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, the companies could modify the service provided to 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. 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 algorithm 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

Gautam, G. and Yadav, D conducted the sentiment analysis on the Twitter data and focused specifically on the extraction of adjectives within each document. 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.

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). 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.

Pang and Lee proposed a method to combine the traditional bag-of-words model with a minimum-cut framework to retain the polarity information. 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.

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 non-linear 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 x^*, the algorithm computes

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

where $||\cdot||$ can be 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 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. In our case, 100 trees are being trained, and at each node, a random set of features are chosen for the purpose of splitting. The prediction is the majority vote of the 100 trees. 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,

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

where $\boldsymbol{\beta}$ is estimated from the training set by minimizing

$$\begin{aligned} &-2\log L(\boldsymbol{X_1}, ..., \boldsymbol{X_N}) \\ &= -2\log(\prod_{i=1}^{N} P_i^{1-Y_i}(1-P_i)^{Y_i}) \\ &= -2\sum_{i=1}^{N}(Y_i\boldsymbol{\beta}^\top \boldsymbol{X_i} - \log(1+\exp(\boldsymbol{\beta}^\top \boldsymbol{X_i}))). \end{aligned}$$

However, 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 Language Processing

4.1.1 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 remove these irrelevant components:

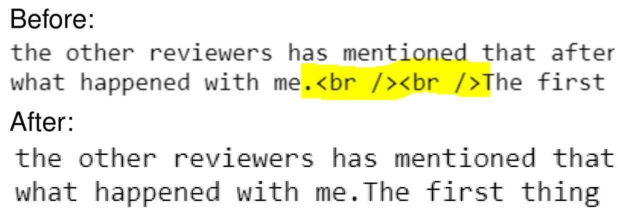

Figure 1: Removal of HTML tags

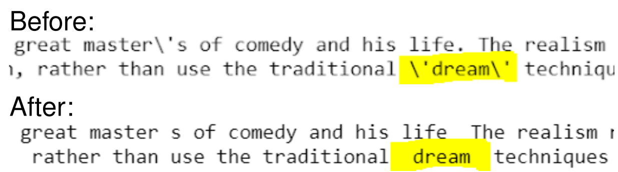

Figure 2: Removal of special characters

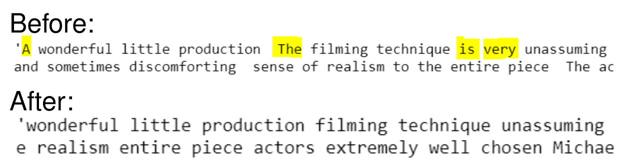

Figure 3: Removal of stop words

4.1.2 Text Stemming

Words such as "go" and "went" do not make essential difference. They are only different due to the English grammar. We therefore stem the text to coerce such words into one and the same feature:

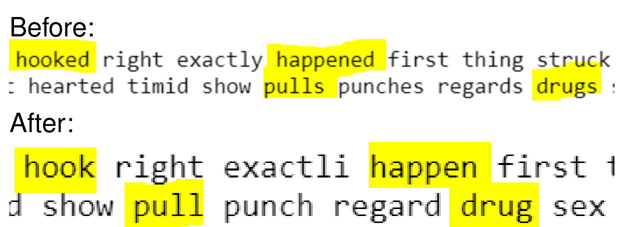

Figure 4: Text Stemming

4.1.3 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

4.2 Model Training

The dataset is divided into 70% of training data and 30% of test data. The split is stratified by the labels to guarantee that the training and the test sets contain homogeneous features. A random seed of 1 is used whenever randomness is involved. Hyperparameter tuning, when there are hypermarameters in an algorithm, is conducted by cross validation with k set to 5.

4.2.1 SVM with RBF kernel

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 6, 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.

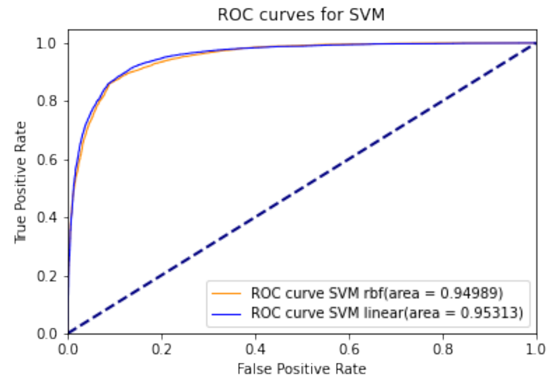

Figure 6: Comparison of SVM models with rbf and linear kernels

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 7, the McNemar's test statistic is

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

whose associated 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.

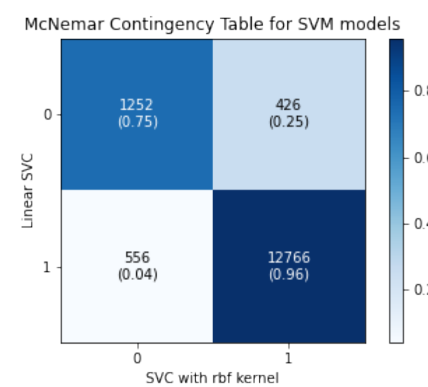

Figure 7: McNemar's Test

5.2 KNN

References