20 Newsgroup Document Classification Report

Definition

Project Overview

Document classification or document categorization is a problem in library science, information science as well as computer science. The task is to assgin a document to one or more classes. This may be done "manually" (or "intellectually") or algorithmically, and the documents to be classified may be texts, images, music, etc. Each kind of document possesses its special classification problems. Documents may be classified according to their subjects or according to other attributes (such as document type, author, printing year etc.)[1].

This project will focus on algorithmical methods, exactly machine learning algorithms which are widely used in information science and computer science. There are many classification algorithms such as KNN, Naive Bayes, Decision Tree and etc, all of which have their advantages and disadvantages.

There are many public text dataset online for classification, here, I will apply classification algorithms on famous 20_newsgroup dataset from CMU Text Learning Group Data Archives, which has a collection of 20,000 messages, collected from 20 different netnews newsgroups. The news will be classified according to their contents.

Problem Statement

The classification of 20_newsgroup dataset is a **supervised classification problem**, there are news of 20 categories, each piece of news belongs to one category, the goal is to **extract proper features and build an effective model to assign each piece of news to the correct category**.

I will explore the dataset in the beginning on the training part, then extract useful keywords and build vectors of features from the texts of news, based on those vectors I will use several classfication methods to do classification, compare the efficiency of these classifiers on the testing data and choose one as final model. Finally I will validate the performance of the classifiers on different dataset and parameters.

In this project, if I take all the news into account, it will be a huge computing and memory demand for my computer, so as to speed computing, **I will only select 4 categories of them intially**, and validate the model on the rest topics.

Metrics

As it is a multi-class classification problem with thousands of samples, I will use two metrics here:

- accuracy: the proportion of correct labels that we made if we apply our model to the training dataset. Ideal accuracy is 100%.

$$accuracy = \frac{\sum_{i=1}^{n} I(y_i = \hat{y}_i)}{n}$$

(y_i , the label of the *ith* sample, $\hat{y}_i$, predicted label if the ith sample, if they are equal, I=1, otherwise, I=0)

- timing: the time the algorithm takes to do classification, a scalable algorithm is expected to work fast because corpus usually includes thousands or more text files.

Analysis

Data Exploration

The dataset can be acquired online directly with built-in Python scikit-llearn tools. There are 20 categories of news in all, here I just choose 4 categories to do analysis, analogously, the other categories can be also handled in this way.

The dataset has been splitted into training part and testing part, each part consists of news of different categories. There are 2374 samples in training set, and 1580 in testing set, and each sample has been labeled by researchers. They belong to four categories involved with computer science, baseball, medical, religion. Let's look at the content of one text.

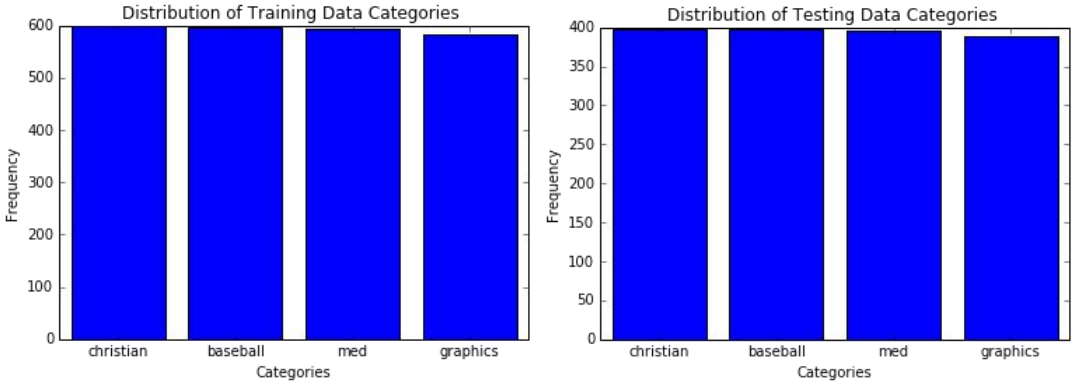
From: lyford@dagny.webo.dg.com (Lyford Beverage)
Subject: Re: Notes on Jays vs. Indians Series
Distribution: na
Organization: Data General Corporation, Research Triangle Park, NC
Lines: 22
In article <1993Apr13.202037.9485@cs.cornell.edu>, tedward@cs.cornell.edu (Edward [Ted] Fischer) writes:
|> In article <rudyC5Fr3q.1CL@netcom.com> rudy@netcom.com (Rudy Wade) writes:
|> >In article <C5FMxD.2pM@cs.dal.ca> niguma@ug.cs.dal.ca (Gord Niguma) writes:
|> >>reference to history because he certainly didn't have the best season for
|> >>second basemen in history. He probably didn't even have as good a season as
|> >>Alomar last year.
|> >
|> >What? Do you have some measure (like popularity in Toronto doesn't count)
|> >that you are basing this statement on?
|>
|> Uh, yes. Baerga has a lot of flash, but Alomar was the better hitter
|> last year.
|>
|> BATTERS BA SLG OBP G AB R H TB 2B 3B HR RBI BB SO SB CS E
|> BAERGA,C .312 .455 .354 161 657 92 205 299 32 1 20 105 35 76 10 2 19
|> ALOMAR,R .310 .427 .405 152 571 105 177 244 27 8 8 76 87 52 49 9 5
|>

This is fascinating. You say that Alomar was the better hitter last year, and immediately follow that up with numbers showing that Baerga had a better year. The only category that I see which shows an advantage for Alomar is OBP.

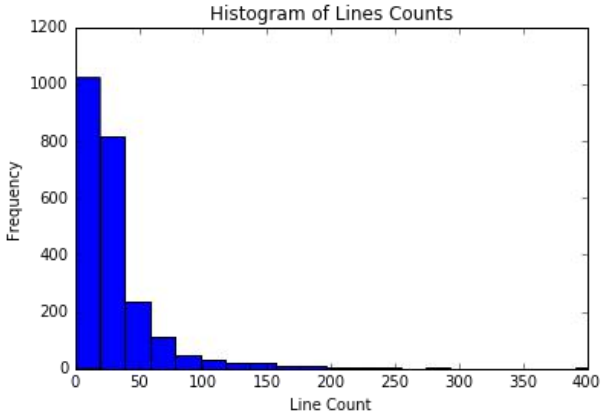
It looks like an email, there are a subject, an organization, an email address and a main body, the subject and content can contain some useful words in terms of classification, for example, this article mentioned 'basemen', 'hitter', terms often refer to baseball. Aside from these, there are many signs such as punctuation, digits in the content, which may be redundant for classification.

Exploratory Visualization

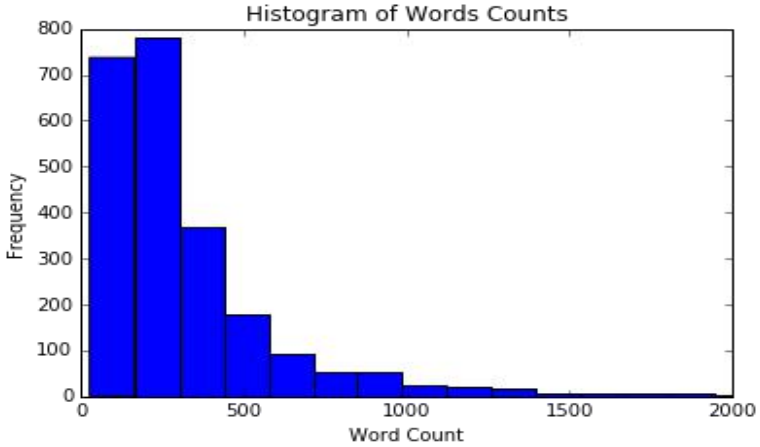
I have checked some statistical characteristics of the corpus through barplots, histograms and etc.



Each category has almost the same articles, both the training and testing data are balanced. We need not reshuffle and split them.



Most articles have less than 50 lines, the median is 22, so these news consist of quite short articles indeed. Note: there are just 1 article with 0 line, it is not necessary to remove it.

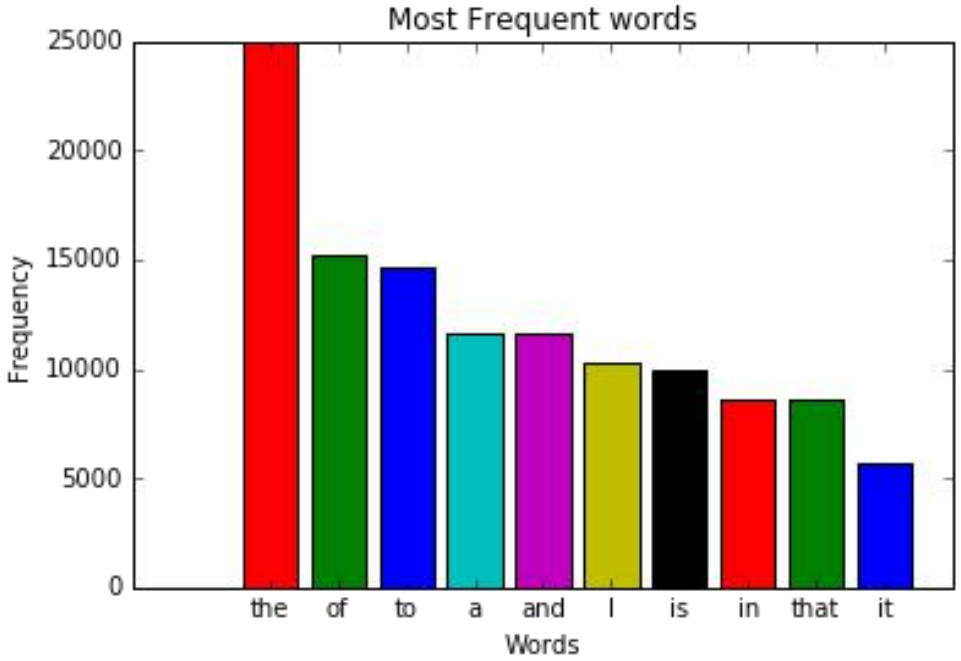


I have tokenized those articles using TextBlob package[2], namely split the texts into words using textblob package. Most articles have less than 500 words, max word count is 11002,

min word count is 26. There are no obvious outliers. Note: the header of each article accounts for over 20 words, so the contents should contain more than 20 words, otherwise the articles are not useful. Actually there are only 3 articles with less than 30 words. It is not necessary to remove them.

The total number of unique words can be 671564, a very huge number if we treat all of them as features, we need extract only a proper proportion of words as keywords.

Anyway, in this project, it is likely that the features(selected words) outnumber samples, so it's also a problem of high dimension.



I have also found out the most frequent words in the given corpus. It is clear that the most frequent words are stop words such as 'the', 'of', commonly seen almost in every article, but without much meaning in term of classification, they can be removed to reduce the dimension of our problem.

Algorithms and Techniques

First of all, I need extract features from the news, these features can be words, sentences or phrases or combination of them all. I only considered using single words as features allowing for model complexity and the computation capacity. I used Tf-Idf algorithm to extract features rather than simple word frequencies. The tf-idf value increases proportionally to the number of times a word appears in the document, but is offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general[3].

This problem has over 2000 training samples with four categories, a multi-class classification problem with high dimension. KNN is not suitable here because of computing efficiency in

high dimension. I will try *Multinomial Naive Bayes*[4], *Decision Tree*[5] and *Feedforward Neural Network*[6] methods, they can work even on large dataset, the characteristics are as following:

- Multinomial Naive Bayes
  Pros: only requires a small amount of training data for classification, highly scalable, used very often in text retrieval.
  Cons: Oversimple for certain data, it needs assumption that the value of a particular feature is independent of the value of any other feature.
- Decision Tree
  Pros: easy to interpret the results, scalable
  Cons: the accuracy may be not excellent, prone to overfitting.
- Feedforward Neural Network
  Pros: can produce state-of-the-art results on various tasks
  Cons: computationally expensive, prone to overfitting

I used Python *scikit-learn* package[7] to do the feature extraction, as well as Naive Bayes learning and Decision Tree learning. The corpus of news do not provide default parameters for *NB* and *Decision Tree* models, the parameters should be set in advance, and need tuned if necessary.

As for neural network, I used *tensorflow*[8]. But for *deep learning*, the number of features can determine the size of weights of the first layer, as for the hidden layer, the size should also be set in advance.

Benchmark

Many researchers have applied classifiers on 20_newsgroup dataset. A group in Stanford University applied Stanford Classifier to it, and their accuracy was around 0.85 on all the 20 topics. Meanwhile they listed results of other experiments, the accuracy were also below 0.90[9].

I just selected 4 from the 20 topics, my model is less complex than that of Stanford Experiment, so I expect my accuracy higher than them. Here we consider two factors: bias and variance.

- Bias: the ideal model should assign each testing article to the right category, I set the accuracy threshold as 90% on testing dataset.

- Variance: the model performance should not change on different dataset if we have trained it, try to avoid overfitting. Ideally the accuracy doesn't vary much with parameters and dataset size. Here I limit the accuracy difference between training and testing data is below 8%.

Methodology

Data Preprocessing

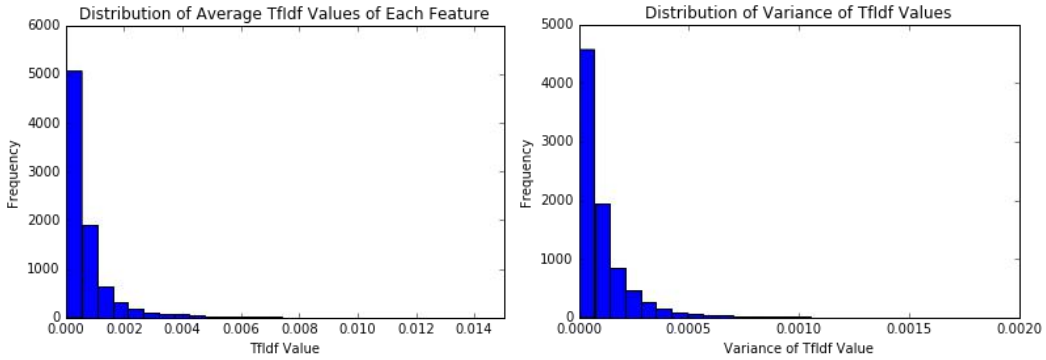
I wrote a function and removed punctuation, digits and turned letters into lowercase before feature extraction, after that there were 35257 words in all, but many words were repetitions of letters like 'aaaaa' , perhaps typos.

I set a minimum frequency for the words extracted, those words appeared just once or twice might be typos or less useful, and ran the feature extracting function again, this time there were only 7626 words. And the feature matrix was (2374, 7626).

After feature extraction, the vector looked like:

```
array([[ 0.        ,  0.        ,  0.09797326,  0.        ,  0.        ,
         0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
         0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
         0.        ,  0.38527571,  0.        ,  0.        ,  0.        ,
         0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
         0.        ,  0.        ,  0.        ,  0.        ,  0.        ]])
```

Actually most values are zeros in the matrix, so *sklearn* package transformed them into sparse matrix to save space. In later analysis, I need transform them into normal matrix again.



From the figures above, both the means and variances of each column(feature) were in similiar scales, **no need to normalize them.**

Implementation

I used NB, Decision Tree and deep neural network methods to train the data, the metrics were accuracy and timing. The parameters of NB and Decision Tree were default. First I imported Multinomial Naive Bayes classifer from *sklearn.naive_bayes,* trained them on the extracted feature matrix and predicted the categories on testing feature matrix. Then, I imported tree.DecisionTreeClassifier() from *sklearn,* trained them on the extracted feature matrix and predicted the categories on testing feature matrix.

I also designed a feedforward neural network classifier based on *tensorflow*. There are over 7626 features, single-unit perceptrons are only capable of learning linearly separable patterns[6]. I doubted those articles were linearly separable, so I created a hidden layer, processed the output of the hidden layer by *Rectifier Linear Units*[10]. In order to speed up learning, I applied **stochastic gradient descending method** , as there were just 2374 training samples, I set min-batch size as 8, the number of iteration was 251, so it could use almost all the samples. Initially, I **set the size of hidden layer as 512 ,** it didn't work well, and I increased it to 1024, 2048, finally at 4096, the testing accuracy could reach 0.88 after 251 iterations.

All the computing time were recorded. The result turned out to be in this way: the more simple, the better it performed. The **accuracy of *NB* classifier was 0.954**, and the computing time was 0.006s. The a**ccuracy of *Decision Tree* classifier was 0.757** and the computing time was 0.485s. The **accuracy of *neural network* was 0.88** and the computing time reached 63s. It seemed *NB model* worked pretty well on the dataset in terms of both accuracy and timing.

From the computing results, I realized that overfitting happened when using deep neural network, the training accuracy has been 100%, after 100 iterations, yet the testing accuracy was not stable and less than 90%.

Refinement

I used grid search method to tune parameters of Naive Bayes, exactly the values of ***alpha*** . As the timing difference were very obvious, I would not discuss them later. I set the *alpha* ranged from 0.01 to 1, and when alpha = 0.01, the model had a best accuracy of 0.961 on testing data, an improvement of default one whose accuracy was 0.954. It seemed that smaller alpha could lead to a better accuracy.

I also tuned the **maximum depth**, *max_depth* parameter of Decision Tree classifier. I set the *max_depth* as (3, 4 ,6 ,8, 10). Overfitting happened if I tried to improve training accuracy, the grid search method chose *max*_depth=10, while the accuracy on the testing data was less than 0.6. Decision Tree model didn't work well in this problem.

I have tuned the size of hidden layer above, more perceptrons meaned higher accuracy on training dataset. In order to prevent overfitting in neural network, I **added regularization** part to the loss function. I took the all the weights into consideration, and tuned the coefficient of regularization as well. **When the coefficient was 0.000001,** the regularization part was the same scale as the original loss function. I also increased the number of iteration to 801. Finally, the accuracy improved and reached 0.9.

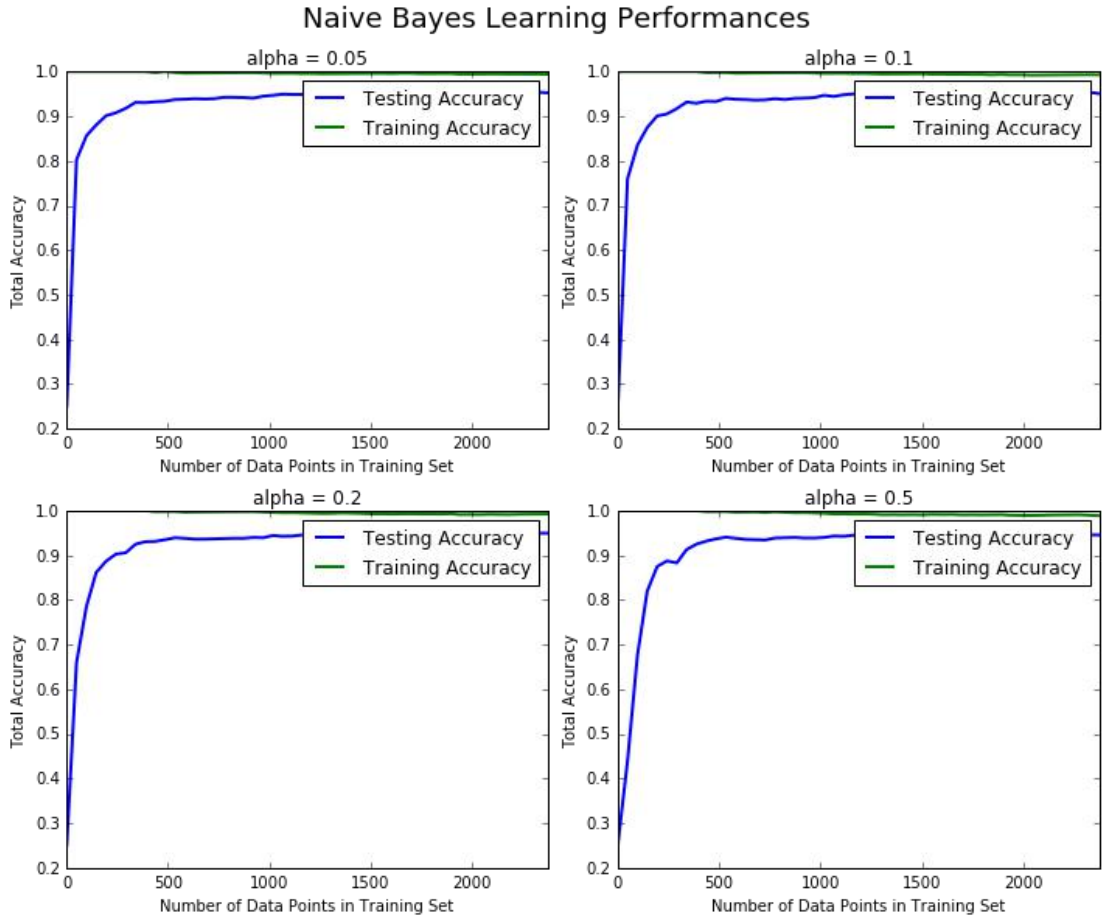
Results

Model Evaluation and Validation

Mutinomial Naive Bayes

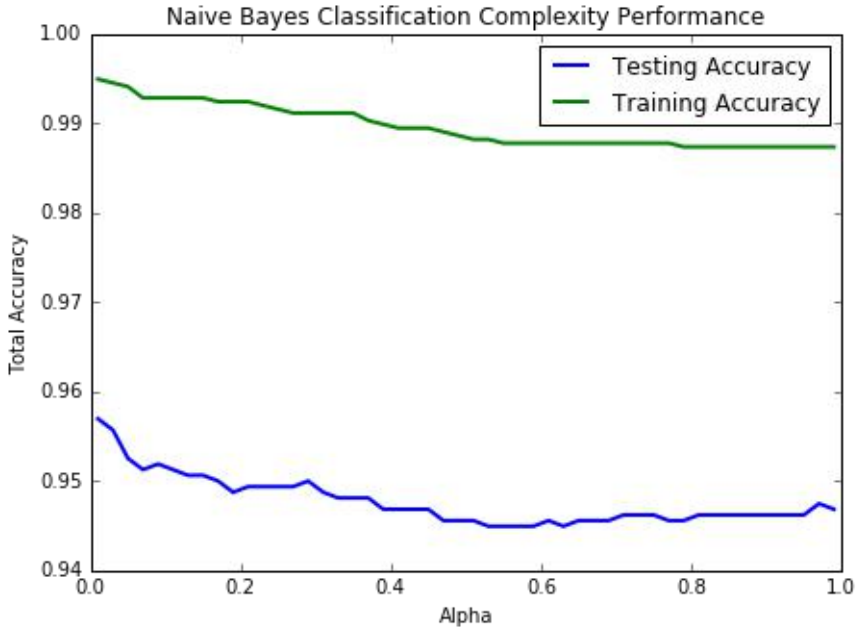
In this part, I took a look at several models' learning and testing accuracy on various subsets of training data, to check bias and variances of the models. In addition, I investigated NB classifier with an increasing alpha parameter on the full training set to observe how model complexity affects learning and testing accuracy.

Based on my previous study, I wrote functions to calculate the performance of several models with varying sizes of training data. The learning and testing accuracy for each model were then plotted.



Those figures indicate that NB classifier works well on those dataset in terms of variance and bias. The accuracy grows as the number of samples increases initally, When the number of samples are over 500, the accuracy become quite stable. The training accuracy reaches almost 100%, and testing accuracy can reach as high as 95%. No overfitting here. The classifier is robust in terms of sample scales.

I wrote a function to calculate the performance of the model as model complexity increased as well. The learning and testing accuracy were then plotted.



As the alpha increases, the accuracy of training dataset and testing dataset go down slowly. Then after 0.4, the accuracy seem to be stable. The testing accuracy is around 95%, and when alpha is close to zero, it has large accuracy. The differences between training dataset and testing dataset are small, no overfitting here.

Above all, NB classifier is quite robust to different training dataset and alpha. **I will set alpha= 0.01 as the parameter of our final NB model.**

Feedforward Neural Network

I made a table of *feedforward neural network* performance,. The mini batch training reached 100% after 200 iterations, and the testing accuracy increased gradually, but not quite stable. Overfitting was still obvious although regularization was considered. Perhaps with more samples, it would converge better.

feedforward Neural Network Performance Table

Iteration	0	100	200	300	400	500	600	700	800
Training Accuracy	0.25	0.375	0.625	1	1	1	1	0.875	1
Testing Accuracy	0.254	0.487	0.837	0.796	0.832	0.891	0.848	0.889	0.906
Loss	54.09	49.29	18.68	12.05	12.05	12.05	12.04	12.8	12.05

Justification

As analyzed above, Multinomial NB model performed well on the classification tasks in terms of both bias and variance. The accuracy of classification could reach 95%, and the variance was also small, that meaned the model generalizes well.

Feedforward neural network could have accuracy of 0.9, but the variance was not ideal. It was not quite stable even after 801 iterations, and really time-consuming. Therefore, **I preferred *Multinomial NB* model to *Feedforward Neural Network model*.**

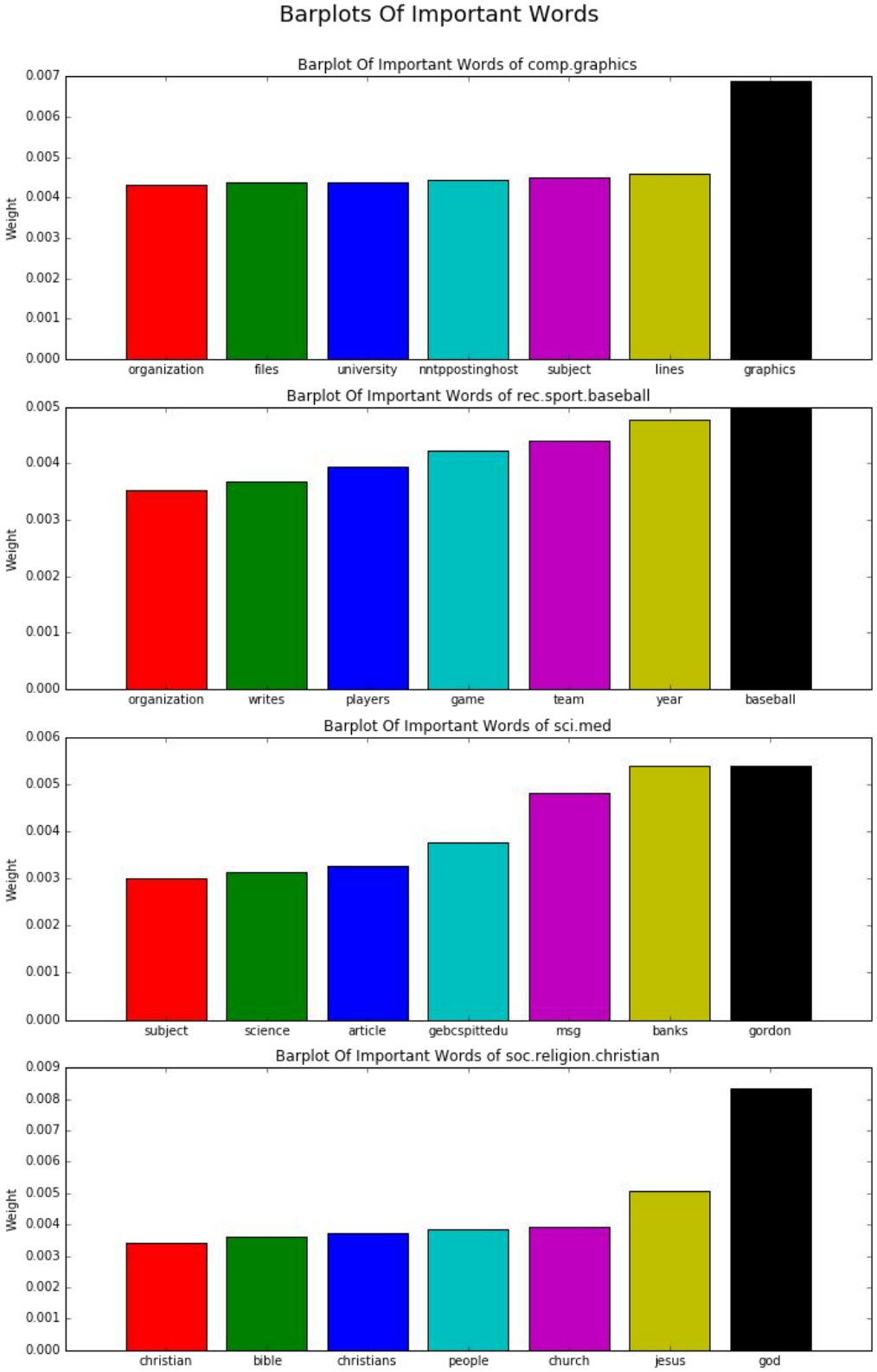
In our original dataset, many dataset of other categories remain left, I could use them to validate my model. This time I would chooseanother four topics(‘rec.motorcycles', 'talk.politics.misc', 'comp.windows.x', 'sci.space') to validate NB model.

The model still performed quite well. The accuracy was above 0.96 and the timing was still very small. So Multinomial Naive Bayes model could classify these articles precisely and quickly.

Conclusion

Free-Form Visualization

I extracted word weights for each topic from Mutinomial NB model, and listed the largest weights along with corresponding words as important features for each topics.



The barplots showed that certain important words were indeed related to specific categories. For example, in the first figure, 'graphics' is very important for 'comp.graphics' topics, and in the last figure, 'God', 'Christians', 'Jesus' point to topics of religion. These also proved that the Multinomial model works well on this dataset.

Reflection

This project is to solve a multi-class document classification problem of hign dimension. I downloaded 20newsgroups dataset, selected 4 topics of them in order to speed up computing, extracted features using Tf-Idf algorithms from the corpus. I applied Naive Bayes, Decision Tree and Feedforward neural network classifier to the dataset, and found that NB performed best in terms of accuracy and timing on the testing dataset.

Then I tuned parameters of NB and Decision Tree classifier with grid search method, refined neural network model. Decision Tree performance didn't improve much, the accuracy of neural network could be over 0.9. Afterwards, I analysed peformance of neural network model on different iteration numbers, as well as NB model by exploring models' learning and testing accuracy on various subsets of training data, and comparing accuracy as changing values of alpha.

Mutinomial NB classifier has balanced bias and variance, it generalized well on different dataset if the sample number is over 500, and the variance of classifier is not sensitive to alpha. Feedforward neural network had a higher variance after 801 iterations. In addition, I validated the NB classifer on other catergories of dataset, it still worked well. Finally, I listed weights and words of important features of each category, which were consistent with real world meanings.

Improvement

In this project, I did not consider different forms of the same words, for example, 'Christian', 'Christians', 'christian' are the same in meaning, if I combine them into one word, I can reduce the redundance of features and improve efficiency of the model.

Furthermore, if I took phrases such bigrams or trigrams into consideration, i.e., 'big data', 'machine learning', it would be likely to increase the classification accuracy, but require more computing capabilities and memories.

Reference

[1]Document Classification: en.wikipedia.org/wiki/Document_classification
[2]TexBlob Package: textblob.readthedocs.io/en/latest/
[3]Tf-Idf: en.wikipedia.org/wiki/Tf%E2%80%93idf
[4]Naive Bayes Classifier: en.wikipedia.org/wiki/Naive_Bayes_classifier
[5]Decision Tree: en.wikipedia.org/wiki/Decision_tree

[6]Feedforward Neural Network: en.wikipedia.org/wiki/Feedforward_neural_network
[7]scikit-learn userguide: scikit-learndevelopers, Release0.16.1
[8]Deep Learning Class: cn.udacity.com/course/deep-learning--ud730
[9]NLP Stanford Classifier: nlp.stanford.edu/wiki/Software/Classifier/20_Newsgroups
[10]Relu function: en.wikipedia.org/wiki/Rectifier_(neural_networks)