Sentimental Analysis of IMDb Movie Reviews

Haoming Chen hchen549@wisc.edu
Ruiting Tong rtong5@wisc.edu
Dingyi Li dli283@wisc.edu

Abstract

The objective of this project is to predict sentimental representation from the review of movies listed on IMDb via several classification machine learning algorithms. We collected 50,000 movie reviews and their sentiment scores (0 and 1) and processed them for analysis. The data are divided into 70% of training set and 30% of test set. Algorithms of SVM, KNN, Decision Tree, Logistic Regression, and Random Forest are trained and tested for accuracy. In particular, we used both the RBF and the linear kernels for SVM, and we applied bagging to Logistic Regression to obtain and verify an increased accuracy. As a result, we have found that the Logistic Regression is the best option for this task among others.

1. Introduction

With the Internet and modern technology, people are able to produce tons of text data every single day. We can find such data in emails, comments on articles, reviews on products, and social media, and there is a lot of information contained within such text data. When people write things down, they are motivated by certain emotions or attitudes. If they were happy with a product or an experience, they would compliment it and recommend it to others. If they had a terrible experience, they would criticize it and caution others. If we are able to organize and analyze the text data in a way that could accurately reflect users' emotion or attitude, which is referred to as sentimental analysis, relevant companies could modify the service for each individual accordingly.

Internet Movie Database (IMDb) is one of the most popular online databases for movies where millions of users read and write movie reviews. The users' attitude toward one movie could be inferred by looking at the reviews they made. The objective of this project is to predict the sentimental representation from the 50,000 review of movies listed on IMDb[1] via several classification machine learning models. In this project, we consider the simplest case where the sentiment is represented by a binary indicator: 1 meaning positive and 0 meaning negative. Despite such a simplification, it would be impossible for humans to read through millions of comments and reviews made by all movie watchers and label them each with a sentiment score. Therefore, it is necessary that we train and implement a machine learning model to accomplish this task in our place.

Solving this classification problem is the first step to uncover the underlying habits of different users. One of the beneficiaries of such analysis would be streaming sites such as Netflix that need such data and methods to find out the movie types that particularly interest each user, and make accurate recommendations. On top of that, they can filter some junk ratings that are not conducive to helping others correctly understand and evaluate a movie, since they can be purely emotional and not based on reasons.

To find out the most suitable classification algorithm for this task, we experiment with SVM (RBF and linear kernels), KNN, Decision Tree, Random Forest, and Logistic Regression (with and without bagging). First of all, we extract 50,000 movie reviews and their corresponding 0-1 labels, and clean up the data by *Natural Language Toolkit* (NLTK) package of Python. Then, we use 70% of the data for training and the rest for testing. All the aforementioned algorithms are trained using the features selected in the first step. Lastly, test accuracy is computed for each algorithm, and further analyses such as McNemar's test and ROC comparisons are implemented to rank the algorithms.

2. Related work

There have been some previous studies on sentimental analysis. Back in 2004,Pang and Lee proposed a method to combine the traditional bag-of-words model with a minimum-cut framework to retain the polarity information[2]. Their approach will not only reduce computational complexity by reducing the length of texts but also achieve significantly better results compared to the model trained on the original full dataset.

Wilson et al's work presented an approach to phrase-level sentiment analysis, which determines whether an expression is neutral or polar at first and then disambiguates the polarity of the polar expressions (Wilson, Wiebe, & Hoffmann, 2005)[3]. By considering prior polarity and contextual priority, they are able to identify negation and intensification on a phrase level and achieve much better results

compared to the baseline model.

The recent studies have been focusing on extracting information on social media. Gautam, G. and Yadav, D conducted the sentiment analysis on the Twitter data and focused specifically on the extraction of adjectives within each document[4]. They then progressed to several machine learning algorithms, including Naive Bayes, Support Vector Machine, and Maximum Entropy, followed by semantic analysis. The model performance was measured using precision, recall, and accuracy.

However, the above researches focus more on extracting accurate and compact information from the raw data and limited to the use of a single machine learning model to improve prediction accuracy. Instead, this report will put a great emphasis on examining the power of ensembling methods in the case of natural language processing. We will extend on the previous machine learning models and combine them into a new model with different techniques.

3. Proposed Methods

3.1. Support Vector Machine

Support Vector Machine (SVM) is a very commonly used algorithm for binary classification. The motivation behind it is relatively easy to understand. Basically, in an n-dimensional feature space, we want to find a (n-1)-dimensional hyperplane that separates the datapoints of two different classes, assuming that they are indeed separable by a line. Suppose that the hyperplane is

$$\boldsymbol{\omega}^\top \boldsymbol{x} + b = 0.$$

and that

$$\boldsymbol{\omega}^\top \boldsymbol{x} + b = \pm 1 \tag{1}$$

happen to be boundaries of the two classes that touch on at least one sample from each class respectively. Our goal is then to maximize the distance between the two hyperplanes in 1. Such a distance is given by

$$\frac{2}{||\boldsymbol{\omega}||},$$

which can be interpreted as the distance between the two classes. The optimal hyperplane obtained from solving this maximization problem is used for prediction of new data.

So far, the idea of SVM has been introduced for the linear kernel. In fact, we can replace $\boldsymbol{x}$ in the constraint to an arbitrary function $\phi(\boldsymbol{x})$. Then the separating surface may no longer be a hyperplane. For example, a nonlinear kernel $\kappa(\boldsymbol{x}, \boldsymbol{y}) = \phi(\boldsymbol{x})^\top \phi(\boldsymbol{y})$ that we have also employed is the Radial Basis Function (RBF), which is given by

$$\kappa(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(-\frac{||\boldsymbol{x} - \boldsymbol{y}||^2}{2\sigma^2}\right),$$

where $||\cdot||$ denotes the Euclidean distance and σ represents the bandwidth.

L_1 regularization is applied to SVM with linear kernel, and the coefficient c is tuned.

3.2. k Nearest Neighbors

k Nearest Neighbors (KNN) is a classification algorithm whose training step only involves remembering the data. Suppose the size of the training set is N. In the fitting step, for each new sample with features $\boldsymbol{x}^*$, the algorithm computes

$$d_i = ||\boldsymbol{x}^* - \boldsymbol{x}_{\text{train},i}|| \text{ for } i = 1, 2, ..., N,$$

where $\boldsymbol{x}_{\text{train},i}$ denotes the features of the i^{th} training sample, N is the size of the training sample, and $||\cdot||$ is any distance function. Then we pick the k indices from $\{1, 2, ..., N\}$ that are associated with the k smallest distances. The predicted label for $\boldsymbol{x}^*$ is the majority vote of the labels corresponding to these indices.

For our problem, we only consider the Euclidean distance, and k is the only parameter we need to tune for. Considering that the optimal k could be very large given a very high-dimensional data set, we search for such k manually by narrowing down the search range.

3.3. Decision Tree

A decision tree predicts the label of a new sample by searching for the leaf node it belongs to. A tree can be either binary or non-binary, the information criterion for splitting a node can be "Gini", "Entropy", and so on, and the maximum depth is yet another hyperparameter that is set to prevent overfitting. In our project, we require the tree to be binary, while letting information criterion and maximum depth be unknown hyperparameters to optimize. Similar to the k in k Nearest Neighbors algorithm, the maximum depth is not searched for exhaustively. It is pinpointed by manually narrowing the search range.

3.4. Random Forest

Random Forest is an ensemble method augmented from decision trees based on bagging. For each iteration, we train a weak tree from a bootstrap sample obtained from the training set. At each node of this tree, a random set of features are chosen for the purpose of splitting. In our case, 100 trees are being trained from bootstrap samples. Notice that we do not perform any hyperparameter tuning for random forest for the sake of runtime. The maximum depth is set to "None" and the criterion is fixed as "Entropy".

3.5. Logistic Regression

Unlike the previous algorithms, the logistic regression is a soft classifier that predicts the conditional probability of a

label being 0 or 1. Explicitly, we predict

$$P(y=0) = \frac{1}{1+\exp(\boldsymbol{\beta}^\top \boldsymbol{x})},$$

where $\boldsymbol{x}$ is the feature vector of a new sample and the coefficient vector $\boldsymbol{\beta}$ is estimated from the training set by minimizing the negative log likelihood:

$$\begin{aligned} & -\log L(\boldsymbol{\beta}) \\ &= -\log(\prod_{i=1}^{N} P_i^{1-Y_i}(1-P_i)^{Y_i}) \\ &= -\sum_{i=1}^{N}(Y_i \boldsymbol{\beta}^\top \boldsymbol{x}_i - \log(1+\exp(\boldsymbol{\beta}^\top \boldsymbol{x}_i))), \end{aligned}$$

where N is the training sample size, $\boldsymbol{x}_i$ denotes the features of the i^{th} sample in the training set, and Y_i is its label. P_i, defined by $\frac{1}{1+\exp(\boldsymbol{\beta}^\top \boldsymbol{x}_i)}$ in this model, is the probability of Y_i equal to 0. In order to avoid overfitting, we add an L_1 regularization term to the cost function. Its coefficient is c, which is a hyperparameter we need to optimize.

3.6. Bagging

Bagging is an ensemble method to improve the performance of an algorithm. In our project, we only apply it to the logistic regression. This seemingly arbitrary decision is made not only because the logistic regression has a high test accuracy to be worthy of a further uplift, but also because it is the fastest algorithm among others so as to make bagging very applicable. The L_1 regularization coefficient is inherited from the regular logistic regression.

Different bagging sizes are attempted to help us visualize a pattern of increase in test accuracy.

4. Experiment

4.1. Dataset

This dataset, originally obtained from IMDb database, is now published on Kaggle after initial preprocessing. The dataset provides a perfectly balanced class between positive and negative reviews, 25000 reviews for each label, 50000 reviews in total. The original movie rating is on a scale from 1 to 10. When the dataset was first processed by previous researchers, there was a threshold where a review will be labeled as negative when the rating is lower than 4 and positive when the rating is higher than 7. Though the dataset has been preprocessed, there are still some clean up work need to be done for our analysis.

4.2. Language Processing

4.2.1 Removing HTML tags, special characters, and stop words

Being read the internet, raw review data contains tags that constitute the grammar of HTML, punctuation, and words that are not related to sentiments (stop words). We eliminate these irrelevant components by three different functions. `BeautifulSoup` function from package bs4 is first applied to the original texts to remove HTML tags. Then all special characters are replaced by white spaces by the `sub` function of package re. Lastly, we download from NLTK a collection of stop words, and remove them from the texts with the help of `ToktokTokenizer` from NLTK. Figures 1, 2, and 3 display the results.

Before:

```
the other reviewers has mentioned that after
what happened with me.<br /><br />The first
```

After:

```
the other reviewers has mentioned that
what happened with me.The first thing
```

Figure 1: Removal of HTML tags

Before:

```
great master\'s of comedy and his life. The realism
), rather than use the traditional \'dream\' techniqu
```

After:

```
great master s of comedy and his life  The realism r
 rather than use the traditional  dream  techniques
```

Figure 2: Removal of special characters

Before:

```
'A wonderful little production  The filming technique is very unassuming
and sometimes discomforting  sense of realism to the entire piece  The ac
```

After:

```
'wonderful little production filming technique unassuming
e realism entire piece actors extremely well chosen Michae
```

Figure 3: Removal of stop words

4.2.2 Text Stemming

Words such as ”go” and ”went” do not make essential difference. They are only different due to the English grammar. We therefore lemmatize and stem the texts to coerce such words into one and the same feature. Classes `WordNetLemmatizer` and `PortStemmer` from the nltk package are used to accomplish this goal.

Before:

```
hooked right exactly happened first thing struck
: hearted timid show pulls punches regards drugs
```

After:

```
hook right exactli happen first t
d show pull punch regard drug sex
```

Figure 4: Text Stemming

4.2.3 Vectorization

Now we need to turn features (words) into a design matrix. This step is performed by the `CountVectorizer` class imported from scikit-learn. A total of 70847 features are extracted.

```
from sklearn.feature_extraction.text import CountVectorizer
cv=CountVectorizer()
cv_train=cv.fit_transform(df["review"])
cv_train
```

```
<50000x70847 sparse matrix of type '<class 'numpy.int64'>'
        with 4637943 stored elements in Compressed Sparse Row format>
```

Figure 5: The code and result for vectorizing reviews

4.3. Model Training

After processing, the texts are turned into word counts and are stored in a 50000×70847 sparse matrix. The dataset is then divided into 70% of training data and 30% of test data using `train_test_split`. The split is stratified by the labels to guarantee that the training and the test sets both contain an equal proportion of positive (negative) labels. A random seed of 1 is used whenever randomness is involved. Hyperparameter tuning, when there are hypermarameters in an algorithm, is conducted by 5-fold cross validation, accomplished by `GridSearchCV`.

4.3.1 RBF-kernel SVM

RBF kernel is the default parameter setting for class `SVC`, which is a scikit-learn SVM classifier. We apply `SVC().fit` directly to our training data. There is no hyperparameter tuning because it is computationally expensive. We quote from documentation of this class:

> *The fit time scales at quadratically with the number of samples and may be impractical beyond tens of thousands of samples.*

Indeed, the training time can reach up to an hour in our case.

4.3.2 Linear SVM

An fast alternative to the preceding RBF-kernal SVM is linear SVM. The classifier we use is `LinearSVC`. We change the L_1 regularization parameter c from 0.001 to 100 with a step factor of 10. The best c is found to be 0.01 by `GridSearchCV`.

4.3.3 KNN

We use `KNeighborsClassifier` for KNN. The parameter k is tuned by grid search. We do not brute force through every possible k due to inefficiency and hardware limitation. As we can see from Figure 6, we first examine k from 50 to 300 by a step size of 50. Then we narrow our range down to around 200 and reduced the step size, and eventually achieve an optimal k of 182. Figure 6 shows this process together with validation scores.

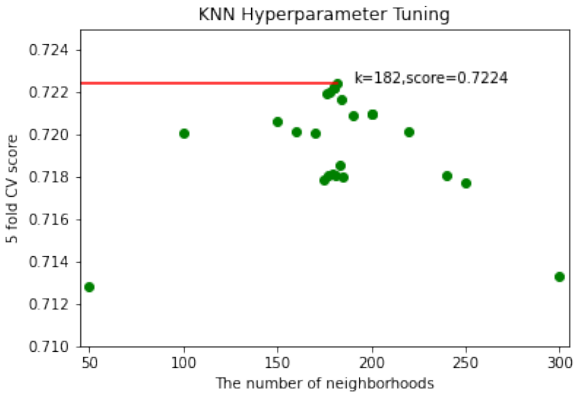

Figure 6: KNN Hypertunning

4.3.4 Decision Tree

There are two hyperparameters we tune in `DecisionTreeClassifier`: maximal depth of the tree and information criterion. We search for the optimal max_depth in the same way as k in KNN except for coupling it with criterion = ['entropy','gini'] at each iteration. We first examine max_depth from 5 to 30 with a step of 5, and then from 15 to 25 with a step of 1. Eventually, we find that a maximal depth of 18 with criterion = 'gini' has the highest validation score.

4.3.5 Random Forest

Similar to Decision Tree, two hyperparameters that we tune in `Random Forest` are maximal depth of the tree and information criterion. The parameter values specified for max_depth are [1, 10, 50, 100, 500, none], while values specified for criterion are ['entropy', 'gini']. By `GridSearCV`, we are given 50 as the most optimal value for max_depth, and 'entropy' for criterion.

4.3.6 Logistic Regression

For logistic regression, we use the class `LogisticRegression` from scikit-learn. When bagging is not applied, we change the L_1 regularization parameter c from 0.001 to 100 with a step factor of 10. The best c is found to be 0.1 by `GridSearchCV`. When bagging is applied, we inherit the hyperparameter c=0.1 from the case without bagging, and use `BaggingClassifier`. Bagging sizes ranging from 10 to 80 with a step size of 10 are experimented.

5. Results

Table 1 is a summary of the results obtained from each model.

Model	Hyperparameter	Accuracy
SVM (RBF kernel)	NAN	0.87947
SVM (Linear kernel)	C=0.01	0.88813
KNN (L2 Norm)	k = 182	0.72893
Decision Tree	max_depth = 18 criterion = 'gini'	0.74087
Random Forest	max_depth = None criterion = 'entropy'	0.85853
Logistic Regression	C=0.1	0.88967
Logistic Regression with Bagging	C=0.1(inherited)	0.89167 (size = 60)

Table 1: Test Accuracies and Hyperparameters for each Model

5.1. SVM

The SVM model with RBF kernel achieves a test accuracy of 87.95%, which is slightly less than the one with linear kernel, whose score is 88.82%. However, for our problem, the training time of SVM with RBF kernel can take over an hour while linear SVM, even coupled with hyperparameter tuning, takes just a few minutes. We also compare these two methods by ROC curves and McNemar's Test. From Figure 7, we can see that the rendered ROC curve for linear SVM is above that of SVM with RBF kernel for a large portion of [0,1] but not completely.

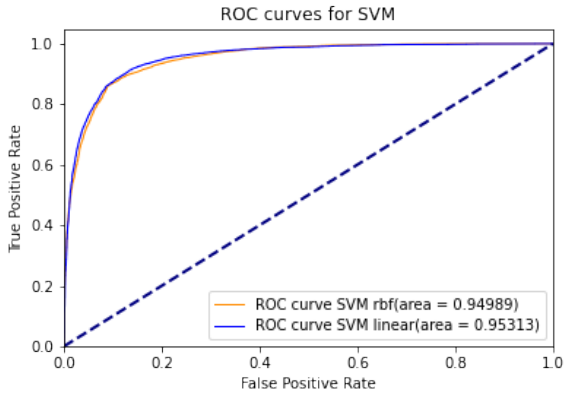

Figure 7: Comparison of SVM models with rbf and linear kernels

The difference of performance in the models can also be detected by McNemar's test. We set the significance level to 0.05, and from Figure 8, the McNemar's test statistic is

$$\chi_1^2 = \frac{(|556 - 426| - 1)^2}{426 + 556} = 16.946,$$

whose p-value is $3.845827e - 05 < 0.05$. While the test results indicates a significant difference, notice that we did not add a regularization term for SVM with RBF kernel. Adding it could possibly reduce overfitting and improve the performance. Yet considering an enormous amount of runtime needed for training this model, linear SVM is arguably better.

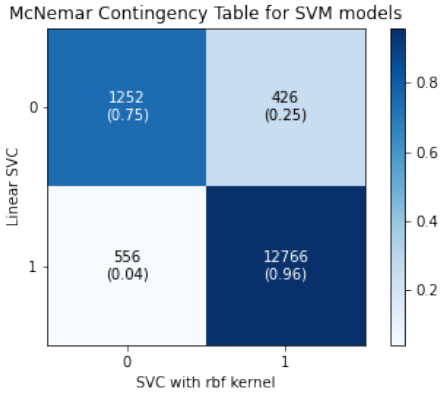

Figure 8: McNemar's Test

5.2. KNN

The KNN model achieves 72.89% accuracy with optimal k of 182, which is the lowest accuracy achieved among all models we fitted in the project. The result is not surprising as we expected KNN to serve as a baseline model for comparison in our project because of its simplicity. Though it

is possible that we did not find the best parameter value because of the interval search process we used, it is not worthy of going further on KNN since we are not expecting a dramatic improvement on accuracy caused by parameter value change.

5.3. Decision Tree & Random Forest

The Decision Tree model achieves a test accuracy of 74.09% while Randomn Forest achieves a test accuracy of 85.85%. The result is not surprsing as Random Forest is an ensemble method that leveragges the power of multiple decision trees for making decisions.The performance can be illustrated with Figure 9, from which we can see that the ROC curve for Random Forest is much futhur above Decision Tree.

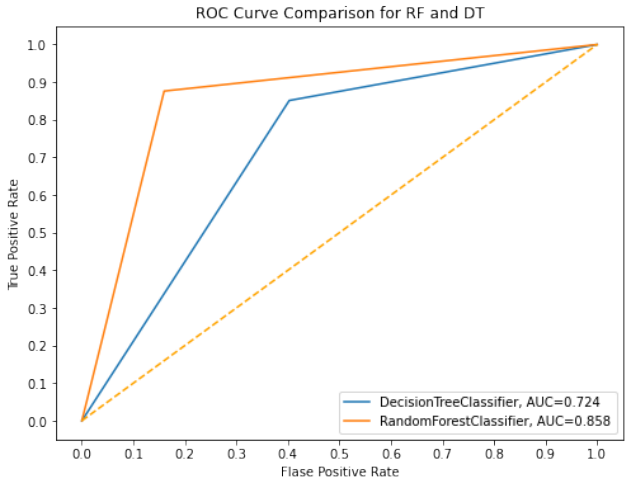

Figure 9: Comparison of Decision Tree and Random Forest

The disadvantage of using Random Forest over Decision Tree is the overwhelming computional complexity.The parameter tunning process for Random Forest generally takes hours to run given a few specified parameter. However, considering the dramatic performance improvement, Random Forest is undoubtedly peferred over Decision Tree.

5.4. Logistic Regression

The Logistic Regression achieves a test accuracy of 88.97%, which is the highest accuracy among all the models we have discussed so far. To see if we can furthur improve Logistic Regression performance, we applied bagging to it. We have changed the bagging size from 10 to 80 by a step of 10,fit the model, test on the testing set and record the accuracy. From Figure 10, we observe an overall improvement on accuracy and the highest accuracy, 89.17%, is achieved at bagging size of 60.

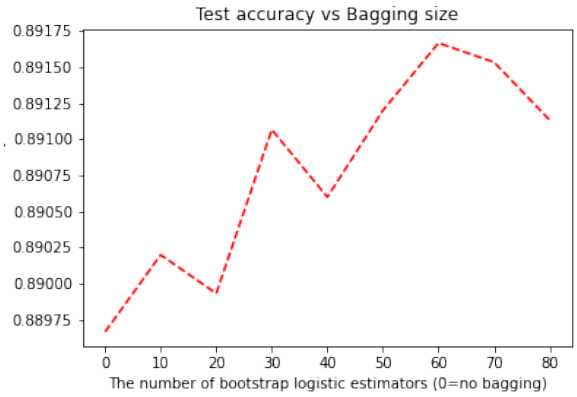

Figure 10: Test Accuracy vs Bagging Size

Furthurmore, by plotting ROC curves for logistic regression with and without bagging in Figure 11, we can see that ROC curve with bagging lies entirely above the one without bagging. Therefore, we can confirm the usefullness of the ensemble method of bagging on Logistic Regression.

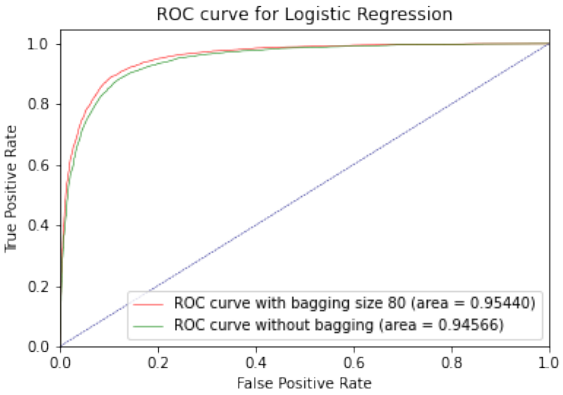

Figure 11: ROC curve for Logistic Regression

6. Discussion

It is important to note that the performance of each model varies significantly from each other. KNN and Decision Tree turn out to perform the worst with 72.893% and 74.08% over all accuracy, which is much less accurate than the other 5 models. We conclude that KNN and Decision tree is not suitable for sentimental analysis.

However, we were expecting Random Forest to outperform other fitted models while it turns out to be the least accurate model excluding KNN and Decision Tree. It is possible that we did not find the best parameter for Random Forest. The reason is that Random Forest is computationally expensive.The grid search process took hours to

run even though only a few parameter values were specified. Therefore, is it likely that the selected parameter value in our model is not the best. We are curious to see the performance of Random Forest if we are able to find the best parameter values with either a enhanced hardware or cloud computing.

7. Conclusion

The analysis presented in the report aims to find the best algorithm for sentiment analysis of IMDb reviews. We demonstrated several potential algorithm including SVM, KNN, Decision Tree, Random Forest and Logistic Regression, train them on the same dataset with paramenter tunning and different machine learning techiniques, and evaluate the performance in terms of accuracy. We also use ROC curve, Confusion Matrix and Learning Curve as supplementary techniques to help us better evaluate and understand the fitted models.

From Table 1, we cconclude that Logistic Regression performs the bestamong all fitted models, with an overall 89.167% accuracy and Bagging size of 60. Nonetheless, it is possible that Random Forest could be improved during parameter tunning stage. Also, from the comparison of Logistic Regression with and without bagging, we conclude that bagging could help to improve the model accuracy.

Finally, we hope our analysis lays a solid foundation for sentimental analysis. We hope our current analysis could be expand to Multi-label analysis since most of the rating system implemented in the business world is multi-lable. In the future, we hope more researchers could gather a dictionary will be helpful for organizations to effectively finish sentimental analysis on the collected reviews.

8. Contributions

Haoming Chen found the original dataset online and cleaned up the data for furthur analysis. Ruiting Tong coded the following models with parameter tunning: SVM(RBF kernel), Decision Tree, KNN and Logistic Regression with and with out bagging. Dingyi Li coded the rest models, SVM(Linear kernel) and Random Forest, and added code to generate confusion matrix, ROC curve and learning curve.

As for the report, Haoming drafted Introduction and Related Work. Ruiting finished Abstract, Proposed Methods, Experiments and SVM section under Results section. He also worked on the overall structure of the paper including paper style and reference. Dingyi finished the rest of Results section, Discussion and Conclusion.

9. Code

All of the code used to generate the plots and results for this project is on GitHub: `https://github.com/hchen549/stat451`

References

[1] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, (Portland, Oregon, USA), pp. 142–150, Association for Computational Linguistics, June 2011.

[2] B. Pang and L. Lee, "A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts," *CoRR*, vol. cs.CL/0409058, 2004.

[3] T. Wilson, J. Wiebe, and P. Hoffmann, "Recognizing contextual polarity in phrase-level sentiment analysis," in *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, (Vancouver, British Columbia, Canada), pp. 347–354, Association for Computational Linguistics, Oct. 2005.

[4] G. Gautam and D. Yadav, "Sentiment analysis of twitter data using machine learning approaches and semantic analysis," in *2014 Seventh International Conference on Contemporary Computing (IC3)*, pp. 437–442, 2014.