

Text Classifiers for Political Ideologies

Maneesh Bhand, Dan Robinson, Conal Sathi

CS 224N Final Project

1. Introduction

Machine learning techniques have become very popular for a number of text classification tasks, from spam filters to sentiment analysis to topic classification. In this paper, we will apply a number of machine learning techniques to classifying text by the political ideology of the speaker (i.e. Republican or Democrat). This is a potentially useful application of text classification, as when readers read online blogs, political discussions, news/encyclopedia articles, history textbooks, etc., it would be helpful for them to know the ideology of the writer (which is difficult to find especially when the writer is not a politician), as then they can uncover bias, allowing them to get a better perspective of the writing.

We found a corpus of data from the following website:
<http://www.cs.cornell.edu/home/llee/data/convote.html>

The data consists of a collection of annotated speech segments from members of Congress in 2005. Each speaker is given a numerical ID and each speech segment is annotated with a political ideology (Democrat or Republican). The speech segments come from debates on various issues.

2. Language Modeling

As a first approach to text classification, we attempted to use a variety of language models to determine differences between the modes of language used by people with differing sentiments. That is, given our underlying hypothesis that differences in sentiment and ideology are reflected in the frequencies of specific unigrams, bigrams, and trigrams, we sought to model these linguistic distinctions using a library of models coded during previous assignments.

Thus, our first classification approach was to select a specific language model and train a copy of it on each class of our classification (e.g. a language model for each party). Then, given a sample of text, we labeled it with the class label corresponding to the language model that agreed most with the sample. This agreement can be measured by probability assigned to the test sample, which we sought to maximize, or perplexity of the sample under each model, which we attempted to minimize, but ultimately we found that the former performed consistently better.

In these attempts, we utilized an absolute discounted unigram language model, an unsmoothed bigram language model, a Kneser-Ney smoothed bigram language model, a Good-Turing smoothed trigram language model, and linear interpolations of various combinations of these models, with and without expectation-maximization of our linear weights. Our results are as follows.

Language Model	Comparison Method	Republican F1	Democrat F1	Aggregate F1
Absolute Discounted Unigrams	Document Probability	0.045455	0.640000	0.477178
Unsmoothed Bigrams	Document Probability	0.061069	0.649573	0.489627
Kneser-Ney Smoothed Bigrams	Document Probability	0.076336	0.655271	0.497925
Good-Turing Smoothed Trigrams	Document Probability	0.077519	0.662890	0.506224
Linear Interpolation of Above Smoothed Models	Document Probability	0.090909	0.657143	0.502075
Linear Interpolation with Expectation Maximization	Document Probability	0.092308	0.664773	0.510373

Observe that, intuitively, more advanced language models tend to perform better. The most important factor in this specific task was ultimately the amount of smoothing offered by a particular model. Our training set consisted of 860 samples of congressional floor debates taking place in 2005, totaling around 1.6 megabytes of text, giving each model around 430 samples of training text. That said, although our data is stripped of many trivial cases, such as those in which a speaker uses only one sentence and uses the word “yield”, as when yielding all time to the chair, there still is a good deal of procedural text in our training data. Thus, the training data for our models is very sparse, and contains a high proportion of procedural data, which conveys very little about the ideology of a speaker. Especially given that topics of floor debates can be made up of almost any conceivable domestic or foreign issue, and often contain large numbers of facts, figures, names, and places, test data, understandably, often contains a large fraction of sentences in which many or all significant words are unseen in training. In this case, sentences will be evaluated with almost all returned probability mass coming from smoothing, and thus with no significant difference in evaluation of sentences under each political party's language model.

Thus, the overwhelming preference in terms of F1 score of one party over another makes sense given the underlying implementation of the classifier; ties are broken in one direction or another, in this case towards Democrats, not Republicans, giving our language model classifier high recall on the former and high precision but very low recall on the latter. Indeed, on Republican text samples, our EM smoothed linearly interpolated language model recorded a recall of 0.04918, but a precision of 0.75, whereas the same model records a recall of 0.98319 on Democratic text samples.

Ultimately, the best way to improve our language model approach is simply to find more training data. Unfortunately, we were unable to do so in large enough quantities to at least somewhat combat the major sparsity issue outlined above. Preprocessing with a named entity labeler which replaced entities such as names, monetary amounts, and dates

with the names of their classes could conceivably also help with this sparsity problem, but ultimately this did not significantly improve performance. A possible explanation could be that, while such preprocessing reduces sparsity, it also throws away a significant amount of information about the ideology of the speaker, as members of each political party are much more likely to reference certain names than others. (For example, a Republican concerned with a strong defense might reference the “*Reagan* administration”, whereas a Democrat arguing for paying down the national debt might reference prosperity during the “*Clinton* years”.)

A less pointed approach to our sparsity issue is stemming. By reducing the many forms of a word we might come across to a common stem, our evaluation of text becomes much more semantic, and much less obscured by syntax. Results for the same language models with testing and training data preprocessed by a Porter stemming routine are as follows.

Language Model	Comparison Method	Republican F1	Democrat F1	Aggregate F1
Absolute Discounted Unigrams	Document Probability	0.045802	0.643875	0.481328
Unsmoothed Bigrams	Document Probability	0.076923	0.659091	0.502075
Kneser-Ney Smoothed Bigrams	Document Probability	0.090226	0.653295	0.497925
Good-Turing Smoothed Trigrams	Document Probability	0.062992	0.664789	0.506224
Linear Interpolation of Above Smoothed Models	Document Probability	0.090226	0.653295	0.497925
Linear Interpolation with Expectation Maximization	Document Probability	0.076336	0.655271	0.497925

Observe that our F1 score improved most significantly for our unsmoothed bigram model. Indeed, this model suffers the most severely from the sparsity problem, and thus benefits significantly from anything that alleviates this issue. Still, this improvement was minor. For some of the more highly smoothed models, the stemming actually hurt final performance, perhaps by reducing semantically distinct words to a common stem, or by removing specificity in the frequencies of our N-grams.

3. Using Naive Bayes and Support Vector Machines with n-grams as features

3.1 Naïve Bayes

3.1.1 Description

Naïve Bayes is a probabilistic classifier that selects a class Y using a “bag of features” model. Naïve Bayes chooses a class Y that maximizes $P(Y | X_1, X_2, \dots, X_n)$, where n is the number of features. It does so by Bayes’ Rule:

$$P(Y | X_1, X_2, \dots, X_n) = \frac{P(X_1, X_2, \dots, X_n | Y)P(Y)}{P(X_1, X_2, \dots, X_n)}$$

Since the denominator is constant no matter which Y we choose, we can ignore it. Thus, we choose a class Y that maximizes $P(X_1, X_2, \dots, X_n | Y)P(Y)$. Using the Naïve Bayes assumption that each X_i is conditionally independent of Y, we get that:

$$P(X_1, X_2, \dots, X_n | Y) = P(X_1|Y) * P(X_2|Y) * \dots * P(X_n|Y)$$

Thus, we choose a Y that maximizes $P(X_1|Y) * P(X_2|Y) * \dots * P(X_n|Y) * P(Y)$

We used the Naïve Bayes source code from Weka: an open source software for machine learning and data mining.

3.1.2 Motivation

Although Naïve Bayes has its flaws in that the Naïve Bayes assumption is not entirely accurate, it's a simple model to implement, so it is a good first model to start with, and it generally performs well on text classification tasks, such as spam filters.

3.1.3 Results

We employed the Naïve Bayes model using the presence of n-grams as binary features. We used a cutoff to determine which of the n-grams to use. If the number of times we saw the n-gram was at least the cutoff, then we used that n-gram as a feature. We tested different cutoffs to see which performed the best. The table below summarizes our results.

	Features	# of features	Republican F1	Democrat F1	Aggregate F1
(1)	Unigram (cutoff = 1)	20217	0.63235	0.54188	0.59207
(2)	Unigram (cutoff = 2)	12568	0.62434	0.53956	0.58625
(3)	Unigram (cutoff = 3)	9834	0.61407	0.53470	0.57809
(4)	Unigram (cutoff = 4)	8281	0.60897	0.53077	0.57343
(5)	Bigram (cutoff = 1)	208692	0.73433	0.48370	0.64918
(6)	Bigram (cutoff = 2)	73088	0.71600	0.51654	0.64219
(7)	Bigram (cutoff = 3)	43906	0.71656	0.53168	0.64685
(8)	Bigram (cutoff = 4)	31626	0.70822	0.52968	0.63986
(9)	Unigrams +Bigrams (using best cutoffs)	228909	0.71852	0.52201	0.64569
(10)	Trigrams (best cutoff = 1)	503821	0.72670	0.23111	0.59674
(11)	Unigrams + Bigrams + Trigrams (best cutoffs all 1)	732730	0.73720	0.43382	0.64103

3.1.4 Analysis

Overall, the model performed better on lower cutoffs. This makes sense as in lower cutoffs, we have more features, and noisy features are less likely to lower performance, as each feature is considered conditionally independent of one another. Noisy features will have roughly equal $P(x|y)$.

An interesting note was that bigrams performed much better than unigrams. In most papers we looked at that dealt with text classification, most seemed to have better performance on unigrams than bigrams (such as Pang, et. al, and Yu). It may be that members of each party tend to use similar phrases (as opposed to just key words), so bigrams are more effective at capturing these phrases than unigrams. Looking at the data, we noticed that republicans for example often used the phrase “national security” and “enduring freedom.” If only unigrams were used as features, these features would be almost meaningless, as “national” and “enduring” can be used in many contexts. Unfortunately, because of the Naïve Bayes assumption, short phrases cannot be captured in the Naïve Bayes model when using only unigrams as features, as each word is assumed to be conditionally independent of one another.

Another interesting note was that when both unigrams and bigrams were employed as features, performance decreased. This may be because we are now violating the naïve assumption. If a particular bigram is seen, then the two corresponding unigrams belonging to that bigram will also be seen, so the features are definitely no longer conditionally independent, so the Naïve Bayes model is no longer valid. This may also be why performance also decreased when we employed unigrams, bigrams, and trigrams. Trigrams by themselves performed extremely poor. The recall for democrats was found to be .231. This is perhaps because of the sparsity of our data. However, it is interesting that the F1 is so high for Republicans. It may also be possible that Republicans more frequently use longer phrases than Democrats, at least in this training set.

3.2 Support Vector Machines

3.2.1 Description

Unlike Naïve Bayes, which is a probabilistic classifier, Support Vector Machines are *large-margin* classifiers. It seeks to find a hyperplane, represented by some vector \mathbf{w} that not only separates the documents in one class from another class but that maximizes that margin of separation. We obtained the source code for the SVM algorithm again from Weka.

3.2.2 Motivation

SVM's are considered to be one of the best supervised learning algorithms. In addition, unlike Naïve Bayes, they don't make any assumptions of conditional independences between the features. As we covered above, not only are bigrams conditionally dependent with the unigrams they consist of, but unigrams may be conditionally

dependent on one another. For example, given the user is a democrat, the unigrams “abortion” and “pro-choice” are not independent. Thus, it is important that we try out an algorithm that makes no assumptions of conditional independences.

3.2.3 Results

	Features	# of features	Frequency/ presence	Republican F1	Democrat F1	Aggregate F1
(12)	Unigram (cutoff = 1)	20217	Presence	0.63871	0.57252	0.60839
(13)	Unigram (cutoff = 2)	12568	Presence	0.64731	0.58270	0.61772
(14)	Unigram (cutoff = 3)	9834	Presence	0.64370	0.57942	0.61422
(15)	Unigram (cutoff = 4)	8281	Presence	0.64008	0.57614	0.61072
(16)	Bigram (cutoff = 1)	208692	Presence	0.64871	0.58629	0.62004
(17)	Bigram (cutoff = 2)	73088	Presence	0.64957	0.57949	0.61772
(18)	Bigram (cutoff = 3)	43906	Presence	0.65951	0.59617	0.63054
(19)	Bigram (cutoff = 4)	31626	Presence	0.65598	0.58718	0.62471
(20)	Unigrams + Bigrams (using best cutoffs)	64174	Presence	0.64097	0.59653	0.62005
(21)	Trigrams (best cutoff = 1)	503821	Presence	0.61436	0.56967	0.59324
(22)	Unigrams + Bigrams + Trigrams (best cutoffs all 1)	732730	Presence			

3.2.4 Analysis

Overall, SVM had similar trends to the Naïve Bayes algorithm. Like Naïve Bayes, the SVM performed better with bigram features than with unigram features. This again may be due to unigrams losing context. Also, like Naïve Bayes, the SVM performed better with just bigrams than with bigrams and unigrams. This may be because the unigrams overwhelm the bigrams, as many of them are noisy. Again, trigrams proved themselves to be the worst feature, which makes sense, as our data is sparse and it may be because longer set phrases are not as common.

3.3 Comparing Naïve Bayes and Support Vector Machines

As expected, SVM performed better than the Naïve Bayes classifier on unigram features. However, interestingly enough, the Naïve Bayes classifier performed better than SVM on bigram features. This may be because since there are more bigram features, there are more noisy bigram features. NB can handle noise better than SVM's because if a particular feature x_i is noisy, then $P(x_i|y)$ is roughly equal for both values of y , so $P(x_i|y)$ is now a constant, so now its value does not affect which y is chosen that maximizes $P(x_1|y) \dots P(x_n|y) \cdot P(y)$.

3.4 Error Analysis

Our results may be very poor, due to the use of so many features, many of which are noisy. We attempt to fix this issue in the next section. Also, we may have issues of sparsity in our data set, as our data set is not incredibly large. Our training set has 2741 files where each file ranges from 1 sentence of text to 30 or 40 sentences of text. We attempt to fix this issue in Section 5.

4. *Selecting Features using Information Gain*

4.1 Motivation

Looking at the previous tables, we're using a lot of features. When the cutoff is 1, we're using around 20,000 features. It would be better to figure which of these 20,000 features were the most distinctive in classifying the texts into the political ideology classes and just use those features to reduce noise of unnecessary features.

4.2 Implementation

We determine which features are "most distinctive" by choosing the features that have the most information gain. We do so by calculating the information gain of each feature and just take the n features with the highest information gain, where we vary n .

We calculated information gain by the following formula we found in Forman's feature selection paper.

$$e(pos, neg) - [P_{word}e(tp, fp) + (1 - P_{word})e(fn, tn)]$$

where $e(x, y) = -x \log_2\left(\frac{x}{x+y}\right) - y \log_2\left(\frac{y}{x+y}\right)$,
and $x \log_2(x) = x \log_2(x)$

We varied the number of unigram features selected and measured the F1 performance, first using unigram frequency as features and then using unigram presence as features.

4.3 Results

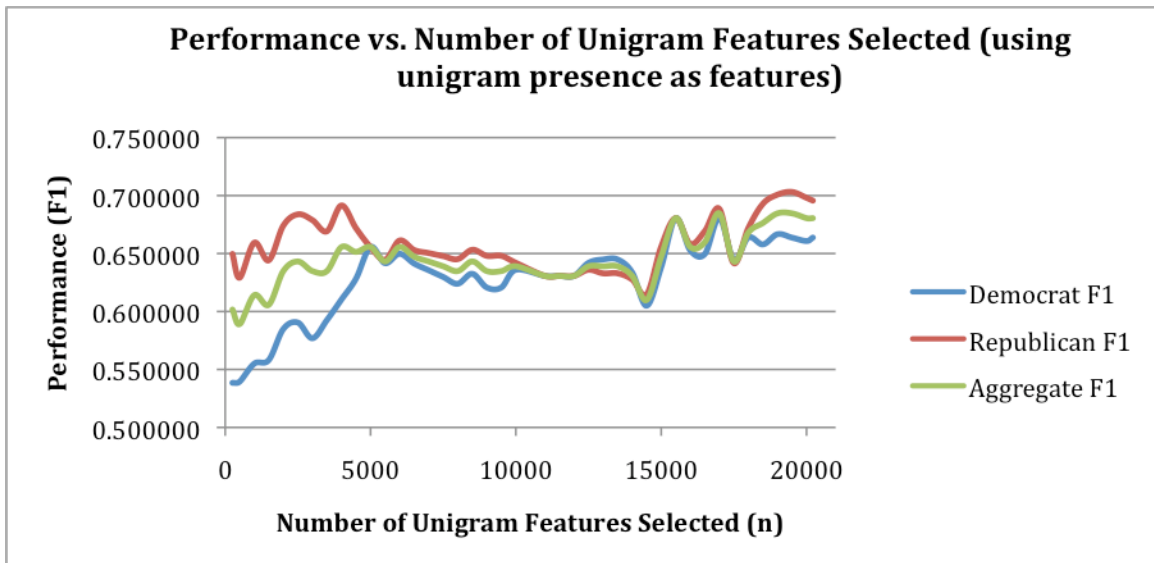
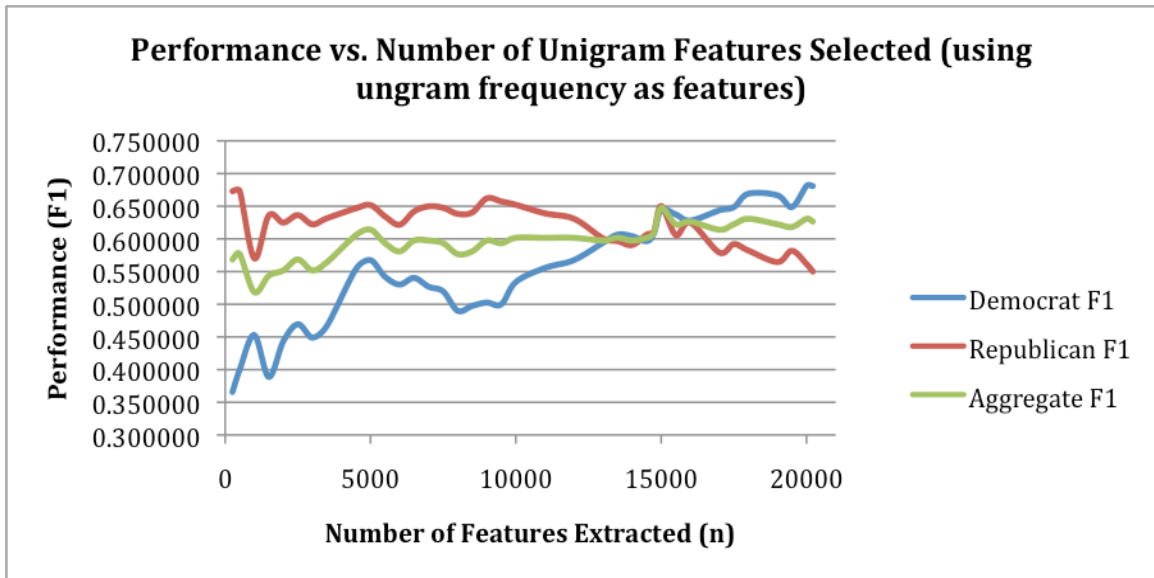
Top 10 Unigrams With Most Information Gain	Top 10 Unigrams With Least Information Gain
1) republican 2) cuts 3) majority 4) opposition 5) cut 6) instead 7) fails 8) ? 9) republicans 10) billion	1) actions 2) particularly 3) legislative 4) sense 5) allow 6) heartland 7) stresses 8) 1948 9) intentioned 10) gloves

The top 10 unigrams with least information gain is somewhat intuitive. They are very vague or random words, such as “gloves,” or “allow” or “intentioned.” Otherwise, they deal with Congress, which all Congressmen face, not just those belonging to a certain party, such as the unigrams “actions,” “legislative,” and “1948.” Nonetheless, it is intriguing that very common words like “a,” “the,” or “person” are not on that list, implying that politicians of different ideologies have different styles of speaking in addition to their sentiments on particular issues and their phrases.

The top 10 unigrams with most information gain are not as intuitive. The question mark is the most interesting case. After looking at documents, it seems that democrats ask more rhetorical questions when questioning the actions of the republicans. As the democrats are of the minority during this time, this makes sense as the republicans are in control, so the democrats cannot command what Congress should do.

For example, in the corpus, you find a democrat speaking the following quote:

“You 'd think that 'd be accounted for in this budget? No. The billions of dollars that will be needed for the Iraq war. In the budget? No. The cost to our children of extending the massive bush tax cuts to the wealthy that will balloon our massive deficit?”



4.4 Analysis

First off, our results confirm our beliefs that using unigram presence as features are superior to using unigram frequency as features, as on average the aggregate F1 score for the former is greater than the later.

We reach a maximum aggregate F1 score of 0.684647. Before, on the SVM classifier, our highest F1 was 0.6172. Thus, performance improved by selecting a limited number of features with the most information gain.

5. Preprocessing our data to extract more information to aid our algorithms

5.1 Stemming

5.1.1 Motivation

Stemming is a method to fix the sparsity in our data, as it stores all the different forms of the same word as the same word. For example, it stores the words “vote,” “votes” and “voting,” all as one word.

5.1.2 Implementation

We employed Porter’s stemming algorithm. The source code was found from <http://tartarus.org/~martin/PorterStemmer/java.txt>

5.1.2 Results

	Features	# of features	Frequency/ presence	Republican F1	Democrat F1	Aggregate F1
(1)	Unigram (cutoff = 1) with NB	20217	Presence	0.63235	0.54188	0.59207
(2)	Unigram with stemming (cutoff = 1) with NB	20217	Presence	0.60087	0.53652	0.57109
(3)	Bigram (cutoff = 1) with NB	208692	Presence	0.73433	0.48370	0.64918
(4)	Bigram with stemming (cutoff = 1) with NB	208692	Presence	0.72414	0.50489	0.64569
(5)	Unigram (cutoff = 2) with SVM	12568	Presence	0.64731	0.58270	0.61772
(6)	Unigram with stemming (cutoff = 2) with SVM	12568	Presence	0.63447	0.58634	0.61189
(7)	Bigram (cutoff = 3) with SVM	43906	Presence	0.65951	0.59617	0.63054
(8)	Bigram with stemming (cutoff =3) with SVM	43906	Presence	0.64751	0.60666	0.62821

5.1.3 Error Analysis

Overall, stemming slightly diminishes our performance. This implies that Republicans tend to employ certain inflections of a word more often than Democrats and vice versa. Future work would be to determine which inflections were more commonly used by which party. Determining the inflection could be done syntactically by looking at the last 2 or 3 letters or semantically by performing some type of lookup on the word.

5.2 Dealing with negations

5.2.1 Motivation:

The problem right now is that if we have the sentence, “I will not stand for pro-choice,” the unigram features will store the unigram “pro-choice” and most likely this sentence will be classified as a democrat, rather than a republican. Currently, we don’t deal with negations other than in bigrams and trigrams. However, it may be that the negation is more than two words before the word it applies to, such as in this example.

5.2.2 Implementation

The solution to this issue was inspired by the Pang, et. all paper. Whenever we encounter a word that denotes negation, such as “not,” “no,” or “n’t,” for every subsequent word, we insert the prefix “NOT_” to it until we reach some sort of punctuation. For example, in the sentence, , “I will not stand for pro-choice, as I am pro-life,” after this processing, we will obtain the sentence, “I will not NOT_stand NOT_for NOT_pro-choice, as I am pro-life.”

5.2.2 Results

	Features	# of features	Frequency/ presence	Republican F1	Democrat F1	Aggregate F1
(1)	Unigram (cutoff = 1) with NB	20217	Presence	0.63235	0.54188	0.59207
(2)	Unigram with negations (cutoff = 1) with NB	20217	Presence	0.63852	0.52886	0.59091
(3)	Bigram (cutoff = 1) with NB	208692	Presence	0.73433	0.48370	0.64918
(4)	Bigram with negations (cutoff = 1) with NB	208692	Presence	0.73293	0.44723	0.63986
(5)	Unigram	12568	Presence	0.64731	0.58270	0.61772

	(cutoff = 2) with SVM					
(6)	Unigram with negations (cutoff = 2) with SVM	12568	Presence	0.64865	0.58913	0.62121
(7)	Using IG to select unigrams with negations with SVM	19000	Presence	0.63797	0.59506	0.61772
(7)	Bigram (cutoff = 3) with SVM	43906	Presence	0.65951	0.59617	0.63054
(8)	Bigram with negations (cutoff =3) with SVM	43906	Presence	0.66030	0.58656	0.62704

5.2.3 Error Analysis

Overall, this feature does not improve our performance at all, but rather it slightly hurts it. The only case where it helps is when we have presence of unigrams as features with a cutoff. A possible reason why there is no clear answer for whether this feature aids performance or not may be because words to express negation are not merely excluded to “not”, “no”, or the contraction “n’t.” In fact, as they are senators they are more likely to use more sophisticated words to express negation. Below’s an excerpt from one speech, which is very negative without explicitly using “not” or “no” or the contraction “n’t” before words they modify to express negation.

“I rise in strong opposition to the republican budget. Republicans dishonestly proclaim their budget is fiscally responsible. The only way their numbers work out is if you use slick accounting gimmicks or fuzzy math. Let me give you some examples of their clever sleight of hand: the republicans' top priority to privatize social security through private accounts will cost billions of dollars.”

A more sophisticated way would be to extract the topic being discussed and analyze the sentiment of the speaker towards that topic, which is a very difficult task, but would be very effective. Alternatively, if negative words could automatically be learned and placed into a list, so that our list does not just include “not,” “no,” or the contraction “n’t.”

5.3 Combining different speech segments of the same user together

One potential improvement we explored was grouping together different speech examples of the same speaker. Since the corpus is a transcription of Congressional discussions, the same speakers show up multiple times. By grouping together instances of the same speaker, when classifying an example, our classifier would have additional

sentences to work with. Grouping examples together like this would also avoid the possibility of classifying the same speaker as a Republican in one instance and a Democrat in another. Although this may seem like a good idea, in practice, grouping multiple examples together decreases performance dramatically. We suspect that this is because each speaker talks about different issues in each example – one example might be a speech on veterans' benefits, while another might be a speech on abortion. By combining these different examples, the resulting feature vectors contain more 1's and become more generic – they reflect a broader range of features rather than a smaller set of features pertaining to a specific topic. As a result, accuracy worsens. Therefore, we did not explore this strategy further.

6. Using a parser to extract more features

6.1 Motivation

One idea we explored was the use of parsers in generating features. The principle behind this is that parsers are able to impart structure into sentences and establish relationships between words that are potentially beneficial as features for a classifier. The idea was to take a set of training data, and use a parser to apply some transformation to that data, generating a new set of data from which a better set of features can be extracted.

6.2 Implementation

One method of using parsers to transform the data was to use a lexicalized parser and, given an input sentence, to construct new, shorter sentences based on the head words of the sentence's parse tree. For example, if a sentence in the training data were "The quick brown fox jumps over the lazy dog," then the lexicalized parser might construct the phrase "fox jumps dog." We used the parser to construct many such phrases and to write each of them as a new sentence in the transformed data set. While these head word phrases are often redundant and do not represent well-formed English sentences, they nevertheless capture relationships that an n-gram model based on the untransformed sentence would not, because they filter out the less important words in a sentence. Our method of constructing head word phrases was as follows: choose an integer k , and expand the nodes of the parse tree up to depth k . The head words of all the nodes of depth k are concatenated to form a phrase. We ran the algorithm for k values from 2 to 6; these values were arbitrary chosen, but are sufficient to accommodate most sentences. The leaves of the parse tree were not used. We employed the Stanford Parser – we used a `LexicalizedParser` running on the file `englishFactored.ser`.

6.3 Results

Information Gain (4500 features) on Unigram data using a SVM

before transformation: $F1 = 0.585081585081585$

after transformation: $F1 = 0.585081585081585$

Bigram model (cutoff=1) on a Naïve Bayes classifier:

before transformation: $F1 = 0.6421911421911422$
after transformation: $F1 = 0.6526806526806527$

Combination Unigram/Bigram/Trigram model on a Naïve Bayes classifier:

before transformation: $F1 = 0.6503496503496503$
after transformation: $F1 = 0.655011655011655$

Clearly, results are mixed. In some cases, transforming the data improves F1 scores, while in other cases it decreases F1 scores. For obvious reasons, the unigram presence models are not affected by the transformation (since the same set of words is present in both the untransformed and transformed data sets). Performance on the bigram model seems to improve slightly, while performance on the trigram model worsens. Even so, transforming the training data is an idea worth exploring, as it may lead to better features in the future.

7. Conclusion

7.1 Summary

The problem of text classification is one that has been thoroughly explored and found to have many solutions. There is no definitive way to go about classifying text, but there are certain machine learning techniques which are generally successful at these problems. However, the best strategy for classifying text will depend on the details of each individual problem and the corpora involved. For our instance of the problem, the 2-party classification of Congressional speeches, we implemented various different feature ideas and classification algorithms, but in the end, we discovered presence detecting unigram and bigram features with the aid of information gain on an SVM classifier seemed to work the best. With that technique, we got an F1 for the Republican class to be 0.70079, an F1 for the Democratic class to be 0.666667, and an aggregate F1 to be 0.684647.

7.1 Future Work

We have several ideas to try in the future. First off, it may be interesting to train and test on a corpus of written political data, rather than transcribed spoken data, as it has a very different style to it. Politicians may be more verbose when writing, allowing us to look for more features in their style of writing. When people write, more glaring differences may result. Also, we could extend our information gain algorithm for selecting features to bigrams and trigrams in addition to unigrams.

Works Cited Page

Forman, George. Feature Selection for Text Classification. *Computational Methods of Feature Selection*. 2007

Pang, B., L. Lee, and S. Vaithyanathan. 2002. Thumbs up? sentiment classification using machine learning techniques. In EMNLP 2002, 79–86

Yu, Bei, Kaufmann, Stefan and Diermeier, Daniel, Ideology Classifiers for Political Speech (November 1, 2007). Available at SSRN: <http://ssrn.com/abstract=1026925>