Analyzing Donations to 2016 Presidential Candidates

Raphael Palefsky-Smith and Christina Wadsworth
Stanford University
`rpalefsk@stanford.edu, cwads@stanford.edu`

1 Introduction

With an election cycle coming up, there's a huge amount of discussion and media focus on which groups - and what kinds of individuals - will vote Democrat and which will vote Republican. We wanted to apply a machine learning approach - given details about a person, can we determine whether they're a Democrat or Republican? Thankfully for society but unfortunately for our project, there's no public dataset on how individuals vote. So for our approach, we used the next best thing: donations to presidential candidates. Whenever anyone gives to a candidate, the candidate is required by law to publicly disclose the donor's (self-reported) name, occupation, workplace, donation amount, date of donation, and full address. These records are available as huge CSV's from the Federal Election Commission, and so our data collection consisted of a simple download!

The input to our algorithm was one of these donation records, minus the candidate. We then used Logistic Regression to output a predicted political party for the donor, "Democrat" or "Republican." This is a simple binary classification problem, so most of the work was in the pre-processing and feature optimization stages.

Our baseline for accuracy was a human's performance on the same task. We pulled 20 random samples from the dataset, and attempted to classify them ourselves using every Democrat/Republican stereotype in the book. To our surprise, we only got half of them right - our intuition was no better than guessing. So, we were particularly excited to buck the "computers can only approximate human ability" trend and potentially create a classification model that was better than a human.

This was a joint project with CS221, but our work for that class was only related in that it uses the same dataset - our goal in CS221 was to use constrained k-means to find optimal locations for candidate fundraisers. Christina Wadsworth is not in 229, so her work was limited to the k-means efforts. So, all of the work presented here (beyond basic data acquisition) was solely for CS229 and performed by Raphael Palefsky-Smith.

2 Related Work

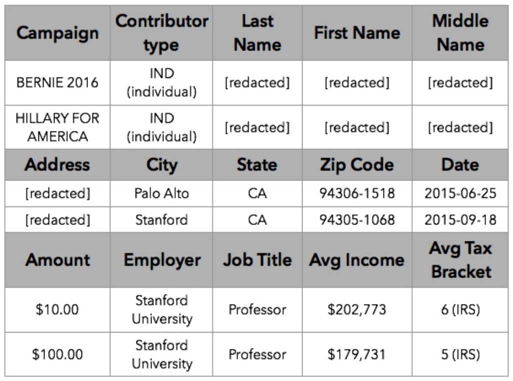
The first paper we found was "Political Campaigns and Big Data" [1] by David W. Nickerson and Todd Rogers, which gave a general overview of data processing techniques employed by real-world political campaigns. While it didn't delve into our specific problem, it noted that campaigns had used supervised learning, including logistic regression, to analyze "propensity to donate." Despite its generality, we felt that the paper validated that our model was appropriate for the task at hand. After all, if real campaigns used these techniques, they're probably good for our purposes, too.

Then, we looked at concrete approaches to predicting donations and political sentiment using other data sources, namely blog posts and tweets. In "Mining Sentiment Classification from Political Web Logs" [2], Kathleen T. Durant and Michael D. Smith predict a blog post's political affiliation with 78% accuracy using Naive Bayes. Similarly, Colin Conrad, in his paper "Predicting Political Donations Using Data Driven Lifestyle Profiles Generated from Character N-Gram

Analysis of Heterogeneous Online Sources" [3], used Common N-Gram techniques to predict the donations of Twitter users with over 65% accuracy. He used the exact same donations dataset we used, similarly employing the candidate as a label, but using tweet data as his algorithm's input as opposed to our use of the donor's personal information. The approaches of Durant/Smith and Conrad are fascinating, but they require the donor to have written blog posts or tweets. Our algorithm has the advantage of operating solely on the donor's personal information, without the donor having written a single word.

We then examined an approach that, like us, used demographic data to draw conclusions. In "An analysis of the 2002 presidential elections using logistic regression" [4], Jairo Nicolau used surveys of voters in a 2002 Brazilian election. Respondants were asked for their age, gender, skin color, schooling, religion, party sympathies, and who they voted for. Instead of using ML techniques as we did and building a classifier, Nicolau took a "pure statistics" approach and used logistic regression to determine the most important factors in an individual's vote. His methodology had distinct advantages over ours - his data contained more demographic attributes, and he had actual data on voting behavior rather than our "analogue" of donations. However, our approach doesn't require expensive polling, instead relying on freely available public information.

Finally, we found a non-academic project that approached our exact problem with the same data and same algorithms that we used. Jasmine Wilkerson's Political Party Affiliation Predictor [5] used the same FEC dataset to predict Republican or Democrat using logistic regression. Her model used slightly different features, incorporating the donation amount feature and including records from Senate and House candidates. Since she didn't report her model's accuracy, we could not compare it to our own, but it was reassuring to see that others have attacked the problem with almost identical methods!



Campaign	Contributor type	Last Name	First Name	Middle Name
BERNIE 2016	IND (individual)	[redacted]	[redacted]	[redacted]
HILLARY FOR AMERICA	IND (individual)	[redacted]	[redacted]	[redacted]

Address	City	State	Zip Code	Date
[redacted]	Palo Alto	CA	94306-1518	2015-06-25
[redacted]	Stanford	CA	94305-1068	2015-09-18

Amount	Employer	Job Title	Avg Income	Avg Tax Bracket
$10.00	Stanford University	Professor	$202,773	6 (IRS)
$100.00	Stanford University	Professor	$179,731	5 (IRS)

Figure 1: Actual Stanford Professors in Dataset

3 Dataset and Features

Our dataset was comprised of donations to the four candidates with the most money raised at the time: Hillary Clinton, Bernie Sanders, Jeb Bush, and Ben Carson. This was downloaded from the Sunlight Foundation's Influence Explorer [6]. We also augmented our data with income-by-ZIP-code statistics from the IRS [7]. All together, our data consisted of almost 300,000 individual donations and 27,000 ZIP codes.

Most of our pre-processing was filtration of the donation set. The raw data contained donations from PACs and businesses, which we removed in order to only analyze individuals. We then removed duplicate donors (ones who had donated multiple times and therefore showed up more than once) to arrive at a little more than 100,000 unique individuals. We then removed donation amount and data information, so our model would generalize beyond an individual "donation event." After building a model with over 90% accuracy on a test set, we tried it out on our friends, asking them to provide their information and see how our classifier faired. To our surprise, despite its incredible accuracy on the test set, it was *wildly* inaccurate in the real world, correctly classifying only 1 out of 7 of our friends. After manually digging throug our data, it turned out that the donor's title (Mr. / Ms.) was only reported by Republican candidates! This was a great reminder that accuracy can be misleading, and one must always fully understand their data. So, the "title" feature was removed as well. Then, we augmented each individual with the average income in their ZIP code, and the IRS tax bracket corresponding to that average in order to discretize and contextualize the raw number. Sample data is shown in Figure 1. Finally, we use Scikit-Learn's [8] DictVectorizer to transform our data into templated features. For instance, a donor with zip code 95062 will have zip_95062 = 1 and zip_12345 = 0.

4 Methods

To build our model, we settled on the Stochastic Gradient Descent learning algorithm because of its balance of speed and accuracy. After trying out several objectives (hinge, logistic regression, ordinary least squares), we settled on Logistic Regression, as it provided the best accuracy - SVM had 0.4% higher test error, and least squares added over 60% error!. The Stochastic Gradient descent rule is as follows (where h_θ is the objective function):

```
Loop until convergence {
    For $i = 1$ to $m$ {
        For $j = 1$ to $n$ {
            $\theta_j := \theta_j + \alpha(y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)}$
        }
    }
}
```

The algorithm is pretty simple. For each training example, it loops through each feature. For each of these features, it makes a prediction using the current "best guess" objective function h_θ. When this is subtracted from the label y, we get a measure of "how wrong" the guess was. This "error factor" is multiplied by a learning rate α and the value of the feature itself to form a delta, which is then used to update the best-guess parameter vector θ.
In more practical terms, Stochastic Gradient Descent works by evaluating one feature at a time, computing the gradient for the objective at that point, and nudging the parameters in the opposite direction. Bit by bit, these "nudges" push the parameters to a local minimum. Since the loss functions used have only one minimum, this local minimum ends up being the global minimum as well! As to our specific objective function, Logistic Regression looks like this:

$h_\theta(x) = g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$

where $g(z)$ is called the sigmoid function. The advantage of using g is that it smoothly "squishes" the output of $\theta^T x$ to lie between 0 and 1. This means it works wonderfully for binary classifiction problems like ours, since y will only ever take on the values 0 or 1 as well. Therefore, the decision boundary is just $h_\theta(x) > 0.5$. Simple, and it performs well!

5 Experiments/Results/Discussion

We used Scikit-Learn for all aspects of our application. Specifically, we employed its built-in Stochastic Gradient Descent implementation. This means we got hyperparameters "for free." We still played around with them ourselves, setting the learning rate to a several constant values (our best, 0.1, had 21.2% error), but none of them outperformed Scikit's built-in defaults based on years of research. For instance, it defines the learning rate as $\frac{1}{(\alpha*(t+t_0))}$ where $\alpha = 0.0001$ and t_0 is determined by a heuristic proposed by Leon Bottou. We opted not to use mini-batches, since we had no need to formulate the problem as online learning.

Our primary metric was simple accuracy on a test set - while we eventually evaluated precision, recall, and F1 (the harmonic mean of precision and recall), our "how well are we doing" metric was accuracy, the number of examples correctly classified divided by the total number of examples. Since we had relatively plentiful data, we were content to use the much faster and simpler hold-out cross validation instead of k-folds. We held out 33% of our samples, chosen randomly, as a test set and used the rest for training.

To pick the optimal feature set, we used a rudimentary but incredibly effective method: brute-force evaluation. That is, we simply computed every possible subset of our features and evaluated the test error on each candidate. We chose our final features as the subset with the highest accuracy. Why did we use this approach instead of more sophisticated methods like backwards search or filter selection? We had only 11 features, so there were only $2^{11} - 1 = 2047$ possible subsets. And since evaluating error was embarassingly parallel, we took advantage of the Stanford CORN system's 16-core machines and distributed the workload. All in all, it took less than ten minutes to find our optimal feature set, and it was great knowing we had the guaranteed global maximum for accuracy. Figure 2 plots test and training error as we evaluated each feature subset, re-arranged from maximum to minimum. It demonstrates the huge difference feature selection can make!

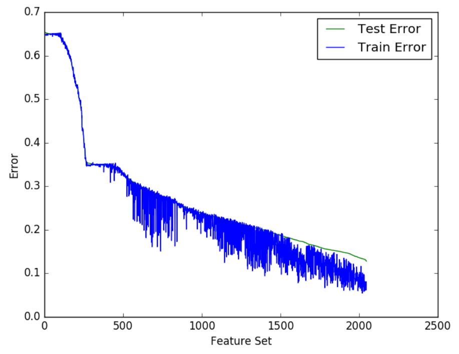


Figure 2: Feature Subset vs Error

Our worst feature set, "area income, city, tax bracket", had over 65% error. Our best set, "first/middle/last name, employer, occupation, state, zip code, tax bracket" had under 13%! We were surprised that average income in the donor's ZIP code actually hurt our accuracy. Furthermore, while that figure discretized into tax bracket improved our accuracy, it contributed less than 0.3%. So, our stereotypes about Republicans being accross-the-board richer ended up not helping much.

Once our features were chosen, we plotted the classic test/training/desired error curves (Figure 3), with an admittedly overly-optimistic desired error of 5%. We got two main takeaways from

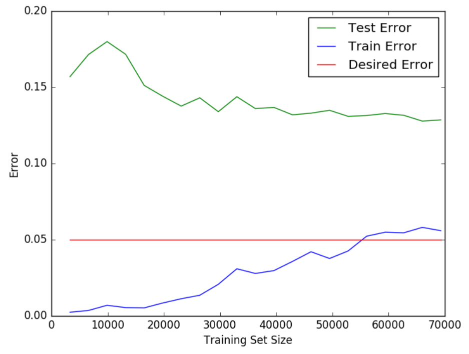
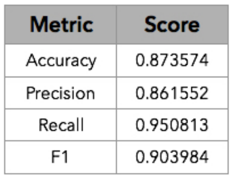


Figure 3: Training Set Size vs Error

this graph. Firstly, our test error seemed to be pretty much flat as we added more training examples. Because of this plateau, we felt confident that we didn't need to acquire more data. Secondly, the relatively large gap between training and test error pointed to a variance problem.

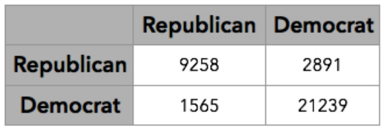
The rest of our experimentation focused on reducing this variance. We recieved three suggestions for tackling the problem: using a simpler algorithm, using fewer features, and adding in regularization. It's hard to get simpler than logistic regression, but just for kicks, we tried least-squares regression and got over 65% error. So that was a non-starter. Using fewer features was also a no-go, since our brute-force feature selection showed that we couldn't get a lower error with any other set. Finally, we tried out multiple forms of regularization, including L1 and L2.

Admittedly, we did not exhaustively explore the hyperparameters for these regularizations. Rather, we tried a couple different constant factors for each, and were disappointed with the results. The best we could get with L1 was a 1.2% *increase* in error, and L2 was even worse: it brough our error up to 35%! So, regularization didn't bear much fruit. However, we were still pretty pleased with our results. Here are our key metrics and confusion matrix:



Metric	Score
Accuracy	0.873574
Precision	0.861552
Recall	0.950813
F1	0.903984

(a) Key Metrics



	Republican	Democrat
Republican	9258	2891
Democrat	1565	21239

(b) Confusion Matrix (rows actual, columns predicted)

Figure 4: Evaluating Results

From the confusion matrix, we can determine that our false-Democrat rate is 23.8%, whereas our false-Republican rate is 6.86%. This means we are much, much more likely to falsely categorize someone as a Democrat, which makes sense given that our dataset contains nearly twice as many Democrats as Republicans. One such false-Democrat was a Realtor from Wilsonville, Oregon. While Oregon is an incredibly Democratic state (which our model learned), there are always exceptions!

Why did Logistic Regression perform best? In our trials, it outperformed SVM by less than a percent, so it's not as if Logistic Regression blew everyone else out of the water. We attribute this slight performance boost to its absolute simplicity - the simpler the algorithm, the less chance of overfitting. Nevertheless, due to the variance problem evident from our graph, we believe we still had a slight case of overfitting. While we attempted to solve this issue through regularization, we didn't succeed, and removing features only increased our error. So, our (not-too-terrible) overfitting remained unmitigated.

6 Conclusion/Future Work

Despite our variance issue, we were still surprised by our accuracy: given just someone's name, job, and address, we can predict their political affiliation with over 87% accuracy! And thanks to the relatively small number of features, we could compute the guaranteed optimal feature set via brute-force optimization. Our accuracy is especially impressive considering the abysmal (50%) performance of humans. Evidently, our model picks up on trends that a human will miss!
Our best performing algorithm, Logistic Regression, is also one of the simplest. We believe this simplicity, coupled with our relatively few features, reduced overfitting and kept our variance much lower than it could have been. Sometimes, simpler is better!
Finally, if we had more time and (wo)manpower, we would definitely attack the variance problem. We believe it would possible, with the right sort of regularization or more nuanced learning algorithms, to close the training-test error gap by a couple more percent at least. That is to say, given a year of work by a team of five, our hunch is that we could get above 90% accuracy. But given our constraints and "rudimentary" methods, 87% isn't half bad.

7 References

[1] Nickerson, David W., and Todd Rogers. Political Campaigns and Big Data. The Journal of Economic Perspectives 28.2 (2014): 5173. Web...

[2] Durant, Kathleen T., and Michael D. Smith. ”Mining sentiment classification from political web logs.” Proceedings of Workshop on Web Mining and Web Usage Analysis of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (WebKDD-2006), Philadelphia, PA. 2006.

[3] Conrad, Colin. ”Predicting Political Donations Using Data Driven Lifestyle Profiles Generated from Character N-Gram Analysis of Heterogeneous Online Sources.” (2015).

[4] Nicolau, Jairo. ”An analysis of the 2002 presidential elections using logistic regression.” Brazilian Political Science Review 1.1 (2011): 125-135.

[5] Wilkerson, Jasmine. ”Political Party Affiliation Predictor.” DataScienceDojo. DataScienceDojo, 30 June 2015. Web. 10 Dec. 2015.
`<http://demos.datasciencedojo.com/demo/political-party/>`.

[6] ”Real-Time Federal Campaign Finance.” Influence Explorer. Sunlight Foundation, 10 Dec. 2015. Web. 10 Dec. 2015.
`<http://realtime.influenceexplorer.com/newest-filings/#?ordering=-filing_number&min_raised=1&committee_class=P&time_range=2016_cycle&report_type=monthly>`.

[7] ”SOI Tax Stats - Individual Income Tax Statistics - ZIP Code Data (SOI).” IRS. IRS, 27 Aug. 2015. Web. 10 Dec. 2015.
`<https://www.irs.gov/uac/SOI-Tax-Stats-Individual-Income-Tax-Statistics-ZIP-Code-Data-(SOI)>`.

[8] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.