Cmpsci585

Final Report

Patrick Carron

Raymond Zhu

12/20/2016

Predicting Political Party Affiliation from Speech

1 Abstract

We attempt to classify a speaker's party affiliation from their word usage by analyzing a corpus of congressional speeches. We investigate the impact of preprocessing techniques on cross-validation accuracy and test Multinomial Naive Bayes and Stochastic Gradient Descent classifiers. We perform 3-fold cross-validation on each classifier to find optimal hyper-perameters. We conclude that bigram language modeling with Tf-Idf weighting results in optimal preprocessing and Stochastic Gradient Descent classification with Hinge loss, $l2$ regularization, and a regularization constant of $\alpha = .0001$ result in an optimal observed test accuracy of 74.9%.

2 Introduction

This research attempts to predict a congress person's political party affiliation based on the language used in their congressional speeches. We are using the Congressional speeches dataset created by Lillian Lee at Cornell [?]. The dataset is preprocessed somewhat and split into train, validation, and test sets. We first perform exploratory data analysis to ensure high data quality and to understand some idiosyncrasies of this data asset. We then look at differences between the parties by analyzing commonly used terms, highlighting differences in policy positions reflected in language. We then use the [?] Scikit Learn library to create a data analysis pipeline to ensure that we have consistency in data preprocessing while testing both Multinomial Naive Bayes and Stochastic Gradient Descent classifiers. For each we use 3-fold cross validation to search for an optimal

pre-processing strategy along with optimal hyper-parameters for our classifiers. We compare cross validation accuracy between unigram and bigram language models and also train our classifiers both including Tf-Idf weighting and excluding Tf-Idf weighting. Ultimately we found that Stochastic Gradient Descent with Hinge loss, l_2 penalty, $\alpha = .0001$, Bigram language modeling with Tf-Idf included gave us the highest test accuracy at 74.9%.

3 Related Works

The first research performed on this corpus was [?] **Get out the vote: Determining support or opposition from Congressional floor-debate transcripts** by Thomas et al.. They attempt to classify support or opposition for legislation based solely on the same corpus used in this research. They use Support Vector Machine classification on individual documents using a unigram unigram language model For the cases involving the same speaker they implement weighted links to allow for "soft" preferences. These "soft" preferences show changes in a speaker's opinion over time during the debate. When compared to "hard" assignments, in which all labels of that speaker are the same, "soft" preferences outperform "hard" assignments. In this approach they also utilize speech-segment relationships in their classification.

In [?] **Classifying Party Affiliation from Political Speech** by Yu et al.. each document is mapped to a vector of word frequencies, and stop words, selected based on high and low frequency thresholds, are removed from the document. Naive Bayes and Support Vector Machine classifiers are tested. Leave-one-out cross validation as well as hold-out tests are used during evaluation. They also examine differences in speech by component of the legislature. They train on speeches made by congress people in the House while the test set is comprised entirely of speeches made by Senators. They then swap training and test sets. Finally, In [?] **Political Ideology Detection Using Recursive Neural Networks** Iyyer identifies ideological bias within individual sentences within this corpus. Instead of using Naive Bayes and Support Vector Machine classifiers, they use Recursive Neural Networks. Also, instead of analyzing a speech, such as in our approach, they

analyze at the sentence-level. This approach does not employee bag-of-words modeling. They detect bias in complex sentences that a baseline using bag-of-words does not.

4 Data

Some preprocessing has already been preformed on the dataset. Each token is lower case and spaces exist around punctuation marks in order to count them as separate tokens. There are some exceptions to this, such as the token "mr.", which uses the period as part of the token. Congressional names are remoced from the text and replaced with unique identifiers. Each filename within the data set contains information about the speech, and to our interest it includes the political party affiliation of the speaker. We parse the file name to record the party affiliation of each speech, which would serve as our labels for both the training and test data sets. Due to our dataset being originally for a research done to classify support or opposition, the documents of the data set are organized around proposed legislation to some speaker voting for or against the bill. The training corpus composed of 6,362 documents containing 1,544,279 tokens with a vocabulary size of 24,564 terms. The test set is composed of 1,759 documents. We merge the development set with the training set because we use 3-fold cross validation instead of a train, validation, test split to optimize our hyper-parameters. For both the training and test sets, the contents of the document and its respective labels are recorded through the same method.

As a quality check we look see that Zipf's law holds for our data set and also check high and low frequency terms. We created a log scaled Zipf's law representation in **??**.This visualization meets our expectations, and gives us confidence that our data set is high quality. The highest terms are largely contractions and determiners, while the single frequency terms seem to be proper nouns or rarely used verbs which matches our expectations. With our documents read-in and a high level of confidence in the quality of our data set and representation, we proceed to implement a data pipeline and perform baseline experiments on our corpus.
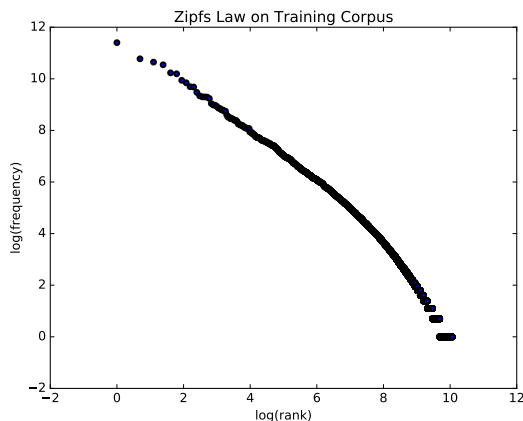
Figure 1: Zipf's Law

	Most Common		Least Common (a subset)
Word	Frequency	Word	Frequency
the	89452	herzog	1
to	47827	alfonso	1
of	42111	thimerosal	1
and	37916	nehf	1
that	27776	maize	1
in	26750	dearth	1
is	20759	hulshof	1
this	18897	boor	1
for	16213	eneryville	1
we	16039	nassau	1

4.1 Term Differences by Party

We split the corpus by party affiliation and analyzed the word frequencies for each party to create lists of most frequently and least frequently used terms for each. While there are many common terms which are determiners, prepositions and conjunctions, there are some interesting differences. Democrats use terms affiliated with education, healthcare, and energy much more frequently than Republicans do in our corpus. To be more specific, Democrats have bigrams such as "energy saving", "nuclear energy", "wind energy", "clean energy", etc. for the word energy. In contrast, Republicans have bigrams that do not show up in the Democratic bigrams dictionary. These bigrams include, but aren't limited to, "energy cost", "energy increasing", "cost energy". These bigrams match Republican sentiment which seems to suggest that using a different energy source would cost users more money. While, on the other hand, Democrats are more eager to change to

a better energy source. With regard to education, Democrats talked about "disabilities education" and "education reform" as well as other bigrams. Some bigrams that do not show up on the Democrats dictionary, but do in the Republican bigrams are "improve education" and "increase education". Lastly, in terms of healthcare we see that Democrats use bigrams such as "healthcare providing" and "healthcare assistance", in which the Democrats believe government intervention is needed for healthcare. On the other hand, Republicans use the intriguing bigram "healthcare distribution". This bigram does not show up in the Democrat's dictionary of bigrams, which could mean the Republicans are worried about the distribution of healthcare to the general population while Democrats are in favor government intervention to increase capacity of the system.

5   Method

We use [**?**] Scikit Learn to create identical data preprocessing pipelines for both our Multinomial Naive Bayes and Stochastic Gradient Descent classifiers. Since the corpus is comprised of a high volume of terms and documents we select a sparse representation using the CountVectorizer function which allows dictionaries containing tokens as keys and frequencies as values to represent each document in the corpus. We include a Tf-Idf transform in the data pipeline as well. These functions allow us to search across a range of options for preprocessing and look at their impact on cross validation accuracy. For CountVectorizer we search both Unigram and Bigram language model counting. For the Tf-Idf transform we compare accuracy including the option and excluding the option. This allows us to search for an optimal preprocessing methodology. The classification function is the final method included in the data pipeline and these methods and their hyper-parameters are the only distinguishable characteristic for each experiment allowing us to compare the impact of each classification technique. First we test each classifier using default settings. Then we perform an exhaustive grid search utilizing 3-fold cross validation to find each classifiers respective optimal hyper-parameters with optimal preprocessing to obtain our highest achievable test accuracy.

6 Proposed Solution and Experiments

6.1 Optimal Preprocessing

6.1.1 Unigram vs. Bigram Language Model

For each classifier we will compare unigram language modeling against bigram language modeling. Both of these language models are considered to be bag of words models meaning that positional information is largely ignored and documents are viewed as frequency counts of terms. Unigram language modeling is based strictly on the counts of each term in each class whereas bigram language models include counts for terms following other terms. This means that bigram language models count term frequencies along with the frequencies with which each word follows a preceding term. Essentially bigram counting allows for conditional probabilities to be calculated conditioning on a given previous word. We compare 3-fold cross-validation accuracies for our models using both unigram and bigram counts to find which results in higher accuracy.

6.1.2 Tf-Idf Weighting

When Tf-Idf weighting is included in preprocessing, term frequencies are adjusted by their inverse document frequencies. Scikit Learn [**?**] describes this mathematically as:

$$tf - idf(t, d) = tf(t, d)idf(t)$$

Where $tf(t, d)$ is the term frequency in a document and $idf(t)$ is the inverse of the number of documents containing the term t in the corpus which is represented as:

$$idf(t) = \log(\frac{1 + n_d}{1 + df(d, t)}) + 1$$

Here n_d is the total number of documents in the corpus and $df(d, t)$ is the count of documents including the term. When inverse document frequency is included, high frequency terms like determiners which also exist in all documents are essentially meaningless while terms that are used in fewer documents but in higher frequency in certain documents are up-weighted during counting. By varying including Idf weighting and excluding Idf

weighting on our classifiers we will observe the impact of the weighting scheme on our classifier's cross-validation accuracy.

6.2 Classifiers and Hyperparameters

Once the pipeline is constructed we perform an exhaustive search for optimal hyper-parameters using 3-fold cross-validation over a range of hyper-parameters for each classification algorithm. The algorithms that will be tested are Multinomial Naive Bayes, and Stochastic Gradient Descent. Each classifier has its own set of hyper-parameters which are selected based on their performance on the validation set. Once optimal hyper-parameters are found, each classifier is run on the test data and accuracies are compared.

6.2.1 Multinomial Naive Bayes Experiments

The equation for the Naive Bayes classifier is expressed mathematically in the Scikit Learn documentation [?] as:

$$f_{NB}(y) = \arg \max_{y \in Y} P(y) \prod_{i=1}^{n} P(x_i|y)$$

Where Y is the set of classes y and $P(y)$ is a given class's probability and $P(x_i|y)$ is the marginal class conditional distributions. $P(y)$ and $P(x_i|y)$ are learned from the training data. Since Multinomial Naive Bayes pertains to multi-class classification, $\phi_{y_i} = \frac{N_{y_i} + \alpha}{N_y + \alpha n}$. Here ϕ_{y_i} returns the maximum likelihood that a term is a member of a given class. The hyper-parameter for Multinomial Naive Bayes is a smoothing term α to correct for out of data observations. Initially no smoothing is included to set a baseline accuracy score. Then, a wide range of α differing in multiples of 10 from from .0001 to 1000 will are searched initially and then a specific range around the best performing power of 10 is searched more exhaustively.

6.2.2 Stochastic Gradient Descent Experiments

The Stochastic Gradient Descent classifier in Scikit Learn allows for different loss functions to be passed as a hyper-perameter, where gradient descent is performed to find the

7

optimal local minima of the training error corresponding to highest accuracy. The Scikit Learn documentation [**?**] expresses this mathematically as

$$E(w, b) = \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i)) + \alpha R(w)$$

where L corresponds to a loss function, R is a regularization term and α is a regularization multiple. Selecting the Logistic loss function $\log(1 + e^{-y_i g(\boldsymbol{x})})$ where $g(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + b$ effectively turns this classifier into Logistic Regression, while the Hinge loss function $\max(0, 1 - y_i g(\boldsymbol{x}_i))$ implements a Linear Support Vector Machine classifier and the Perceptron loss implements the Perceptron algorithm. Each of these classifiers share a regularization hyper-perameter that can be set to $l1$ corresponding to the l_1 norm $||x||_1 = \sum_{i=1}^{n} |x_i|$, $l2$ corresponding to the l_2 norm $||x||_2 = \frac{1}{2} \sum_{i=1}^{n} x_i^2$ or to *none* for no regularization. Again initially a baseline is recorded using default parameters followed by a search of a wide range of α's differing in multiples of 10 from from .0001 to 1000. Once a region of high cross validation accuracy is found a search around the best performing power of 10 is conducted to find the highest accuracy.

7 Experiments and Results

7.1 Baseline Algorithms

Our first experiment found baseline accuracies for our classifiers with no hyper parameter tuning, no Tf-Idf weighting, and unigram language modeling. The Multinomial Naive Bayes classifier with $\alpha = 0$ achieved 62.25% on the test set while Stochastic Gradient Descent classifier with default parameters achieved 69.98% accuracy. The defaults for the Stochastic Gradient Descent classifier implement the Hinge loss function so it is essentially performing Support Vector Machine classification.

7.2 Optimal Preprocessing

For both classifiers we found that bigram language models and including Tf-idf weighting increased our accuracy. **??** below is representative of the impact of including Tf-Idf

weighting. Regardless of the type of classifier or the type of loss function implemented we found that there was a uniform increase in cross-validation accuracy for including Tf-Idf weighting. Likewise, bigram language model counting consistently dominated unigram counting as shown in **??**. The results of these tests lead us to conclude that optimal preprocessing for our corpus includes bigram language modeling and Tf-Idf weighting.

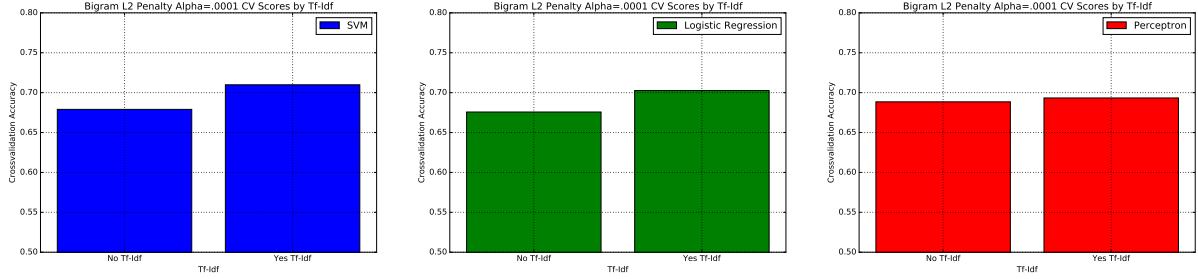
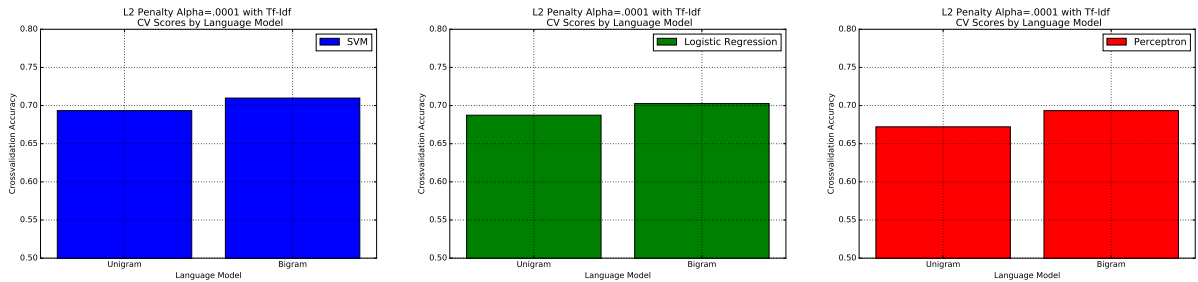Figure 2: Tf-Idf On vs. Off for SGDClassifier with Varied Loss Functions

Figure 3: Unigram vs. Bigram for SGDClassifier with Varied Loss Functions

7.3 Optimal Hyper-parameters and Results

For both Multinomial Naive Bayes and Stochastic Gradient Descent experiments we included Tf-Idf weighting and bigram language models. For Multinomial Naive Bayes we found an optimal α smoothing parameter at $\alpha = .05$ which resulted in a 3-fold cross-validation accuracy of 68.9% and a test accuracy of 70.6%. The Stochastic Gradient Descent classifier has many more hyper-parameters. The most important of these is the loss function because it determines the classification function. The Hinge loss function was found to be the most accurate in initial tests but we include the other loss functions in all cross-validation grid searches. While each loss function has its own set of optimal hyper-parameters, in general we find that the $l2$ norm penalty with a small regularization

constant multiple α worked well with the Hinge loss function. We then searched a range of α's close to 0 as seen in **??**. An optimal value of $\alpha = 0.0001$ resulted in a 3-fold cross-validation accuracy of 70.9% and a test accuracy of 74.8%, which is the highest test accuracy we observed.
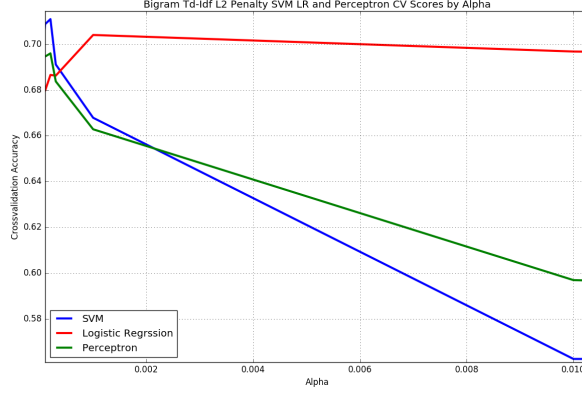

Figure 4: SGDClassifier with Hinge Loss and $l2$ Penalty Over Range of α's

8   Discussion and Future Work

Overall our results seem to match those found in previous research. Thomas et al. note [**?**] that Support Vector Machines achieved the highest test accuracy of 71.2% in predicting if a speaker would vote for or against a bill using this same corpus. This is similar to our result in that the Hinge loss function is the same used by Support Vector Machines and political party affiliation may be highly correlated with positions on a given bill. Our higher accuracy may be do to many factors including our performing a more exhaustive hyper-parameter search, or even the fact that not all votes are cast along party affiliation lines. An interesting future project to perform in the same vein as this research would be to test a recurrent neural network's performance, as mentioned by Iyyer [**?**], and see if accuracy increases.

References

[1] Mohit Iyyer, Peter Enns, Jordan Boyd-Graber, and Philip Resnik. Political ideology detection using recursive neural networks. In *Proceedings of the Association for Computational Linguistics*, 2014.

[2] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.

[3] Matt Thomas, Bo Pang, and Lillian Lee. Get out the vote: Determining support or opposition from congressional floor-debate transcripts. In *Proceedings of the 2006 conference on empirical methods in natural language processing*, pages 327–335. Association for Computational Linguistics, 2006.

[4] Bei Yu, Stefan Kaufmann, and Daniel Diermeier. Classifying party affiliation from political speech. *Journal of Information Technology & Politics*, 5(1):33–48, 2008.