Newspaper Article Classification Contest

Final Report

Terry Li — *tianweil@andrew.cmu.edu*
Tao Yu — *taoyu@andrew.cmu.edu*

Abstract

This is the final report for the Newspaper Article Classification Contest project. The content includes how we retrieve the features from articles, which classifier we used, how to train the classifier and the performance of our classifier.

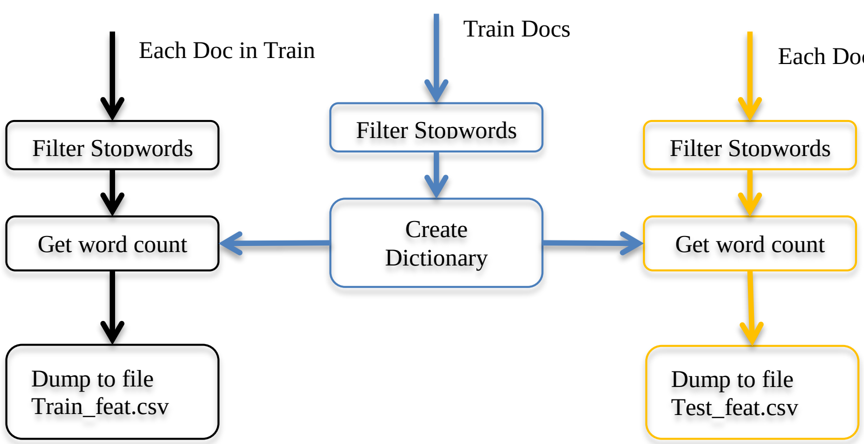
1 Project members

Terry Li (Andrew ID: tianweil) and Tao Yu (Andrew ID: taoyu)

2 Feature Extraction

2.1 Basic Approach

We use the word count as the features of our classifier. The process to extract features for train articles and test articles is as below.



For each article:

1. Parse the articles and separate it to a bunch of words
2. Remove the stop words
3. Count the word count for each word in the dictionary for each article
4. Use the word count as the features for classification

After filtering the stop words, we have 38800 words left. Then we use these words to classify the articles.

2.2 Word Stemming

To reduce the number of features, we leveraged the public Porter Stemming Algorithm [6] to pre-process the documents. After stemming, the number features reduced to 25313.

2.3 TF-IDF

To reduce the number of features, we also implemented the tf-idf feature selection algorithm. The original tf-idf algorithm does not take the class information into account. It only calculates the score of a certain word in a bunch of documents. The original equation is

$$idf(word) = \log\left(\frac{doc\ count}{doc\ count\ that\ conatins\ word}\right)$$

To introduce the class information into this equation, we used a modified version of tf-idf:

$$idf(word|c) = \log\left(\frac{doc\ count * doc\ count\ in\ class\ c\ and\ contains\ word}{doc\ count\ that\ conatins\ word}\right)$$

For the tf, the equation is the same:

$$tf(word|c) = \frac{word\ count\ in\ class\ c}{all\ word\ count\ in\ class\ c}$$

Because the word count for a certain word in a certain class could be 0, and the doc count that contains a certain word in certain class also could be 0, we introduced add-1 smooth in the process of calculating both tf and idf.

After calculating tf-idf for every feature, we selected 2500 features with the highest scores to be the features of our classifier.

3 Naive Bayes Classifier

For Naïve Bayes classifier,

$$C_{NB} = argmax\ P(x_1, x_2, \dots x_n|c)P(c) = argmax\ P(c) \prod P(x_i|c)$$

We use the word count of each to calculate the $P(x_i\ |c)$ for every class c.

In the beginning, the equation we used is:

$$P(x_i\ |c) = \frac{N_{ci}}{N_c}$$

In which N_{ci} is the number of times word x_i appears in the documents in class c.
N_c is the total number of words appears in documents in class c.

Because a certain word may have zero count in a class, then the $P(x_i|c)$ is 0. This will cause the posterior probability will become 0. Because there are always some words have 0 count in some classes. It makes all posterior probability for every class is 0, which makes our classifier useless.

To solve this problem, we introduced add-1 smooth to the probability calculation. That is we add 1 count for every feature (different word). To make the total probability of all feature still 1, we need to add the number of features to the total word count.

Then the equation becomes:

$$P(x_i \mid c) = \frac{N_{ci} + 1}{N_c + n}$$

In which n is the number of features.

For the Prior probability of each class, we just use MLE to calculate them.

After get the $P(c)$ and $P(x_i|c)$ for every word and every class, we finished training the classifier.

4 Multinomial Naive Bayes

Multinomial Naive Bayes models the distribution of words in a document as a multinomial. The likelihood of a document is a product of the probability of the words that appear in the document.

So the probability of one document belongs to class c is:

$$P(c|doc) = P(c) \prod P(x_i|c)^{f_i}$$

Where the f_i is the frequency of word i in doc.

To simplify the calculation, we use the log likelihood instead of original probability.

$$L(c|doc) = logP(c) + \sum_{i=0}^{n} f_i * \log P(x_i|c)$$

Then we can classify the doc into the class that has the highest $L(c|doc)$

5 Modified Naïve Bayes Classifier

According to paper [5], the performance of Naïve Bayes Classifier is limited by the prior probability $P(c)$ and the probability of word i in each class c $P(x_i|c)$. We tried the modification introduced to Naïve Bayes Classifier introduced by paper [5].

5.1 Complement Naive Bayes (CNB)

Instead of calculating the probability of word i in class c:

$$P(x_i \mid c) = \frac{N_{ci} + 1}{N_c + n}$$

We calculate the probability of word i that is not in class c:

$$P(x_i \mid \tilde{c}) = \frac{N_{\tilde{c}i} + 1}{N_{\tilde{c}} + n}$$

Then the log likelihood of doc belongs to class c is:

$$L(c|doc) = logP(c) - \sum_{i=0}^{n} f_i * \log P(x_i|\tilde{c})$$

5.2 Transformed Naïve Bayes (TNB)

Instead of using of $P(x_i|c)$ as the probability of word i belong to class c, we can also use the tf-idf score of the word i.

Then the log likelihood of doc belongs to class c is:

$$L(c|doc) = logP(c) + \sum_{i=0}^{n} f_i * \mathrm{score}_{\mathrm{tfidf}}(\mathrm{i}, \mathrm{c})$$

6 Support Vector Machine

We implement a simple version of support vector machine algorithm with kernel parameters of linear, Poly and RBF. For convenience, we choose Quadprog as dual problem solver. We do experiments on linear SVM as well as with RBF kernel as below:

6.1 Linear SVM

SVM are shown to handle feature redundancy well, because of the reason that we have 38863 features. It is reasonable to use linear SVM. For features, we use wordcount and TF-IDF features to train the linear SVM model. Every feature will contribute to the improvement of the linear SVM model.

C is essentially a regularization parameter, which controls the trade-off between achieving a low error on the training data and minimizing the norm of the weights.

The C parameter tells the SVM optimization how much we want to avoid misclassifying each training example. For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points. For very tiny values of C, we should get misclassified examples, often even if our training data is linearly separable.

We adjust C from 2^(-10) to 2^5, and find that the accuracy is highest when C is 2^(-10).

6.2 SVM with RBF kernel

For SVM with RBF kernel, we replace natural product with kernel function. So, at the end, w*would look like,

$$w^* = \sum_{i \in SV} h_i y_i \Phi(x_i)$$

and hence,

$$\langle w^*, \Phi(x) \rangle = \sum_{i \in SV} h_i y_i \langle \Phi(x_i), \Phi(x) \rangle$$

Similarly,

$$b^* = \frac{1}{|SV|} \sum_{i \in SV} \left(y_i - \sum_{j=1}^{N} \left(h_j y_j \langle \Phi(x_j), \Phi(x_i) \rangle \right) \right)$$

and our classification looks like

$$c_x = \text{sign}(\langle w, \Phi(x) \rangle + b)$$

After fine-tuned our learner parameters on evaluation data, we test our fitted SVM's performance on the testing data that's previously unseen in the training and fine-tuning stages. Because of time limitation, we use the grid search: Set a range of feasible values for C, for instance C in [2, 2^2, 2^3, 2^4, 2^5, 2^6]. Set a range of feasible values for Gamma [1, 10, 100, 1000]. Then we use grid search to find the best C and Gamma.

6.3 SVM Experimental Result

Feature Vector Kernel	Kernel Type	Prediction Success Rate	Kernel Variable
wordcount	Linear	66.88%	C = 2^(-10)
TFIDF	Linear	66.94%	C = 2^(-10)
wordcount	RBF	67.01%	G = 0.5
TFIDF	RBF	67.05%	G = 0.5

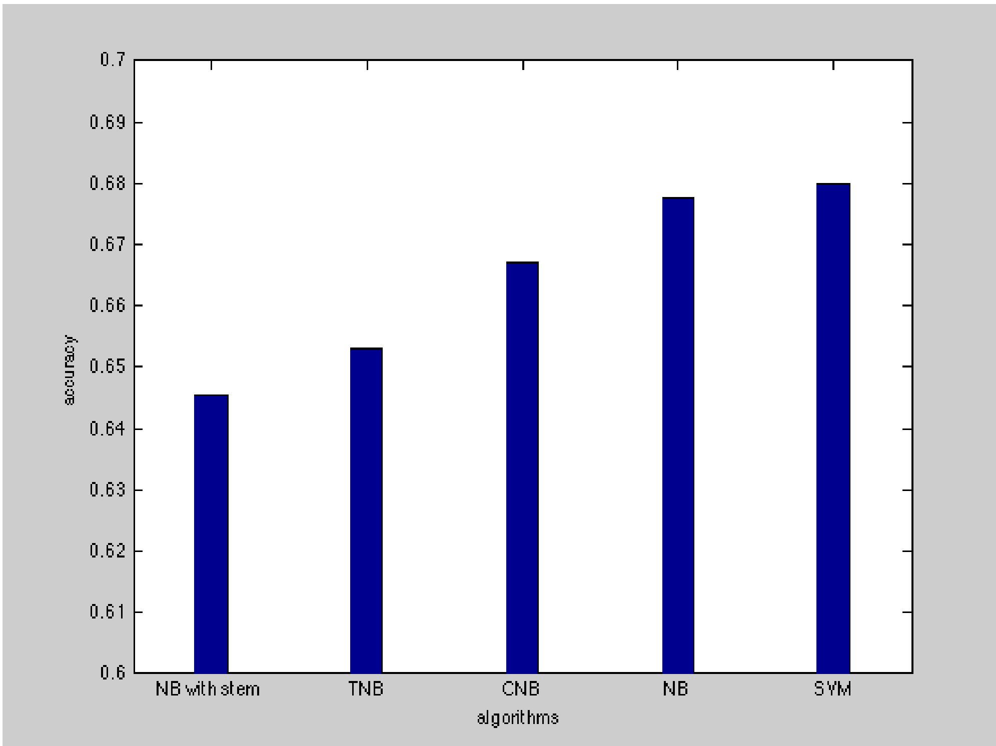
Compare wordcount with TFIDF features, TFIDF is a little better than original wordcount information.

Alternatively, we can use n-cross validation to estimate our SVM's performance. If we have a limited amount of annotated texts, n-cross validation is recommended as it takes advantage of using all the data we have.

The experimental results show that SVM consistently achieve performance on text classification tasks. SVMs eliminate the need for feature selection, making the application of text classification easier. Meanwhile, SVM has the advantage of robustness.

7 Performance

Feature selection method	Accuracy
Na ïve Bayes with stemming	64.55%
Transformed Na ïve Bayes	65.3%
Complement Na ïve Bayes	66.7%
Basic Na ïve Bayes	66.75%
SVM	67.05%



Performance Analysis:

Based on our classification result, Na ïve Bayes Classifier does not have overfitting problem, so the more features the better. That’s why the basic Multinomial Na ïve Bayes Classifier has a better performance than Na ïve Bayes with tf-idf.

Dealing with text classification, we have to consider very many (more than 30000) features. SVMs use overfitting protection, which has the potential to handle these large feature spaces.

Although there are original 38663 features，chances are that features ranked lowest still contain useful information. As a result, aggressive feature selection may lead to information loss. Na ïve Bayes and SVM with all the features achieve good performance.

When we convert texts into vector representations, they contain only few entries which are not zero. Thus the vectors are very sparse, SVMs is well suited for problems with sparse instances. Meanwhile, most of the text classification problems are linearly separable. SVMs with (linear, RBF) is good at finding such a separator.

References

[1] http://en.wikipedia.org/wiki/Latent_Dirichlet_allocation

[2] http://nlp.stanford.edu/IR-book/html/htmledition/feature-selection-1.html

[3] Joachims, Thorsten. Text categorization with support vector machines: Learning with many relevant features. Springer Berlin Heidelberg, 1998.

[4] Yang, Yiming, and Jan O. Pedersen. "A comparative study on feature selection in text categorization." ICML. Vol. 97. 1997.

[5] Jason D. M. Rennie, Lawrence Shih, Jaime Teevan, David R. Karger "Tackling the Poor Assumptions of Naive Bayes Text Classifiers"

[6] http://tartarus.org/martin/PorterStemmer/index-old.html

[7] http://www.cs.cornell.edu/people/tj/publications/joachims_98a.pdf

[8] http://www.cs.cornell.edu/People/tj/svmtcatbook/