IMDb Movie Reviews Classification and Prediction: Rule-based, Supervised Learning, Neural Network

Isabella Xue rxue8@wisc.edu
Shuyuan Jia sjia39@wisc.edu
Freya Wan zwan26@wisc.edu

Abstract

Movie reviews usually reflect the audience's attitude towards a movie, but it is time-consuming for human to read every review to classify the emotion polarity of the review. In this project, we are interested in conducting sentiment analysis on IMDb movie reviews. We use 50000 IMDb movie reviews to build model and make predictions. We experiment with the rule-based method, supervised learning algorithms such as Random Forest, Logistic Regression, XGBoost, LightGBM, Stacking classifier(with logistic regression, XGBoost, and LightGBM), and Voting classifier (with logistic regression, XGBoost, and LightGBM), and Neural Network method. Our best test accuracy score is 88.909%, obtained by Stacking Classifier. We conclude that rule-based method is able to perform baseline predictions of positive and negative review, and although with bigger dataset and more tuning, neural network method could achieve better performance, supervised learning models are most efficient and effective to perform classification on movie review.

1. Introduction

In the time that the internet provides a broader-than-ever-imagined platform, people are able to share their thoughts freely online. A problem also arises due to this huge flood of comments or information. IMDb is one of the flooded online platforms that serve to present critical reviews of movies. People will praise the movies they like and criticize those they don't like on IMDb. The reviews contain information related to the reviewer's attitude towards a movie. But how do we interpret these amounts of information efficiently since we obviously can't read every review? This is where word representation and sentiment analysis come into place. There are numerous research studies on word representation. Many of them have combined supervised and unsupervised learning to improve accuracy.

In this project, we aim to perform sentiment analysis on movie reviews using various techniques and models to predict label- Positive or Negative. We used a balanced dataset containing 50000 IMDb movie reviews with 25000 positive and negative reviews to conduct sentiment analysis. The reviews have 2 labels of sentiment- positive and negative, so we convert the sentiment variable from a string variable to a numeric indicator, with 0 representing negative and 1 representing positive. We implement rule-based method, supervised learning models and neural network models to do the classification and then compare their performances to figure out the advantages and disadvantages for each method.

To find the model with the best performance, we use a rule and lexicon-based model as our baseline model and experiment with Random Forest, Logistic Regression, XGBoost, LightGBM, Stacking(with logistic regression, XGBoost, and LightGBM), Voting(with logistic regression, XGBoost, and LightGBM), and Neural Network models(full-connected model, LSTM models). We clean data in the procedure shown in Figure 3 by using NLTK packages in Python and select the features by using word frequency and TF-IDF. Then, we split the data into 67% for training and 33% for testing. We calculate the testing accuracy for every method we experiment with to evaluate the model and use McNemar's test and ROC curves to choose the best model.

2. Related Work

Machine learning has been widely used for sentiment analysis, and researchers have been already implemented methods including supervised learning, unsupervised learning, semi-supervised learning. Sahu and Ahuja used six classification techniques, including random forest, decision tree, Naïve Bayes, etc., and random forest gives the best performance with 88.95% accuracy[8].

Singh et al. utilized two supervised learning algorithms (Naïve Bayes and SVM), one unsupervised learning algorithm (SO-PMI-IR algorithm), and the SentiWordNet lexical method to conduct movie review classification [9]. The researchers concluded that the unsupervised learning algorithm had achieved the best performance, followed by the supervised learning algorithm. In addition, although the lexical method achieved the lowest accuracy, it is "computationally most favorable."

Turian et al. proposed to combine unsupervised word representations as extra word features with supervised learning baseline models [10]. They concluded that their semi-supervised method, word representation with supervised learning model, was able to improve model performance than the supervised and unsupervised learning model alone.

The above papers proposed different approaches to perform classification on text data. However, we aim to optimize the performance of rule-based, supervised and neural network method. And most importantly, we further compare and weigh the pros and cons of above methods.

3. Proposed Method

3.1. Rule-based Method

In this project, we use a rule and lexicon-based method to build baseline model: VADER (Valence Aware Dictionary and sEntiment Reasoner). VADER is a weakly supervised model that was created to analyze sentiment in social media [5]. It was built on 9,000 lexical feature candidates and 7,500 of them were kept with validated valence scores that indicate sentiment polarity (positive or negative) and intensity (scale of -4 to 4).

To analyze intensity of the text, the algorithm incorporate punctuation (e.g., "I hate this movie!!!" is more intense than "I hate this movie!"), capitalization (e.g. "I HATE this movie" is more intense than "I hate this movie"), degree modifiers (e.g., "I hate this movie so much" is more intense than "I hate this movie"), and the contrastive conjunction "but" (e.g., "the CGI was ok, but the acting was horrible").

The outcome produced by the algorithm is "pos", "neu", "neg", and "compound" scores of the text, and each of them is displayed as proportion in the text so that "pos", "neu" and "neg" add up to 1. In this project, we use the "compound" score to label positive or negative review.

3.2. Supervised Learning

3.2.1 Logistic Regression

Logistic regression is a widely used machine learning algorithm. It is built based on logistic function to predict the probability for two or more discrete classes given some input variables. In this project, we use movie reviews to predict whether the review is negative or positive. There are two classes which are "negative"(0) and "positive"(1), so we implement binary logistic regression. Suppose there are p features extracted from the reviews, then we would write the predicted possibility for as the following:

$$P(y^{(i)} = 1) = \frac{1}{1 + exp(-(\beta_0 + \beta_1 x_1^{(i)} + ... + \beta_p x_p^{(i)}))}$$

3.2.2 Random Forest

Random forest is a robust supervised learning algorithm for classification. The idea of random forest is fitting decision trees based on different bootstrap samples and randomly choosing a subset of features at every node to decide the optimal split. We select some tokens in review text for our project as the features to build random forest. However, if the features have a strong correlation with each other, in this case, there is a high possibility to overfit. Also, our dataset contains 500,000 reviews, and the model will be built in super high dimensions. That is, both the number of trees and the depth of each decision tree could be large, which could lead to an overfitting problem.

3.2.3 XGB

XGBoost is a popular supervised learning algorithm used in classification. It is an ensemble method based on decision trees. XGBoost will build new trees based on the errors of previous trees and combine trees for prediction until we cannot make any further improvements to the model. It efficiently implements gradient boosted decision trees by constructing decision trees concurrently to increase speed. We use *GridSearchCV* with cross-validation for tuning to find the best hyperparameter for our model.

3.2.4 LightGBM

LightGBM is one of the efficient implementations for ensemble methods that help us combine some "weak learners," such as shallow decision trees, into a "strong learner." Compared to other ensemble methods, such as XGBoost, LightGBM has the advantages of its efficiency and smaller memory consumption. It can handle large datasets and perform relatively well [6].

3.2.5 Stacking

Stacking is an ensemble method that combines multiple first layer classifiers through a meta-classifier, as shown in figure 1. The first layer classifiers are trained on the training dataset. Then the second level meta-classifier uses either the predicted label from first layer classifiers or the probabilities from the ensemble (by setting use_probas=True). Stacking classifier is most helpful when the first layer classifiers have advantages in different skills. Therefore, using diverse first layer classifiers elevate the performance of stacking classifier. It is safe to conclude that stacking classifier is very effective in improving model performance.

3.2.6 Voting

Voting classifier is not an actual classifier, but it trains on an ensemble of models and generates predictions using the

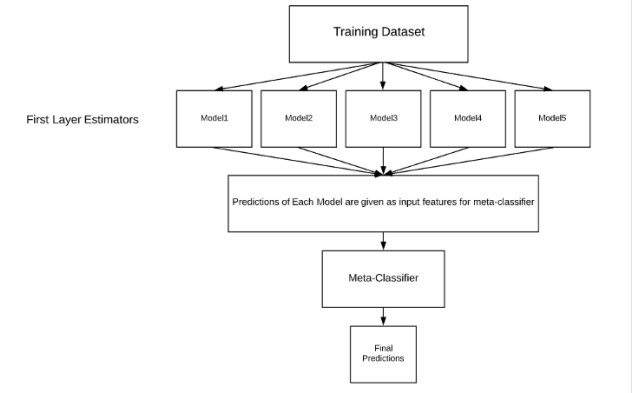

Figure 1. An illustration of Stacking classifier. Image source: [7]

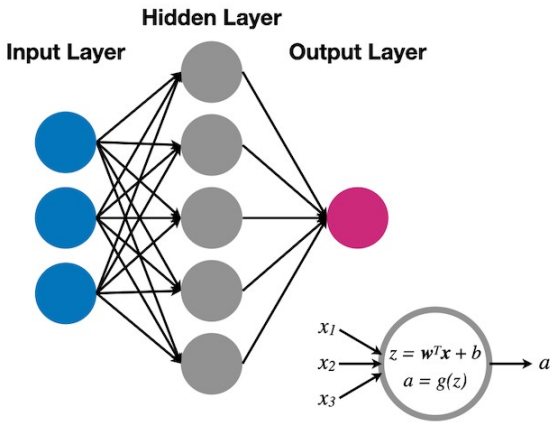

Figure 2. A shallow neural network with only one hidden layer. Image source: [3]

majority vote. That is, the voting classifier predicts labels using hard or soft voting. In soft voting, the model generates labels based on predicted probabilities for each classifier, as seen in the equation,

$$\widehat{y} = \underset{i}{\operatorname{argmax}} \sum_{j=1}^{m} w_j p_{ij}.$$

and we also choose to specify the weight w_j of each classifier. In contrast, hard voting predicts labels simply by the majority vote of each classifier. It is crucial to choose a diverse group of classifiers to train the Voting classifier since similar classifiers may fall prey to the same type of errors.

3.3. Neural Network

Neural network is a part of machine learning, and the idea is similar to how the human brain's neural network works. For a standard neural network model, there is a layer of input nodes, the layer(s) of hidden nodes, and the layer of output nodes (Figure2). Every node has its own model similar to a multiple linear regression composed of some input data, weights, bias, and outputs. The output can be written as the following [4]:

$$Output = f(x) = \begin{cases} 1, & \text{If } \sum_{i=1}^{m} w_i x_i + bias \geq 0 \\ 0, & \text{Otherwise} \end{cases}$$

There are various neural network models; in this project, we implemented fully-connected model and LSTM models. A dense layer means that the layer is fully connected. That is, for every neuron in a dense layer, it will receive the input from all neurons in the previous layer. LSTM is a special case of RNN models. RNN is a class of neural networks that uses the previous output as the inputs while having hidden states. RNN takes advantage of having feedback connections that are different from the standard feedforward neural networks methods. The tokens that appear in the front in one review sentence are likely to affect the tokens that appear in the back; thereby, this may affect the results, meaning that the order of tokens matters. Therefore, we choose to implement some sequential models. To deal with the vanishing gradient problems, we implement LSTM, which uses a memory cell to handle some long-range information.

4. Experiments

4.1. Dataset

In our project, we use the Large Movie Review Dataset from Stanford University, which was compiled by Andrew Maas and can be found here: `http://ai.stanford.edu/~amaas/data/sentiment/`. The full dataset contains 50,000 movie reviews, and they are split into 25,000 reviews for training and 25,000 for testing the classifier. Additionally, each data subset has 12,500 positive and 12,500 negative reviews. Users can rate movies on a scale from 1 to 10 on IMDb. The reviews with ≤ 4 stars are labeled negative, and reviews with ≥ 7 stars are labeled positive in the dataset. Reviews with 5 or 6 stars were left out.

4.2. Data Processing

4.2.1 Data Preprocessing

Firstly, We remove all the HTML tags and the special characters from the review text, then convert all letters into lowercase. After that, we clean the extra white space and remove stop words in the sentences, since most stop words in English do not give us much information, such as "a", "of", etc. We also do the lemmatization, which groups the different forms of a word as a single word.

Raw text	A masterful production about one of the great master\'s of comedy and his life. The realism really comes home with the little things
Remove html tags	A masterful production about one of the great master\'s of comedy and his life. The realism really comes home with the little things
Remove special characters	A masterful production about one of the great master s of comedy and his life The realism really comes home with the little things
Convert letters into lower case	a masterful production about one of the great master s of comedy and his life the realism really comes home with the little things
Remove extra space and stop words	masterful production one great masters comedy life the realism really comes home little things

Figure 3. Data Preprocessing

4.2.2 Feature Extraction

We use package *TfidfVectorizer* in Python, which combines *CountVectorizer* and *TfidfTransformer* to convert tokens into vectors by their importance. The package *CountVectorizer* helps us to get the word frequency in text reviews and TF-IDF helps us to map text data into vectors and reflects the importance of a word in a corpus. The term frequency tf(t,d) calculate how frequent a term *t* appears in a document *d*, while the inverse document frequency idf(t, D) calculate how important a term *t* is in some documents *D*. Here are the formulas for TF, IDF and TFIDF[1]:

$$\text{tf(t, d)} = \log(1 + \text{freq(t, d)})$$
$$\text{idf(t, D)} = \log \frac{\text{N}}{|\text{d} \in \text{D:t} \in \text{d}|}$$
$$\text{tfidf(t, d, D)} = \text{tf(t, d)} \times \text{idf(t, D)}$$

4.3. Modeling

4.3.1 Rule-based Method

We use the VADER Rule-based Classifier in this project. For the rule-based method, we only process the data by removing the HTML tags in the reviews and expanding the contractions in the reviews to their respective actual form. After processing, we used *RegexpTokenizer* to tokenize sentences to a list of words. Then we create a *SentimentIntensityAnalyzer* object directly from nltk package. This object can be directly applied to the unlabeled data. Calling the polarity_scores method of *SentimentIntensityAnalyzer* generates the positive, neutral, negative, and compound score, as seen in the examples in the figure 4 . A **-1** compound score is extremely negative and a **+1** compound score is extremely positive.

To predict positive or negative labels, we set a threshold that a review having a compound score bigger than **0.3** will be classified as a positive review; otherwise, it is labeled as a negative review. We chose not to create another class "neutral since this project only perform binary classification. It is important to note that this rule-based implementation did not require any training.

```
I can't believe that I spend frikin 4 hours watching this ho
rrible movie. A waste of my time!! {'neg': 0.385, 'neu': 0.6
15, 'pos': 0.0, 'compound': -0.8181}
[illegible]
the movie is visually stunning to watch and the casting was
just perfect!! {'neg': 0.0, 'neu': 0.615, 'pos': 0.385, 'com
pound': 0.7835}
[illegible]
```

Figure 4. Examples of sentiment score generated by VADER SentimentIntensityAnalyzer

4.3.2 Supervised Learning

4.3.2.1 Logistic Regression For logistic regression, random_state is set to 123, and max_iter is set to 1000. Then we use *GridSearchCV* in sklearn library to find best hyperparameter of solver, penalty, and C (CV is set to 10). At first, the parameter values specified for solver are ['newton-cg', 'lbfgs', 'sag'], ['l2', 'l1'] for penalty, and [10, 1.0, 0.1, 100] for C. The tuning results gives 'C': 10, 'penalty': 'l2', 'solver': 'newton-cg' for highest validation score 88.934%. Next, we change the parameter values specified for C to [0.1, 1.0] to rerun *GridSearchCV* to find best hyperparameter. We concluded with 'C': 1.0, 'penalty': 'l2', 'solver': 'newton-cg', with validation score of 88.635%.

4.3.2.2 Random Forest For the Random Forest model, we set random_state to 123 and then fit the random forest classifier on our training set using the default parameters without tuning because it takes too long to tune the model and find the best parameters.

4.3.2.3 XGB For XGBoost, we set random_state to 123, and use GridSearchCV with cross-validation for tuning the hyperparameters to find the best parameters, setting cv to be 10. At first, we set the 'max_depth' to be [4, 6, 8] and ''min_child_weight' to be [1, 3], as shown in Table 1. The tuning result specifies that 'max_depth': 6, and 'min_child_weight': 1 achieve the highest cross-validation score. Then, we change the parameters to the best hyperparameter we found to build the model.

Hyper-Parameters	Value Range	Best Choice
max_depth	[4, 6, 8]	6
min_child_weight	[1, 3]	1

Table 1. XGBoost hyper-parameters

4.3.2.4 LightGBM For LightGBM, we use randomized search with cross-validation for tuning the hyperparameters to find the best parameters, setting random_state to 123. The parameters we specified for solver are 0.6 for 'bagging_fraction', 5 for 'bagging_freq', [0.01, 0.05, 0.1] for 'learning_rate', and [100, 200] for 'num_iterations', as shown in Table 2. At first, we include the parameter 'num_leaves' with a value of 80, but the model shows overfitting, so we remove this parameter from our model to resolve the overfitting problem. The best hyperparameters we get are 'bagging_fraction': 0.6, 'bagging_freq': 5, 'learning_rate': 0.1, and 'num_iterations': 100. Next, we apply the best hyperparameters to build our model.

Hyper-Parameters	Value Range	Best Choice
bagging_fraction	[0.6]	0.6
bagging_freq	[5]	5
learning_rate	[0.01, 0.05, 0.1]	0.1
num_iterations	[100, 200]	200

Table 2. LightGBM hyper-parameters

4.3.2.5 Stacking For stacking classifier, the first layer classifiers are *LogisticRegression*, *XGBClassifier*, and *LGBMClassifier* formerly used in the project. We adopt this method to improve the accuracy score (CV is set to 10). In addition, we use *LogisticRegression*() as meta-classifier. Since it already takes about twenty minutes to run the stacking classifier, we decided not to apply any tuning method.

4.3.2.6 Voting For voting classifier, we choose to include *LogisticRegression*, *XGBClassifier*, and *LGBMClassifier* formerly used in the project as ensemble of models. We only adopt the soft voting method in this project and specify the weight [2, 1, 3] accordingly. Setting the specific weight shows how much we trust each model when making predictions.

4.3.3 Neural Network

4.3.3.1 Fully-connected Model We first build a full connected model. In this model, we set the input size 250 × 50, which will be transformed into a vector in the Flatten layer, giving a feature space of width 12500. Then we use a hidden dense layer, and this dense implementation is based on a 10-unit layer followed by the final dense layer computing the sigmoid probabilities.

4.3.3.2 LSTM Model For a simple LSTM model, we start with defining an Embedding layer with a vocabulary of 250 a vector space of 32 dimensions in which words will be embedded, and input documents with 250 words each. Then we define the LSTM with 32 units to pass on a dense output layer.

4.3.3.3 LSTM with more Layers In this case, we start with defining an Embedding layer with a vocabulary of 250 a vector space of 128 dimensions in which words will be embedded, and each input documents has 250 words. This time, we change LSTM into bidirectional LSTM, consisting of two LSTMs. That is, one LSTM uses the input in a forward direction, while the other LSTM takes inputs in a backwards direction[2]. Then, we use the MaxPooling1D layer to down sample the inputs. Then pass the output into a 10-unit Dense layer. Also, to avoid the overfitting problem, we use Dropout layer, which could deal with the strong dependencies between the close layers by dropping out the random portion of the outputs, which is similar to the boosting technique for decision trees.

4.4. Software

In this project, we use Jupyter Notebook as the computing platform, and use Python to preprocess data, select features, train models, evaluate models and also do some data visualization. The packages we used are Numpy, Pandas, Matplotlib, Scikit-Learn, NLTK, etc.

4.5. Hardware

Each group member uses their laptops.

5. Results and Discussion

5.1. Rule-based Method

The rule-based method VADER SentimentIntensityAnalyzer serves as a baseline model and generates 70.52% accuracy score, 72.02% precision score, 70.52% recall score, and 70.01% F1 score. The accuracy score is a good metric for evaluating the model since both precision and recall scores are high, and the dataset is balanced. We can see that the performance of this rule-based method is better than chance (50%). One factor that we propose that may hinder the performance of the model is that it cannot analyze the context of the text. In other words, it only cares about individual words. The figure 5 shows an example of this. Clearly, the commenter is not praising the movie, but the compound score (0.9482) is highly positive.

But this is not to say that one should not use this rule-based method in any circumstances. It is straightforward to implement and does not require any model training and fitting. When a higher accuracy score is not required, it is fair to adopt this method.

5.2. Supervised Learning

Summary of model results is displayed in Table 3.

```
Phil the Alien is one of those quirky films where the humour is ba
sed around the oddness of everything rather than actual punchline
s.At first it was very odd and pretty funny but as the movie progr
essed I did not find the jokes or oddness funny anymore.Its a low
budget film (that is never a problem in itself), there were some p
retty interesting characters, but eventually I just lost interest.
I imagine this film would appeal to a stoner who is currently part
aking.For something similar but better try "Brother from another p
lanet" {'neg': 0.088, 'neu': 0.679, 'pos': 0.233, 'compound': 0.94
82}
```

Figure 5. Example of how VADER SentimentIntensityAnalyzer can fail sometimes

Model	Accuracy	ROC AUC
LogisticRegression	0.88842	0.925
RandomForestClassifier	0.84994	0.956
XGBClassifier	0.85242	0.930
LGBMClassifier	0.87467	0.945
StackingClassifier	0.88909	0.956
VotingClassifier	0.88418	0.955

Table 3. Supervised learning models performance summary

5.2.1 Logistic Regression

In the first version of Logistic regression model with hyperparameter 'C': 10, 'penalty': 'l2', 'solver': 'newton-cg', we achieved test accuracy score 89.133%, which is fairly high. However, the train accuracy score is high as 98.490%, which implies overfitting. We aim to address this problem; therefore, we retrain the model with hyperparameter 'C': 1.0, 'penalty': 'l2', 'solver': 'newton-cg' and achieved train accuracy score 92.904% and test accuracy score 88.842%, which shows less sign of overfitting than the former model. The improved model can also be seen in the learning curve 6.

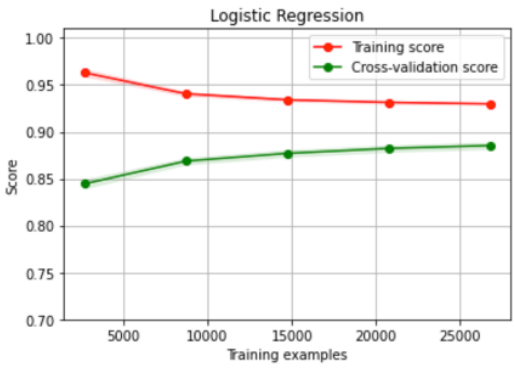

Figure 6. Logistic regression learning curve

Although test accuracy score is reduced after treating for overfitting issue, we conclude that it is wise to take this step because overfitting may affect the accuracy of predicting future observations.

5.2.2 Random Forest

The random forest model without tuning achieves a test accuracy score of 84.994% but has a training accuracy of 100%, which means that the model is strongly overfitting. Since the test accuracy of random forest model is lower than other models and the model has serious overfitting problem, we tend to choose other methods with higher test accuracy and more optimal training accuracy to be used in Stacking classifier and Voting classifier.

5.2.3 XGB

The test accuracy of XGBoost model using parameters 'max_depth' of 6, and 'min_child_weight' of 1 is 85.242%, higher than the score using random forest method, and the training accuracy is 93.91%, which reflects that there is no overfitting problem in the model.

5.2.4 LightGBM

The test accuracy of LightGBM using hyper-parameters 'bagging_fraction' of 0.6, 'bagging_freq' of 5, 'learning_rate' of 0.1, and 'num_iterations' of 100 is 87.218%, higher than the score using random forest and XGBoost. The training accuracy is 93.725%, showing that there is no overfitting problem in the model.

5.2.5 Stacking

The stacking classifier achieves a test accuracy score of 88.909%, which is by far the highest accuracy score we have achieved. The training accuracy score is 93.310%, which does not indicate a strong sign of overfitting.

It is natural to conclude that this is the best model so far. However, we concluded differently after comparing the ROC AUC of *LogisticRegression* model and *StackingClassifier*. In addition, we conducted McNemar's test with significance level of 0.05 to check if the difference of performance of *LogisticRegression* and *StackingClassifier* is significant. The result gives a test statistic of 96.000, p-value of 0.48286 (bigger than 0.05), which indicates that the two classifiers have a similar proportion of errors on the test set. Therefore, we concluded that although *StackingClassifier* test accuracy score is higher than *LogisticRegression* test accuracy score, it is reasonable to adopt *LogisticRegression* as the best choice if ones consider running *StackingClassifier* too expensive.

5.2.6 Voting

The voting classifier achieves a train accuracy score of 94.173% and a test accuracy score of 88.418%. We're surprised to find out that this test accuracy score is not higher than the test accuracy score for LogisticRegression. And

we conduct McNemar's test with significance level of 0.05 to check if the difference of performance of VotingClassifier and StackingClassifier is significant. The result gives a test statistic of 296.000, p-value of 0.00728 (smaller than 0.05). Therefore, we reject the null hypothesis of McNemar's test and conclude that VotingClassifier and StackingClassifier have different errors on the test set.

5.3. Neural Network

The accuracy and F1 scores for the test dataset is shown in Table 4. The accuracy for both fully-connected model and LSTM with more layers are larger than 0.87, which is higher than the simple LSTM model. However, even if we spent much more time training the neural network models, their performances are not better than the logistics regression model. One of the possible reasons is that the training data set is large enough. Furthermore, we could do model tuning to enhance the model's performance.

Model	Accuracy	F1 score
Fully-connected Model	0.8773	0.8783
LSTM	0.7557	0.7392
LSTM with more layers	0.8727	0.8766

Table 4. Results for Neural Network Models.

We also plot the accuracy curve and loss curve for each model to diagnose the neural network models. The full-connected model has a good learning rate, but there may be a little overfitting. The simple LSTM model and 'LSTM with more layers' model do not have either significant underfitting problem or overfitting problem.

6. Conclusions

In this project, our goal is to compare the performance among the rule-based model, supervised learning algorithms, and the neural network model. We weigh the pros and cons of each model to select the most advantageous method.

We used the rule-based model as our baseline because it has the lowest testing accuracy (70.52%). It is easy to implement, but the accuracy is not very high. If higher accuracy is not required, the rule-based model may be enough. Stacking achieves the highest accuracy among the supervised learning algorithms (88.909%), but it is relatively more expensive. The accuracy of logistic regression (88.842%) is high enough, and it is less expensive to train, so we can choose logistic regression if it is not worth it to make more effort. The Neural Network model is too expensive to train to get the same accuracy as supervised learning algorithms since we need further model tuning. If we have a bigger dataset to train on (such as millions of data), Neural Network may perform better and achieve higher accuracy. However, it is too time-consuming and effort-consuming. Therefore, we recommend adopting supervised learning if higher accuracy is not required.

If the dataset is small, using supervised learning is efficient and will perform better. Further studies should compare and combine supervised, semi-supervised, and unsupervised learning to investigate the efficiency of different methods for practical purposes. Our project sets a baseline for future study on comparing different models.

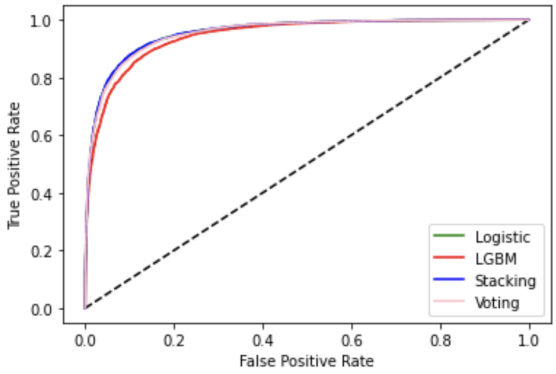

Figure 7. ROC curve of top 4 models

7. Acknowledgements

We truly appreciate the tremendous help and support from our professor Dr. Sebastian Raschka. His online deep learning lectures are extremely helpful to understanding new concepts about neural networks. The online resources can be found here: `https://sebastianraschka.com/blog/2021/dl-course.html`

8. Contributions

Isabella Xue implemented supervised learning models, including random forest, XGB, LightGBM, stacking, and voting with parameter tuning. She also coded for the rule-based method and did some data visualization. Shuyuan Jia coded for the data processing, including data cleaning and feature selection. She also coded the neural network models and did some data visualization. Freya Wan found the original data set and tried all other supervised learning models, including logistic regression, KNN, and SVM. She also tried to implement the semi-supervised algorithm. All team members, Isabella Xue, Shuyuan Jia, and Freya Wan worked together to finish the project report.

References

[1] tf-idf— Wikipedia, the free encyclopedia, 2021. [Online; accessed 5-Dec-2021].

[2] J. Brownlee. A gentle introduction to long short-term memory networks by the experts. *Machine Learning Mastery*, 19, 2017.

[3] Christian. A shallow neural network for simple nonlinear classification. 2020.

[4] I. C. Education. Neural networks. 2021.

[5] C. Hutto and E. Gilbert. Vader: A parsimonious rule-based model for sentiment analysis of social media text. *ICWSM*, 11(1):216–225, 6 2014.

[6] P. Khandelwal. Which algorithm takes the crown: Light gbm vs xgboost? *Analytics Vidhya*, 12, 2017.

[7] Koushik222. Stacking in machine learning.

[8] T. P. Sahu and S. Ahuja. Sentiment analysis of movie reviews: A study on feature selection amp; classification algorithms. pages 1–6, 2016.

[9] V. Singh, R. Piryani, A. Uddin, P. Waila, and Marisha. Sentiment analysis of textual reviews; evaluating machine learning, unsupervised and sentiwordnet approaches. pages 122–127, 2013.

[10] J. Turian, L.-A. Ratinov, and Y. Bengio. Word representations: A simple and general method for semi-supervised learning. pages 384–394, July 2010.