Tweet classification: A comparison between Naive Bayes, ensemble methods and LSTM

Mohsen Pirmoradiyan
mohpi286

Abstract

Natural Language Processing as one of the branches of artificial intelligence deals with text document processing. Text classification as one of the important tasks in NLP addresses the problem of classifying a collection of textual documents into two or more categories which are labeled in a way that each group reflects a particular subject. The classification based on sentiment analysis focuses on categorizing the documents according to the attitude and emotions of the writer towards a particular object or subject under discussion. In this project a collection of manually labeled tweets about the Corona virus was selected to classify them into three classes, 'Negative', 'Neutral', and 'Positive', based on the underlying emotions conveyed by them. Naive Bayes, Ensemble methods, and LSTM models were built to carry out the task. From the performance perspective, LSTM outperforms the other methods, and ensemble methods did a better job compared with Naive Bayes. Considering the assumptions of the models we may infer that the dependency between the features in the contextual data processing is an important aspect to be considered.

I. INTRODUCTION

Classification, in general, is a supervised learning process attempting to classify or to categorize the data into two or more predefined categories. With this definition, we may define the text classification as a process aiming to assign the textual documents a specific label or tag as a class label. Depending on the classification goal, the class labels could be chosen so that each category reflects a specific topic, language, author, sentiment, etc., among which the sentiment-based classification is the aim of this project.

Sentiment refers to the underlying feeling and emotion in the text document expressed by the writer. People often have different opinions about any of the controversial issues. With the dramatic increase in using social media as the most widely used means to communicate with the world, people always share their beliefs about every matter of the world ranging from very local issues to global circumstances. The tweeter is one of the most worldwide used social media by which people all around the world share their opinion and attitude. Nowadays, there is no even a small issue which we can not find any discussion around it within the Daily published tweets. Sentiment analysis application in tweets analysis aims to label the tweets based on the underlying emotions. This kind of classification can always be helpful to extract the most predominant opinions and attitudes. Figuring out the most widely supported notions may help the politicians, decision-makers, and businesses to make a decision accordingly in order to either make a benefit such as winning an election campaign. It is also helpful in order to understand the trending points of view in favor of/against a topic, a policy, a hotel, a movie, etc.

Machine learning algorithms are very helpful to classify and predict whether a document represents positive or negative sentiment. As mentioned before, classification is a supervised machine learning task which uses a labeled dataset where each document of the training set has been labeled with appropriate sentiment.[10]

Many studies have been focused on sentiment-based classification.

Abinash et.al. attempted to perform a binary classification in which they categorized the textual movie reviews into positive and negative classes. They have implemented Naive Bayes and Support Vector machine in order to perform the task.

Abien Fred et.al. tried to implement deep neural networks to carry out a text classification to categorize the reviews on a women clothing e-commerce. Their result supports the claim that using recurrent neural networks better captures the context of review texts which leads to better predictive performance.

Peng Zhou et.al. studied the performance of deep neural networks by combining recurrent neural networks and convolutional neural networks to perform text classifications. Their experiment results demonstrate that their proposed model can outperform any of the RNN and CNN models implemented standalone.

	UserName	ScreenName	Location	TweetAt	OriginalTweet	Sentiment
0	3799	48751	London	16-03-2020	@MeNyrbie @Phil_Gahan @Chrisitv https://t.co/i...	Neutral
1	3800	48752	UK	16-03-2020	advice Talk to your neighbours family to excha...	Positive
2	3801	48753	Vagabonds	16-03-2020	Coronavirus Australia: Woolworths to give elde...	Positive
3	3802	48754	NaN	16-03-2020	My food stock is not the only one which is emp...	Positive
4	3803	48755	NaN	16-03-2020	Me, ready to go at supermarket during the #COV...	Extremely Negative

FIG. 1. First 5 rows of the Coronavirus tweets-NLP dataset. It contains six columns including: UserName, ScreenName, Location, TweetAt, OriginalTweet, and Sentiment. OriginalTweet and Sentimen are the main columns used in this project

In this project, machine learning algorithms were implemented to perform a text classification on a dataset containing tweets posted about the coronavirus. The purpose is to categorize the tweets into three sentiment groups including 'positive', 'neutral', and 'negative'. Naive Bayes firstly was used as a baseline method. Ensemble methods then were implemented from which two algorithms were selected: Random Forest and Adaboost classifiers as the representative of the averaging and boosting families respectively. Finally, recurrent neural networks (RNNs) were performed. The models were trained on the training set and then evaluated on the test set.

The remainder of this report is organized as follows. Section II introduces the data and tries to explore some aspects of the data visually. Section III presents a brief explanation of the theoretical parts of the methods. Sections IV describes the approach followed in this work. The results are then reported and discussed in Section V. Finally, some conclusions and suggestions for further studies are provided in Section VI.

II. DATA DESCRIPTION

The dataset is called Coronavirus tweets-NLP which is a Kaggle dataset proposing for text classification purposes. The tweets have been pulled from Twitter and manual tagging has been done then. It contains six columns including UserName, ScreenName, Location, TweetAt, OriginalTweet, and sentiment. Fig. 1 shows the first five rows of the dataset. As we can see the Sentiment column contains the labels which manually have been assigned to each tweet that exists in the OriginilTweet column. These two columns are the main columns used in this project. Totally 44955 samples exist in the dataset. The original class labels were 'Neutral', 'Extremely Negative', 'Negative','Positive', 'Extremely Positive'. However, in order to increase the number of samples within each group, the samples labeled as 'Extremely Negative' and 'Negative' were merged into a single group and tagged as 'Negative', and those in 'Positive' and 'Extremely Positive' classes were combined together to form a single 'Positive' group.

Fig. 2 illustrates the distribution of the class labels.

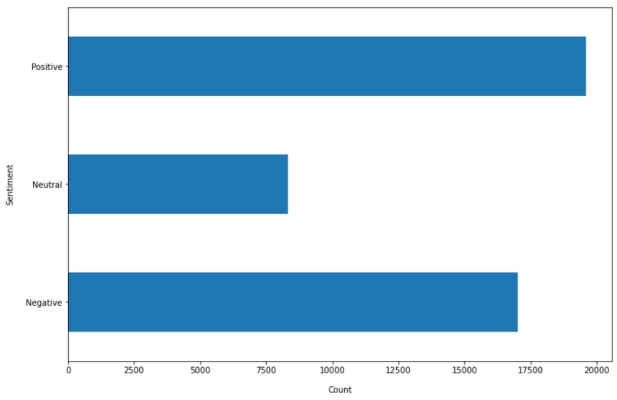

FIG. 2. Sentiment distribution

As we can see the Negative and Positive groups contain almost the same amount of tweets, however, the number of samples per Neutral class is much less than of the other two groups which makes an imbalanced dataset.

Every tweet consists of some words. Those words in fact can describe the feelings and opinions. Each of those words is potentially a feature to be used as input data to the learning algorithm. Nevertheless, not all of the consisting words and characters are meaningful. It is required to preprocess the contexts to remove all the meaningless characters and all the words that convey no specific emotions. The following section explains the preprocessing steps in detail. Clean tweets are used for further visualization in this section.

It is often of interest to have a visual insight into the frequent words used by people when they have different feelings. In fact, those words and the frequency of their occurrence in sentences can lead us to infer conveying emotions. WordCloud [7] provide us with such an interesting visualization. We can have a cloud of arbitrary shapes filled by different words each with a specific size which determines how frequently that word has been used in the text. In other words, the size of a word in the cloud represents the frequency of occurrence of that word. Fig. 3 shows the word clouds for the dataset under consideration. We may notice that the words 'covid', 'supermarket', 'grocery store', 'people', 'food' are the most commonly used words in all groups. It seems that the people who had a negative feeling were engulfed by economical concerns as the words 'oil price', 'work', 'stock', 'crisis', 'panic' are noticeable in the corresponding cloud. Positive tweets, on the other hand, conveyed much more optimistic words such as 'good', 'help', 'hand sanitizer', 'thank', 'great'. Disappointingly Neutral cloud does not show any distinguishable words. In another word, the neutral tweets consist of the same words as the other two categories do. What we can conclude from this analysis is that in spite of the existence of some outstanding words distinguishing the positive and negative groups, all the three categories share plenty of words in common. This result may suggest that using the words independent of each other might not lead the algorithm to perform a

FIG. 3. Word Clouds

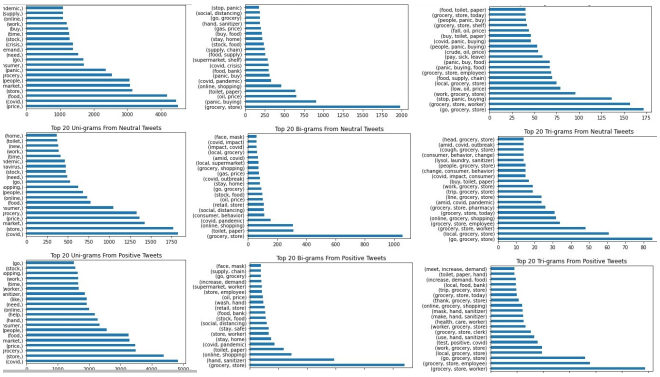

FIG. 4. N-grams distribution. Columns from left to right are related to Uni-grams, Bi-grams, and Tri-grams distribution respectively. Rows from top to bottom associate to Negative, Neutral, and Positive categories respectively.

perfect classification.

As another exploratory data analysis, N-Grams analysis could be quite interesting. It is very informative, particularly for sentiment-based text classification, to know how the words have been used in a review or a tweet. In other words, we would like to analyze the consecutive words that appear frequently together. Analyzing the frequency of words that often appear consecutively can help us to extract more distinguishing features for our classification. In other words, although words alone may be present in all sentences, the way that they are joined to each other to express a feeling might be different for each sentiment group. An n-gram is simply a sequence of n words where n is a discrete number that can range from 1 to infinity. In n-gram ranking, we simply rank the n-grams according to how many times they appear in the body of a text document. Fig. 4 depicts the result of this analysis. The columns from left to right are related to Uni-grams, Bi-grams, and Tri-grams distribution respectively. The top row corresponds to negative tweets, the bottom row describes the positive group, and the figures at the middle associate with the neutral class. For the sake of readability, only the top 20 frequent n-grams have been shown. As we may clearly notice the Unii-grams distributions show that all the classes share most of the words in common as we also noticed in WordCloud analysis. Interestingly, as we increase the number of consecutive words, it seems that we will have more separating features which can be used efficiently for text classification purposes.

We conclude this section by summarizing the results

we obtained through our exploratory data analysis. In a nutshell, the results may prove an important issue in text analysis, and that is the importance of the dependency between the words as features within a context. In other words, a sentence is not only a container of some words but is in fact a sequence of words whose positions matter considerably. Two texts consisting of the same words as their constituents can express totally different meaning if the positions of the words within the context change.

III. THEORY

Classification lies at the heart of both human and machine intelligence.[5]. The problem involves a wide variety range of tasks such as image labeling, face recognition, spam detection, voice recognition. The underlying task in all of these examples is to assign a class label to input data.

In the context of NLP, text classification refers to the task of splitting a document into two, or more categories each of which could reflect a different conceptual subject. As an example, we may are interested to divide a bunch of textual documents into two groups: those which contain political content, and those which focus on economical issues. In that case, in which the task is to divide the inputs into two categories is referred to as a binary classification. In cases the class labels are more than two, the task is called multi-class classification. This project is focused on multi-class classification aiming to categorize the input tweets into three sentiment categories; 'Negative', 'Neutral', and 'Positive'. Sentiment analysis is defined as the extraction of the underlying emotions from a text which simply would be the positive or negative orientation that a writer expresses toward some object.[5]. This section involves two subsections: Preprocessing, and Machine learning classifiers.

A. Preprocessing

The workflow of a text classification begins with the preprocessing of the unstructured textual data. In our case, it means that each tweet, as the input data, must be preprocessed at the first step. The data preprocessing step is often considered an important step in machine learning tasks. Collected data usually contains a lot of noisy samples which are not conveying any informative information. It is, therefore, of crucial importance to remove out these noisy data in order to improve the signal-to-noise ratio, hence enabling the learning algorithms to recognize the informative patterns properly. Lowercasing all the tweets was the first step in this process to make the context consistent. This step ensures us that all the words are only compared based on their underlying message and not according to their style of typing. Tweet data always contain a lot of punctuations and URL links which must be removed. Dropping out these uninformative strings will reduce the length of the text, so the feature extraction may be performed more appropriately which will result in a much more efficient feature set.

Having purified the unstructured textual data, tokenization as another step of the preprocessing phase comes in. Tokenization is referred to as the task of splitting a sequence of strings into its constituents known as tokens. This process is an important step as the tokens which are the words composing a text or document will be used as the input features in any learning procedures. Removing the stop words is the next process in preprocessing. According to Wikipedia, stop words usually refers to the most common words in any language. In English the words such as 'the', 'in', 'which', 'each', 'about' are considered as the stop words which often carry little knowledge about the context in which they appear. Therefore, these words are mostly uninformative words, and to ensure the efficiency of the learning algorithm, it is necessary to be removed in the preprocessing step. By removing such frequent uninformative words we actually increase the signal to noise ratio, consequently leading to an improvement in the feature selection process.[9]

After tokenizing the words and removing the stop words lemmatization must be performed. Referring to Wikipedia, lemmatization in natural language processing is defined as the process of grouping together the inflected forms of a word so they can be analyzed as a single item, identified by the word's lemma. Lemmatization algorithms try to find the lemma of a word based on the meaning of a word in a sentence. In other words, the Lemma of a word may be defined as the root of a word considering the surrounding words and sentences in the context.

The output of the preprocessing phase is supposed to be the most important words which have been responsible for conveying the essence of the text document. In other words, those lemmatized tokens will be the input features into our machine learning algorithm. These features, however, still are in a textual format. In order to make a text document learnable by a machine learning algorithm, it first must be converted to a numeric representation since the machine learning algorithms operate on numeric input spaces. Therefore, we need to represent the obtained tokens as the numeric arrays so that it is possible to perform a machine learning procedure. This process is referred to as text vectorization and is an essential step before applying any machine learning algorithm. There are different approaches to transform a text document into a numeric vector. Below introduced some of the vectorization techniques[8]:

Count Vectorization

Count vectorization or term frequency is the simplest approach in which the vector is filled in by the number of occurrences of the corresponding term t in document d and is denoted by $tf(t, d)$.

One-hot encoding

A unique index is assigned to each token. Then each text data is represented as a vector containing 1s and 0s.

1 indicates the presence of the corresponding token, and 0 symbolizes the absence of a token in the text.

TF-IDF vectorization

The frequency vector takes only a text document into account without considering the other documents existing in the collection. Term Frequency–Inverse Document Frequency(TF-IDF) is an approach with which not only the frequency of a word in a document is considered, but also its relative frequency in that document against its frequency in other documents of the corpus will be taken into account. Considering a collection containing N documents, document frequency is referred to as the number of documents that contain a term t and is denoted as $df(t)$. The inverse document frequency is then defined as the multiplicative inverse of the document frequency and is denoted by $idf(t)$,

$$idf(t) = \log \frac{N}{df(t)} \tag{1}$$

tf–idf weight of a term t in a document d is then defined as

$$tf - idf(t,d) = tf(t,d).\log \frac{N}{df(t)} \tag{2}$$

Word Embedding

By this approach, the meaning of the word is taken into account in a way that the words that have the same meaning have a similar representation. The representing vector in this case is not a simple mapping from token position to token score. Instead, the document is represented in a feature space that has been embedded to represent word similarity.[8]

B. Machine Learning classifiers

As it was mentioned earlier, a classification is a supervised machine learning algorithm. By supervised, we mean that there exists a dataset with known values for the target variable. This dataset is called the training set and is utilized for training the system. Building an automated system starts with choosing and developing a machine learning algorithm that can be trained on the training dataset. The trained model can then be implemented as an automated system to predict the target variable values for future data. The following presents a brief description of the algorithms considered in this project.

Naive Bayes

Naive Bayes method as a probabilistic classifier is a learning algorithm based on applying Bayes' theorem. Let $(x_1, x_2, ..., x_n)$ be the features and y be the class variable,Bayes' theorem then states the following relationship:

$$P(y|x_1, x_2, ..., x_n) = \frac{P(x_1, x_2, ..., x_n|y)P(y)}{P(x_1, x_2, ..., x_n)} \tag{3}$$

Naive Bayes classifier has a "naive" assumption, and that is the conditional independence between features. It assumes every pair of features are conditionally independent given the value of the class variable. Using this assumption;

$$P(x_i|x_1, x_2, ..., x_n, y) = P(x_i|y), \quad i = 1, 2, ..., n \tag{4}$$

and hence,

$$P(y|x_1, x_2, ..., x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1, x_2, ..., x_n)} \tag{5}$$

The denominator is constant for all classes. Therefore a class label for the inputs is chosen which maximizes the numerator,

$$\hat{y} = argmax_y P(y)\prod_{i=1}^{n} P(x_i|y) \tag{6}$$

Ensemble methods Ensemble methods address a class of methods that create and combine several base estimators in order to improve the performance. In fact, instead of predicting based on a single estimator, a number of estimators are built, and then the predictions of those individual models are combined to obtain the final result. These methods are categorized into two main families: Averaging methods, and boosting methods. In averaging methods, several estimators are built independently, and then their predictions will be aggregated by taking an average. The intuition is that, on average, the combined estimator is usually better than any of the single base estimator because its variance is reduced. In boosting methods, the base estimators are no longer independent but they are built sequentially each of which tries to reduce the bias of previous estimators. In this project, Random Forest and Adaboost classifiers have been selected as the representatives of those two families.

Random Forest The base estimators in this method are the decision trees. A forest of multiple trees are built and then the predictions of those trees are aggregated as the final result. Each tree in the ensemble is built from a subset drawn from the training set. Bootstrap sampling is used for the sampling in which the samples are sampled with replacement. In addition, when splitting each node during the construction of a tree, the best split is found either from all input features or a random subset of them. These random samplings attempt to decrease the variance of the forest estimator and hence improving the performance.

Adaboost Similar to the Random Forest, Adaboost also creates a forest of trees, however, the trees are not built independently in a full-sized style. In the forest made by Adaboost, the trees are usually just a node and two leaves which is referred to as a stump. Stumps are technically weak learners as they do not perform accurately. In the sequence, each stump is built in a way that

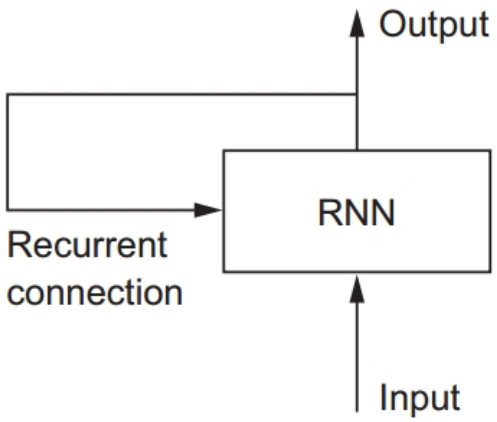

FIG. 5. A simple RNN.[3]

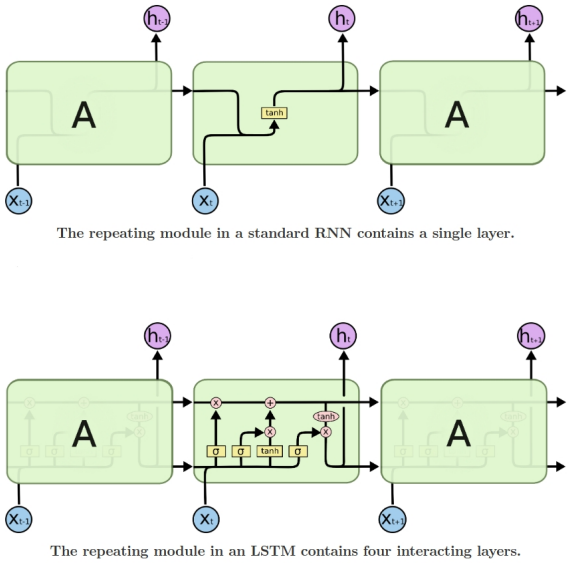

FIG. 6. A simple RNN vs. LSTM.[1]

decreases the bias introduced by the previous stump. The predictions from all of them are then combined through a weighted majority vote to produce the final prediction. Having plenty of weak learners in such a forest, we can expect that the final prediction will be accurate.

RNN The recurrent neural network is a class of deep neural network which have been effectively implemented for sequential data. In contrast to traditional networks that have no memory, RNNs process a sequence of data by not only seeing the current input data but also maintains a state containing information relative to the previous time steps. In other words, RNNs allow historical information to be kept over the process. Fig. 5 illustrates a simple RNN. The neuron is fed by some input and outputs a value. A loop allows information to be passed from one step of the network to the next.[1]

The loop in the architecture of the RNNs makes them possessing a chain-like nature in their characteristics. This capability enables the RNNs to model the sequential data dynamically since they can handle the sequential dependencies.[6] The range of the dependencies, however, matters in the sense that how far information in the past RNN needs to maintain in order to solve the problem. In cases, where the prediction only depends on the very recent information, a simple RNN as indicated in Fig. 5 can learn to use the past information.[1], however, as the range of the dependency grows, it becomes harder and harder for the simple RNN to make the connection due to the vanishing gradient problem. This problem was first addressed (Hochreiter and Schmidhuber, 1997) by introducing Long Short-Term Memory or simply LSTM.[4]

The core principle in designing LSTMs is to enable the network to remember the information for long periods of time.

Fig. 5 illustrates the differences between simple RNN and LSTM structures. Similar to simple RNN, LSTM has also a chain-like structure, but the repeating module has a different structure. The modules or memory blocks are referred to as cells. The memory blocks are responsible for remembering the information. The relevant information manipulations are carried out through the so-called gates.[2] In the figure the circles containing multiplication operation sign represent the gates. The gates actually control the input, output into/out of the cell state so that the LSTM can decide to remember or forget the information in the recurrent layer.[4] The first sigmoid layer is called the *forget gate layer* and decides on what information is allowed to pass through.

$$f_t = \sigma(W_f.[h_{t-1}, x_t] + b_f) \tag{7}$$

The next sigmoid layer called the *input gate layer* and the tanh layer together are responsible for updating the cell state. In this step, it is decided what new information is going to be stored in the cell. The sigmoid layer decides about the values that should be updated and the tanh layer creates a vector of new candidate values $\tilde{C}_t$, and in combination together they create an update to the state.

$$i_t = \sigma(W_i.[h_{t-1}, x_t] + b_i) \tag{8}$$

$$\tilde{C}_t = tanh(W_C.[h_{t-1}, x_t] + b_C) \tag{9}$$

$$C_t = f_t.C_{t-1} + i_t.\tilde{C}_t \tag{10}$$

Where i_t and $\tilde{C}_t$ are the outputs from sigmoid and tanh layers respectively. C_{t-1} is the old state cell comming in from the old state cell through the upper horizontal arrow. The old state is multiplied by f_t which a decision is taken on whether the old information can let in or not, and then the result of update step is added to that. The output would be the current cell state denoted by C_t.

The final step is actually a filter stage applied on the output. First, a sigmoid layer decides what parts of the cell state should go to output. Then, the result is multiplied by the the cell state that has been passed through the tanh layer. In this way we can only output the parts we decided to.[1]

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{11}$$

$$h_t = o_t * tanh(C_t) \tag{12}$$

Where o_t is the output from the sigmoid layer.

As we can notice, such a structure enables the system to remember old information and use it for the current processing instead of using only the current input. Such a characteristic provides the LSTM with the capability of learning the long-term dependencies. In our data set which contains the tweets, the ability to maintain the long-range dependencies makes this possible for LSTM to focus more on the important words and their relationships and so generate a more meaningful representation for the tweets. That is the reason we decided to implement this algorithm.

IV. APPROACH

Figure 7 illustrates a workflow of the steps considered in this project.

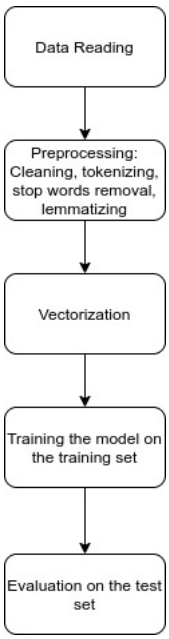

FIG. 7. The workflow of the approach considered in this project

Having imported the data, each tweet first undergoes preprocessing steps including removing the URLs, HTML tags, numbers and digits, punctuations, mentions, and hashtags. The cleaned tweets were then tokenized and the stop words were taken out. As the final step in preprocessing, lemmatization was performed to have a prepared feature set for the following steps.

After vectorization of the features, the data was then split into training and test sets. The training set was used to train the models. cross-validation through a grid search was implemented to tune the hyperparameters. Uni-grams, Bi-grams, and Tri-grams were tried as the input features. For each input set, the algorithms were applied and the best candidate was selected.

For the LSTM model, several different architectures were attempted. The embedding layer was implemented in the architecture. The best result obtained by implementing the following structure and parameters:

- Embedding layer, with the size of the vocabulary as the input dimension, and 100 assigned to output dimension.
- LSTM layer with 100 units
- GlobalMaxPool1D
- Dense layer, with 3 units, activation:softmax
- batch size: 16
- epoch: 7
- Optimizer: adam
- loss function: SparseCategoricalCrossentropy

After selecting the best estimator for each of the abovementioned models, the classifier was fit to the trainset and evaluated on the test set. The evaluation scores chosen to be reported are as follows:

Accuracy The fraction of the samples which have been correctly classified.

$$Accuracy = \frac{NumberOfCorrectlyClassified}{TotalNumberOfSamples} \tag{13}$$

Precision Precision of prediction for class c is a measure which determines the proportion of the samples classified into class c which truly belong to this category.

$$Precision = \frac{CorrectlyClassifiedToClass\ c}{TotalNumberOfPredictionsForClass\ c} \tag{14}$$

Recall Recall is also defined for a particular class. It returns the percentage of the samples belong to actual category c which have been correctly classified into class c.

$$Recall = \frac{NumberOfCorrectlyClassifiedSamples}{TotalNumberOfSamplesWithGoldStandrdClass} \tag{15}$$

F1 F1 is a harminic mean of precision and recall

$$F1 = \frac{2.precision.recall}{precision + recall} \tag{16}$$

For each of the aforementioned models, these scores were computed and reported based on which a comparison between models' performance can be made.

V. RESULTS AND DISCUSSION

This section presents the results obtained by the implementation of the above-mentioned algorithms. To select the best performance, F1 score was considered as it cater for both precision and recall. Naive Bayes was implemented as the baseline method to carry out the task under consideration. The best result obtained with the uni-grams features. Figure 8 illustrates the classification report. As we can see 67 is the best accuracy obtained by Naive Bayes.

Performance on the test data

	precision	recall	f1-score	support
Negative	0.68	0.72	0.70	2555
Neutral	0.59	0.39	0.47	1250
Positive	0.69	0.76	0.72	2939
accuracy			0.67	6744
macro avg	0.65	0.62	0.63	6744
weighted avg	0.67	0.67	0.67	6744

FIG. 8. Performance report for Naive Bayes classifier.

Ensemble methods then were applied to the data. A combination of uni-grams, bi-grams, and tri-grams was the best candidate for input features to Adaboost. Random Forest, however, worked better with the uni-grams. Figure 9 and Figure 10 show the result for RandomForest and Adaboost classifiers respectively.

Performance on the test data

	precision	recall	f1-score	support
Negative	0.81	0.74	0.78	2555
Neutral	0.66	0.81	0.73	1250
Positive	0.83	0.81	0.82	2939
accuracy			0.79	6744
macro avg	0.77	0.79	0.78	6744
weighted avg	0.79	0.79	0.79	6744

FIG. 9. Performance report for Random Forest classifier.

The main purpose of this project was to compare the four different classifiers. In terms of the performance scores, it can be inferred that ensemble methods outperform the Naive Bayes considerably. As it was mentioned, Naive Bayes has conditional independence between the features as its main assumption. This assumption could

Performance on the test data

	precision	recall	f1-score	support
Negative	0.86	0.78	0.82	2555
Neutral	0.66	0.86	0.75	1250
Positive	0.86	0.81	0.84	2939
accuracy			0.81	6744
macro avg	0.79	0.82	0.80	6744
weighted avg	0.82	0.81	0.81	6744

FIG. 10. Performance report for Adaboost classifier.

simplify the real problems unrealistically but still could be efficient in cases that data points are not sequentially dependent on each other. In sequential data, on the other hand, this assumption actually is a very naive assumption as it simply ignores the dependency between the data. The textual data consist of words and the words are the elements of the feature space. Basically, the positions of the words inside a sentence determine the final meaning of that text. Therefore, the dependency between the words matters. Ensemble methods, on the other hand, did a better job. There is no assumption of the independence of features in these methods. Instead, by drawing random samples from the data and fitting the several base estimators the best features and the best combination of them will be used in computing the final prediction.

Performance on the test data

	precision	recall	f1-score	support
Negative	0.89	0.86	0.88	2555
Neutral	0.87	0.83	0.85	1250
Positive	0.87	0.91	0.89	2939
accuracy			0.88	6744
macro avg	0.88	0.87	0.87	6744
weighted avg	0.88	0.88	0.88	6744

FIG. 11. Performance report for LSTM.

Finally, we have the LSTM model result. Figure 11 illustrates the performance report. As we can see, all the scores show a notable improvement. The accuracy increased by about 8 percent in comparison with the ensemble methods. The other scores also report an enhancement in the performance. The reason again should be looked for in the dependency subject. As it was explained in III LSTM is able to connect the current state to the old information, and so use the old outputs to be used in updating the new state. This capability ensures the system not to ignore the dependency between the features even within a long-range lag. This outstanding property has made the recurrent neural networks to be widely used for processing the sequential data.

Another noticeable result that should be taken into account is the significant improvement in scores related to the 'Neutral' class by implementing LSTM. As it was explained in IV precision is defined for a particular class

and measures the proportion of correctly classified samples among all the classified samples into that class. As we can notice, the least precision for Naive Bayes and ensemble methods were reported for the 'Neutral' class. The low number of samples per this class in the training data set could be mentioned as the reason for this decrease. Interestingly, this score improved significantly by implementing the LSTM model.

VI. CONCLUSION AND SUGGESTION

The classification task in this project was a sentiment-based classification aiming to categorize a collection of tweets data into three categories labeled with the underlying emotions expressed by the writers. Naive Bayes algorithm as the baseline which ignores any dependency between the features was the start point of our project. Ensemble methods, then, were applied. We noticed that the improvement in the performance was considerable. Finally, an LSTM model was implemented which increased the performance scores. Incremental Increasing of the performance through this pipeline seems to highlight the importance of considering the dependency in the contextual data processing. A contextual document must be considered as a sequence of words together that convey a particular subject. Ignoring the dependency between words as the features might result in ignoring informative patterns within our data.

Of crucial importance to obtain perfect results from a deep learning network is the size of the data. The more the training dataset, the better the performance will result. Therefore, it will be appropriate if the size of data could be increased for future studies. Another important issue that could be potentially be investigated is the effect of imbalanced data on machine learning systems. As we noticed in II our data is imbalanced in the sense that the values of the target variable are not distributed uniformly. The amount of documents within the "Neutral" class is considerably less than of the other two categories. The impact of this issue is suggested to be studied to see whether it may influence the models' performances.

ACKNOWLEDGMENTS

I would like to acknowledge all the efforts Marco Kuhlman and Hao Chi Kiang put into the course. New interesting topics were covered and presented. It is really appreciated.

APPENDIX

Github Repository:
https://github.com/mpirmoradiyan/TextMiningProject

[1] Understanding LSTM Networks. https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[2] Architecture Of LSTM- https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/, Dec. 2017.

[3] Chollet, F. *Deep Learning with Python*. Manning Publications Co, Shelter Island, New York, 2018.

[4] Huang, M., Cao, Y., and Dong, C. Modeling Rich Contexts for Sentiment Classification with LSTM. *arXiv:1605.01478 [cs]* (May 2016).

[5] Jurafsky, D., and Martin, J. H. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall, 2009.

[6] Liu, G., and Guo, J. Bidirectional LSTM with attention mechanism and convolutional layer for text classification. *Neurocomputing 337* (Apr. 2019), 325–338.

[7] Mueller, A. Amueller/word_cloud- https://github.com/amueller/word_cloud, Dec. 2020.

[8] Rebecca Bilbro, Tony Ojeda, B. B. Applied Text Analysis with Python [Book]. https://www.oreilly.com/library/view/applied-text-analysis/9781491963036/.

[9] Sarica, S., and Luo, J. Stopwords in Technical Language Processing. *arXiv:2006.02633 [cs]* (June 2020).

[10] Tripathy, A., Agrawal, A., and Rath, S. K. Classification of Sentimental Reviews Using Machine Learning Techniques. *Procedia Computer Science 57* (Jan. 2015), 821–829.