StackExchange Question Topic Classifier

Haiyu Yao <hyao4@ncsu.edu>, Yihuan Dong <ydong2@ncsu.edu>, Yao Lu <ylu31@ncsu.edu>

1. Project Idea and Description

Nowadays, Q&A communities are becoming more and more popular. Almost all of those communities allow users to tag the topic (category) of their questions. The main purpose of having topic tags is to help users find posts which focus on a specific area from the extremely large question pool. Despite such benefits, users may still feel troublesome to add tags to their posts. One of the reasons is that people don't know which tag is appropriate for their posts. Another reason for this can be that people just simply feel it troublesome to think of and add tags to their posts. Our project intends to address this problem by building a classifier to find appropriate topics for questions based on their text content. This classifier can potentially be used as a tag recommendation engine or topic tag generator in real scenarios.

We used about 22000 questions (attributes include title, excerpt and topic) from 10 different topics as training and testing data. To preprocess our raw data, we use several technology including document-term matrix and word pruning. Then we will apply Naive Bayes, SVM (Support Vector Machine), Decision Tree, ANN (Artificial Neural Network) and k-NN (k-Nearest Neighbor) algorithms on the data. Together with cross validation, we then compare the performance (prediction, error rate, recall, F-Score) of these algorithms. To boost our classifiers' performance, we introduced LSA (Latent Semantic Analysis) and Ensemble Method.

2. Material and Methods

2.1. Data Specification:

The data used for building the classifier is a sample data of questions on StackExchange which is available on https://s3.amazonaws.com/hr-testcases/845/assets/training.json [1]. The language of these questions are restricted in English. There are more than 22000 data records of 10 topics. Each record is a question entity represented in JSON format. The content of a record consists of topic, question and excerpt. All of the three fields are text data. An example of the data is listed below:

*{**"topic"**:"electronics","**question**":"What is the effective differencial effective of this circuit","**excerpt**":"I'm trying to work out, in general terms, the effective capacitance of this circuit (see diagram: http://i.stack.imgur.com/BS85b.png). \n\nWhat is the effective capacitance of this circuit and will the ...\r\n "}*

2.2. Methods of Transforming Text Data into Structured Data:

Text data is unstructured data and is difficult to analysis directly, so we need find a way to transform text data into structured data. The common methods are bag-of-words and word-frequency.
Bag-of-word transforms a document (which is a question entity in our case) into a document-term matrix. Each row in the matrix is a document-term vector of which each attribute is a boolean representing if a word exist in the document. Word-frequency also transforms a document into a similar

document-term matrix. Each attribute in a document-term vector is an integer representing the occurrence frequency of a word. For example, there are two questions:

1. *"How to learn data mining?"*
2. *"To learn data mining, do I need to learn R?"*

The transformed records will look like the records shown below:

Bag-of-words:

	HOW	TO	LEARN	DATA	MINING	DO	I	NEED	R
Q1	T	T	T	T	T	F	F	F	F
Q2	F	T	T	T	T	T	T	T	T

Word-frequency:

	HOW	TO	LEARN	DATA	MINING	DO	I	NEED	R
Q1	1	1	1	1	1	0	0	0	0
Q2	0	2	2	1	1	1	1	1	1

The computing cost of transformation is close between the two methods, but the later one reserves more information, so we chose word-frequency method to transform text data into structured data. This task is accomplished by using R package "tm".

Most of APIs in R for creating document-term matrix is document-oriented, so we also wrote a python program to split the original JSON document that contains all questions into multiple documents of which each contains only one question.

2.3. Methods of Data Pre-processing:

Although the word-frequency transformation gave us a structured representation of text, it also caused the curse of dimensionality. The dimension of a document-term vector is 35024 before any preprocessing, which is so large that nearly impossible to do any analysis within reasonable time. To reduce the dimension, we stemmed word, transformed all words to lowercase, removed punctuations and stopwords. We also discovered the distribution of words, in terms of their frequency, is very skewed. This is illustrated in the figure 2.1. There are 19401 words that occur only once in all questions. Only 1464 words occur more than 50 times and 842 words occur more than 100 times. Intuitively, we know if a word is too frequent or too rare, the word usually is not special to the topic we are talking about, which means it does not help much for classification. Therefore, we tried to filter out low frequency words and high frequency words, and we used cross validation to find a good upper bound and lower bound. The candidate lower bound are 50, 100, 150 and the candidate higher bound are 700, 1000, 1100. The only exception is that for k-NN we chose 70 instead of 50 and 120 instead of

150 as a candidate lower bound and, because k-NN runs too slow to get a result within reasonable time when the low bound is set to 50 and we get “too many ties error” for lower bound = 150. The reason for choosing these numbers as bounds is that we found the improvement of accuracy of classification is not notable beyond this range.

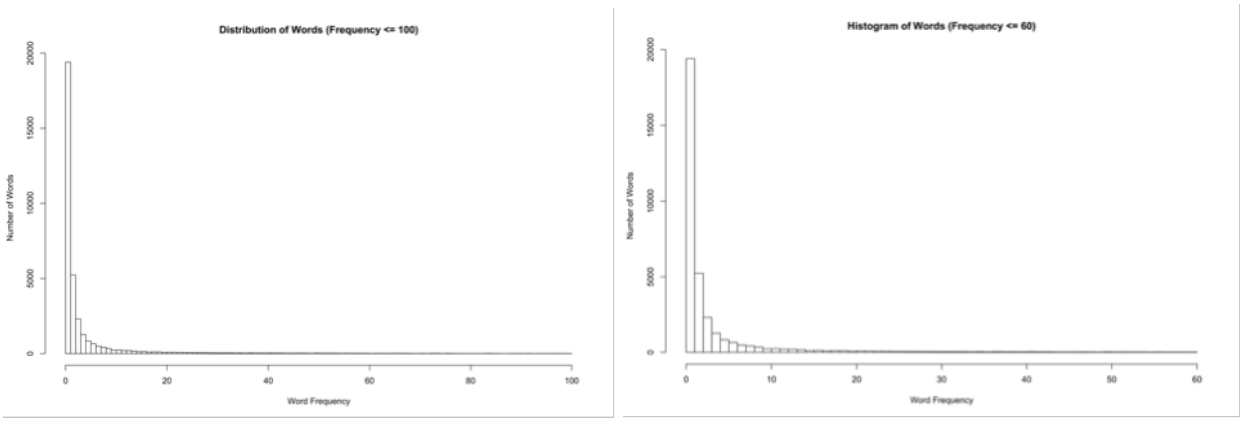

Figure 2.1 Distribution of Words

2.4. Methods of Building Classifiers and Evaluation:

We used k-NN (knn of class package in R), decision tree (rpart of rpart package in R), SVM (svm of e1071 package in R), Naive Bayes (naiveBayes of e1071 package in R) and artificial neural network (nnet of nnet package in R) to build different classifier based on the same data. We chose these algorithms because they are commonly used and have been shown useful in classifier research.

Some of these algorithms require parameters to achieve best performance, so we tried out different parameters: K = 1, 5, 10 for k-NN, size = 5,6,7,8,9,10 for ANN. Size for ANN means the number of units in hidden layers. We chose these values as candidates because when we choose bigger values the running time increased too much to build a classifier in reasonable time. To evaluate which parameter setting gave us the best performance, the evaluation is done by 5-fold cross validation on each attempt of parameters including the lower bound and upper bound of filtering out unuseful words.

After first attempt to build classifier based on the data preprocessed in the way mentioned in the previous section, we found the performance of these classifier was not good enough. Then, we introduced Latent Semantic Analysis method (LSA, also called Latent Semantic Index i.e. LSI) to transform data further to boost performance. The R package we used is “lsa”.

“Latent semantic analysis (LSA) is a robust unsupervised technique for deriving an implicit representation of text semantics based on observed co-occurrence of words.”[3] It also creates a document-term matrix of which each entry a_{ij} corresponds to a weight instead of frequency of word j in document i. The weight is called TF*IDF weight. It is calculated by multiplying Term Frequency and Inverse Document Frequency of a word. TF weight is a measure of the frequence of a word in a particular document, so it is often called local weight. IDF weight is a measure telling whether a word is common or rare across all documents, so it is often called global weight. There are lots of weight functions can be used for the two kinds of frequencies. We chose binary TF weight (lw_bintf of tm

package in R) and inverse frequency IDF weight (gw_idf of tm package in R). Binary TF weight is a binary value which is 1 if a word occurs in a document or 0 if no so. Inverse frequency IDF weight for a word i in a document j is $\log(N/N_{ij})$ where N is the total frequency of all words and N_{ij} is the frequency of the word i in the document j. After the document-term matrix is created, a SVD is performed on it so that we get a new matrix which is called LSA vector space. The matrix can give a better but implicit representation of text semantics.

After LSA is performed on our data, we ran another around classifiers training and did similar cross validation to evaluate the performance. The result is much better than the first attempts. Nevertheless, we did not stop here. We tried to make use of ensemble method to continue boost performance. We intended to use the predicted results of different classifiers to train a random forest classifier. Each first round classifier's result is an attribute of the training data for the random forest classifier and the actual topic label is the label of the new training data. We divided data into 3 parts: 90% data as training data for first round classifiers, 9% data as test data for first round classifiers (also training data for random forest classifier) and 1% data as testing data for the final classifier. Since the running time of this method is too long to do a cross validation (ANN already took 4 whole days to complete 5-folder cross validation), so we just use hold-out method to evaluate the performance of the final classifier. Since the random forest often offers robust performance, we think the final test result is still trustable. The figure 2.2 illustrates the architecture of the final classifier.

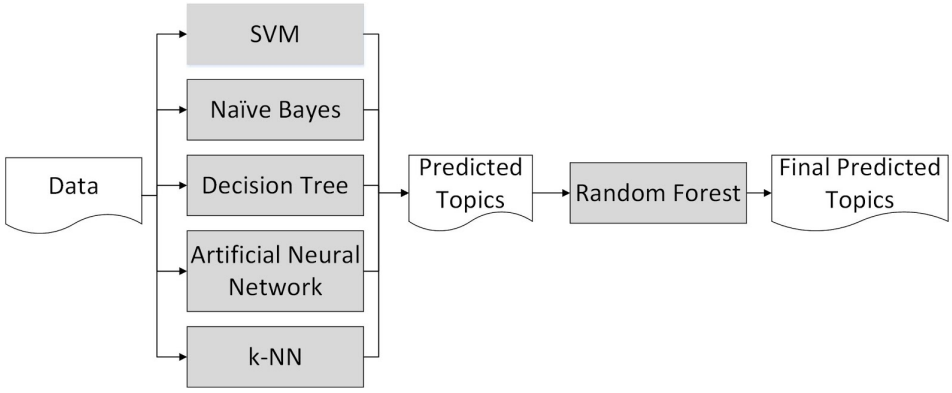

Figure 2.2 Architecture of the classifier using ensemble method

3. Results

Algorithm (word frequency bound)	Error Rate	Error Rate Variance	Precision	Recall	F-Score
SVM (L=50, H=1000)	0.176	1.50E-06	0.972	0.844	0.903
NB (L=150, H=700)	0.931	1.04E-07	NaN	0.101	NaN
DT (L=100, H=700)	0.416	2.72E-06	0.846	0.580	0.688
ANN (L=50, H=1100) (Size=6)	0.094	2.33E-04	0.945	0.920	0.932
KNN(L=120, H=1100) (K=1)	0.297	5.53E-06	0.725	0.704	0.714

Table 3.1 Evaluation metric for 5 methods before using LSA

In the evaluation part, we used Error Rate, Error Rate Variance, Precision, Recall and F-Score to measure the performance of SVM, NB, DT, ANN and k-NN. In the first column lies the method name and a list of attribute used in cross-validation. These attribute values are not selected randomly; they are chosen because they're shown to provide the best performance for the method after experiment with different values. In the tables, L stands for lower bound, H stands for upper bound, size is number of units in hidden layer and k is the number of nearest neighbors for a testing data point considered in classification.

Table 3.1 shows the evaluation result for the 5 methods before using LSA. As we can see from Table 3.1, different method requires different upper and lower word frequency bound to provide the best performance. ANN gave a significantly lower error rate, 0.094, in the tag classification task. SVM also provides an acceptable error rate of 0.176. k-NN has an error rate of 0.297. However, for this task, Decision Tree and Naive Bayes provided unsatisfying result. In particular, Decision tree has an error rate of 0.416, which classify almost half the data points into the wrong class. Naive Bayes had an error rate of 0.931, which is even worse than random selection. When we look deeper into Naive Bayes's result, we found actually it classify all test data to a single topic. And this is also why its precision and F-score are NaN (The dividers are 0 for most topics.).

Algorithm (word frequency bound)	Error Rate	Error Rate Variance	Precision	Recall	F-Score
SVM (L=50, H=1000)	0.092	4.08E-06	0.957	0.893	0.924
NB (L=50, H=1000)	0.114	2.45E-06	0.947	0.893	0.927
DT (L=100, H=1100)	0.158	1.44E-05	0.860	0.840	0.849
ANN (L=50, H=1100) (Size=10)	0.106	1.14E-04	0.935	0.891	0.913
KNN(L=120, H=1000) (K=1)	0.254	3.10E-06	0.725	0.747	0.736

Table 3.2 Evaluation metric of 5 methods after using LSA

Table 3.2 shows the evaluation metric for the 5 methods after using LSA. The optimum upper and lower word frequency bound became wider for Naive Bayes classifier, from 150 - 700 to 50 - 1000. After using LSA to reconstruct the features of original data, half of the methods experienced a great boost in their classification performance. Specifically, the error rate for Naive Bayes dropped from 0.931 to 0.114 and the error rate for Decision Tree dropped from 0.416 to 0.158. However, LSA did not have obvious impact on the performance for ANN and k-NN. The variance of ANN is 1.14E-04, indicating that its performance is not very stable compared to other methods.

		TRUE									
		android	**apple**	**electronics**	**gis**	**math**	**photo**	**scifi**	**security**	**unix**	**wordpress**
Predict	**android**	**535**	22	1	1	0	0	0	3	3	0
	apple	33	**453**	6	3	1	2	0	9	22	2
	electronics	2	4	**477**	10	8	5	11	13	8	0
	gis	1	0	7	**582**	2	0	0	4	9	5
	math	2	2	16	14	**339**	3	8	8	9	7
	photo	3	1	6	4	1	**499**	4	4	0	1
	scifi	4	11	32	2	7	11	**607**	8	7	4
	security	2	11	5	5	4	3	0	**433**	8	3
	unix	21	53	12	18	5	2	0	25	**464**	7
	wordpress	2	1	0	5	3	1	0	6	1	**496**
Error Rate		**0.106**									

Table 3.3 Confusion matrix of the final result using ensemble method

Table 3.3 is the confusion matrix of our ensemble method after one execution. From the table, we can see that we did most of the classification correctly, with an error rate of only 0.106.

4. Discussion

First, to build a good text classifier, data pre-processing is a critical part. Because of the nature of language, to find a good representation is already a difficult task. It is intuitive to try to keep as much information as possible so that we can get better performance, but it also introduces complexity and high dimensionality of data. According to the result of our project, this intuition might backfire the good intention to get better performance. We pruned more than 90 percent words, but still got good enough result. LSA is also a surprisingly powerful tool to get better representation of text. Only by using LSA, we boosted the performance of almost all classifiers except for ANN (which had already gotten good performance before LSA) . LSA also made SVM beat the ANN and get accuracy more than 90%. However, we have to admit LSA also have the weakness such as its implicit meaning. From the value in LSA space, we cannot get a good interpretation of what it represents. LSA is also not a silver bullet, because we saw the performance of ANN was worse after LSA transformation.

Second, different classifier algorithms perform very different for text mining tasks. According to our result, ANN and SVM are good candidates for this task. k-NN is not good at this task no matter whether LSA is used. Naive Bayes and decision tree perform bad without the help of LSA. Moreover, among the two good candidate, if we take into account the training time and performance robustness, SVM is better than ANN. ANN took us 4 whole days to do cross validation which is much longer than what SVM took us. The variance of error rate of ANN is 2 orders of magnitude higher than the one of SVM. We also found the performance of ANN is very sensitive to initial weights. On the other hand, by leveraging LSA, SVM could get better performance than ANN did.

Finally, ensemble method gave a better and robust performance even if the performance of the underlying classifiers varies a lot. From our result, it seems the final performance (error rate = 0.1059663) is worse than the best classifier SVM's (error rate = 0.09243697). However, during the final evaluation, the SVM's performance was actually not that good (error rate = 0.1369809), and none of other classifier beat the final random forest classifier in that evaluation.

References

[1] Samle data source from hackerrank.com
[https://www.hackerrank.com/challenges/stack-exchange-question-classifier]
[2] Provost, Foster, and Fawcett, Tom. Data Science for Business : What You Need to Know about Data Mining and Data-analytic Thinking. Sebastopol, CA, USA: O'Reilly Media, 2013. ProQuest ebrary. Web. 16 April 2015.
[3] Nenkova, Ani, and Kathleen McKeown. "A survey of text summarization techniques." Mining Text Data. Springer US, 2012. 43-76.
[4] Tan, Pang-Ning, Michael Steinbach, and Vipin Kumar. Introduction to data mining. : Pearson Addison Wesley, 2005.
[5] Latent semantic indexing from online Wikipedia
[http://en.wikipedia.org/wiki/Latent_semantic_indexing]
[6] CRAN. tm: Text Mining Package[http://cran.r-project.org/web/packages/tm/index.html]
[7] CRAN. lsa: Latent Semantic Analysis [http://cran.r-project.org/web/packages/lsa/]
[8] Timothy DAuria. How to Build a Text Mining, Machine Learning Document Classification System in R! [https://www.youtube.com/watch?v=j1V2McKbkLo]