CS G220 Project Report : Ideology Detection Engine

Daniel Mauer

Abstract

In this report, I present an early iteration of a system intended to detect the ideological bent of an arbitrary block of text along the scale from liberal to conservative. This is attempted via the use of an Adaboost-generated ensemble of decision stumps which split upon the presence or absence of a given n-gram in the text. The training data consists of floor statements taken from the U.S. Congressional Record, paired with ideological scores for each elected representative, such that each statement is labeled according to its speaker's ideology.

1 Introduction

George Lakoff's book *Don't Think of an Elephant*[2], published in 2004, theorized that a defining characteristic of recent political discourse is a concerted effort by ideological groups (particularly the two major political parties, and to greatest effect by the Republican party) to *frame* various issues with the clever and subtle use of key words and phrases. The prototypical example is the phrase "Death Tax", which is used regularly by opponents of the federal Estate Tax. The use of the phrase was famously encouraged in a widely circulated memorandum by Republican "message man" Frank Luntz because the term "kindled voter resentment in a way that 'inheritance tax' and 'estate tax' do not". Other examples of these sorts of phrases are myriad: "energy exploration" (oil drilling); "Pro-Life" (against legalized abortion) and "Pro-Choice" (in favor of the same); and so forth.

The nature of this technique dictates that a given "loaded phrase" will likely be used frequently by members of a particular ideology and rarely by opponents of that ideology. In fact, a cursory search through the past few years of the Congressional Record turns up a great many mentions of the phrase "death tax" by Republican members, and nearly none by Democrats. It stands to reason that such phrases may, in fact, be among the only phrases which are significantly more likely to be uttered by members of one party over the other; and, conversely, that it may be the case that most phrases uttered significantly more frequently by members of a particular ideology are, in fact, "loaded".

This final possibility was the inspiration for this project: Is it possible to determine the ideology behind a given text by learning from data? With that in mind, I set out to write a system to be trained on ten years' worth of the full text of the Congressional Record (which includes all floor statements made by elected officials in both houses of Congress), along with the *CommonSpace* ideological scores[3] for all speakers, essentially using a simple unigram/bigram model to seek out terms and phrases which are much more likely to appear in statements made by members of one ideology over another.

While the system is still in its early stages, and significant optimizations must be made before it is capable of working on the large corpus, I have made significant progress toward building a working ideology detector.

Prior Work

Automated text categorization is an active and wide area of research[4], and much of this research consists of applying machine learning algorithms to text corpora. However, the majority of this research differs from this project in two significant ways. First, traditional text category classification learners generally learn from preclassified documents, whose class is known; the data I am using is likely to be much noisier, as my data is classified by the ideology of the *speaker*, not of the statement – and there is no guarantee that a given statement made by a person of a particular ideology will in fact contain any text that would imply such an ideology. Second, most categorization systems deal with discrete categories – "sports", "politics", "entertainment", and so forth. For this system, each piece of text in the corpus is scored in a continuous, two-dimensional range. Whether it will prove valuable to utilize the full continuous values, or whether it will be better to discretize the ideology scores, is another

unanswered question.

Method

Data

I obtained the full text of the congressional record for the 105th through the 109th congresses (years 1997-2006) as flat HTML files, as well as the CommonSpace scores (consisting of positive or negative values in two "dimensions": liberal/conservative and social issue standing) for every member of Congress (as well as presidents Clinton and Bush) over that time period. I developed software which, using a series of regular expressions, parsed the text of the Record, removing non-floor-statement text (e.g. roll calls, bills, etc.), separating out individual statements and associating them with their speaker, and coupled this information with the speaker's ideological scores. The result is a large corpus of text (approximately 750MB of floor statement transcriptions), fully annotated with a significant amount of information about the speaker of each statement, and organized in a relational database; I have made the database available online.

Language Model

As stated above, a simple language model of unigrams and bigrams was utilized. Ideally, trigrams would be included as well, as most loaded phrases are 2- to 3-word terms; however, at present, even without trigrams, computational tractability is an issue. It is also important to note that for all aspects of this project, n-grams which appear fewer than three times in the corpus, as well as those which appear in more than a fourth of all statements in the corpus, are thrown out, as they are nearly certainly meaningless. It turns out that such n-grams in fact make up the majority of all n-grams in the corpus, and their inclusion would result in a massive waste of computation. N-grams are picked out from each statement by a set of fairly simple heuristics which attempt to disallow ngrams to cross sentence and phrase boundaries. For example, the phrase Mr. President, we should all go to Mars. It'll be fun. contains the unigrams mr, president, i, think, we, should, all, go, to, mars, it'll, be, fun, and the bigrams mr president, we should, should all, all go, go to, to mars, it'll be, be fun. Note that president we and mars it'll are excluded. This further reduces the number of n-grams in the corpus and makes intuitive sense as well. All n-grams are stored case-insensitively.

Possible additions to the language model might include preprocessing with a stemmer (to recognize different morphologies of the same word people and person, for example as such, instead of as entirely different word), or other very rudimentary transformations of the corpus; however, the use of such transformations, or more sophisticated language models which attempt to detect phrases or other syntactic structure, are beyond the scope of this project.

Learning

For learning, I decided to focus on a single algorithm, AdaBoost [1], using decision stumps as a weak learner. Due to the nature of the domain, I made some minor modifications to the AdaBoost algorithm, as well as to the WeakLearn algorithm which it calls; I will detail these changes below. Initially, I attempted to build a variant of AdaBoost which used a continuous error function to attempt to make use of the fact that the ideological scores in the database are continuous in $[-1, 1]$; however, its behavior (upon a small amount of testing) was unconvincing in terms of effectiveness, so for the time being that avenue was abandoned. The current version of the software reduces all ideological scores to -1/+1.

AdaBoost

The general AdaBoost algorithm is as follows (X is the vector of training instances, Y is the vector of instance labels):

AdaBoost$(X, Y, WeakLearn)$
1 $\forall i,\ w_i \leftarrow \frac{1}{|X|}$
2 **for** $t \leftarrow 1$ **to** T
3 **do** $(h_t, \epsilon_t) \leftarrow$ WeakLearn(X, Y, D)
4 $\alpha_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
5 $\forall i,\ D_i \leftarrow D_i e^{-\alpha_t y_i h_t(x_i)}$
6 Normalize(D)
7 $H \leftarrow \lambda x. \sum_{t=1}^{T} \alpha_t h_t(x)$
8 **return** H

Generally, H is thresholded at 0 such that the hypothesis returned will always classify as -1/+1; leaving the Sign function out has the effect of giving essentially an "expected class" in [-1,+1] which is more appropriate in a continuous scale such as ideology.

0/1 Decision Stumps

Decision stumps are an often-used weak learner for boosting; I chose a slightly unorthodox form of the decision stump for this project. Normally, such classifiers split on some attribute's value and return a +1/-1 classification. For this domain, the splitting attribute is

presence or absence of a given term in the given text. However, the standard type of decision stump seems ill-suited for one main reason: While the *presence* of a term in a text is likely meaningful, the *absence* of that term likely is not. For example, the fact that the U.S. Constitution contains the word "freedom" is meaningful. The fact that it does not contain the word "microprocessor" is meaningless. So it seems not only unintuitive but counterproductive to use a classifier which labels a text +1 if a given term appears and -1 otherwise. Rather, it stands to reason that the correct type of weak classifier would label a text either +1 or -1 if its term appears, and 0 (i.e., a neutral classification) otherwise.

In the particular context of use as a weak learner for AdaBoost, the use of this form of decision stump is even more appropriate: In the final ensemble, the classification will be based on input from those ensemble members whose terms which appear in the text, and all other ensemble members will essentially bow out of the voting.

WeakLearn

The purpose of the WeakLearn algorithm, as called by AdaBoost, is to find the particular weak hypothesis which best classifies the weighted sample of the data. The algorithm attempts to return $h_t = \arg\min_{h_j \in H} \epsilon_j$, where ϵ_j is defined as the weighted error rate of classifier h_j. The standard calculation of e_j for a given h_j is $\sum_{i=1}^{|X|} D_i[h_j(x_i) \neq y_i]$. I have chosen a slightly altered version of this error function:

$$\epsilon_j = \sum_{i=1}^{|X|} D_i \begin{cases} [y_i \neq h_j(x_i)] & \text{if term in text} \\ .5 & \text{otherwise} \end{cases}$$

The effect of using this altered error function is that those texts which do not contain the decision stump's term are assumed to be equally likely to be classified correctly or incorrectly by that stump; this effectively removes at least some of the bias introduced by a training corpus with significantly more instances of one class than another (as the .5 term would essentially be replaced by the proportion of the corpus classified as +1). Another nontrivial side effect is that error must only be directly calculated for those texts in which the stump's term appears. This provides a very significant performance gain.

Efficiency and Optimization

Code efficiency and optimization are crucial when dealing with such a large corpus, and this has been a sticking point in my evaluation of the system. Significant further optimizations must be made before I will be able to test a sample of any reasonable size. The WeakLearn algorithm is the source of the massive slowdown, as it must determine the classification error of each possible decision stump during every round of boosting. Even with the optimization gained through the alternate error function mentioned above, this is a very time-consuming process, especially on a computer such as mine which has nowhere near the amount of RAM needed to hold the data entirely in memory.

So, until I am able to spend more time optimizing my code, and until I gain access to a computer with somewhere on the order of 6GB of RAM, I am able only to test on a tiny fraction of the corpus.

Results

The results I present here are absolutely not representative of the corpus as a whole; they are based on the first 2,000 floor statements of the 105th congress only. While this data can not likely be considered an argument for or against the probability of success on the full corpus, some interesting results were obtained.

Figure 1 shows the ensemble learner generated by running the boosting algorithm on those 2,000 training examples for 25 rounds. There are a few interesting results here. First, note that "mr. speaker" was the first classifier chosen. This happened because it appears in a large percentage of all the training examples, and it turns out there are more statements in the training set by conservatives than by liberals – sort of the inverse of the issue I attempted to resolve with the altered error function. Most such n-grams which appear very frequently are ignored by the algorithm (as outlined in the Data section), but this one seems to have gotten through.

The other classifiers in the list (well, some of them) are interesting because, despite the small size of the training corpus, the words chosen were, more often than not, words that intuitively seem politically charged to some degree. Conservatives, for example, seem to talk about taxes and government a great deal. Liberals seem to talk about children, health and leadership. While, again, this sample size is far too small to truly mean a great deal, the contents of this table seem to hint at the possibility that this technique may be successful; at the very least, many "meaningful" words were chosen.

The only actual test results I've produced so far, in terms of classifying data, are based on this same test run, with 100 rounds of boosting. The results were essentially slightly worse than random guessing for the

Figure 1. Example Ensemble

t	term	class	α_t
0	mr speaker	+1	0.1008
1	government	+1	0.0764
2	tax	+1	0.0699
3	tax	+1	0.0601
4	washington	+1	0.0524
5	must	-1	0.0505
6	tax	+1	0.0517
7	leadership	-1	0.0481
8	you	+1	0.0495
9	health	-1	0.0477
10	ask	+1	0.0454
11	rise	-1	0.0459
12	government	+1	0.0455
13	must	-1	0.0468
14	leadership	-1	0.0445
15	washington	+1	0.0441
16	colleagues	-1	0.0438
17	tax	+1	0.0438
18	health	-1	0.0415
19	going	+1	0.0417
20	children	-1	0.0417
21	ask	+1	0.0410
22	leadership	-1	0.0406
23	taxes	+1	0.0405
24	democratic	-1	0.0397

+1 = conservative
-1 = liberal

set of 109 test cases; however, testing was done only by comparing the guessed class to the scored ideology of each test statement's speaker. This is really not the appropriate manner of testing, as a statement whose speaker has a strong score will still say many things which have no hint of ideology ("Mr. Speaker, I would like to take this opportunity to thank the Gentlewoman from New York.") as well as some things which do ("Mr. Speaker, the Gentlewoman from New York is engaging in class warfare!"). So, in order to truly test this system, it would be appropriate to hand-label a number of statements which in and of themselves contain bias one way or the other, as well as some neutral statements. In addition, using 2,000 test cases is nowhere near sufficient to build a meaningful classifier.

Conclusion

This is clearly still a work in progress. However, preliminary results, as represented by the table in Figure 1, indicate that at the very least, the weak learners are focusing (at least in part) on terms that seem intuitively reasonable. More development and testing are clearly required before I pass judgment on the concept as a whole; however, a great deal has been learned in the process so far, and while I haven't yet shown that the idea has merit, I also certainly haven't encountered any strong evidence to the contrary.

Acknowledgements

Thanks to Dan Schulman, Prof. Javed Aslam and Prof. Ronald Williams for their input and advice regarding the implementation of this system.

References

[1] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *International Conference on Machine Learning*, pages 148–156, 1996.

[2] George Lakoff. *Don't Think of an Elephant.* Chelsea Green Publishing Company, 2004.

[3] Keith T. Poole. Recovering a basic space from a set of issue scales. *American Journal of Political Science*, 42(3):954, 1998.

[4] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Comput. Surv.*, 34(1):1–47, 2002.