Movie Review Sentiment Analysis Report

Chenzi Zhang, NetID chenziz2; Zhaoxu Qiu, NetID zhaoxuq2

4/16/2019

Introduction

In our project, we are interested in analyzing the positive or negative sentiment of Movie Review from IMDB. The goal is to build a binary classification model based on the original 50,000 rows and 3 columns in `Project2_data.tsv`. The 3 train data sets we use to build models have 25,000 rows with 3 columns for each. The 3 test data sets we used to examine accuracy have 25,000 reviews with 3 columns. In every data set, Column 1 "new_id" is the ID for each review, Column 2 "sentiment" is the binary response and Column 3 is the review we will use as features.

In this report, we use three models. The direct Lasso model reduces more work on generate new features before fitting a model and efficiently conduct predicting process with high accuracy. The Naive Bayes model needs XGBoost process to generate new features, resulting in prediction not as efficient as lasso. The LDA model is built based on features selected before Lasso, performing with relatively higher accuracy compared to Naive Bayes. These three models have AUC much more than 0.90 and even Lasso can gain AUC much larger than 0.96 with the highest efficiency.

Method

We choose to use three models: Lasso, Naive Bayes + XGBoost and LDA. Before calling these three models, we need to build vocabulary and construct DT matrix. We divide data into three training/test splits. Then we use the third training split and pick the top 3000 words by the ordered magnitude of their t-statistics to trim the vocabulary. In sth loop, we use the trimmed vocabulary to clean the DT matrix, get the new `dtm_train` and `dtm_test` for sth split, build model, predict the probability and classify the result and finally do comperation between predicited class and the actual class of `sentiment` in test data by calculating AUC. For certain split, we will get three txt file for three model containing 25000's probability of having `sentiment` = 1 as results. Based on these results, we can compare the performaces of three models.

Model 1: Logistic regression model with Lasso:

Before calling models, we need to trim the vocabulary. There are three steps:

Step 1: Clean html tags, and split data into training and test, and then use R package `text2vec` to built vocabulary and construct DT matrix (maximum 4-grams). In this step, we use `i`, `me`, `my`, `myself`, `we`, `our`, `ours`, `ourselves`, `you`, `your`, `yours`, `their`, `they`, `his`, `her`, `she`, `he`, `a`, `an`, `and`, `is`, `was`, `are`, `were`, `him`, `himself`, `has`, `have`, `it`, `its`, `of`, `one`, `for`, `the`, `us` and `this` as stopwords and use function `create_vocabulary()` to create the original vocabulary. Use function `prune_vocabulary()` to prune the original one. Finally, use function `create_dtm()` to obtain `dtm_train` and `dtm_test` with close to 30000 features which are bigger than the sample size.

Step 2: Switch to a screening method and pick the top 3000 words by the magnitude of their t-statistics. In the result of step1, some of the selected variables, i.e., terms/words with non-zero logistic regression coefficients are difficult to interpret. Some words may be meaningful, e.g., `great` or `awful`, but they are mixed with a large amount of hard-to-explain words, so we do the two-sample t-test on the third training/test split. Since `dtm_train` is a large sparse matrix, we use commands from the R package `slam` to efficiently

compute the mean and var for each column of `dtm_train` and calculate the corresponding t-statistics. Finally, order the words by the magnitude of their t-statistics, pick the top 3000 words.

Step 3: Save the 3000 chosen words in a file `myvocab.txt`. Use the same vocabulary on all three training/test splits to clean the DT matrix and get the new `dtm_train` and `dtm_test` for each split.

Use the function `cv.glmnet()` to find the lambda that minimizes the cv error, and then we use the 1se lambda to predict. Finally, we get the estimated probability of having `sentiment` = 1 from Lasso model and then caculate the AUC.

Model 2: Naive Bayes Based on XGBoost

We choose to use boosting trees to generate new nonlinear features and then do discriminate analysis. To build this model, we have two parts of process - `XGBoost` and `Naïve Bayes`:

Process 1: Use `XGboost` to generate new features. There are two steps:

- Step 1: Fit 1800 trees of depth = 2. The 1800 trees end up using 1365 terms/words (in the thrid split). We use `XGBoost` to generate `xgb.train` which is named `bst`. By tunning, we choose the parameters with the best performance. Here, we have `nrounds = ntrees = 1800`, `max_depth = 2`,`subsample = 0.5` and `nthread = 2`. Then, we input model `xgb.train` into `xgb.model.dt.tree` function, gaining the the model we need. In this boosting trees model, we have 1800 trees. Each tree has 2^2=4 leaves. So the 1800 trees correspond to 1800 categorical variables, each having 4 levels. By selecting unique features which are not leaves, we can obtain the words to be used in generate new features.
- Step 2: Generate design matrices for new 1800 categorical features. By using command `xgb.create.features`, we generate binary design matrices (dummy coding) for training and testing. In the third split, the new feature matrix has 5217 columns.

Process 2: Fit a `Navie Bayes` model using new features. There are two steps:

- Step 1 Training: Obtain the distribution from training data (mean, standard deviation and weight of class) assuming data satisfying a Gaussian/Normal distribution. By calculating the mean and standard deviation for every level of each new feature, we can generate mean matrix and standard deviation matrix. The number of rows for these two matrices is the total number of levels in new feature train data matrix. The number of columns for these two matrices is the number of features for new feature train data matrix. In addtion, by calculating the percentage of class 0 in training data, we obtain the frequency/probability for class 0.
- Step 2 Predicting: Based on the optimal decision rule,

$$\underset{k}{\text{argmax}}\mathbb{P}(Y = k|X = x) = \underset{k}{\text{argmax}}\pi_k f_k(x)$$

  we use test data x and for loop to calculate

$$log(\pi_k f_k(x)) = log\pi_k + \sum_{i=1}^{p} log f_{ki}(x_i))$$

  where $k = 0$ or $k = 1$. From this result, we conduct prediction based on:

$$\frac{1}{1 + \pi_1 f_1(x) - \pi_0 f_0(x)}$$

  At the end of the step, by `roc` function and coresponding test column data, we can get the AUC of our prediction, checking the accuracy of our model.

Problem Solving:
We also deal with the zero standard deviation problem in the last Step. Beacause the values for new features

might be the same, our sd matrix has zero values. These zero values will produce `NaN` as denominators in $\frac{(x_{ki}-\mu_{ki})^2}{2\sigma^2}$. To deal with this problem, we use `if` function to jump the zero value condition seperately for each class. Since, with zero sd, we can assume it has probability 1, leading to zero log value. An advantage of this idea is that we avoid removing all zero sd columns unreasonablly in `mysd` matrix.

Model 3: Linear Discriminant Analysis:

Use `dtm_train` and function `lda()` to do the linear discriminant analysis and make prediction for `dtm_test`. Finally, we get the estimated probability of having `sentiment` = 1 from LDA model and then caculate the AUC.

Results

Evalution on Prediction Accuracy

Table 1: AUC

model	split1	split2	split3
Lasso	0.9617078	0.9623652	0.9603678
NB+XGBoost	0.9103324	0.9091808	0.9066478
LDA	0.9488070	0.9512139	0.9482446

In general, logistic regression model with Lasso has the best prediction performance and can produce AUC equal to or bigger than 0.96 over all three test data. LDA is second-best and the results of AUC are approximately 0.95 over all three test data. However, Naive Bayes with XGBoost model is the worst and AUC of three model just exceeds 0.9. In addition, the first two test splits perform better than the remaining one training/test splits for all these three models. The selection of top 3000 words may largely influence the result for third split. Overall, Lasso model fits the dataset best among these three models. LDA model fits better than Naive Bayes and XGBoost.

Evalution on Efficiency

Table 2: Running Time (Unit: Mins)

model	split1	split2	split3
Lasso	1.053422	1.148159	1.109986
NB+XGBoost	26.57378	26.21393	27.35376
LDA	13.04404	12.92226	13.68386

My computer system is iMac which has 3 GHz Intel Core i5 Processor, 16 GB 2400 MHz DDR4 Memory. After comparing running time of these three models, we find NB+XGBoost has the longest running time and Lasso has the shortest running time. Since logistic regression model with Lasso has the best prediction performance and can produce AUC equal to or bigger than 0.96 over all three test data and the AUC of NB+XGBoost and LDA don't reach the benchmark 0.96, we can conclude that logistic regression model with Lasso has the best efficiency – best performance and shortest running time.

What we found interesting

By reviewing the 3x3 AUC table in Accuracy Evaluation section, we find that split3 has the lowest accuracy. Since we generated the `myvocab.txt` based on split3, we even more strongly wonder why it happens. After going back to the corresponding sentiments for top 50 postive words and top 50 negative words, we find an interesting thing. By averaging the sentiment value corresponding to one word, if positive takes the most parts, the result should larger than 0.5, vice versa. In this way, we try 100 times and find that the most important words of positive and negative sentiment has different sentiment percentages. Here is what we get from our codes:

Table 3: The percentage of positive sentiment for top1 words

Top1 Words	split1	split2	split3
"bad"	0.2520967	0.2568189	0.2591563
"great"	0.6801352	0.6797667	0.6709904

In this table, "bad" is the most important word in negative sentiment expression. In split3, 25.92% reviews which contain "bad" has positive sentiment. Split2 has a percentage of 25.68% and split1 has a percentage of 25.21%, which are lower than the percentage in split3. This means that the negative "bad" selected from split3 have better performance while classifying split1 and split2.

In the meanwhile, "great" is the most importnat word in positive sentiment expression. In split3, 67.10% reviews which contain "great" has postivie sentiment. Split2 has a percentage of 67.98% and split1 has a percentage of 68.01%, which are higher than the percentage in split3. This means that positive "great" selected from split3 have better performance while classifying split1 and split2.

Therefore, the reason why split1 and split2 have better accuracy than split3 can be found from the vocabulary we choose.

Conclusion

For logistic regression model with Lasso, we don't need to tune many parameters manually and just use cv to select lambda 1se to make prediction, which is time-saving. More importantly, the prediction performance seems the best among these three model and Lasso also has the shortest running time. For NaiveBayes+XGBoost, the steps to preprocess is much more than other models. We need to tune several parameters manually such as `max_depth`, `ntrees`, `subsample`, `nrouds` to build xgboost model and obtain new features, which is time-consuming. And the prediction performance seems the worst and it has the longest running time. For LDA, the prediction performance seems good enough but it has limitation of dimensions. After trimming the vocabulary, it's suitable to use LDA method in our case.

To sum up, it's better to choose logistic regression model with Lasso to do binary classification especially when we don't need to do much data preprocess and expect a more accurate prediction. It's time efficient. It's better to chosse NB+XGBoost when we need many steps to fix our original data such as obtaining more new features. It's better to choose LDA when we don't want to do much parameter tuning and the data matrix is not sparse enough.