Stock market prediction

王嬰(Student ID:18210980068)

Abstract—This program is to predict whether the price of a stock will go up or down based on financial news. The Naïve Bayes model is asked to be implemented that maximizes the jointly likelihood of word features and target labels for text classification. This project mainly has three steps: processing data, generating features and training classifier. For preprocessing data, I use Jieba. To better cut the words, I've added two financial texts to Jieba and removed stopwords. As for generating features, I try mainly two methods to extract the useful variables by using tf-idf, positive words and negative words. In the third step, I train six models containing Naïve Bayes, SVC, Logistic Regression and so on. Moreover, the model parameters were saved into pickle files so that we can load directly when testing. Finally, the best accuracy was reached at 71% with NuSVC and the best F1 score was reached at 80% with SVC.

I. Introduction

Text classification is a fundamental problem in natural language processing(NLP).There are numerous applications of text classification such as document organization, news filtering, spam detection, opinion mining and computational phenotyping. An essential intermediate step for text classification is text representation. Traditional methods traditional methods represent text with hand-crafted features, such as bag of words and tf-idf. In this project, I choose tf-idf to represent the text. Moreover, I have tried to use positive and negative words for extracting more features. Another necessary step is training classifier. Although deep learning models have been widely used to classification, including convolutional neural networks(CNN)(Kim 2014) and recurrent neural network(RNN) such as long short-term memory(LSTM). Only the basic machine learning methods such as Naïve Bayes, Logistic Regression and SVM are implemented in this project.

II. Theory Foundations

Naive Bayes is a probabilistic classifier, meaning that for a document d, out of all classes c∈C the classifier returns the class $\hat{c}$ which has the maximum posterior probability given the document. Use the hat notation ˆ to mean "our estimate of the correct class".

$$\hat{c} = \underset{c \in C}{arg\max} P(c|d) \tag{2.1}$$

The intuition of Bayesian classification is to use Bayes'rule to transform Eq.2.1 into other probabilities that have some useful properties. Bayes' rule is presented in Eq. 2.2; it gives us a way to break down any conditional probability P(x|y) into three other probabilities:

$$\mathrm{P(x|y)} = \frac{P(y|x)P(x)}{P(y)} \tag{2.2}$$

Therefore the Eq.2.1 can be changed to Eq.2.3:

$$\hat{c} = \underset{c \in C}{arg\max} P(c|d) = \underset{c \in C}{arg\max} \frac{P(d|c)P(c)}{P(d)} \quad (2.3)$$

P(d) doesn't change for each class; We can conveniently simplify Eq. 2.3 by dropping the denominator P(d). Thus, we can choose the class that maximizes this simpler formula:

$$\hat{c} = \underset{c \in C}{arg\max} P(c|d) = \underset{c \in C}{arg\max} P(d|c)P(c) \quad (2.4)$$

We thus compute the most probable class $\hat{c}$ given some document d by choosing the class which has the highest product of two probabilities: the prior probability prior probability of the class P(c) and the likelihood of the document P(d|c):

$$\hat{c} = \underset{c \in C}{arg\max} P(f_1, f_2, \dots, f_n|c)P(\mathrm{c}) \quad (2.5)$$

Naive Bayes classifiers make assumptions: this is the conditional independence assumption that the probabilities $\mathrm{P}(f_i|\mathrm{c})$ are independent given the class c and hence can be 'naively' multiplied as follows:

$$\mathrm{P}(f_1, f_2, \dots, f_n|\mathrm{c}) = \mathrm{P}(f_1|\mathrm{c})\mathrm{P}(f_2|\mathrm{c}) \dots \mathrm{P}(f_n|\mathrm{c}) \quad (2.6)$$

The final equation for the class chosen by a naive Bayes classifier is thus:

$$c_{NB} = \underset{c \in C}{arg\max} P(c) \prod_{f \in F} P(f|c) \quad (2.7)$$

III. Experiments

The environment is Ubuntu 16.04.4 LTS.

Datasets can be three parts: 'News.txt','Train.txt' and 'Test.txt'. There are 7954 samples in train data and 3000 samples in test data.

There are mainly three parts in my program.

（1）Preprocessing Data

- Take 'train.txt' file as an example, make every stock be connected with the related title and content of news. Use jieba to cut each sentence, To better cut the words, I've added two financial texts to Jieba and then removed stopwords. Finally the trainset has format as ([[title1],[content1],[title2],[content2],...],'+1'), testset has the same format as trainset. In order to accelerate the speed to read data, I change the processed data into pickle fille and saved as 'trainset.pkl' and 'test.pkl' respectively.

- Construct a corpus containing all the tokenized words in order to calculate the term frequency–inverse document frequency value(tf-idf value). I choose the top 400 words as variable and saved as a list. Furthermore, not only 400 feature words were used, but also 600 and 3000 tf-idf feature words were tested to compare the accuracy and F1 score.(600 tf-idf feature words correspond to the 600 most large tf-idf value)

(2) Generating Features

- Loop each stock's corresponding news: if the word in tf-idf words list, then features['contain the word'] = True. Conversely, features['contain the word'] = False. Therefore,

I finally get a feature matrix which has 400 columns(if I choose 400 tf-idf feature words).

- Moreover, the positive words and negative words were used to extracting more features.

For example, if the word is positive, then features['word'] = w where w is a weight hyper-parameter . Conversely, if the word is negative, features['word'] = -w. The default value of w is 1.

{'股份': True, '万股': True, '减持': True, '增持': 1, '净利': True, '公司': True, '亿元': True, '点评': True, '封板': True, '净利润': True, '同比增长': False, '万元': False, ' 恶性':-1…,}

The first method to generate features were saved in 'prediction.py'. Then I explore how to add the sentiment words to the features so that I can get higher accuracy and F1 score. The second method were written in different python files including 'load_data.py', 'prepare_data.py', 'model.py' and 'main.py'. The files order is the running order.

(3) Train Classifier

- I trained the NaiveBayes model and BernoulliNB, LinearSVC, NuSVC, SVC, Logistic Regression models in Sklearn. I also save the parameters of models into files so that I can load them directly when using that.

- For each model, I show the most informative features which was saved in '4_result/nohup_M1.out'. Use the 'score.py' file to evaluate the models and compare results which will be explained in Section IV.

Overall, I have already saved the above three tuned model parameters in '4_result', '7_result2' and '8_result3' corresponding to tf-idf-400, tf-idf-300 and tf-idf & senti-words.

IV. Results

This section I will compare the results among different models and different features.

(1) TF-IDF 400 features

Table 1: The evaluation Results of tf-idf 400

	NaiveBayes	BernoulliNB	LinearSVC	NuSVC	SVC	LR
Recall	0.59	0.59	0.83	0.81	**0.95**	0.81
Precision	0.68	0.68	0.69	**0.73**	0.67	0.69
Accuracy	0.56	0.56	0.66	**0.70**	0.67	0.65
F1 score	0.63	0.63	0.75	0.77	**0.79**	0.75

SVC model has the best recall value and F1 score. It get the best precision and accuracy when choose the NuSVC. Comparing the two models, I will choose NuSVC when I prefer to higher accuracy. However, the SVC is not a bad choice when higher F1 score is needed.

(2)TF-IDF 3000 features

Table2: The evaluation Results of tf-idf 3000

	NaiveBayes	BernoulliNB	LinearSVC	NuSVC	SVC	LR
Recall	0.56	0.56	0.71	0.82	**0.97**	0.74
Precision	0.70	0.70	0.73	**0.74**	0.67	**0.74**
Accuracy	0.57	0.57	0.64	**0.70**	0.68	0.67
F1 score	0.62	0.62	0.72	0.78	**0.80**	0.74

Change the number of tf-idf feature words from 400 to 3000, it performs a little better.
It shows two many features may lead to a sparse feature matrix, which is not beneficial for training model and improving results.

(3)TF-IDF &Senti-words

Table 3: The evaluation Results of tf-idf & Senti-words

	NaiveBayes	BernoulliNB	LinearSVC	NuSVC	SVC	LR
Recall	0.90	0.60	0.72	0.83	**0.98**	0.75
Precision	0.67	0.71	0.73	**0.75**	0.66	0.74
Accuracy	0.65	0.59	0.65	**0.71**	0.67	0.67
F1 score	0.77	0.65	0.72	0.78	**0.79**	0.74

Overall, I change 3 times of the features. Finally, the accuracy of NuSVC can reach at 71% when using tf-idf 400 words and adding sentiment words to justify the context.

V. Conclusion

Only use tf-idf words as features can reach at a good accuracy. Although I have tried to add more features such as the sentiment words, it didn't perform too much better which confuse me. I think it may be that the using more tf-idf words can represent the stock news content well. Finally, the best accuracy was reached at 71% with NuSVC and the best F1 score was reached at 79% with SVC. As for the assignment for text classification, there are two parts to construct model. One is text representation, another is training classifier models. Text representation methods contain n-grams, tf-idf, word2vec and so on. This program only use the tf-idf method. Training classifier models include the traditional machine learning algorithm which were used in this project , deep neural network methods such as CNN, RNN, Attention and Graph convolutional methods such as text GCN and SGC. In the future work, I will try to use the deep learning methods to increase the performance.

VI. Reference

[1] Tushare pro
[2] Words of finance
[3] Stopwords
[4] Kim, Y. 2014. Convolutional Neural Networks for Sentence Classification.In EMNLP, 1746-1751.
[5] Wu F, Zhang T, Souza A H D, et al. Simplifying Graph Convolutional Networks[J]. 2019.