Evaluation of Machine Learning Algorithms on Twitter Sentiment Detection

Final Project Report

Proud CHAREESRI
DSBA
ESSEC&
CentraleSupélec
Paris IDF France
B00803841@essec.edu

Jiuduo WANG
DSBA
ESSEC&
CentraleSupélec
Paris IDF France
B00806059@essec.edu

Shuya WU
DSBA
ESSEC&
CentraleSupélec
Paris IDF France
B00798610@essec.edu

Yujun ZHAI
DSBA
ESSEC&
CentraleSupélec
Paris IDF France
B00805282@essec.edu

ABSTRACT

Sentiment analysis using machine learning and deep learning techniques has gained a lot of attention in the natural language processing field these days. Natural language sentiment analysis has significant applications in various domains, such as social media monitoring, customer service, public opinion measurement, etc. In this project, we use the "Sentiment140 dataset with 1.6 million tweets" datasets[1] on Kaggle to analyze sentiment with tweets. We use traditional machine learning techniques with hyperparameter tuning, including logistic regression, naïve bayes, random forest, k-nearest neighbors (kNN), support vector machines (SVM), and several ensemble learning methods, such as AdaBoost, LightGBM and XGBoost. These methods are applied to data prepared using the TF-IDF (Term Frequency-Inverse Document Frequency). We also try other data processing methods including normal tokenizer, Word2Vec and GloVe, then we develop deep learning models using ANN, CNN, RNN (LSTM and GRU). After training and evaluations, we find that logistic regression model, naïve Bayes model, random forest model, SVM model, LSTM model and bidirectional model perform well in this text sentiment classification task and LSTM model performs the best. Future works may involve considering the sentiment effect of noise, slang, abbreviations and punctuations, conducting further hyperparameter tuning and fine-tune model architectures, introducing more classes in reality into the datasets. Also, we find that the mislabeling could be the reason limiting the performance of all models.

1 INTRODUCTION & MOTIVATION

1.1 Introduction

In recent years, the maturity of machine learning techniques and the continuous development of deep learning techniques has greatly promoted the development of the field of natural language processing (NLP). NLP is a branch of artificial intelligence that helps computers understand, interpret and manipulate human language. It can fill the gap between human communication and computer understanding and can be applied to several key domains. Our project targets the domain of sentiment analysis within natural language processing. Social media platforms like Twitter have become an integral part of our daily lives. Billions of messages are posted on social media every day, so the data on the platform serves as a natural corpus for NLP, providing insights into sentiments expressed by users. The aim of our project is to leverage existing machine learning and deep learning techniques to develop an effective model to analyze sentiment in text on social media platforms. It may help brands or businesses with their business performance by brand public relations and customer service. For our analysis, we use the Kaggle dataset "Sentiment140 dataset with 1.6 million tweets". This dataset includes id, date, flag (query), user, text and our target variable which is labeled as 0 (negative) or 4 (positive) of 1.6 million tweets in several different languages. Our objective is to process the "text" column to make the classification within positive and negative. The project's success is contingent on developing accurate and efficient models for sentiment analysis within the dynamic and noisy environment of social media.

1.2 Significance and Potential Applications

The importance of this project lies in its ability to gain insights from numerous chaotic information through sentiment analysis which is particularly important in this era of information explosion. Sentiment analysis helps businesses to manage the brand reputation and make quick decisions. By analyzing sentiments related to their products or services from customers, businesses can better understand customer needs and respond to customer feedback effectively. Furthermore, those platforms will be able to create value for brands by better helping them in identifying trends, positioning themselves and promoting their marketing efforts.

Sentiment analysis is critical in this digital communication era. It can help not only in business but also many other aspects. Community managers and event organizers can leverage Twitter data to gauge community sentiment, measure event success, and tailor future initiatives based on feedback. As for the financial market, sentiment analysis can be used to predict market trends and investor sentiment. In the healthcare sector, sentiment analysis can be used to understand emotional feedback on services and treatments, thus helping improve the quality of medical services. In the education field, sentiment analysis helps to understand the student's real meaning and thoughts, thus helping in making

[1] https://www.kaggle.com/datasets/kazanova/sentiment140

personalized learning plans and improving teaching methods. In addition, sentiment analysis can be applied to Conversational AI, aiding in a better understanding of the true meaning of people's intentions. In conclusion, sentiment analysis is important and plays an important role in many aspects.

2 PROBLEM DEFINITION

2.1 Objective

In this project, the primary problem we are trying to solve is to identify and detect users' sentiment by analyzing text on twitter. The primary objective is to train a classification model that maximizes the accuracy of sentiment prediction on unseen Twitter text data. The accuracy is defined as the ratio of correctly predicted sentiments to the total number of tweets in the dataset. We also combine other metrics including precision, recall, F1 score, ROC AUC to measure model performance.

2.2 Notation

X represents the feature space.

Y represents the output space, indicating the sentiment target of the text.

d is the dimensionality of the vectorized text features after tokenization.

$(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ is the dataset, where x_i represents the vectorized and tokenized representation of a Twitter text, and y_i represents its corresponding sentiment label.

$\hat{f}(x)$ is the predictor, the model to predict y of a x.

2.3 Formal Definition

Feature space X:

$$X = R^d,$$

Output space Y:

$$Y = \{4 \text{ (positive)}, 0 \text{ (negative)}\},$$

Dataset:

$$\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\},$$

The predictor of the function mapping X to Y:

$$\hat{f}(x) : \mathrm{X} \rightarrow Y$$

2.4 Constraints and Restrictions

The input data comprises natural language text from Twitter, possibly containing noise, slang, and abbreviations. The sentiment analysis task is strictly framed as a binary classification problem, where tweets are categorized as either "positive" or "negative." These factors constraints the model in some aspects and we will elaborate more in conclusion part.

3 RELATED WORKS

Sentiment Classification The idea of classifying documents by sentiment using machine learning techniques has a long history. Bo Pang, Lillian Lee and Shivakumar Vaithyanathan used three machine learning methods to classify movie reviews in 2002 using standard bag-of-features framework. Our task is like theirs, but we use TF-IDF and more deep learning techniques.

TF-IDF TF-IDF is a statistical method used to measure the importance of a word in a document. It was initially introduced by British computer scientist Karen Spärck Jones. Bijoyan Das, Sarit Chakraborty use TF-IDF along with next word negation model for text classification, finding that Linear Support vector machine (LSVM) is the most appropriate technique. In our task, we will implement TF-IDF to more machine learning models.

Word Embedding Tomas Mikolov et al. introduced word embedding 2013. In' Efficient Estimation of Word Representations in Vector Space, they introduced Word2Vec, a technique including two novel model architectures to compute continuous vector representations of words. They also published pre-trained vectors trained on part of Google News dataset (about 100 billion words). The model contains 300-dimensional vectors for 3 million words and phrases. Another popular word embedding technique is GloVe (Global Vectors for Word Representation) introduced by Stanford NLP Group in 2014. It was performed on aggregated global word-word co-occurrence statistics from a corpus. They published the pre-trained word vectors using Wikipedia 2014 + Gigaword 5, Common Crawl and Twitter. In our work, we use both techniques and compare their results.

Deep Learning Methods There are also many deep learning models implemented on sentiment analysis. BERT (Bidirectional Encoder Representations from Transformers) was designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers by Jacob Devlin et al. Yoon Kim (2014) use CNN to do sentiment analysis and question classification. Yequan Wang et al. use Attention-based LSTM for Aspect-level Sentiment Classification and achieves state-of-the-art performance on aspect-level sentiment classification. There are many papers using different deep learning models to do the text sentiment analysis. In our task, we explore many existing deep learning models and compare their performances.

4 METHODOLOGY

4.1 Data Import and Visualization

We first download the datasets from Kaggle and import the dataset into "data" dataframe. On inspection of the data, we can see that there are six columns ("target", "ids", "date", "flag", "user", "text") and 1600000 rows from 1598315 users. After filtering for useful columns ("target", "text") and checking if there is null, we can find that there are no null values in our datasets. Checking the balance of the datasets, we can see that the dataset is well-balanced and well distributed regarding the target value. We use langdetect to check the languages in our dataset finding that there are 33 languages, and some are still unknown in our dataset.

4.2 Data Preprocessing

Considering the process procedures, stopwords and models are quite different in different language sentiment analysis, also we cannot guarantee the balance of the data among each language, so we filter the English language text. Checking the language again, we find that the dataset is still quite balanced regarding the target value. We randomly sample 4000 data from negative and positive data (2000 in each category), then combine them together to get our datasets for modeling to save run time. Then we preprocess the "text" column, following:

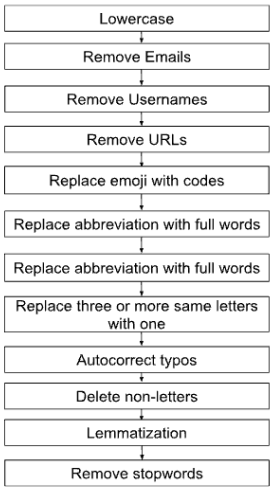

Figure 1: Text Preprocessing Steps

We try lemmatization, after transferring all the words to lower case, removing emails, usernames and URLs, replacing emojis with their words, replacing abbreviation with full words, replacing three or more same letters with one, autocorrecting typos using "autocorrect" package, deleting non-letters. Lemmatization is a tool to normalize words by converting words into their lemmas, which helps reduce the variations of words and at the same time retain the true meaning of words. Unlike stemming, which simply removes the suffix of a word without considering the context, lemmatization typically involves using lexical and grammatical rules to convert words to lemmas. After lemmatization, it will be easier to identify vocabulary during sentiment analysis. As a final step as normal text preprocessing, we delete the stopwords to extract highly informative words.

Example	Stemming	Lemmatization
am	am	be
having	hav	have
wolves	wolv	Wolf
happiest	happi	happy

Table 1: Stemming VS Lemmatization Example

At first, we used WordNetLemmatizer from nltk to do the lemmatization, but after checking the word clouds, we have identified some abnormal words such as "wa" which appeared very frequently but were not deleted when removing stopwords. After checking the preprocessing procedures one by one, we found that the WordNetLemmatizer from nltk transformed "was" to "wa", so when we removed stop words, the word "wa" was not removed and appeared very large in the word clouds. Then we tried lemmatizer from other packages, finding that nlp from spacy performed optimally. So, in our last edition, we use spacy to do the lemmatization.

4.3 Word Cloud

We make word clouds to check with the words and verify the effectiveness of the preprocessing steps.

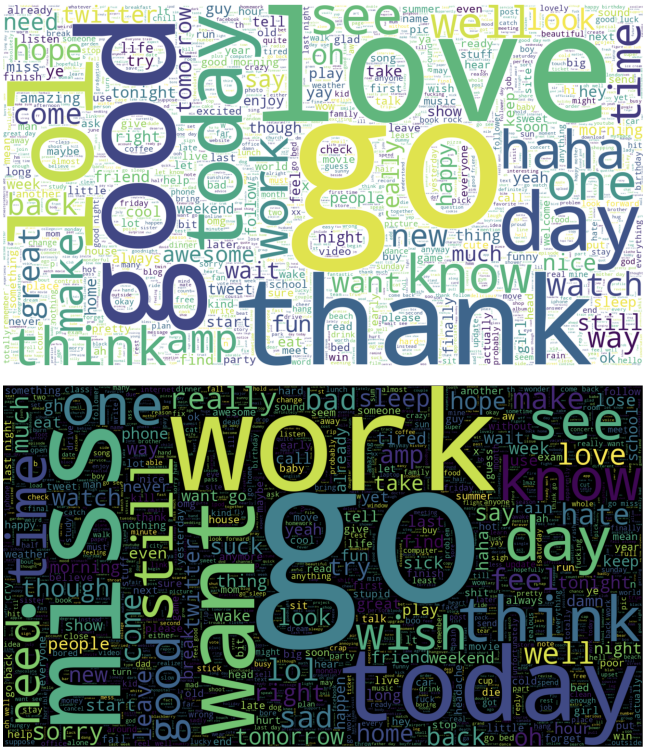

Figure 2: Positive (Top) & Negative Wordcloud (Below)

4.4 Data Processing

We split the datasets into two parts, training and test sets according to the ratio of 7:3, resulting in 27,906 training samples and 11,960 test samples. Then we use cross-validation in machine learning models to tune the hyperparameters. Since we need validation set in deep learning model, we further split the test data into validation and test sets with the ratio of 1:1, resulting in 5,980 validation samples, and 5,980 test samples for deep learning models.

Data processing for Machine Learning

TF-IDF (Term Frequency-Inverse Document Frequency) measures the importance of a particular term in a document collection. It

contains 2 parts: TF assesses the frequency of a term in a document and IDF evaluates the importance of a word in the whole document collection.

$$\mathrm{TF}(t,d) = \frac{\mathrm{count}(t,d)}{\sum_i \mathrm{count}(i,d)}$$

$$\mathrm{IDF}(t,D) = \log\left(\frac{|D|}{|\{d \in D : t \in d\}| + 1}\right)$$

$$\text{TF-IDF}(t,d,D) = \mathrm{TF}(t,d) \times \mathrm{IDF}(t,D)$$

t *representsa term,* d *represents a document, and* D *represents the document collection.*

To do TF-IDF, vectorization need to be first executed. We choose the vectorizer considering both unigrams (single words) and bigrams (two consecutive words) and limiting the number of features to 20000. We choose both unigrams and bigrams because this helps capture not only individual words but also pairs of words that might convey additional meaning together. And also we did many experiments, finding that the parameter "ngram_range=(1,2), max_features=20000" works optimally.

Data processing for Deep Learning

Another step is to tokenize the data for deep learning models. Tokenization is a technique to set index to the words in the whole database, then convert sentences into vectors of numbers. Getting the word_index in preprocessed_data['preprocessed_text'], we turn X into tokenized X. Then we pad the tokenized X to the same length, which is the length of the longest sentence, 25. This helps convert text data into a form that the neural network can process.

Word - Index

i	love	dogs	this	one	is	cute	so	really	it
1	2	3	4	5	6	7	8	9	10

Encoding & Padding

i love dogs	1	2	3	0	0	0	0
this one is cute	4	5	6	7	0	0	0
i love cute one	1	2	7	5	0	0	0
i really love dogs	1	9	2	3	0	0	0
this is really cute i love it	4	6	9	7	1	2	10

Table 2: Tokenization Example

4.5 Machine Learning Model Application

In this section, we explore the application of various basic machine learning models to our datasets. The following algorithms are employed to analyze and classify the data. In addition, applying the models from scikit-learn, hyperparameter tuning work has also been conducted to the model. Methods involved include Grid Search, Random Search, Bayes Search and accuracy is chosen as the metric.

4.5.1 Logistic Regression. Logistic Regression is a supervised algorithm which is simple and efficient for binary classification problems. It applies a logistic function to a linear combination of input features and then estimates the mathematical probability of whether an instance belongs to a specific category or not.

For Logistic Regression, grid search is employed. Grid search helps define the hyperparameter space based on the parameters we determine to tune. Then it will find the best parameter combinations among all through training the model on all possible points in the hyperparameter space. After getting the best parameters, cross-validation is conducted to ensure there is no overfitting problem occurring. It helped tune hyperparameters including:

C: regularization strength, a larger C implies weaker regularization;
class_weight: used to handle unbalanced dataset if applies
max_iter: maximum number of iterations, a higher value improve; performance but increase costs
solver: algorithms for optimization including 'lbfgs', 'liblinear', which should be considered with data size and other characteristics.

Through cross-validation, we find the best parameters are: {'C': 1, 'class_weight': 'balanced', 'max_iter': 10, 'solver': 'liblinear'}, which acheived the best cross-validation accuracy 73.75%.

4.5.2 Naïve Bayes. Naïve Bayes is a supervised probabilistic classifier based on Bayes' theorem. It assumes conditional independence among features, making it particularly effective for text classification tasks.we use grid search to tune hyperparameter:
alpha: used for Laplace smoothing, which addresses the issue of zero probabilities; a lower value makes it sensitive to noise

Through grid search and cross-validation, we find the best parameter is: {'alpha': 10}, which acheived the best cross-validation accuracy 73.44%.

4.5.3 Random Forest. Random Forest is an ensemble learning method that constructs a multitude of decision trees using different random subsets of the data and features and outputs the most popular prediction as the result.

For Random Forest model, gird search is less suitable because the computational cost of exploring the space could be very high. Instead, Random Search is more suitable for it randomly sample hyperparameter combinations to evaluate instead of exclusively exploring every point of the hyperparameter space. It helps tune hyperparameters including:

max_features: the maximum of features a tree, a higher value may lead to overfitting
n_estimators: number of trees in the forest, a higher value improve performance but increase costs
max_depth: the depth of a tree, help controls overfitting
min_samples_split: minimum number of samples to decide if a tree should be split, a higher value help not to capture noise

Through random search and cross-validation, we find the best parameter is: {'n_estimators': 200, 'min_samples_split': 200, 'max_features': 'log2', 'max_depth': None}, which acheived the best cross-validation accuracy 73.24%.

4.5.4 kNN. kNN (k-nearest neighbors) is a non-parametric supervised classification algorithm. It assigns a data point to the majority class among its k-nearest neighbors based on a distance metric. We use grid search to tune hyperparameter:

n_neighbors: numbers of neighbors used for classification, a higher value means smoother boundary but also may lose inner patterns

weights: weights function, can be 'uniform' (equal weights) or 'distance' (inverse of distance)

algorithm: algorithm used to computes nearest neighbors, 'auto' can choose the best algorithm automatically

Through grid search and cross-validation, we find the best parameter is: {'algorithm': 'auto', 'n_neighbors': 300, 'weights': 'uniform'}, which acheived the best cross-validation accuracy 71.90%.

4.5.5 SVM. SVM (Support Vector Machine) is a supervised learning algorithm used for both classification and regression. It finds the hyperplane that best separates the data into different classes. We use grid search to tune hyperparameter:

kernel: kernel function, decides the decision boundary type such as 'linear', 'sigmoid'

C: regularization strength, a larger C implies weaker regularization.

gamma: a smaller value makes the decision boundary smoother, which may invoke overfitting

Through grid search and cross-validation, we find the best parameter is: {'C': 1, 'gamma': 1, 'kernel': 'rbf'}, which acheived the best cross-validation accuracy 74.82%.

4.5.6 AdaBoost. AdaBoost is an ensemble learning method that combines weak learners sequentially, which adjusts weight to misclassified instances such that subsequent classifiers focus more on difficult case. For Adaboost model, Bayesian search is chosen as the optimization algorithm. Grid search can't be applied on continuous space and random search may get stuck in the local optimum. Same occasions go on following ensemble learning model including LightGBM and XGBoost. Bayesian search optimizes its parameter selection in each round according to the previous round score, which performs better for it reaches the optimum faster. It helps tune hyperparameters including:

n_estimators: number of weak learners, a high value may cause overfitting

learning_rate: contribution of each weak learner to the total model

base_estimator__max_features: number of features to consider, similar as those in random forest

base_estimator__max_depth: depth of the weak learners, similar as those in random forest

Through cross-validation, we find the best parameter is: { ('base_estimator__max_depth',2),('base_estimator__max_features', None), ('learning_rate', 0.5), ('n_estimators', 300) }, which acheived the best cross-validation accuracy 71.71%.

4.5.7 LightGBM. LightGBM (Light Gradient-Boosting Machine) is a gradient boosting framework based on decision trees developed by Microsoft. It is very efficient on large datasets. We use Bayesian search to tune hyperparameter:

max_depth: the depth of a tree, help controls overfitting

subsample: subsample ratio of the training instances, affects the randomness and diversity

colsample_bytree: subsample ratio of features, affects the inner feature diversity of a tree

reg_alpha: L1 regularization term on weights

reg_lambda: L2 regularization term on weights

Through cross-validation, we find the best parameter is: {('colsample_bytree', 0.7), ('max_depth', 6), ('reg_alpha', 0.01), ('reg_lambda', 0.1), ('subsample', 0.9)}, which acheived the best cross-validation accuracy 70.03%.

4.5.8 XGBoost. XGBoost (Extreme Gradient Boosting) is a scalable, distributed gradient-boosted decision tree model, where trees are built in parallel, instead of sequentially. At each level, it evaluates the quality of splits according to the gradient, thus making itself very efficient. We use Bayesian search to tune hyperparameter:

n_estimators: number of boosting trees, a high value may cause overfitting

learning_rate: contribution of each tree to the total model

base_estimator__colsample_bytree: subsample ratio of features, affects the inner feature diversity of a tree

base_estimator__max_depth: maximum depth of a tree

Through cross-validation, we find the best parameter is: {('base_estimator__colsample_bytree', 0.5), ('base_estimator__max_depth', 4), ('learning_rate', 0.1), ('n_estimators', 1000)}, which acheived the best cross-validation accuracy 72.15%.

These machine learning models provide a comprehensive analysis of the dataset, capturing different aspects of the data patterns and relationships. The algorithms employed in this section serve as a comparison to the deep learning models discussed in the subsequent 4.6 chapter.

4.6 Deep Learning Model Application

In this chapter, we apply some deep learning models to our datasets, such as ANN, CNN, LSTM, GRU, LSTM-Bidirectional, GRU-Bidirectional with different forms of embedding methods.

Word Embedding is a technique which maps words in datasets into a high-dimensional vector space. In this vector space, the words sharing similar meanings will be closer compared to the words are quite different. By doing word embedding, we can better capture semantic information. The vector space can be transformed to a matrix, a row (vector) is represented as the coordinate of a word, and the number of columns is the dimension of the vector space. We can use embedding layer in our deep learning model, the first method we use is self-learning embedding. The weights of the embedding layer will be randomly initialized and learned during the training process of the model. The second word embedding

method we use is custom Word2Vec. In this method, we pretrained the embedding matrix by Word2Vec using our own dataset. Word2Vec uses two training methods, Skip-gram and Continuous Bag of Words (CBOW). Skip-gram predicts the context words before and after a given word, while CBOW predicts the target word given the context words. From the results, we can see that these two methods don't work as well as it should be. We think that the reason is that Self-training methods use our own dataset, which is very limited. Methods like Word2Vec normally require large amounts of data to learn useful information, since our training dataset is quite small, we consider using a pre-trained word vector model, such as pretrained Word2Vec and GloVe. Pre-trained Word2Vec vectors[2] are trained on a part of the Google News dataset (about 100 billion words). The model contains 300-dimensional vectors for 3 million words and phrases. GloVe (Global Vectors for Word Representation)[3] is another unsupervised learning algorithm by Stanford NLP Group for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space. The pre-trained word vectors we use is Common Crawl (42B tokens, 1.9M vocab, uncased, 300d vectors).

The models we use are:

ANN: This model consists of an embedding layer, followed by flattening and dense layers with ReLU activation functions, and concludes with a sigmoid activation output layer for binary classification.

CNN: This model adds one 1D convolutional layer with 128 filters and a kernel size of 5, followed by a global max pooling layer. We also use dropout technique to avoid overfitting.

LSTM: This model adds two Long Short-Term Memory (LSTM) layers to capture long-range dependencies in sequential data and dropout to avoid overfitting.

GRU: Another way to capture long-range dependencies in sequential data is Gated Recurrent Unit (GRU) layers with dropout to avoid overfitting.

LSTM-Bidirectional: This model adds Bidirectional layer to processes the input sequence in both forward and backward directions to the LSTM model.

GRU-Bidirectional: This model adds Bidirectional layer to processes the input sequence in both forward and backward directions to the GRU model.

Model	Self-learning Embedding	Custom Word2Vec	Word2Vec	GloVe
ANN	0.7067	0.6008	0.6711	0.6796
CNN	0.7125	0.6045	0.7286	0.7226
LSTM	0.7261	0.6263	0.7492	0.7365
GRU	0.4965	0.6013	0.7413	0.7385
LSTM-BI	0.6997	0.6013	0.7423	0.7400
GRU-BI	0.6980	0.6130	0.7326	0.7314

Table 3: Validation Accuracy of the Last Epoch

From the result, we can see that the models with pre-trained word vectors in Word2Vec and GloVe perform better than models with self-learning embedding and custom Wod2Vec. LSTM model with and without bidirectional perform better than others with pre-trained word vectors in Word2Vec and GloVe. Since the models with pre-trained word vectors in Word2Vec is slightly better than the ones with GloVe, we decide to choose the LSTM model with and without bidirectional using pre-trained word vectors by Word2Vec.

Checking the history of validation accuracies of each epoch, we find that these two models are not overfitting. Therefore, we consider adjusting the architecture and hyperparameters of these two models. We do the Bayesian optimization using gp_minimize to tuning the learning_rate and batch_size. We find that the optimized learning_rate is 0.001, batch_size is 128.

5 EVALUATION

In this section, we choose four machine learning model (lr_model, nb_model, rf_model, svm_model) which perform well in 4.5 and two deep learning model (model_lstm, model_bid) which perform well in 4.6. We compare their performances on test set using several metrics, including accuracy, precision, recall, F1 score and ROC AUC.

Model	Accuracy	Precision	Recall	F1 Score	ROC AUC
lr	0.7489	0.7419	0.7627	0.7522	0.8284
nb	0.7486	0.7638	0.7192	0.7408	0.8267
rf	0.7462	0.7446	0.7488	0.7467	0.8233
svm	0.7482	0.7394	0.7660	0.7525	0.8258
lstm	0.7522	0.7483	0.7662	0.7571	0.8297
bid	0.7485	0.7682	0.7177	0.7421	0.8281

Table 4: Metrics Comparison

We also plot the confusion matrix and ROC curve of each model.

[2] Word2Vec Website: https://code.google.com/archive/p/word2vec/

[3] GloVe Website: https://nlp.stanford.edu/projects/glove/

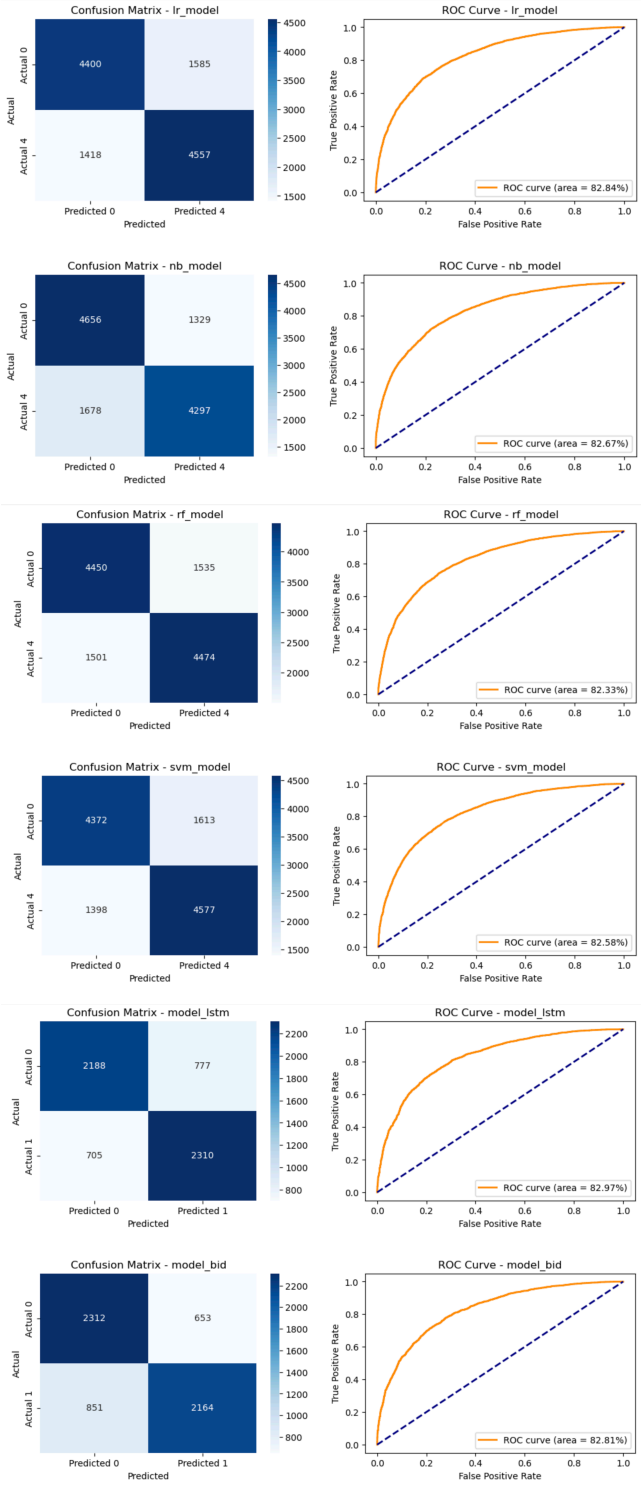

Figure 3: Confusion Matrix and ROC Curve Plots

From the table and plots above, we can find that the model using LSTM gain the highest accuracy. Model using LSTM and bidirectional method has the highest precision, which means that the model makes fewer mistakes when predicting positive categories. Both SVM and LSTM model perform well regarding recall, at the same time, naïve Bayes and bidirectional model perform worse than other models. A higher recall means that the model is good at identifying positive samples. Balancing precision and recall, we can see that LSTM model performs the best with F1 score. ROC AUC also indicates that the LSTM model performs the best. The values of ROC AUC of all the models are higher than 80%, which means that these models have relatively good ability to distinguish between positive and negative categories. We can also observe that there is minimal variation in these metrics among the selected models.

6 CONCLUSIONS

In conclusion, after preprocessing, word cloud inspection and different ways of data processing (TF-IDF and tokenizer), we find that through hyperparameter tuning, logistic regression model, naïve Bayes model, random forest model, SVM model, LSTM model and bidirectional model perform well in this text sentiment classification task.

Among these models, LSTM model from deep learning performs slightly better than others. However, we can find that the highest accuracy with 75.22% is not as high as we expected. Future work could explore further enhancements in model performance.

From the perspective of preprocessing, although we use autocorrection to correct the typos, there may still be some instances of noise, slang, and abbreviations. We also consider whether certain punctuation marks can convey emotions. From the perspective of the datasets, the original dataset consists of data target of only either “positive” or “negative”, which makes the case a binary classification problem. However, the twitter text in reality could be neutral, which calls for the introduction of more classes for better classification performance. Also, after we observe that all the models that we choose reach their optimal performance around 75%, we inspect the datasets, and find that there are some tweets that are actually neither positive nor negative, despite being labelled as positive or negative. This might be the reason that whatever model we use, the accuracy cannot be higher. From the perspective of models, we can conduct more refined hyperparameter tuning and fine-tune the architectures of deep learning models to optimize the performance of models.

REFERENCES

[1] Go, A., Bhayani, R. and Huang, L., 2009. Twitter sentiment classification using distant supervision. *CS224N Project Report, Stanford, 1(2009), p.12*.

[2] Das, B., & Chakraborty, S. (2018, June 17). An improved text sentiment classification model using TF-IDF and next word negation. arXiv.org. https://arxiv.org/abs/1806.06407

[3] Google. (n.d.). Google code archive - long-term storage for google code project hosting. Google. https://code.google.com/archive/p/word2vec/

[4] Gorodetski, M. (2021, September 6). Hyperparameter tuning methods - grid, random or Bayesian search?. Medium. https://towardsdatascience.com/bayesian-optimization-for-hyperparameter-tuning-how-and-why-655b0ee0b399

[5] Natural language processing (NLP): What it is and why it matters. SAS. (n.d.). https://www.sas.com/en_us/insights/analytics/what-is-natural-language-processing-nlp.html

[6] Pennington, J. (n.d.). GloVe: Global Vectors for Word Representation. Glove: Global vectors for word representation. https://nlp.stanford.edu/projects/glove/

[7] Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global vectors for word representation. ACL Anthology. https://aclanthology.org/D14-1162/

[8] Person. (2021, October 6). Understanding TF-IDF for Machine Learning. Capital One. https://www.capitalone.com/tech/machine-learning/understanding-tf-idf/

[9] Scikit-Learn Documentation. scikit. (n.d.). https://scikit-learn.org/stable/index.html

[10] Shafi, A. (2023, February 24). Random Forest classification with Scikit-Learn. DataCamp. https://www.datacamp.com/tutorial/random-forests-classifier-python

[11] What is the K-nearest neighbors algorithm?. IBM. (n.d.). https://www.ibm.com/topics/knn#:~:text=Resources-,Next%20steps,of%20an%20individual%20data%20point

[12] What is XGBoost?. NVIDIA Data Science Glossary. (n.d.). https://www.nvidia.com/en-us/glossary/xgboost/ Year:2018

[13] Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv:1810.04805.*

[14] Pang, B., Lee, L., & Vaithyanathan, S. (2002). Thumbs up? Sentiment Classification using Machine Learning Techniques. *ACL-02 Conference on Empirical Methods in Natural Language Processing*, 79-86.

[15] Das, B., & Chakraborty, S. (2018). An Improved Text Sentiment Classification Model Using TF-IDF and Next Word Negation. *arXiv preprint* arXiv:1806.06407.

[16] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

[17] Pennington, J., Socher, R., & Manning, C. D. (2014, October). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 1532-1543).

[18] Devlin J, Chang M W, Lee K, et al. Bert: Pre-training of deep bidirectional transformers for language understanding[J]. arXiv preprint arXiv:1810.04805, 2018.

[19] Kim, Y. (2014). Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

[20] Wang, Y., Huang, M., Zhu, X., & Zhao, L. (2016, November). Attention-based LSTM for aspect-level sentiment classification. In *Proceedings of the 2016 conference on empirical methods in natural language processing* (pp. 606-615).

[21] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., ... & Liu, T. Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, *30*.