).

RNNs excel in processing sequential data, making them suitable for tasks involving time series analysis, natural language processing, and other sequence-based tasks. However, RNN models often struggle to capture long-range dependencies due to the vanishing/exploding gradient problem (Nir Arbel). It is when the parameter updates carried through backpropagation become very small or large (Nir Arbel).

To address these limitations, LSTM models were introduced by Hochreiter and Schmidhuber in 1997 (Shipra Saxena).The vanishing gradient problem impedes the learning of long-range dependencies in RNNs. The LSTM algorithm overcomes this by incorporating a new component into the network architecture: gates. Gates act as filters to selectively allow or prevent information flow. They consist of a sigmoid neural network layer and a pointwise multiplication operation. An LSTM cell comprises three types of gates: the input gate, forget gate, and output gate.
As shown in Fig. 2, the input gate receives the previous hidden state and the current input, passing them through a sigmoid function to control the amount of new information added to the cell state. Simultaneously, the same inputs go through a tanh function to generate new candidate values, which are scaled between -1 and 1 to regulate the network. The output of the tanh function is then multiplied pointwise by the output of the sigmoid function, which determines the relevant information to retain.

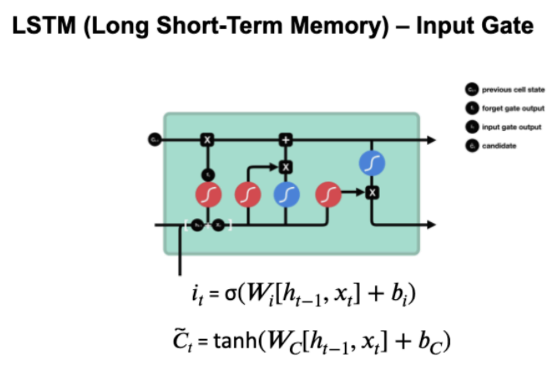

LSTM (Long Short-Term Memory) – Input Gate
previous cell state
forget gate output
input gate output
candidate
$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$
$\tilde{C}_t = \tanh(W_C[h_{t-1}, x_t] + b_C)$

Figure. 2.: LSTM input gate structure and formula (*Jonathan*)

The cell state undergoes transformation, being element-wise multiplied by the forget vector that decides the proportion of past information to retain. As shown in Fig. 3, the output from the input gate is then element-wise added to update the cell state with new values considered relevant by the neural network, resulting in the new cell state.

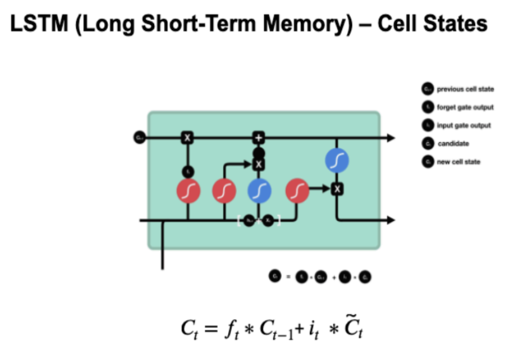
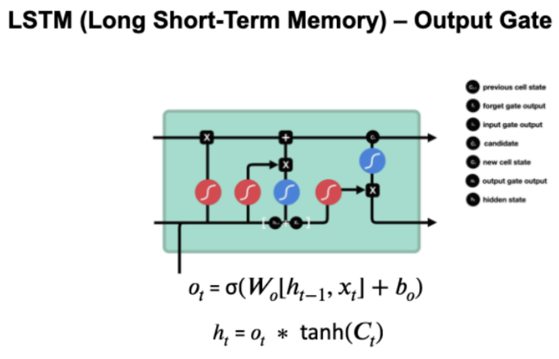

LSTM (Long Short-Term Memory) – Cell States
previous cell state
forget gate output
input gate output
candidate
new cell state
$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$

Figure. 3.: LSTM cell gate structure and formula (*Jonathan*)



In the output gate, as shown in Fig. 4,It takes the previous hidden state and the current input, passing them through a sigmoid function. Simultaneously, the updated cell state passes through a tanh function. The output of the sigmoid function is element-wise multiplied with the output of the tanh function, determining the information to be carried by the hidden state. This computed value becomes the next hidden state for the subsequent time step.

Figure. 4.: LSTM output gate structure and formula (*Jonathan*)

Implementation

Both logistic regression and SVM were adjusted with tuned parameters. For the hyperparameter tuning process, we utilised the *GridSearchCV()* method from the scikit-learn library, fitting models with all the possible combinations of parameters from the parameter grid and obtaining the set for the best performing model in terms of the mean accuracy across a 5-fold cross validation.

For the visualisation of the results of tuning of several hyperparameters, we used a plotting function *plot_search_results(grid)* ("python - How to graph grid scores from GridSearchCV?"). It plots the result of changes of one parameter with the rest of the parameters to their value found for the best performing model. This enables us to get the visualisations for how the different parameters influence the score of the model.

Logistic Regression

For the implementation of the model, we utilised scikit-learn linear model Logistic Regression library. We fit it on the training data (vectorised text, result of the preprocessing steps), and scored against the testing set.

Then, we tuned the parameters of the model with the grid search method on the whole training set. The solvers we tried out for the model (algorithm for the optimisation problem) are *A Library for Large Linear Classification* ('liblinear') and *Stochastic Average Gradient* ('sag'). Both of them are designed to handle large-scale problems. For the parameter grid, we specified the penalties for the solver, with both the 'l1' and 'l2' methods for 'liblinear' and only 'l2' regularisation for 'sag'.

For each one of the solvers, we run the grid search with the appropriate list of penalties, and different values of *C* (from 100 to 0.01) for the strength of the regularisation.

Support Vector Machine

We followed a similar approach with developing the SVM classifier.

The time to fit a single model with the whole training data set took about ~200 minutes, It would not be feasible to fit 320 fits (5 folds for 64 candidates). We therefore decided to perform the parameter tuning on a smaller subset of the training data. The training subset for the hyperparameter tuning consists of 1% of the original training data, and was stratified with respect to the sentiment value, ensuring the same ratio of the sentiments for the two sets.

The grid search was performed for the SVM classifier using a parameter grid consisting of different values for 'C', 'gamma', and 'kernel'. The values considered for 'C' were [0.1, 1, 10, 100], 'gamma' had [1, 0.1, 0.01, 0.001], and 'kernel' had ['rbf', 'poly', 'sigmoid', 'linear'] as options. The GridSearchCV function was utilised with refit=True to identify the best combination of hyperparameters. The process was conducted with verbosity level set to 2, allowing for detailed output, and parallelized with n_jobs=-1 to leverage all available CPU cores. After fitting the grid search to the training data, the best score achieved and the corresponding best parameters were printed.

We plotted the results of the grid search, one parameter against the set of other best parameters found for the model. We then set up the model with the obtained tuned parameters, noting its training time, mean accuracy and f1 score. We plotted a normalised confusion matrix for the evaluation of the models performance on the

LSTM

Before training, the data underwent padding to ensure a uniform size of input for the LSTM model. This step is necessary as LSTM requires input sequences of the same length for efficient matrix operations. Padding involves appending zero values to the end of each sequence until they reach a predetermined length of 30. Additionally, text values and sentiment labels were encoded into numerical form to facilitate computation. The chosen encoding approach for text values is word embedding, specifically using the Global Vectors for Word Representation (GloVe) method developed by Stanford. GloVe generates a vector for each word, capturing the semantic information within it. This vector representation places semantically similar words closer to each other in the vector space, enhancing the model's ability to understand word relationships. The code for encoding, embedding, and the layers of the LSTM model is sourced from Kaggle's "NLP Beginner - Text Classification using LSTM" project.

The implementation of the LSTM model is contained in a single script, allowing for a seamless flow from data preprocessing to model definition, training, and evaluation. Key Python libraries used in the implementation include TensorFlow and Keras. TensorFlow serves as the foundation for building, training, and evaluating the deep learning model, while Keras provides user-friendly, high-level building blocks such as layers and activation functions. The codebase exhibits modularity, with core functionalities encapsulated in separate functions. For instance, the create_model() function generates an instance of the LSTM model, enhancing code readability and reusability. It takes parameters such as learning rate, batch size, and dropout rate, and returns the trained model along with accuracy and running time metrics.

Several critical hyperparameters influence the performance of the model. The number of layers determines the depth of the network, with deeper networks capable of capturing intricate patterns. However, deeper networks may be prone to overfitting and are more computationally intensive (Karsten Eckhardt). The model in this report employs various layers: an Embedding layer, a Spatial Dropout layer, a Conv1D layer, a Bidirectional LSTM layer, two Dense layers, and an output layer.

The number of units in each layer specifies the dimensionality of that layer's output space or the complexity of features the layer can learn. More units enable the layer to learn complex representations but increase computational costs. In this model, the Conv1D layer has 64 filters, the Bidirectional LSTM layer contains 64 units, and the two Dense layers have 512 units each. Activation functions such as the Rectified Linear Unit (ReLU) are employed in the Conv1D and Dense layers, while the 'sigmoid' function is used in the output layer. The embedding dimension, which remains constant at 300 in this model, represents the size of the vector space where words are embedded, defining the output vectors' size from this layer for each word. The number of epochs set to 10 means the model iterates over the entire training dataset 10 times. While more epochs could enhance performance by offering the model more learning opportunities which might also lead to overfitting. Conversely, training for too few epochs may not provide sufficient learning which leads to underfitting, where the model fails to capture the data patterns effectively (Kuldeep Chowdhury).

A systematic approach to hyperparameter tuning has been adopted, involving iterations over different combinations of learning rates, batch sizes, and dropout rates. This process determines the optimal configuration for the model from the available parameter combinations, enhancing its performance.

The learning rate, chosen from [0.001, 0.01] in this LSTM model, plays a crucial role as it determines the speed at which weight updates occur during training. A learning rate that is too high may overshoot the optimal solution, while a very low rate may require more epochs to converge (Jason Brownlee).

The batch size is chosen from [512, 1024] where it defines the number of samples processed before updating the internal model parameters. A smaller batch size results in more frequent weight updates, potentially leading to faster but less stable learning. Larger batch sizes entail less frequent updates, potentially providing more stable learning but at a slower pace.

Lastly, the dropout rate where chosen from [0.2, 0.3], indicates the fraction of neurons dropped during training. Dropout is a regularisation technique that prevents overfitting by ignoring randomly selected neurons during training (Kuldeep Chowdhury).

Experimental Results

Experimental setting

Our sentiment analysis project was implemented on a high-performance personal computer to ensure efficient data processing and modelling. The computer is equipped with an 11th Generation Intel Core i7-11700F processor, running at a base frequency of 2.50 GHz. This advanced processor enables it to train and tune complex models such as the LSTM. The PC has 16 GB of installed RAM, out of which 15.8 GB is usable. This large memory capacity is essential for handling large datasets and ensures data preprocessing and model training simultaneously.

The code was executed on Jupyter Notebook with a python version of 3.9.7. Critical libraries for the project included TensorFlow (version 2.12.0), NLTK (version 3.6.5), Keras (version 2.12.0) and Pandas (version 1.3.4). All the software was run on a Windows 11 operating system.

Results

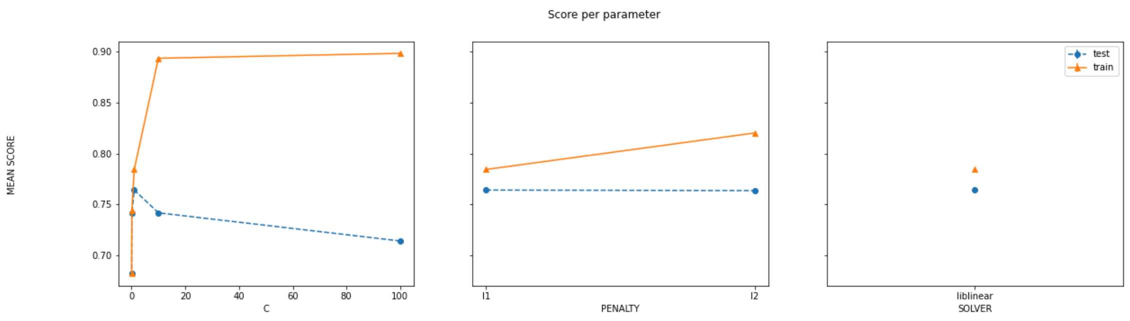
The performance of the models was evaluated using accuracy, precision, recall, confusion matrix, and runtime metrics. The results show each model has quite similar accuracy however their computation time varied.

Logistic Regression

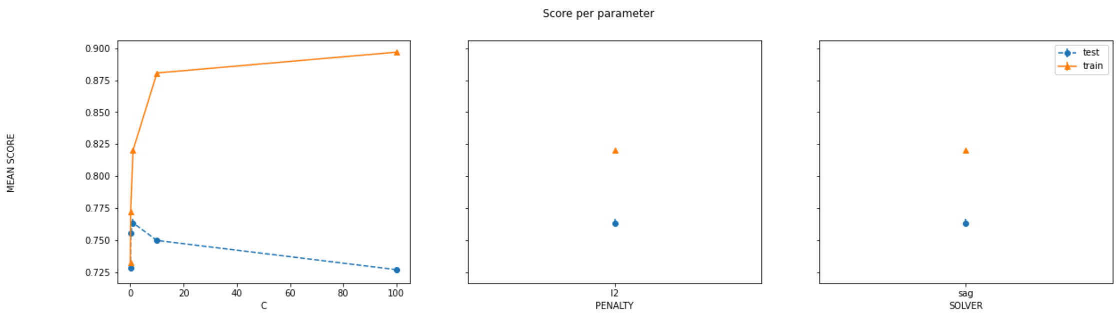
The logistic regression model was tuned with the help of the grid search method, which was conducted separately for the two solvers, 'liblinear' and 'sag' using different penalty options, both with the same set of possible values for the 'C' parameter.

The model is assessed based on the f1 score metric, which takes into account both precision and recall, ensuring a balance between false positives and false negatives. The final parameters are taken for the best performing model in terms of the testing set (created in the cross validation). Combined with regularisation methods, it lowers the chances of the model overfitting its parameters to the training data.

As illustrated in the figure below, the model performance differs depending on the used parameters. In both cases, its score is the highest with *'C = 1.0'*. Even Though the training set score with the Ridge Regression ('l2' penalty) is higher than for the Lasso Regression ('l1' penalty), its score for the testing set is slightly lower (see Fig.5.(a)). This means that in order for our model to be better at generalisation and prevent overfitting, the 'l1' penalty is a better solution.



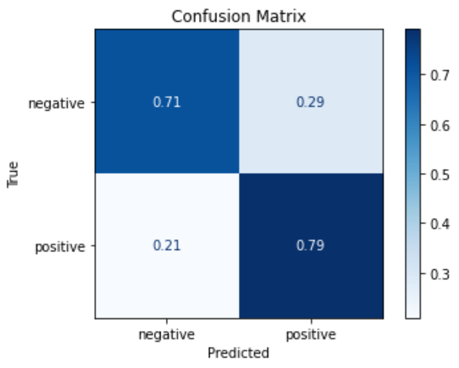
(a)

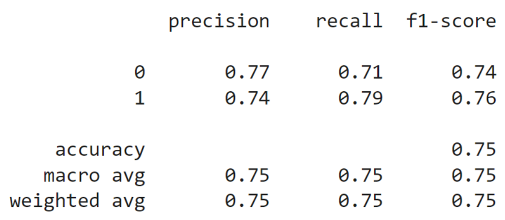


(b)

Fig. 5. Plot of grid search results for the Logistic Regression with corresponding accuracy. Code for visualisation obtained from an online source (“python - How to graph grid scores from GridSearchCV?”). (a) Visualisation of grid search for ‘liblinear’ solver. (b) Visualisation of grid search for ‘sag’ solver.

The best parameters found for the logistic regression are {'solver': 'liblinear', 'penalty': 'l1', 'C': 1.0}. The training time for the model is 0.64 seconds, with the final accuracy of 0.75292. The model has a reasonably balanced performance, with similar precision, recall and F1-scores for both classes (see Fig. 6 below). The correctly classified samples form the majority for both of the classes, with 71% for the negative class and 79% for the positive. The misclassified samples constitute for under a third of the class, for both negative and positive sentiment.





```
              precision    recall  f1-score

           0       0.77      0.71      0.74
           1       0.74      0.79      0.76

    accuracy                           0.75
   macro avg       0.75      0.75      0.75
weighted avg       0.75      0.75      0.75
```

Fig. 6. Performance of the Logistic Regression model on the testing data. On the left: Normalised Confusion Matrix for Logistic regression with tuned hyperparameters. On the right: part of the classification report, class: 0 for ‘negative’ sentiment, 1 for ‘positive’ sentiment.

SVM

The result of the grid search for the SVM with the subset of the training data, shows that the accuracy of the model for the testing set is the highest for *'C = 1.0'* (see Fig. 7.). It was also observed that as the value of 'gamma' approaches zero, the accuracy tends to decrease. As 'gamma' increases, the accuracy for the training set experiences a more significant improvement compared to the testing set. This suggests that a higher 'gamma' value might lead to overfitting, as the model becomes overly sensitive to the training data.

With the different kernels providing different ways of mapping the input data to higher-dimensional spaces, they have a significant impact on the accuracy of the SVM classifier. The sigmoid function achieves the highest accuracy for the testing data, simultaneously with the lowest training accuracy. However, all of the different kernels have a significant discrepancy between the training and testing accuracy, suggesting that the model fits the data too closely and has reduced generalisation capabilities.

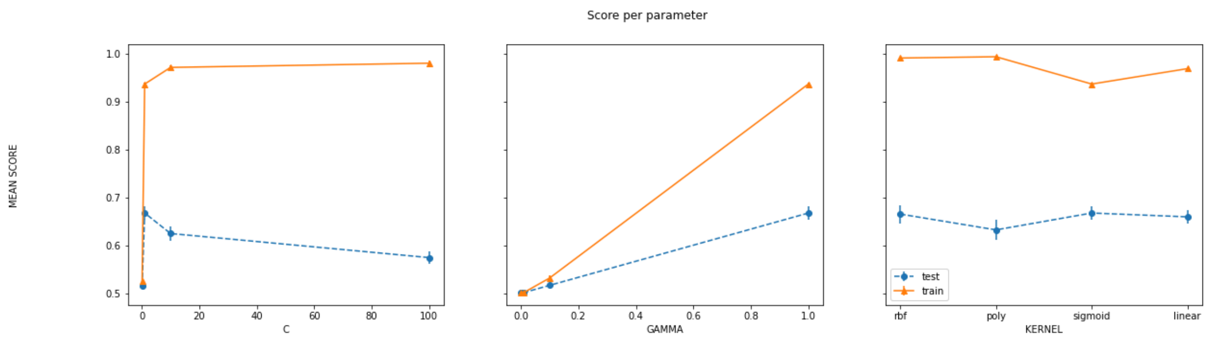
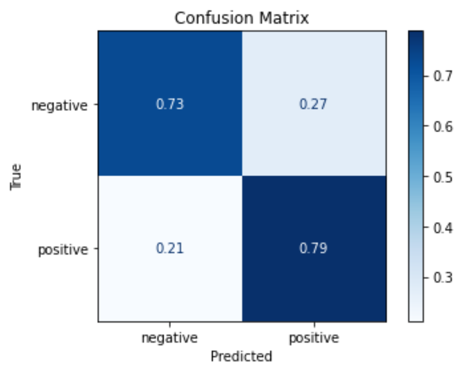


Fig. 7. Plot of grid search results for the Support Vector Machine with corresponding accuracy. Code for visualisation obtained from an online source ("python - How to graph grid scores from GridSearchCV?").

The best parameters found for the SVM are {'C': 1, 'gamma': 1, 'kernel': 'sigmoid'}, with the training time around 148 minutes and final accuracy 0.75783. The model achieved an overall accuracy of 0.76, which indicates that it correctly predicted 76% of the instances. The precision, recall, and F1-scores for both classes (positive and negative sentiment) are reasonably balanced, with values ranging from 0.73 to 0.79 (illustrated in Fig. 8. below).



```
              precision    recall  f1-score

           0       0.77      0.73      0.75
           1       0.74      0.79      0.77

    accuracy                           0.76
   macro avg       0.76      0.76      0.76
weighted avg       0.76      0.76      0.76
```

Fig. 8. Performance of the tuned Support Vector Classifier model on the testing data. On the left: Normalised Confusion Matrix for the SVC with tuned hyperparameters. On the right: part of the classification report, class: 0 for 'negative' sentiment, 1 for 'positive' sentiment.

LSTM

The LSTM model with the highest accuracy was found to have a learning rate of 0.001, a batch size of 1024, and a dropout rate of 0.3, yielding an accuracy of approximately 74.8% with a running time of 1951.294 seconds (see Fig.9.). While the other hyperparameters showed similar accuracy levels, the computational time varies. For instance, the model with a learning rate of 0.001, a batch size of 512, and a dropout rate of 0.3 presents a similar accuracy of 74.7% but executes relatively faster, around 1058.92 seconds. Hence, considering both accuracy and efficiency, the optimal LSTM model is chosen with hyperparameters {learning rate: 0.001, batch size: 512, dropout rate: 0.3}. Also, it is notable that the model with learning rate 0.01, batch size of 512 and drop rate 0.2 has a longest training time of 4395.948 second. The training time across a higher learning rate of 0.01 resulting slower computation time might due to the high learning rate cause the model to overshoot the optimal point during training and never converge. With smaller batch size,the model weights are updating more frequently hence resulting in a longer training time. For lower dropout rate, fewer neurons are dropped out during training, meaning the network is more complicated. Consequently, the training process can become slower because more parameters need to be updated.

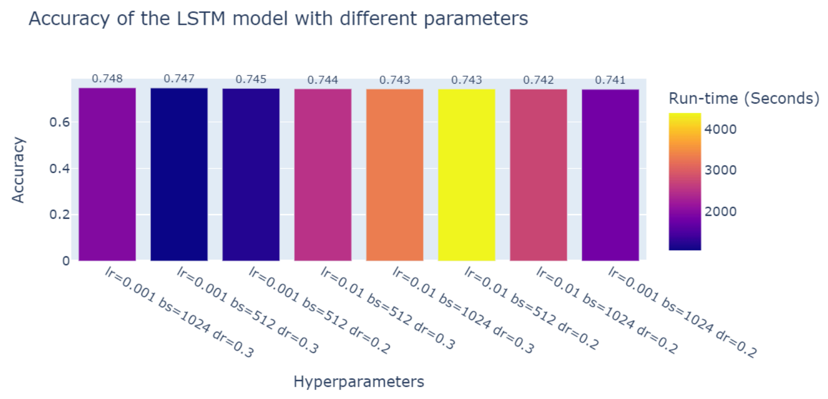
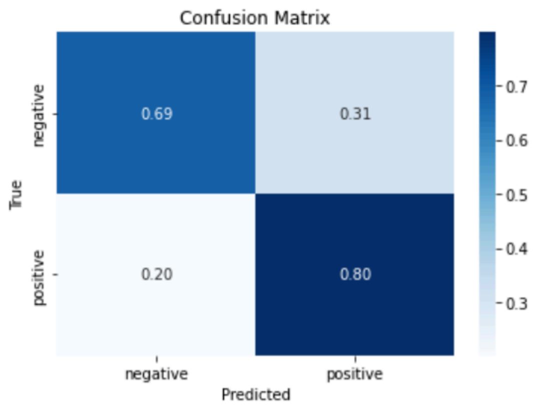


Fig. 9. Accuracy and run-time in seconds of the LSTM model with different parameters.



The true negative rate of 0.69 and true positive rate of 0.80 (Fig. 10.) indicates that the final model of Lstm is quite good at predicting positive sentiment, with a correct rate of 80%. However, it fails to predict the negative sentiment with an accuracy of only 69%. This relatively high false negative rate of 0.31 shows a weakness of the model in correctly identifying negative sentiments.

Fig. 10. Normalised Confusion matrix for tuned LSTM model.

Comparing the performance of models

The performance values of the different models, including accuracy, precision, recall, and F1-score, are presented in Table 1. SVM achieved the highest accuracy with a score of 0.758, followed closely by Logistic Regression with a score of 0.753. LSTM had a slightly lower accuracy of 0.747.

In order to evaluate the overall performance of the models across the two classes, we utilised the macro average calculation for the metrics. The strategy gives the average of a given metric for the classes. As the dataset has an equal amount of positive and negative sentiment tweest, this approach provides a balanced representation of the model's performance. Precision, Recall, and F1-score is similar for all 3 models, with scores of 0.74-0.76 for each metric. There is no significant difference in the ability of the models to balance precision and recall.

There is however a big contrast between the training time. Logistic Regression had the shortest training time, completing in just 0.65 seconds. On the other hand, both SVM and LSTM required significantly longer training times, with SVM taking 9859.16 seconds and LSTM taking 1058.92 seconds.

	Logistic regression	SVM	LSTM
Accuracy	0.753	0.758	0.747
Precision (macro avg)	0.75	0.76	0.75
Recall (macro avg)	0.75	0.76	0.74
F1-score (macro avg)	0.75	0.76	0.74
Training time(s)	0.65	8935.21	1058.92

Table 1. Comparison of the final models, with regards to the mean accuracy, macro average precision, macro average recall, macro average F1-score and training time in seconds.

The higher the model complexity, the more time is needed for all the computations required. The LSTM model has multiple layers, which take a considerable amount of time to train. With all models performing similarly, Logistic regression took much less time to train. Considering the comparable performance of the classifiers, Logistic Regression is selected as the best model due to the fast training.

Conclusion

In this project, we implemented sentiment analysis using three different models: Support Vector Machines (SVM), Logistic Regression, and Long Short-Term Memory networks (LSTM). All three models exhibited similar performance in terms of accuracy, precision, recall, and F1-score. However, Logistic Regression had the shortest training time, while SVM and LSTM required considerably more time for training. Logistic Regression was obtained as the best model, considering the trade-off between accuracy and training time.

To further the sentiment analysis, more advanced model architectures like BERT can be implemented. This pre-trained model has been proved to provide better understanding of the word hence improving the NLP industry (Tarique Akhtar), and we believe their application could significantly improve sentiment analysis.

Due to the long time needed to fit the SVM classifier, we utilised a subset of the training data for the hyperparameter tuning of the model. The SVC implementation in the *scikit-learn* library is based on *libsvm*, with the time required to fit the model growing at least quadratically with the number of samples ("sklearn.svm.SVC — scikit-learn 1.2.2 documentation"). A possible extension to the project would be to implement the model with a different function, which would require less time for fitting the data (e.g. SGDClassifier for linear SVM), and therefore be able to utilise the whole training dataset for the development of the classifier.

Works Cited

Akhtar, Tarique. “Introduction to BERT and its application in Sentiment Analysis.” *Medium*, 30 November 2021, https://medium.com/analytics-vidhya/introduction-to-bert-and-its-application-in-sentiment-analysis-9c593e955560. Accessed 17 May 2023.

Amidi, Afshine, and Shervine Amidi. “Recurrent Neural Networks cheatsheet.” *CS 230 - Recurrent Neural Networks Cheatsheet*, https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks. Accessed 15 May 2023.

Arbel, Nir. “How LSTM networks solve the problem of vanishing gradients.” *DataDrivenInvestor*, https://medium.datadriveninvestor.com/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577. Accessed 16 May 2023.

Balakrishnan, Vimala, and Ethel Lloyd-Yemoh. “Stemming and lemmatization: A comparison of retrieval performances.” *Proceedings of SCEI Seoul Conferences*, 2014, https://eprints.um.edu.my/13423/.

Brownlee, Jason. “Understand the Impact of Learning Rate on Neural Network Performance - MachineLearningMastery.com.” *Machine Learning Mastery*, 25 January 2019, https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/. Accessed 16 May 2023.

Chowdhury, Kuldeep. “10 Hyperparameters to keep an eye on for your LSTM model — and other tips.” *Medium*, 24 May 2021, https://medium.com/geekculture/10-hyperparameters-to-keep-an-eye-on-for-your-lstm-model-and-other-tips-f0ff5b63fcd4. Accessed 16 May 2023.

Cortes, Corinna, and Vladimir Vapnik. “Support-vector networks.” *Machine Learning*, vol. 20, 1995, pp. 273-297. https://doi.org/10.1007/BF00994018.

“Counting characters | Docs.” *Twitter Developer*, https://developer.twitter.com/en/docs/counting-characters. Accessed 12 May 2023.

Czepiel, Scott. “Theory.” *Maximum Likelihood Estimation of Logistic Regression Models: Theory and Implementation*, pp. 2 - 14, https://czep.net/stat/mlelr.pdf.

Eckhardt, Karsten. “Choosing the right Hyperparameters for a simple LSTM using Keras.” *Towards Data Science*, 29 November 2018, https://towardsdatascience.com/choosing-the-right-hyperparameters-for-a-simple-lstm-using-keras-f8e9ed76f046. Accessed 16 May 2023.

Gandhi, Usha Devi, et al. “Sentiment Analysis on Twitter Data by Using Convolutional Neural Network (CNN) and Long Short Term Memory (LSTM).” *Wireless Personal Communications*, 2021, https://doi.org/10.1007/s11277-021-08580-3.

Gräbner, Dietmar, et al. “Classification of Customer Reviews based on Sentiment Analysis.” *Information and Communication Technologies in Tourism*, 2012, pp. 460-470, https://doi.org/10.1007/978-3-7091-1142-0_40.

Guide, Step, and Debasish Kalita. “A Brief Overview of Recurrent Neural Networks (RNN).” *Analytics Vidhya*, 11 March 2022, https://www.analyticsvidhya.com/blog/2022/03/a-brief-overview-of-recurrent-neural-networks-rnn/. Accessed 17 May 2023.

Guide, Step, and Shipra Saxena. “LSTM | Introduction to LSTM | Long Short Term Memory Algorithms.” *Analytics Vidhya*, 16 March 2021, https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/. Accessed 15 May 2023.

KAZANOVA, ΜΑΡΙΟΣ ΜΙΧΑΗΛΙΔΗΣ. “Sentiment140 dataset with 1.6 million tweets.” *Kaggle*, 2016-2017, https://www.kaggle.com/datasets/kazanova/sentiment140. Accessed 12 May 2023.

Korkontzelos, Ioannis, et al. “Analysis of the effect of sentiment analysis on extracting adverse drug reactions from tweets and forum posts.” *J. Biomend. Inform.*, vol. 62, 2016, pp. 148-158, https://doi.org/10.1016/j.jbi.2016.06.007.

Kummerfeld, Jonathan Kay. “Lecture 4: Word Classification and Machine Learning 2.” lecture notes. *COMP 4446 / 5046 Natural Language Processing*, University of Sydney. Delivered 7 March 2023.

Mostafa, Lamiaa. “Home Proceedings of the International Conference on Advanced Intelligent Systems and Informatics 2020 Conference paper Egyptian Student Sentiment Analysis Using Word2vec During the Coronavirus (Covid-19) Pandemic.” *International Conference on Advanced Intelligent Systems and Informatics*, vol. 1261, 2021, pp. 195-203.

“New user FAQ.” *Twitter Help Center*, https://help.twitter.com/en/resources/new-user-faq. Accessed 12 May 2023.

“NLP Beginner - Text Classification using LSTM.” *Kaggle*, 14 May 2020, https://www.kaggle.com/code/arunrk7/nlp-beginner-text-classification-using-lstm#Word-Embedding. Accessed 1 May 2023.

Phienthrakul, Tanasanee, et al. “Sentiment Classification with Support Vector Machines and Multiple Kernel Functions.” *International Conference on Neural Information Processing*, vol. 5864, 2009, pp. 583-592, https://doi.org/10.1007/978-3-642-10684-2_65.

Pramana, Rio, et al. “Systematic Literature Review of Stemming and Lemmatization Performance for Sentence Similarity.” *IEEE 7th International Conference on Information Technology and Digital Applications (ICITDA)*, 2022, pp. 1-6. doi: 10.1109/ICITDA55840.2022.9971451.

“python - How to graph grid scores from GridSearchCV?” *Stack Overflow*, https://stackoverflow.com/questions/37161563/how-to-graph-grid-scores-from-gridsearchcv. Accessed 16 May 2023.

“RBF SVM parameters.” *Scikit-learn*, https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html. Accessed 16 May 2023.

“sklearn.svm.SVC — scikit-learn 1.2.2 documentation.” *Scikit-learn*, https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html. Accessed 17 May 2023.

Sreenivasa, Sushanth. “Radial Basis Function (RBF) Kernel: The Go-To Kernel.” *Towards Data Science*, 12 October 2020, https://towardsdatascience.com/radial-basis-function-rbf-kernel-the-go-to-kernel-acf0d22c798a. Accessed 16 May 2023.

“Tokenization.” *Introduction to information retrieval*, by Christopher D. Manning, et al., edited by Prabhakar Raghavan and Hinrich Schütze, Cambridge University Press, 2008, https://nlp.stanford.edu/IR-book/html/htmledition/tokenization-1.html. Accessed 15 May 2023.

Travers, Eoin. “Why does logistic regression overfit in high-dimensions?” *Eoin Travers*, 25 August 2020, http://eointravers.com/post/logistic-overfit/. Accessed 16 May 2023.

“Twitter sentiment classification using distant supervision.” *CS224N Project Report*, 2009. *cs-stanford*, https://www-cs.stanford.edu/people/alecmgo/papers/TwitterDistantSupervision09.pdf.

“What are Recurrent Neural Networks?” *IBM*, https://www.ibm.com/topics/recurrent-neural-networks. Accessed 15 May 2023.

“What is LSTM - Introduction to Long Short Term Memory.” *Intellipaat*, https://intellipaat.com/blog/what-is-lstm/?US. Accessed 15 May 2023.

Wijaya, Komang Dhiyo Yonatha, and AAIN Eka Karyawati. “The Effects of Different Kernels in SVM Sentiment Analysis on Mass Social Distancing.” *Jurnal Elektronik Ilmu Komputer Udayana*, vol. 9, no. 2, 2020, pp. 161-168, https://ojs.unud.ac.id/index.php/JLK/article/download/64395/37194.

Appendix

Instructions to run the code

Running the code file requires Python installation with ‘pip’ package management system. The python version used to run the code is 3.9.7, however, it should be compatible with any Python version in the 3.x series. In order to avoid minor differences between the different versions of python, and obtain the same results ensure that you have Python 3.9.7.

Install the required libraries, by executing the commands in the command prompt:

```
pip install tensorflow==2.12.0
pip install nltk==3.6.5
pip install pandas==1.3.4
pip install keras==2.7.0.
```

Download the dataset from the link below and as ‘training.300000.processed.noemoticon.csv’, put it in the same folder as the code file.
https://drive.google.com/file/d/1w9vysV8MII_6LF26XFZIfCbyG2MbDj--/view?usp=sharing

To open the code launch jupyter notebook, select the directory with the folder that contains the code file and click on the file.

To run the code select ‘Kernel’-> Run all.