Classifying Reddit Post Titles to Subreddit

Rohit Talreja Michael Fang Joseph Baena
{ rtalreja, mjfang, jbaena } @stanford.edu

Introduction

Reddit is an online forum where community members share and comment on text, links, photos, and videos. All posts on Reddit have a text-based title that characterize the post. These titles may be directly or only tangentially related to the content of their posts. Posts on Reddit belong to topical sub-forums called *subreddits*. We used a dataset from a large corpus of the top-1000 posts from the top-2500 subreddits [1].

Task Definition

Our primary objective was to build predictors that accurately classify Reddit post titles to their respective subreddits. This can be thought of as an application of the problem of short-document classification. Reddit post titles are generally short phrases or sentences, which makes the classification task more difficult. Other challenges include misspellings in titles and the use of rare domain-specific words.

Model

We define the problem of classifying Reddit post titles by subreddit as follows:

Learn a predictor **F** that accurately predicts which subreddit a post title belongs to, such that $\mathbf{F(\Phi(x)) = y_i}$ (where $\mathbf{\Phi(x)}$ is the feature vector of **x**).

Input: **x** = post title from Reddit

Output: $\mathbf{y} \in \{ y_1, y_2, \ldots, y_K \}$, where y_i are all the possible subreddits the title could belong to.

Algorithm

Our supervised multi-class classification algorithm is divided into two main phases: feature extraction and classification.

In feature extraction, the program takes in as input an arbitrary number of CSV data files of Reddit posts, where each CSV file contains the post titles from a single subreddit. We used two general types of feature vectors: word and character. For word features, the baseline feature vector used the frequency of each word in the post title. A word was defined as a token delimited by whitespace. For character features, we used a variable size

n-gram similar to the “sentiment” assignment of CS 221. The feature vector was then used in the classification phase of the program.

In the classification phase, we tried three main algorithms to classify post titles: linear classification, Multinomial Naive Bayes, and support vector machines (SVM). In training the predictor for linear classification, the objective is to minimize the generalized hinge loss using stochastic gradient descent (SGD). In prediction, we output the class which yields the highest score, equal to $\boldsymbol{w} \cdot \phi(\boldsymbol{x})$, where $\boldsymbol{w}$ is the weight vector and $\phi(\boldsymbol{x})$ is the feature vector of post title $\boldsymbol{x}$ (this is one-vs-all classification). We implemented this feature extraction and classification method ourselves and added several optimizations to improve accuracy; they will be detailed later.

We did Multinomial Naive Bayes (MNB) using sci-kit learn. In MNB, we calculate weights for each word to each class. These weights are computed using simple frequency counts, i.e. the number of times each word is present in the title of a post belonging to a certain subreddit, divided by the total number of times the word is present in any post title, as well as Laplace smoothing. Then, at test time the likelihood score is calculated as the product of the weights of each word within the document, and the prediction is the class with the highest likelihood score.

We used the SVM implementation provided by LibShortText, which also uses a one-versus-all strategy for training SVMs [7]. The library also provided pre-processing strategies such as bi-gram features, stop-word removal, and stemming, which gave us the best results.

We randomly selected subsets of the the Reddit corpus, and randomly broke those up into two parts: a large group for training and smaller group for testing.

Example

An example of prediction using the linear classifier, showing the value of the weights learned per word and class in the table below.

Input: “if you could work in a national park...”

Class	if	you	could	work	in	national	park	**Score**
Archaeology	-0.25	-0.45	0.35	-0.1	0.2	-0.65	-0.25	-0.65
Outdoors	0.05	0.25	-0.15	-0.35	-0.35	0.8	0.95	**1.15**
MachineLearning	-0.15	-0.4	-0.15	-0.35	-0.55	-0.6	-0.5	-2.6
ArtHistory	0.05	-0.15	-0.15	0.55	-0.75	-0.6	-0.65	-1.7

Figure 1: Example feature vector.

Output: Outdoors

Implementation

Reading in data

We used the same method of reading in files for all classification methods. The input was a set of CSV files, each containing about 1000 posts from the same subreddit. The files contained more information about the posts than we needed, for example: URL, number of upvotes/downvotes, author, date created, and many more. We selected just the post title, and created a JSON object, `{'title': [post title], 'subreddit': [subreddit name]}`. This way we could keep track of the true class of each title. Based on a weighted random number, the title object would be added to the training set (with ~90% probability) or the test set. Originally, every 10th post was added to the test set, while the other 9 were in the training set; this was changed because the sets would always have the same contents run-to-run.

Feature Optimizations

Our feature vectors began as simple bag-of-word representations of each document)post title). In the following, we present several optimizations. We also tried bag of character n-grams.

Removing punctuation (called Opt1 or Optimization1 in the code)

The first optimization was to remove all punctuation from the Reddit post titles (with the exception of apostrophes). We reasoned that punctuation interferes with construction of accurate features, since distinct tokens act as features and keeping punctuation causes a feature like “awesome” to be distinguished from “awesome!”, even though they are semantically the same.

Breaking down contractions (called Opt2 or Optimization2 in the code)

The second optimization is to go through each Reddit post title and expands contractions. For example, “can’t” or “he’s” are converted “can not” and “he is” respectively. We believed that expanding known contractions can help improve the accuracy of our algorithm by making contractions and the pair of words they expand into the same feature. We found that this strategy was not effective at increasing accuracy, see data section for more details.

Removing stop words (called Opt3 or Optimization3 in the code)

Another optimization removes stop or “function” words -- words with little or ambiguous meaning -- from the feature vector. These can include: `{the, a, an, is, or, and, which, but}`, etc. We believed that by removing stop words from our list of

features, we can focus our classification algorithm to predict based on words of potentially substantive meaning.

Stemming and Lemmatization

To counteract the challenge of having short post titles, we implemented stemming and lemmatization. In stemming, we reduce words to their root form using the Porter Stemming algorithm. For instance, the term “trees” would be stemmed to the word “tree”. In lemmatization, we use a context corpus to find the lemma or root form of the term. Stemming and lemmatization are similar in spirit; however, stemming determines the root form by language alone whereas lemmatization uses context to determine the root form. For both approaches we used the stemming and lemmatization algorithms provided by nltk.

Constructing the feature vectors

We found that the feature vectors that led to the most accurate classification are:

1. Convert the entire title to lower case
2. Strip the title of punctuation
3. Tokenize the title by whitespace
4. Create mapping from token to frequency of token in the sentence
5. Remove common “stop” words

Original	Looking forward to this summer. The perfect spot to put your feet up and be surrounded by silence (besides the occasional bird's chirp)
Lowercase	looking forward to this summer. the perfect spot to put your feet up and be surrounded by silence (besides the occasional bird's chirp)
No punctuation (opt1)	looking forward to this summer the perfect spot to put your feet up and be surrounded by silence besides the occasional bird's chirp
Tokens	['looking', 'forward', 'to', 'this', 'summer', 'the', 'perfect', 'spot', 'to', 'put', 'your', 'feet', 'up', 'and', 'be', 'surrounded', 'by', 'silence', 'besides', 'the', 'occasional', "bird's", 'chirp']
Feature vector (word features)	{'looking': 1, 'forward': 1, 'to': 2, 'this': 1, 'summer': 1, 'the': 2, 'perfect': 1, 'spot': 1, 'put': 1, 'your': 1, 'feet': 1, 'up': 1, 'and': 1, 'be': 1, 'surrounded': 1, 'by': 1, 'silence': 1, 'besides': 1, 'occasional': 1, "bird's": 1, 'chirp': 1}
Remove common words (opt3)	{'looking': 1, 'forward': 1, 'summer': 1, 'perfect': 1, 'spot': 1, 'put': 1, 'feet': 1, 'up': 1, 'surrounded': 1, 'silence': 1, 'besides': 1, 'occasional': 1, "bird's": 1, 'chirp': 1}

Figure 2: The process of constructing a word feature vector.

We found that the feature vectors that led to the most accurate classification are:

1. Convert the entire title to lower case
2. Strip the title of punctuation
3. Construct 5-grams, or consecutive sequences of 5 characters

Original	Looking forward to this summer. The perfect spot to put your feet up and be surrounded by silence (besides the occasional bird's chirp)
Lowercase	looking forward to this summer. the perfect spot to put your feet up and be surrounded by silence (besides the occasional bird's chirp)
No punctuation (opt1)	looking forward to this summer the perfect spot to put your feet up and be surrounded by silence besides the occasional bird's chirp
Feature vector (character features, n=5)	{'erfec': 1, "rd'sc": 1, 'lbird': 1, 'ssumm': 1, 'looki': 1, 'silen': 1, 'cebes': 1, 'ourfe': 1, 'siona': 1, 'urrou': 1, 'andbe': 1, 'dbesu': 1, 'lence': 1, 'rdtot': 1, 'heocc': 1, "bird'": 1, 'feetu': 1, 'casio': 1, 'eetup': 1, 'mmert': 1, 'edbys': 1, 'oputy': 1, 'forwa': 1, 'dbysi': 1, "ird's": 1, 'ardto': 1, 'round': 1, 'ccasi': 1, 'asion': 1, 'nalbi': 1, 'toput': 1, 'ndedb': 1, 'pandb': 1, 'oking': 1, 'dedby': 1, 'tyour': 1, 'hissu': 1, 'urfee': 1, 'dtoth': 1, 'issum': 1, 'tspot': 1, 'ectsp': 1, 'schir': 1, 'unded': 1, 'ummer': 1, 'othis': 1, 'idest': 1, 'summe': 1, 'besur': 1, 'ional': 1, 'ttopu': 1, 'etupa': 1, 'ebesi': 1, 'esthe': 1, 'eocca': 1, 'rward': 1, 'ctspo': 1, 'ookin': 1, 'kingf': 1, 'putyo': 1, 'ngfor': 1, 'surro': 1, "'schi": 1, 'rfeet': 1, 'potto': 1, 'spott': 1, 'eperf': 1, 'thiss': 1, 'besid': 1, 'onalb': 1, 'esurr': 1, 'stheo': 1, 'rthep': 1, 'fects': 1, 'merth': 1, 'occas': 1, 'bysil': 1, 'ounde': 1, 'rfect': 1, 'heper': 1, 'ncebe': 1, 'desth': 1, 'orwar': 1, 'ilenc': 1, 'ysile': 1, 'eside': 1, 'tothi': 1, 'erthe': 1, 'utyou': 1, 'theoc': 1, 'tupan': 1, 'thepe': 1, 'upand': 1, 'perfe': 1, 'ndbes': 1, 'chirp': 1, 'sides': 1, 'rroun': 1, 'ingfo': 1, 'gforw': 1, 'ottop': 1, 'yourf': 1, 'wardt': 1, "d'sch": 1, 'enceb': 1, 'albir': 1}

Figure 3: The process of constructing a character feature vector.

Data and Experimentation

To test the different classification algorithms, we implemented several different test cases. The focus of our testing was to answer the following questions:

1. How does classifier accuracy vary with the number of possible classes (subreddits)?
2. How do the different optimizations (and combinations of optimizations) improve the accuracy of the classifier?
3. How does accuracy of character features vary with the length of the n-gram features?
4. How do character features compare to word features in terms of classifier accuracy?
5. Which classification algorithm is the most accurate?
6. How does the degree of similarity of subreddits affect classification accuracy?

Analysis

Comparison of Performance of Three Classification Algorithms

To compare the classification performance of Naive Bayes, SVM, and the linear classification baseline without any optimizations, we performed multiple trials with a varying number of subreddits. As we see in Figure 4 below, SVM and Naive Bayes outperformed the baseline in every trial. Additionally, SVM outperformed Naive Bayes in every trial. The difference in performance between Naive Bayes and the baseline, and the difference between SVM and the baseline, are both statistically significant (paired t-test). We suspect that SVM is the algorithm of choice for this text categorization problem as it is well regularized, meaning that it prevents overfitting on high dimensional data (such as features in a Reddit post).

# of Subreddits	Baseline	Naive Bayes	SVM
2	0.9158	0.9604	0.9646
3	0.9043	0.9406	0.9403
4	0.8738	0.9158	0.9371
5	0.8356	0.8792	0.9075
6	0.8152	0.8597	0.8877
7	0.7917	0.8502	0.8819
8	0.7693	0.8392	0.8794
9	0.7674	0.8137	0.8642
10	0.7365	0.7967	0.8491

Figure 4: Comparison of baseline linear classification, Naive Bayes, and SVM classification algorithms.

Performance of Character n-gram features with varying n

We sought to compare the performance of character *n*-gram features with varying *n*. We performed a number of trials comparing the baseline algorithm with word features to the baseline algorithm with character n-gram features with varying n, where n = 1 to n = 7. We found that n = 5 with removal of punctuation (optimization 1) produced the optimal performance among all n that we tested. See Figure 5 below.

					N-gram length						
Input 1	Input 2	Input 3	Input 4	Baseline	4 + opt1	5	5 + opt1	6	6 + opt1	7	7 + opt1
Archaeology	ArtHistory			0.7662	0.8657	0.8706	0.8905	0.8905	0.8905	0.8458	0.8657
Archaeology		MachineLearning		0.9158	0.9653	0.9752	0.9703	0.9851	0.9851	0.9455	0.9307
Archaeology			Outdoors	0.8622	0.9439	0.949	0.9541	0.9388	0.9388	0.9031	0.9133
	ArtHistory	MachineLearning		0.8557	0.9453	0.9403	0.9403	0.9303	0.9254	0.9154	0.9154
	ArtHistory		Outdoors	0.8308	0.8821	0.9282	0.9282	0.9128	0.9179	0.9282	0.9231
		MachineLearning	Outdoors	0.9031	0.9592	0.9286	0.9235	0.9184	0.9133	0.9	0.9082
Archaeology	ArtHistory	MachineLearning		0.8013	0.8609	0.8775	0.8841	0.8775	0.8841	0.8775	0.8709
Archaeology	ArtHistory		Outdoors	0.7804	0.8514	0.8615	0.8649	0.8514	0.8649	0.8378	0.8412
Archaeology		MachineLearning	Outdoors	0.8721	0.936	0.936	0.9327	0.9091	0.9192	0.8889	0.8788
	ArtHistory	MachineLearning	Outdoors	0.8345	0.8851	0.8885	0.8953	0.8784	0.8851	0.8581	0.8615
Archaeology	ArtHistory	MachineLearning	Outdoors	0.7657	0.8489	0.8589	0.8615	0.864	0.8741	0.8363	0.8463

Figure 5: Comparison of character *n*-gram features with varying *n*.

Comparison of Performance of Word Features and Character n-gram Features

We sought to analyze the performance of word features versus character features. Comparing the performance word features and the 5-grams on the baseline algorithm, we found that both had similar performance, and the difference in accuracy between the two was not statistically significant. This could because the words used in Reddit post titles are relatively short, so using space-delimited tokens versus 5-grams do not make much of a difference. However, we found that the classifier consistently performed better when classifying with word features with opt1 and opt3 (removal of punctuation and stop words) over 5-grams with opt1. Our results are in Figure 6 below.

Number of Inputs	Word Features		Character Features	
	Baseline	Opt1 + Opt3	n=5	n=5 + Opt1
2	0.9208	0.9554	0.9208	0.9505
3	0.8878	0.9274	0.8878	0.9241
4	0.8639	0.9059	0.8614	0.8911
5	0.8277	0.8693	0.8297	0.8673
6	0.8086	0.8498	0.8135	0.8482
7	0.786	0.8474	0.7903	0.8217
8	0.778	0.8404	0.7756	0.813
9	0.7616	0.831	0.7627	0.8102
10	0.7427	0.8029	0.7459	0.7873

Figure 6: Comparison of performance of word features and character features.

Performance of Various Feature Extraction Optimizations

We tested the performance of various optimizations and combinations of optimizations on word features. Optimization 1 is removal of punctuation, optimization 2 is breaking down contractions, and optimization 3 is removal of stop words. As we see in Figure 7 below, the combination of optimizations 1+2+3 seemed to perform as well as the combination from optimizations 1+3. This is in line with our intuition that punctuation and stop words do not contribute towards accurately predicting the subreddit. We found that breaking down contractions by itself seemed to be somewhat detrimental or not significant in the classification accuracy. The difference between the words contracted and uncontracted in question may simply be not that important to the classifier relative to other word features.

				Accuracy (fraction of correct classifications)							
Input 1	Input 2	Input 3	Input 4	Baseline	Opt1	Opt2	Opt3	Opt1+2	Opt1+3	Opt2+3	Opt 1+2+3
Archaeology	ArtHistory			0.7662	0.8159	0.7512	0.796	0.8109	0.8209	0.791	0.8209
Archaeology		MachineLearning		0.9158	0.9356	0.9109	0.9158	0.9356	0.9505	0.9158	0.9455
Archaeology			Outdoors	0.8622	0.898	0.8673	0.898	0.9031	0.9235	0.8929	0.9184
	ArtHistory	MachineLearning		0.8557	0.8955	0.8557	0.8806	0.9005	0.8955	0.8806	0.8955
	ArtHistory		Outdoors	0.8308	0.8718	0.8205	0.841	0.8564	0.8821	0.8359	0.8821
		MachineLearning	Outdoors	0.9031	0.9439	0.8929	0.9031	0.9439	0.9388	0.9031	0.9388
Archaeology	ArtHistory	MachineLearning		0.8013	0.8444	0.8046	0.8079	0.8411	0.8678	0.8046	0.8675
Archaeology	ArtHistory		Outdoors	0.7804	0.8142	0.7838	0.7939	0.8243	0.8142	0.7905	0.8176
Archaeology		MachineLearning	Outdoors	0.8721	0.8855	0.8519	0.8552	0.8956	0.9192	0.8653	0.9158
	ArtHistory	MachineLearning	Outdoors	0.8345	0.8818	0.8446	0.8615	0.8818	0.8784	0.848	0.875
Archaeology	ArtHistory	MachineLearning	Outdoors	0.7657	0.8212	0.7809	0.7758	0.8186	0.8388	0.7783	0.8388

best in class performance 2nd best in class worst in class

Figure 7: Comparison of various feature extraction optimizations. There is a full-size version in the data folder

Stemming and Lemmatization with Naive Bayes

We tested the effect of stemming and lemmatization on the performance of the baseline linear classification algorithm. We found that stemming had a slight improvement

over the baseline for 5 to 9 subreddits. However, the difference in performance between lemmatization and baseline is not statistically significant (paired t-test, p-value = 0.2120). Additionally, the difference between stemming and baseline was not statistically significant (paired t-test, p-value = 0.1095).

# of Subreddits	Baseline	Lemmatization	Stemming
2	0.9158	0.9307	0.9158
3	0.9043	0.9010	0.9175
4	0.8738	0.8713	0.8589
5	0.8356	0.8436	0.8436
6	0.8152	0.8251	0.8350
7	0.7917	0.7932	0.8031
8	0.7693	0.7805	0.7843
9	0.7674	0.7662	0.7778
10	0.7365	0.7293	0.7324

Figure 8: Comparison of performance of baseline, baseline with lemmatization, and baseline with stemming.

Analysis of Overall Classifier Performance

We performed a series of experiments to test the accuracy of our classifier with various feature extraction optimizations and classification algorithms. We visualize our results in Figure 9. As we see in the graph, classification accuracy steadily decreases as the number of classes (subreddits) increases. SVM with stemmed, bigram word features performed the best overall compared to the other classification methods.

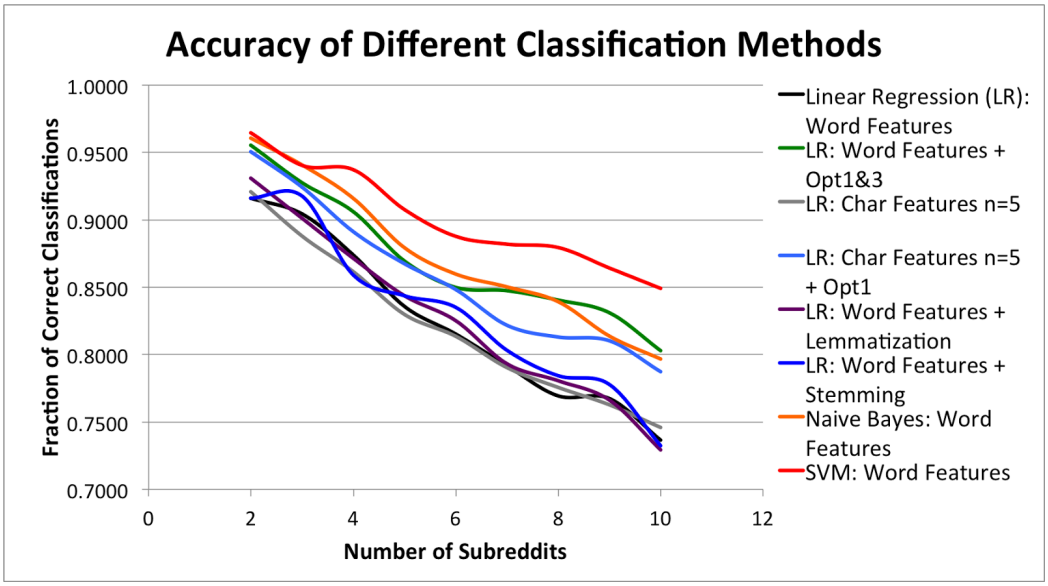

Figure 9: Graph of overall classifier performance with different feature extraction schemes and classification algorithms.

Understanding Misclassifications using a Confusion Matrix

To help visualize where our classifier makes incorrect predictions, we constructed a confusion matrix on our classifier with word features (no optimizations) and the the Naive Bayes classification algorithm. The confusion matrix shows the relationship between the predicted label to actual label. In Figure 10 below, the predicted labels are represented by the columns while the actual labels are represented by the rows. From the confusion matrix we observe that our classifier performed well in distinguishing posts from subreddits that are dissimilar in topic (i.e. Archaeology and MachineLearning), but struggled with posts from subreddits that are somewhat similar in topics (i.e. Otters, Outdoors, and DenverBroncos). We suspect that subreddits with similar topics have posts that contains terms that are common to those subreddits. For instance, the term “river” could appear in both Otters and Outdoors.

	Archaeology	MachineLearning	Pizza	Techno	BBQ	DenverBroncos	Outdoors	Trainporn	Otters	ArtHistory
Archaeology	**0.8317**	0.0099	0.0099	0.0099	0.0099	0.0198	0.0198	0.0198	0.0099	0.0594
MachineLearning	0.0495	**0.8218**	0	0.0198	0.0099	0.0198	0.0297	0.0099	0	0.0396
Pizza	0.0198	0	**0.802**	0.0297	0.0693	0.0396	0	0.0099	0	0.0297
Techno	0.0198	0.0495	0.0297	**0.7624**	0	0.0594	0.0198	0.0099	0	0.0495
BBQ	0.0099	0.0099	0.099	0.0198	**0.7129**	0.0792	0.0297	0.0099	0	0.0297
DenverBroncos	0.0297	0.0099	0.0297	0.0099	0.0396	**0.8119**	0.0198	0	0.0099	0.0396
Outdoors	0.1052	0	0.0316	0.0316	0.0211	0.0842	**0.663158**	0.0316	0	0.0316
Trainporn	0.0495	0.0099	0.0099	0.0396	0	0.0297	0.0099	**0.7822**	0.0099	0.0594
Otters	0.0645	0.0161	0.0645	0.0645	0.0806	0.1129	0.1129	0.0323	**0.3871**	0.0645
ArtHistory	0.22	0.02	0	0.03	0.01	0.03	0.02	0.02	0.01	**0.64**

Figure 10: Confusion matrix with word features (no feature optimizations) and Naive Bayes algorithm. A larger version is included in the “data” folder.

Conclusion

For our research we formally defined the goal of classifying Reddit post titles to their respective subreddits as a multi-class document categorization problem. We tested a multitude of techniques for feature extraction including word versus character features, removal of punctuation, breaking down contractions, removal of stop words, stemming, and lemmatization. We employed three machine learning techniques for the classification phase: a linear classifier (minimizing hinge loss with stochastic gradient descent), Naive Bayes, and SVMs. We found that the stemmed, bigram word features with an SVM classifier performed best on the test set, with an accuracy of 96.46% on 2 subreddits and 84.91% on 10 subreddits. We found that classification accuracy steadily decreases as the number of classes (subreddits) increases. We found that stemming and lemmatization do not have an appreciable effect on classifier performance. We found that 5-grams performed best among character *n*-gram features with n = 1 to n = 7. When comparing word and character features, we found that word features performed about as well as 5-gram features.

For further work, we can extend the testing of our classifier to more than 10 subreddits to see how well it classifies with a larger number of subreddits. In addition, we could incorporate additional information such as post content. We can also interface our

classifier with the Reddit API so that we stream new Reddit posts to the classifier and perform on-the-fly classification, so as to create an “auto-tagging” feature.

References

1. Reddit Corpus: https://github.com/umbrae/reddit-top-2.5-million
2. List of function words: http://www.psych.nyu.edu/pylkkanen/Neural_Bases/13_Function_Words.pdf
3. Cross Validation: http://en.wikipedia.org/wiki/Cross-validation_(statistics)
4. http://nlp.stanford.edu/pubs/sidaw12_simple_sentiment.pdf
5. http://www.cs.ubc.ca/~nando/540-2013/projects/p52.pdf
6. http://en.wikipedia.org/wiki/Document_classification
7. LibShortText: http://www.csie.ntu.edu.tw/~cjlin/papers/libshorttext.pdf