Report of COMPSCI 273A Project -- IMDB Reviews

Jianbo Pei Jingyuan Wang 70168070 Yuxin Jiang 80729166

1. Introduction

The project we did was on constructing classifier models for IMDB reviews, which is a problem related to NLP field. The dataset totally contains 50,000 reviews and splits into 25k training sets and 25k test sets. Then 25k sets split into 1.25k reviews that are labeled as positive and 1.25k reviews that are labeled as negative. Same proportional splits are for the test sets. Also, to avoid correlated ratings, no more than 30 reviews in the dataset collection are collected for any movie because the same movie tends to be reviewed similarly. The rating is on a scale of 0 to 10. A positive rated review has a score >= 7, and a negative rated review has a score <=4. Our project mainly focused on the 25k training sets and constructed various models to test the performance for each model.

In order to build models, we need to process all the review files into an array that can be built for machine learning models. All the reviews are separated into a text file individually. Because each file contains not only English words but also some noise such as stopwords, punctuations, special characters, etc., we need to filter these noises out to obtain better performance in terms of time of complexity and accuracy. The desired processed data is a list of dictionaries in format as*: [{word_1: count, word_2: count, … __FileID__: DocID, __CLASS__: 0 or 1}…]*. Because the processing time was pretty slow inside Jupyter kernel, we did the pre-processing step and saved the results to local for later use.

To generate an array that can be processed, we employed pandas library. Processing all the documents in train folders resulted in a size of 25000 x 121224 array. However, this size exceeds the maximum size of an array that can be allocated. Therefore, we chose only 2000 documents (1000 from neg folder and 1000 from pos folder) in our project instead of all 25k documents. Also, for the words that didn't appear in documents, we set that count to 0. This would help to keep the features consistent. The resulting array has a size of 2000 x 28391, which means there are 2000 documents and 28389 features because we need to count *__FileID__* and *__CLASS__*. Figure 1 shows the first 5 rows of data in the processed array.

	story	man	unnatural	feelings	pig	starts	opening	scene	terrific	example	...	barkers	verges
0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	...	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	...	0.0	0.0
2	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0
3	1.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	...	0.0	0.0
4	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0

5 rows × 28391 columns

Figure 1. Snippet of Processed Array

In our project, we did the test only on these 2000 documents to see the performance of various machine learning models. Therefore, we split the data into training set with 80% data and validation set with 20% data. The library we used is the *train_test_split ()* from sklearn *model_selection*. After the splitting, the training data has a shape of (1600, 28389) and validation data has a shape of (400, 28389).

After we have obtained the processed training data and validation data, we can start to build machine learning models from it. The models we selected for our project included Logistic Regression, Decision Tree, Random Forest, SVM, Naïve Bayes, and SGD. Also, because we have learned the boosting method from class, we implemented AdaBoost as well. Again, running in Jupyter was pretty slow, we ran all the models locally and saved the results.

2. Result Comparison and Analysis

In this selection, our team has chosen six different algorithms to train these data and measure their performance.

2.1 Logistic Regression

Here we trained Logistic Regression Model to classify pos and neg reviews. We first created the model using sklearn.linear_model.LosgisticRegression(). Then logreg.fit() is utilized to fit data and corresponding labels with the model. We showed the training accuracy of perfect 1.0 and validation accuracy of 0.89, from which, the overfitting problem is indicated. Logistic Regression Model is relatively simple to use and easy to understand. With high accuracy and a little cost of computation, it is an ideal choice to perform machine learning.

2.2 Multiple Naïve Bayes

Naïve Bayes Model is trained using MNB(). We received the 0.99 for training accuracy and 0.9325 for validation accuracy. Naïve Bayes Model not only runs in simple way, it also provides the ideal result with highest validation accuracy among all the models we employed in the project. The running time of multiple Naïve Bayes is surprisingly short so that high efficiency is a great advantage for this model.

2.3 Single Decision Tree with Adaboost

To begin with, Decision Tree Model is employed with criterion of ‘entropy’ and other default parameters. Since decision tree is capable of going through maximum purity in each node, we got training accuracy of 1.0. The noise in the sample is so loud that the Single Decision Tree Model captures more features of noise than valuable ones (low validation accuracy of 0.685), leading to the model fit more to the noise in the training set and severely resulting in overfitting.

To get better performance on Single Decision Tree Model, we tested out which value is optimal for a particular parameter through loop. Creating a new Decision Tree Model with manually searched parameters (criterion = "entropy", max_depth = None, max_leaf_nodes = 125, min_samples_leaf = 2, min_samples_split = 60), a brand-new set of accuracy is presented: 0.82 for training accuracy and 0.6625 for validation accuracy. From the result, we were aware that the overfitting is improved in the cost of low validation accuracy.

Using AdaBoost() to Decision Tree Model

We then trained an AdaBoost classifier using the Decision Tree Model above as its base estimator. This model gives us a better validation accuracy (0.845) than Decision Tree Model which is a great improvement. Even if it is still faced with overfitting, this new model is able to predict the general trend of test data, which overweighs the main drawback.

2.4 Random Forest with Adaboost

Random Forest consists of numerous weaker trees which avoids the overfitting. We first trained Random Forest Model with criterion of ‘entropy’ and n_estimators of 100, and obtain results: 1.0 for training accuracy and 0.855 for validation accuracy. Then parameters used by Random Forest Model is adjusted ('max_depth': None, 'max_leaf_nodes': None, 'min_samples_leaf': 1, 'min_samples_split': 5) through the same way as Decision Tree Model above. We succeeded in increasing validation accuracy from 0.855 to 0.88.

Using AdaBoost() to Random Forest Model

We decided to optimize Random Forest Model by adding boost. Random Forest is used as base estimator of AdaBoost classifier and 100 as our number of estimators. We received a 0.988 training accuracy and a testing accuracy of 0.8775, which informs us that no improvement occurs. Hence, adding boost cannot do any contribution to the performance of Random Forest Model here.

2.5 Support Vector Machine

In this part, we train a model of support vector machine (SVM) because in the project we need to divide the sentiment of the reviews into two classes (pos and neg), which kind of problems SVM is good at. And SVM can use different kernel functions to map in high dimension space. This meets the requirement that the model we train has so many features.

After the first train, we can get the basic model with the default parameters and we have measured the performance of the model. The result is that training accuracy is 0.988125 and validation accuracy is 0.8775.

Considering the influence of different parameters including the choice of kernel function on training, we need to change the parameters of the model to improve performance. After trying

different combinations of parameters, we find the best parameters are kernel function is “rbf”, penalty coefficient C is 4 and gamma is 0.001. With the parameters, we can find that C is much larger than the default value and gamma is quite small. And the result of the performance is that training accuracy is 0.985625 and validation accuracy is 0.8975. Compared with the performance of the first SVM model, training accuracy decreases by 0.003 and validation accuracy increases by 0.02. This means the new parameters have improved the problem of overfitting.

2.6 Stochastic Gradient Descent

Gradient Descent is an optimization algorithm based on existing classification algorithms. In this part, we try to find the influence of Gradient Descent on the sentiment of the reviews. Considering that the amount of training dataset is large and the data frame is quite sparse, we choose stochastic gradient descent (SGD) to measure the performance.

In the first training, I choose the loss function is “modified_huber”, which is smoothed soft-margin linear SVM. And the result of the model is that training accuracy is 0.995 and validation accuracy is 0.88. And we also change the related parameter alpha to improve the performance of this model. When alpha is 0.1, the performance is the best. The new result of the model is that training accuracy is 0.99875 and validation accuracy is 0.9025. Compared with the performance of the first SGD model, training accuracy decreases by nearly 0.004 and validation accuracy increases by 0.0225. This means the performance has been improved.

3. Future Work

In the project, we set the DataFrame with training data and test data together to ensure that they can share the same features with each other. Our team only use part of the training data to train the models and these training data only have 25 percent of the whole effective words that are all the features. As a result, if we set the DataFrame with training data and test data separately, they may have different features and we will fail to test the model. Based on this problem, we establish the DataFrame with training data and test data.

However, the lack of features can decrease the performance of the training models and affect the accuracy of the results of test data. So, in the future, we will train the model with all the training data to get all the features. And we will generate the data from the test files by collecting the effective words according to the feature list of training data.