Multi-class Sentiment Analysis on Movie Reviews: Comparative Study of Various Machine Learning & Deep Learning Approaches

Maggie Chu
New York University
Computer Science Department
New York, New York
hcc375@nyu.edu

Abstract

Sentiment analysis (or opinion mining) refers to the use of natural language processing technique to automatically extract, collect, aggregate and classify data into different emotion classes. While much attention has been paid to binary classification of reviews into either positive or negative, less attention has been paid to multi-class classification sentiment analysis. Given the added complexity of multi-labels and difficulty of quantifying human emotions, this proves to be a challenging yet interesting task. This paper discusses the multi-class classification of Rotten Tomatoes' Movie Review dataset (hosted on Kaggle) and will show the different approaches of solving it. We have developed 5 models for approaching this: (1) TF-IDF + Bi-Gram + Multinomial Naïve Bayes, (2) TF-IDF + Bi-Gram + Linear SVC, (3) TF-IDF + Bi-Gram + Logistic Regression, (4) TF-IDF + Bi-Gram + Random Forest, and (5) LSTM (Long Short Term Memory). The aim of this paper is to experiment with these approaches and evaluate their efficacy and impact of feature extraction techniques such as TF-IDF and Bi-Grams have on the performance. The proposed methods above are evaluated based on their accuracy, precision, recall and F-1 measure. Our results suggest that deep learning method such as LSTM outperforms tradition ML techniques even with the inclusion of feature extraction. Out of the ML models above, Linear SVC with TF-IDF and Bi-Gram encodings performs the best in terms of accuracy. However, without feature extraction, Logistic Regression performs the best (excluding LSTM).

KEYWORDS

Machine Learning, Sentiment Analysis, Multi-class, Classification, Movie Reviews

1 Introduction

In the past few years, we have seen the rise and proliferation of Web 2.0 sites and applications such as Twitter, blogs, and online movie reviews, which has further fueled people's interest in analyzing the emotions presented on these platforms, or simply, "sentiment analysis." While sentiment refers to the meaning of words generally associated with an opinion, analysis refers to the process of analyzing data and predicting the result to be either positive or negative. For example, companies selling their products on Amazon will benefit from better understanding their customers' overall opinion on their products. Movie goers and production companies can better understand the overall sentiment towards a movie through online movie reviews. While providing a numerical representation on a movie does quantitively convey the success or failure of a movie, it fails to qualitatively convey the reviewers' overall sentiment like a collection of text movie reviews can.

In recent years, machine learning and deep learning have been applied to sentiment analysis and generally outperform human-produced baselines. We should note, however; that performance of different machine learning models for text classification vary depending on the model variant, feature extraction used, and the dataset (Wang 2012). Common feature extraction methods

include Bag of Words, N-Grams Modeling and TF-IDF (term frequency-inverse document frequency) In this paper, we primarily focus on the performance of different machine learning and deep learning techniques related to the variant of Rotten Tomatoes' Dataset, and TF-IDF and Bi-Gram encodings. Through these experiments, we attempt to show the best ML classification model when paired with TFIDF and Bi-Gram feature extraction. We also attempt to find out if LSTM can outperform traditional ML classifiers..

2 Motivations & Related Works

2.1 Motivations

Realistically speaking, however; sentiment is rarely classified into either positive or negative. For instance, let us take a look at these two movie reviews:

- "This movie is awful."
- "This remake is such a nightmare. Not only do I want a refund, I also want to yell at the producers for turning what is supposed to be a musical masterpiece into a horror film."

We can see clearly that these two movie reviews are both negative. These 2 negative sentiments, however; are expressed in different levels as opposed to one singular level. While the first review suggests a negative attitude towards the film, the second review suggests a strong hatred and disgust towards the movie. While most state-of-the-art approaches of sentiment analysis focus on binary classification (either positive or negative), it would be more interesting to go deeper than that in order to better understand the sentiment behind a review (Bouazizi and Ohtsuki). It would probably be a good idea for the production company to distinguish between these 2 negative reviews in order to gain better insights into their audience.

2.2 Related Works

The field of sentiment analysis is a widely researched field, and therefore, has many ways of implementing it. For instance, sentiment analysis systems can be implemented at different levels such as the word level, attribute level, concept level, sentence level, and the document level. In general, however; sentiment analysis can be categorized into two fields: the (1) dictionary based approach (or lexicon-based approach) that uses predefined set of sentiment dictionaries to identify the sentiments in the text, and (2) the machine learning based approach that work by constructing and training a classifier on a manually annotated corpus in order to identify the sentiments (Jain 2017). In recent years, with the advancements in machine learning, machine-learning based approaches on sentiment analysis have become more popular. Despite their popularity, machine-learning based approaches suffer from disadvantages such as requiring large human effort on manually annotating a corpus for training a classification model. There are many state-of-the-art machine-learning based approaches on sentiment analysis:

(1) Multinomial Naïve Bayes (NB)
(2) Support Vector Machine (SVM)

These two models are perhaps the most used baseline models for text classification and sentiment analysis (Wang 2012). Wang also noted that Multinomial Naïve Bayes (MNB) perform better with snippets while Support Vector Machine (SVM) perform better with full-length reviews. We would also be training these two models in our experiment.

3 Data Exploration & Analysis

3.1 Exploration

The Rotten Tomatoes Movie Review corpus provided by Kaggle is comprised of tab-separated files with phrases. The provided train and test set are used for benchmarking with the sentences shuffled from original order. Each sentence was parsed into many phrases by the Stanford parser. Each phrase has a PhraseID. Each sentence has a SentenceID. Repeated sentences are only included once in the provided dataset.

If we look at the train set, it has 4 columns and 156060 rows of data with PhraseID, SentenceID, Phrase, and Sentiment.

(1) PhraseID is a unique numerical identifier for phrases and we have 156060 PhraseID in this train set.
(2) SentenceID is a unique numerical identifier for sentence.
(3) Phrase is referenced by SentenceID and is of type "string." Phrase stems from Sentence.

(4) Sentiment is the numerical labels ranging from 0 to 4 and represent the target prediction:

- 0 – negative
- 1 – somewhat negative
- 2 – neutral
- 3 – somewhat positive
- 4 – positive

Below is a figure that shows what the train set looks like

	PhraseId	SentenceId	Phrase	Sentiment
0	1	1	A series of escapades demonstrating the adage ...	1
1	2	1	A series of escapades demonstrating the adage ...	2
2	3	1	A series	2
3	4	1	A	2
4	5	1	series	2

Figure 1: Training Set Preview

Below is a graphical representation of the train set that consists of 156059 phrases with their sentiment distribution values:

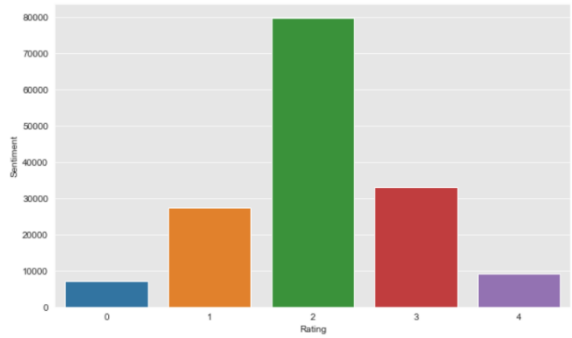

Figure 2: Distribution of Sentiments

As we can see, the train set is unbalanced with majority of phrases having a sentiment value of 2 (neutral sentiment). This means that it is likely that our classifiers may misclassify.

3.2 Data Analysis

When evaluating different machine learning algorithms, we need to keep in mind that the same training set should not be used as this may lead to overfitting the training data and will therefore be unable to predict accurately. To avoid biasing our systems, we can simply split the dataset into train set and test set with a 60/40 split. However, there is also a high possibility of overfitting since we can always adjust the parameters until our expected algorithm performs well. If this happens, our model will become biased and cannot provide accurate information. As a result, we will be splitting the provided data set into 3 sets: training set, development set, and test set. The general workflow for this paper is to train on the train set, evaluate on the dev set, and final test on the test set. After splitting the dataset, the train set has 117045 entries, the test set has 19508 entries, and the dev set has 19507 entries.

4 Pre-Processing

This is the first step in our sentiment analysis experiment. Basic methods include removing non-alphabetic characters, empty strings, lower case conversion, stemming, removing numbers, etc. The provided dataset for this paper does not require extensive pre-processing as it is already cleaned. For the Rotten Tomatoes Corpus, we will be removing stopwords (most common words in English that don't have any contribution to sentiment analysis). For example, we would be removing words such as "are," "the," etc. from our system.

5 Feature Extraction

Feature extraction, simply said, refers to the process of transforming word representations to numerical values that a model can interpret. There are many ways to do this, but for the purpose of this experiment, we will be focusing on TF-IDF and N-Gram Modeling (Bi-Gram).

5.1 TF-IDF

TF-IDF, which stands for Term Frequency – Inverse Document Frequency is a method for evaluating the importance of a word in a single document. We use TF-IDF for this experiment in order to develop feature vector for each phrase.

$$TF - IDF = Term\ Frequency(TF) * Inverse\ Document\ Frequency\ (IDF)$$

Term Frequency (TF) refers to how often words appear in individual documents. It is described here:

$$TF = \frac{number\ of\ times\ a\ term\ appears\ in\ a\ document}{total\ number\ of\ words\ in\ a\ document}$$

Inverse Document Frequency (IDF) is used to diminish the weight of terms (i.e the) that occur

very frequently but are not relevant to the document. It is described here:

$$IDF = \ln\left(\frac{total\ number\ of\ documents}{number\ of\ documents\ with\ term\ in\ them}\right)$$

The final TF-IDF is simply the product of TF and IDF. We will be using TF-IDF for classification algorithms such as Multinomial Naïve Bayes, Linear SVC, Logistic Regression and Random Forest.

5.2 N-Gram Modeling: Bi-Gram

An n-gram is a set of n consecutive words that we can use as building blocks for our classification models and is often useful for dealing with negations. For instance, given the sentence "We really enjoyed this movie and its cinematography," the n-grams will be:

- Unigram: [we],[really],[enjoyed]…[cinematography]
- Bigram: [_,we],[really,enyjoed],[this,movie]…[_,cinematography]
- Trigrams: [_,_,we,][really,enjoyed,this],[movie,and,its][cinematography,_,_]

For this paper, we will be using bi-gram as it was proven that inclusion of bi-gram feature extraction can lead to consistent gains on sentiment analysis tasks (Wang 2012). So far, we have seen a slight increase in the accuracy of our ML classification models, which is detailed more in the Experiments section.

6 Classification Algorithms

The goal of classification is to automatically assign label to an unlabeled example. The classifiers are trained on the train set with features obtained using TF-IDF and N-Gram modeling in order to predict the sentiments and their accuracy are measured against the dev and test set. We experiment with different machine learning and deep learning models and examine which one performs the best (in terms of accuracy score). As a result, we benchmark our Feature Extraction to be TF-IDF and Bi-Grams for all ML models with LSTM being the only algorithm without TF-IDF and Bi-Gram feature extraction. We also experimented with pure ML classifiers without feature extraction to see whether feature extraction could significantly improve the performance of our ML classifiers. We experimented with standard algorithms such as Multinomial Naïve Bayes classification, Linear SVC, Logistic Regression and Random Forest. We also experimented with LSTM to see whether or not deep learning method could outperform traditional machine learning models.

6.1 Multinomial Naïve Bayes

Naïve Bayes is a probabilistic machine learning classifier with a "naïve" assumption of conditional independence between every pair of features. Naïve Bayes is a classification based on Bayes Theorem. In laymen terms, one feature is not related to another feature. This is a commonly used baseline algorithm for classification work for its simplicity and speed on large datasets (Jain 2017).

Bayes' Theorem:

$$\Pr(X = x \mid = y) = \frac{\Pr(Y = y|X = x)\ \Pr(X = x)}{\Pr(Y = y)}$$

Pr(X = x | Y = x) is the conditional probability where the probability of random variable X to have a specific value of x given that another random variable Y has a specific value of x (Burkov 2019).

Multinomial Naïve Bayes (MNB)

For this paper, we are using Multinomial Naïve Bayes model, which implements the naïve Bayes algorithm since we are dealing with multinomially distributed data. Each textual data *d* is considered as a bag of tokens while each entry in ti representing occurrence of a token or TF-IDF value. The data *d* can be shown as vector x where each xi in the x vector entry will show the weight of ti occurred in d. Additionally, each *d* is considered to be the outcome of selecting individually |d| tokens from *F* with replacmenet where each ti has probability p(ti|c).

$$p(\vec{x}|c) = p(|d|) \cdot |d|! \cdot \prod_{i=1}^{m} \frac{p(t_i|c)^{x_i}}{x_i!}$$

Figure 3: multinomial distribution (Jain 2017)

This method has shown significant improvement when combined with combination of unigram, bigram and trigram (Jain 2017). In addition, Wang and Manning also found that Multinomial NB is stronger for snippets than for longer documents (Wang 2017). For this paper, we are using scikit-learn's MulinomialMB. Our experiment will therefore test and see if our results match what they proposed.

6.2 Linear SVC

Support Vector Machines (SVM) are set of supervised learning methods commonly used as baseline for classification. SVM have also been shown to be effective at traditional text classification and outperform Naïve Bayes generally (qtd. Joachinms1998 in Pang 2002). For binary classification, SVM requires a positive label with a numeric value of +1, and the negative label of -1. The decision boundary is chosen with the one that maximizes the distance to the training samples. The model f(x) is defined as such:

$$f(x) = sign(w^*x - b^*)$$

Where w^* is the optimal value and b^* for parameters w and b (Burkov 2019).

Since SVM cannot be naturally extended to solve multi-class problems, we use the scikit-learn library and utilize the kernels provided by SVM and extend it to a Linear SVC ("one-versus-rest) to handle multi-class classification. The idea is that we will be transforming a multi-class problem into *C* binary classification problems and build C binary classifiers. In the case of Rotten Tomatoes' dataset, we have 5 classes,
$y \in \{1,2,3,4,5\}$, so, we will create copies of the original datasets and modify them. For instance, we will replace all labels not equal to 1 by 0 for the first copy. For the second label, we will replace all labels not equal to 2 by 0 and so on so forth until we have five binary classification problems.
Once we have five models, we can get the prediction by applying the five models to the input to classify the new input feature vector x. Below is the certainty of prediction of distance *d* from input x to the decision boundary:

$$\mathrm{d} = \frac{(w^*x+b^*)}{||w||}$$

The larger the distance, the more certain the prediction is (Burkov 2019).

Linear SVC is suitable for many text classification problems due to its versatility and robustness in high dimensional space (Jain 2017). In addition, Wang and Manning also found that SVM is much better at full-length sentiment analysis tasks (Wang 2012). Our experiment will therefore test and see if our results match with what they proposed. For this experiment, we expect the performance of Linear SVC to be better than Multinomial NB.

6.3 TF-IDF +Bi-Gram + Logistic Regression

Logistic Regression (LR) despite its name, is actually a classification learning algorithm as opposed to a regression. It is also known as maximum-entropy classification (MaxEnt) in some literature. While logistic regression is used for binary classification, it can be naturally extended to multi-class classification (Burkov 2019). In logistic regression, we assume the labels to be binary:

$$y^i \in \{0,1\}$$

The logistic regression model looks like this:

$$fw, b(x) = \frac{1}{e^{-(wx+b)}}$$

As opposed to linear regression where we simply model yi as a linear function of xi to predict the value yi for the ith training sample, logistic regression predicts the probability by defining negative label as 0 and positive label as 1 since modeling with binary yi is not straightforward. As a result, logistic regression would just need to find a continuous function whose codomain is (0,1).

To extend our logistic regression into multi-class learning problem, we would replace the sigmoid function with a softmax function. This new multi-class logistic regression is also known as softmax regression or multinomial logistic regression. A softmax function is a generalization of the sigmoid function into multidimensional outputs as softmax simply produces a vector that represents the probability distributions of a list of potential results (Burkov 2019). We will be using sickit-learn's Logistic Regression classifier and will set that mult_class scheme to "multinomial" for cross-entropy loss.

6.4 Random Forest

Random Forest, as the name suggests, creates a forest with a number of decision tress and can be

used for both classification and regression. Random Forest builds on decision trees, which are acyclic graphs used to make decisions. A decision tree is a cyclic graph where each internal node (j) denotes a test on an attribute, each branch represents the outcome of the test, and each terminal node (or leaf) holds a class label. As the leaf node is reached, the decision on which class the example belongs to will be made.

Random forest, on the other hand, grows multiple trees as opposed to a single tree in the classification model. To classify a new object based on attribute, each tree gives a classification and the tree folds for that class. The forest will then choose the tree that has the most votes in the forest. Generally, in Random Forest algorithm, the more trees we have, the better the performance and efficiency of the model. The mathematical expression is given here:

$$y^* = \frac{1}{m}\sum_{j=1}^{m}\sum_{i=1}^{n} W_j(x_i, x')y_i$$
$$= \sum_{i=1}^{n}\left(\frac{1}{m}\sum_{j=1}^{m} W_j(x_i, x')\right)y_i$$

Figure 4: Mathematical Equation for Random Forest (Ayata 2017).

$\{(x_i, y_i)\}_{i=1}^{n}$: *training set,*
$y*$: *predictions,*
x' : *new points to classify,*
$W(x_i, x')y_i$: *weight of the i'th function,*
W : *weight function,*

Figure 5: Random Forest Equation (Ayata 2017).

Compared to decision trees, Random Forest classifier has the power to hold larger datasets with higher dimensionality. However, it should also be noted that while Random Forest does a good job on classification problems, it performs better on regression problems since the predictions are random and non-contiguous. As a result, there is a high possibility that we will be overfitting the model since the bias of a forest usually decrease as the training set increases, which is the case for this experiment.

6.5 LSTM

LSTM stands for Long Term Short Memory and it is an extension (or modified version) of Recurrent Neural Nets (RNN) that address RNN's disadvantage of not being able to remember past data. RNN uses its internal state (memory) to process sequences of inputs and is often used to classify, label, or generate sequences. It is also suitable for text processing since texts are naturally sequences of words or characters and can be easily represented with vectors (Burkov 2019). However, RNN presents a disadvantage of vanishing gradient [1]problem where a gradient exponentially shrinks as it back-propagates through time. If the gradient value becomes too small, it will not contribute to the network layer's (usually early layers) learning. To put it simply, RNN suffers from short-term memory where it forgets what the earlier layers in longer sequences.

LSTM was first introduced in 1997 by Hochreiter & Schmidhuber to address RNN's short-term memory problem (Ayata 2017). It uses the same function and architecture as RNN but is capable of learning long-term dependencies using the "gates [2]" mechanism. In LSTM, for each sequence, we have a cell state [3] and three gates within: input gate, output gate, and forget gate. Gates contain sigmoid activation [4] that squishes values between 0 and 1. This makes it helpful for gates to forget and update data since any number multiplied by 0 will be forgotten while numbers multiplied by 1 will be kept.

We choose to experiment with LSTM classifier since most state-of-the-art results based on neural networks are achieved with either LSTM or GRU (Gated Recurrent Units) and are often used with speech recognition models or speech synthesis (Ayata 2017). In addition, the top sentiment analysis performers for this Kaggle all use LSTM or other neural network to some degree.

[1] Gradient is the value the neural network uses to update the neural network's weight.

[2] Gates are different tensor operation capable of learning what information to add or remove from the hidden state.

[3] Cell states (memory of LSTM) transfer relative information down the sequence of the LSTM chain.

[4] RNN uses tanh activation to squish values between -1 and 1.

7 Experiments & Results

7.1 Experiment Setup & Results

We began our experiment by pre-processing the data, and splitting the dataset into training (117045), development (19508) and test set (19507) with a 75%, 15%, 15% split. We also set the baseline for our system to be to achieve at least 60% accuracy score. This score is derived from looking at Kaggle competition's leaderboard of 861 entries and taking the 430th as baseline.

To assess the performance of our classification models, we defined our evaluation metrics to be accuracy, precision, recall, and F-1 measure. We should also note that Kaggle specifies accuracy as the baseline metric for this competition.

- **Accuracy**: It is defined as the overall correctness of classification. It is calculated by dividing the number of correctly classified examples by the total number of classified examples. This is the most relevant metric for this paper since we also consider errors in our predictions as equally important as correctness.

$$\boldsymbol{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$ [5]

- **Precision**: It is the ratio of correctly classified predictions over the total number of predictions that are classified as belonging to that sentiment. For instance, we will want a high precision if we want to avoid making mistakes by falsely identifying a positive sentiment as negative.

$$\boldsymbol{Precision} = \frac{TP}{TP + FP}$$

- **Recall**: It is the ratio of correctly classified predictions over the total number of predictions that actually belong to that sentiment. For instance, we will want a high recall if we can tolerate some negatives.

$$\boldsymbol{Recall} = \frac{TP}{TP + FN}$$

- **F-1 measure:** It is used to convey the balance between precision and recall.

$$\boldsymbol{F-1} = 2 * \frac{precision * recall}{precision + recall}$$

To observe the impact inclusion of feature extractions such as TF-IDF and Bi-Grams has on the accuracy, precision, recall, and F-1 measure of our models, we conducted our experiment in 3 ways:

(1) Training ML classification models without feature extraction.
(2) Training ML classification models with TF-IDF and Bi-Gram feature extraction.

In addition, we trained a LSTM classifier at the end of our experiment to observe whether or not deep learning classification model outperforms traditional machine learning classification models
.

(3) Training LSTM classification model without feature extraction.

Our LSTM classification model is trained with the help of pytorch as follow: 1 embedding layer, 3 layers of LSTM, stack of 2 dense layers, and optimized with adam (adaptive moment estimation) optimizer. [6] Our loss function is computed with pytorch's CrossEntropyLoss() as this is useful when dealing with multi-class classification problems and with an unbalanced training set. We also set the batch size[7] for training set to be 64, for dev to be the length of itself, and for test to be the length of itself. We used the dev set to tune our LSTM and arrived at 5 epoch[8].

7.2 Experiment Results

(1) ML Classification Models with no Feature Extraction:

[5] True positive (TP) means when a positive prediction is predicted to be positive. False positive (FP) means when a negative prediction is predicted to be positive.
[6] Adam's method is a method of Stochastic Optimization
[7] Batch size is defined as the total number of training examples in a single batch. We want to divide datasets into number of batches since we cannot pass the entire dataset into the neural network at once.
[8] One epoch is the equivalent of passing the entire dataset forward and backward through the neural network.

ML Models:	**Training Time**
Multinomial NB	6.1446
Linear SVC	65.8077
Logistic Regression	17.0487
Random Forest	2654.1420

Table 1: ML Model Prediction without Feature Extraction Training Time

ML Models (Train)	Accuracy	Recall	Precision	F-1 Measure
Multinomial NB	0.6735	0.6735	0.6633	0.6626
Linear SVC	0.7270	0.7270	0.7225	0.7159
LR	0.6945	0.6945	0.6931	0.6741
Random Forest	0.8500	0.8500	0.8487	0.8489

Table 2: ML Model Prediction without Feature Extraction on Train Set.

ML Models (Dev)	Accuracy	Recall	Precision	F-1 Measure
Multinomial NB	0.6091	0.6091	0.5941	0.5973
Linear SVC	0.6289	0.6289	0.6113	0.6118
LR	0.6305	0.6305	0.6122	0.6018
Random Forest	0.6321	0.6321	0.6231	0.6263

Table 3: ML Model Prediction without Feature Extraction on Dev Set.

ML Models (Test)	*Accuracy*	*Recall*	*Precision*	*F-1 Measure*
Multinomial NB	*0.6052*	*0.6052*	*0.5892*	*0.5925*
Linear SVC	*0.6284*	*0.6284*	*0.6104*	*0.6110*
LR	*0.6295*	*0.6295*	*0.6116*	*0.6006*
Random Forest	*0.6297*	*0.6297*	*0.6203*	*0.6237*

Table 4: ML Model Prediction without Feature Extraction on Test Set.

(2) ***ML Classification Models with TF-IDF & Bi-Gram Feature Extractions***

ML Models:	***Training Time***
Multinomial NB	*9.3186*
Linear SVC	*15.0664*
Logistic Regression	*14.1388*
Random Forest	*944.7245*

Table 5: ML Model Prediction with TF-IDF + Bi-Gram Train Time

ML Models (Train)	Accuracy	Recall	Precision	F-1 Measure
Multinomial NB	0.6941	0.6941	0.7190	0.6544
Linear SVC	0.8394	0.8394	0.8379	0.8370
LR	0.7026	0.7026	0.7165	0.6754
Random Forest	0.8514	0.8514	0.8501	0.8503

Table 6: ML Model Prediction with TF-IDF + Bi-Gram on Train Set.

ML Models (Dev)	Accuracy	Recall	Precision	F-1 Measure
Multinomial NB	0.6109	0.6109	0.6072	0.5614
Linear SVC	0.6472	0.6472	0.6371	0.6401
LR	0.6223	0.6223	0.6139	0.5829
Random Forest	0.6353	0.6353	0.6249	0.6282

Table 7: ML Model Prediction with TF-IDF + Bi-Gram on Dev Set

ML Models (Test)	Accuracy	Recall	Precision	F-1 Measure
Multinomial NB	0.6103	0.6103	0.6077	0.5595
Linear SVC	0.6460	0.6460	0.6358	0.6388
LR	0.6174	0.6174	0.6086	0.5762
Random Forest	0.6327	0.6327	0.6217	0.6252

Table 8: ML Model Prediction with TF-IDF + Bi-Gram on Test Set.

(3) LSTM

Model	Training Time	Accuracy	Recall
LSTM (Train)	3561.1926	0.7447	0.7447
LSTM (Test)	N/A	0.6516	0.6516
Model	Precision	F-1 Measure	
LSTM (Train)	0.7439	0.7429	
LSTM (Test)	0.6486	0.6487	

Table 9: LSTM Model Prediction on Train and Test Set

[9] This significant increase in training time is due to the increase in max_iter for Linear SVC to perform properly

8 Conclusion, Analysis & Future Work

8.1 Conclusion & Analysis

From the results above, we can say that out of all ML and deep learning classification models, LSTM achieved the best overall result with 65.16% accuracy at the cost of long computational time of 3561.1926 seconds. This placed us at the top 15% of Kaggle competition's leadership board. If we only consider Machine Learning classifiers (Multinomial Naïve Bayes, Linear SVC, Logistic Regression, and Random Forest), Linear SVC achieved the best overall result with 64.60% accuracy with a relatively fast training time of 15.0664 seconds.

Classification Model	Accuracy	Training Time (seconds)
LSTM	65.16%	3561.1926
Linear SVC	64.60%	15.0664
Random Forest	63.27%	944.7245
Logistic Regression	61.74%	14.1388
Multinomial NB	61.03%	9.3186

Table 10: Classification Models (ML with TF-IDF and Bi-Gram ranked by accuracy)

Classification Model	Accuracy	Training Time (seconds)
LSTM	65.16%	3561.1926
Logistic Regression	62.95%	17.0487
Random Forest	62.97%	2654.1420
Linear SVC	62.84%	65.8077[9]
Multinomial NB	60.05%	6.1446

Table 11: Classification Models (ML without TF-IDF and Bi-Gram ranked by accuracy)

We also observed that, without feature extraction such as TF-IDF and Bi-Grams, the accuracy score of our ML classifiers dropped slightly except for Logistic Regression where feature extraction actually hurt its accuracy score. With the inclusion of feature extraction, we improved the overall accuracy of our ML classifiers with Linear SVC having the most

impact as it went from the 4th best classifier to the 2nd best classifier. In addition, we also observed that our Random Forest classifier was overfitting (or high variance) as it predicts very well on the training data but significantly poorer from the dev set. As mentioned above, this is most likely caused by the complexity of our classifier.

8.2 Future Work

In this paper, we have applied 5 different algorithms on the multi-class classification Rotten Tomatoes corpus with and without Feature Extraction such as TF-IDF and Bi-Gram. We concluded that Feature Extraction would slightly improve the overall accuracy score of most ML classifiers. In the future, we could avoid overfitting in Random Forest classifier by tuning the hyper-parameters. For instance, we could decrease the number of samples in the leaf or to perform regularization on the model. While LSTM was the classifier with the best overall performance, it took a significantly longer time to train. For future work, we could also include feature extraction such as Word2Vec and GloVe for LSTM to see if better accuracy could be achieved.

Acknowledgments

We thank Professor Adam Meyers for his assistance with providing feedback that greatly improves this paper.

References

Ayata, A. (2017). BUSEM at SemEval-2017 Task 4A Sentiment Analysis with Word Embedding and Long Short Term Memory RNN Approaches. In Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017) (pp. 777–783). Association for Computational Linguistics.

Bouazizi, M., & Ohtsuki, T. (Accepted/In press). A Pattern-Based Approach for Multi-Class Sentiment Analysis in Twitter. *IEEE Access*. https://doi.org/10.1109/ACCESS.2017.2740982

Burkov, A. (2019). *The hundred-page machine learning book*. S.l.: Andriy Burkov.

Jain, U. (2017). Sentiment Analysis: An Empirical Comparative Study of Various Machine Learning Approaches. In Proceedings of the 14th International Conference on Natural Language Processing (ICON-2017) (pp. 112–121). NLP Association of India.

Pang, S. (2002). Thumbs up? Sentiment Classification using Machine Learning Techniques. In Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002) (pp. 79–86). Association for Computational Linguistics.

Sida Wang and Christopher D Manning. 2012. Baselines and bigrams: Simple, good sentiment and topic classification. In Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2. Association for Computational Linguistics, pages 90–94.