Introduction

The 2016 election came as a surprise to most political pundits. Almost no well-respected non-partisan pundits predicted that Donald Trump had a realistic chance of winning. Clinton consistently led in polls in several states, and most models gave her a much higher probability of winning, including models like the *538* and the *Huffington Post*, the "gold standard" of election forecasting. While the 538 model incorporates additional complexities (such as economic data, historical performance of polling organization, state biases, etc.), the *Huffington Post* relies more heavily on raw polling data, using a Bayesian Kalam Filter time series that dynamically updates to new polling information. Both models failed to predict the presidential vote in states such as Wisconsin, Michigan, and Pennsylvania. Furthermore, the 538 model incorrectly predicted the senate elections in Pennsylvania and Wisconsin, with the *Huffington Post* also incorrectly predicting Indiana[12].

However, the failures of these models are not entirely surprising, as there are several challenges in creating such a model. First, unlike other data sets that have millions of instances that a model can trained on, polls are often far and few between, as each one requires an extensive effort by a polling organization to perform, often lastly a couple weeks. Second, while in some races polling clearly points to one candidate over another, in tight races, polls are often mixed, with candidate leads very close to (if not within) the poll's margin of error. Third, polls often use different methodologies, the efficacies of which are contested by various analysts (is an online poll of 1000 voters better than a telephone poll of 300?).

It is of obvious concern for us to know which polls and models we should look to. Beyond the appeal to political junkies and campaign operations, knowing how important it is for you to vote in a given

[1] https://www.huffingtonpost.com/entry/pollster-forecast-donald-trump-wrong_us_5823e1e5e4b0e80b02ceca15
[2] https://projects.fivethirtyeight.com/2016-election-forecast/senate/

election is a practical benefit to many citizens. Unfortunately, beyond being incorrect, the underlying mechanics of the aforementioned models are often difficult to understand, due to the layered complexities in both models. What is there was a simpler, “out of the box” method that could perform comparably. Our task will be to try and build such a model: one that, based on previous years’ polls and election outcomes, can accurately predict an election outcome given a poll.

Methods

Polling data was collected using the *Huffington Post*'s *Pollster* API. Senate election polls were collected from years 2012, 2014, and 2016. The results of those elections were collected from *Ballotpedia.* The data was cleaned and preprocessed in *R*. Polls that had multiple candidates were thrown out, as well as polls that included candidates that lost in their party's primaries and polls of elections that resulted in run-offs, in order to simplify the model. The final datasets included 12 variables. These were the Republican polling margin (the amount the Republican candidate led the Democratic candidate by), the length of time over which the poll was conducted, the number of days before the elections the poll was conducted, the sample size of the poll, whether the poll surveyed registered or likely voters, whether the poll had partisan affiliation, and if so, whether those affiliations were Democratic or Republican, whether the poll was conducted online, over the phone with a live person, over the phone with an automated response, or over the phone with an interactive response (these last 4 categories are not exclusive, as many polls used a mixture of methods). Lastly, a binary response variable of whether the Republican candidate won or loss the election was also included.

The training data included polls from 2012 and 2014. There are 887 polls included in that set, with 375 being Republican wins, and the rest being Democratic). The test set included 524 polls from the 2016 senate elections, of which 378 were Republican wins. Although there was a class imbalance, it was not deemed severe enough to justify the creation of synthetic data, and under sampling techniques were ruled out due to the already small nature of the dataset. Instead, 5-fold cross validation will be used.

It is worth taking a look at the distribution of the data. Below are density plots for each variable. As we can see, the distributions of each variable have a high degree of overlap for Republican victories (red) and Democratic victories (blue). The only variable that seems to have some degree of linear separability

(although it's certainly not linearly separable, and we would have to content ourselves with quite a few classification errors) is the Republican margin variable, which is to be expected (races that Republicans

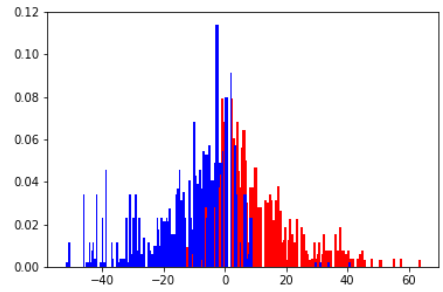

Figure 1: Republican Margin

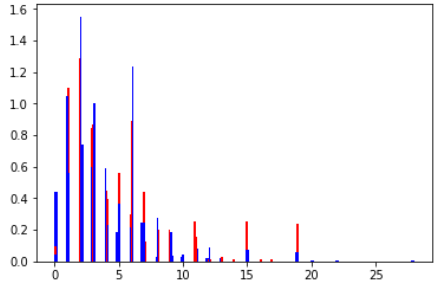

Figure 2: Length of Poll

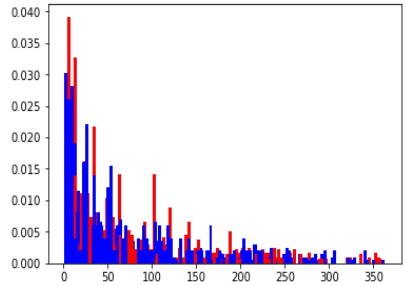

Figure 3: Days Before Election

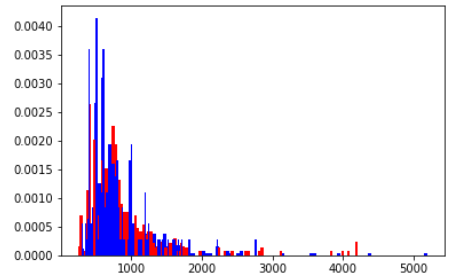

Figure 4: Sample Size

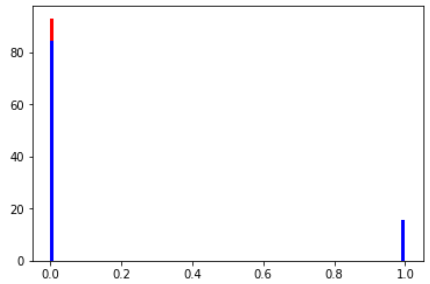

Figure 5: Registered Voters

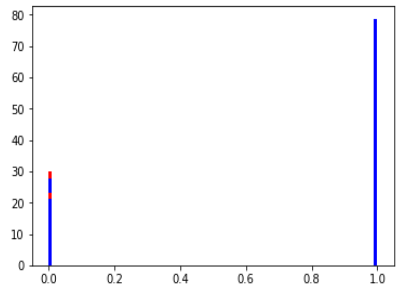

Figure 6: No Partisan Affiliation

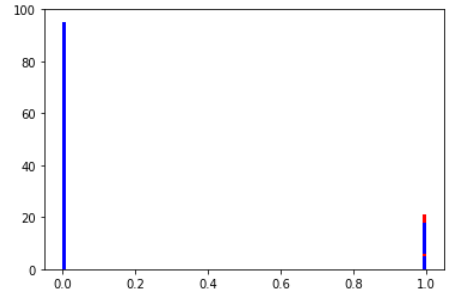

Figure 7: Democratic Affiliation

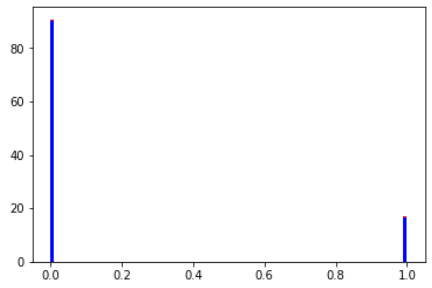

Figure 8: Republican Affiliation

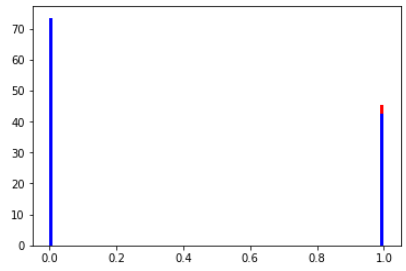

Figure 9: Online

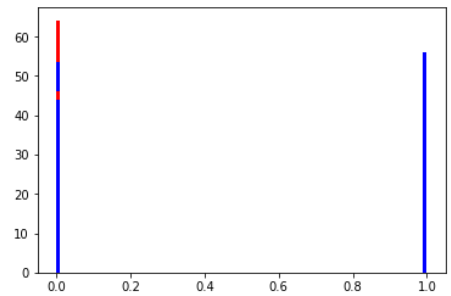

Figure 10: Live Phone

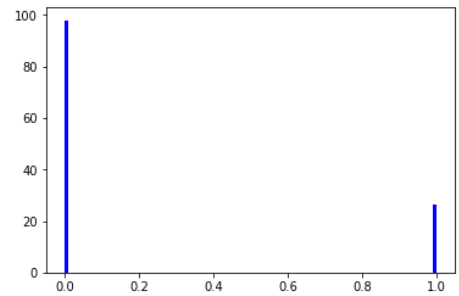

Figure 11: Automated Phone

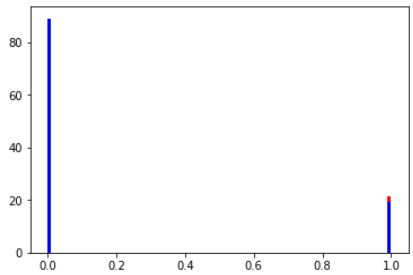

Figure 12: Interactive Voice Response

are leading by over 60 percent rarely go blue). However, there is a great deal of overlap even in this case, with Republican margins as high as 40% still resulting in Democratic victories. All the other variables have a high degree of overlap. This can also be seen by a dimensionality reduced view of the data via TSNE (figure 13).

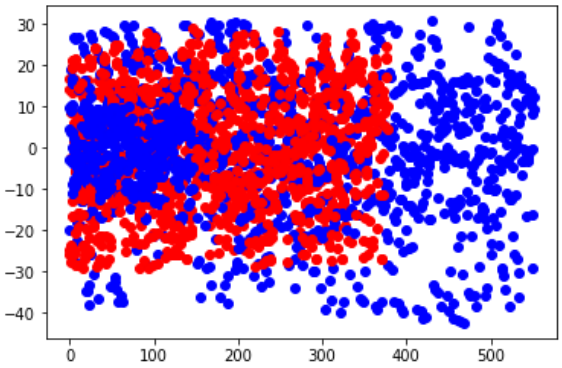

Figure 13: TSNE Plot

Taken together, this information suggests that we need a classification tool that can disentangle overlapping data. Our two tools of choice will be a support vector machine classifier, and a deep neural network.

A support vector classifier tries to fit a boundary to the data that maximizes the margin between the nearest points of each class and the boundary. While support vector classifiers were originally designed to fit linear boundaries, they can fit highly irregular, non-linear boundaries as well. This is done by the

use of a kernel, which projects the data into a higher dimension, increasing the separation between the points. Looking at our data, it doesn't appear as if a simple polynomial or elliptical kernel will do the trick. Instead, we will use a Gaussian, RBF kernel. This kernel actually projects our data to an infinite dimensional space, that is controlled by a few parameters. These parameters (a cost function for misclassified points *C*, and a variance 'gamma') prevent our function from interpolating the data. If our gamma parameter went to zero, then we would have projected to a space that has the same number of dimensions as we have data points, allowing us to classify each data point in the training set perfectly. However, this would clearly perform poorly on our test set. As gamma tends towards infinity, we get a linear boundary back, which will also perform poorly on our training and test sets. Since it will be difficult to fit a boundary perfectly (some points will inevitably cross the boundary), our parameter C imposes a cost onto those misclassifications, with higher values of C penalizing misclassifications more heavily. We will search over a range of hyperparameters in order to determine the best possible combination of these parameters that will maximize our performance on the 5-fold cross validated training set. *Sklearn's* support vector classifier function will be used to perform this analysis.

We will also apply a similar hyperparameter search to the neural network. In particular, we are interested in the number of layers that will exist in our network, the number of artificial neurons in each layer, the 'learning rate', which controls the speed that we will update our connection weights, the batch size of training instances we will feed to the neural network at each step, and the activation function we will use (either a hyperbolic tangent, or an exponential linear unit). Our neural network will also make use of an Adam's optimizer (which is regarded as one of the best available in the field), and a binary cross-entropy loss function that we will minimize. *Tensorflow* will be used to construct and train the neural network, with some helpful code borrowed from Aurélien Géron's *Hands on Machine Learning.*

Results

After running a hyperparameter search over 2200 hyperparameter combinations, the optimal parameters for the support vector classifier were found to be C=98, and gamma = 0.0001. This low value of gamma suggests that our boundary will be highly irregular, and the high value of the cost function suggests that misclassifications were heavily penalized when fitting the boundary. The cross-validation routine prevented egregious overfitting however, as we obtained a test accuracy of 64.3% and the area under our receiver operator characteristic curve was .792. This is noticeably better than a random classifier, as can be seen in the ROC curve plot below (figure 14).

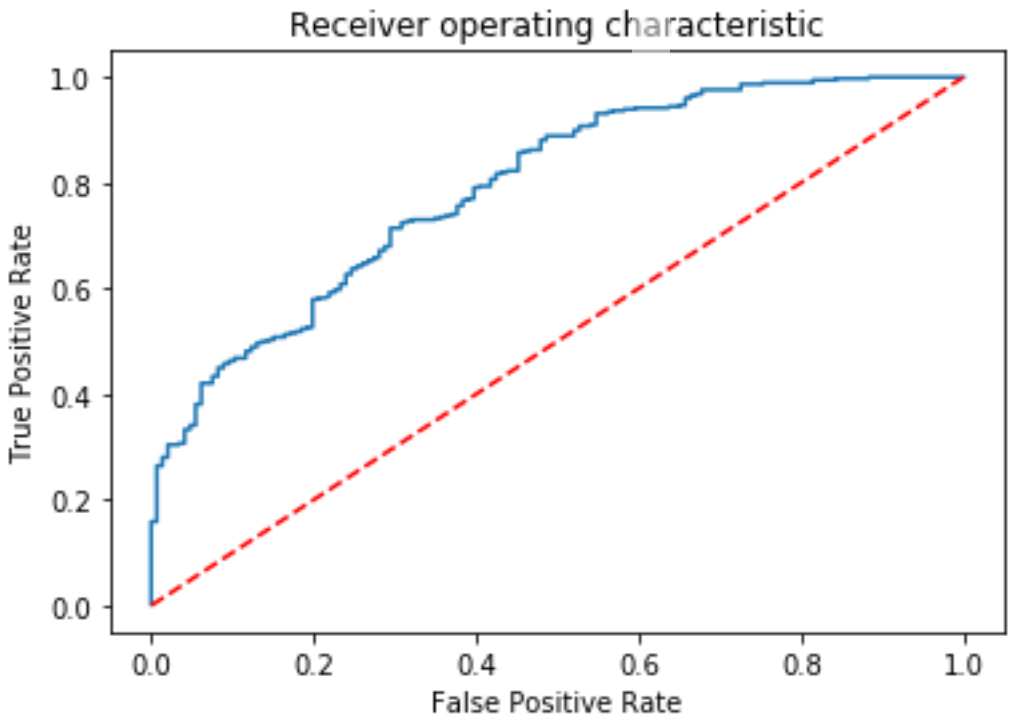

Figure 14

We performed a similar search over the hyperparameters for the neural net. In total, 720 parameter combinations were tested, with each combination being trained until performance on the validation set

remained the same for 20 iterations. The best parameter combination was 5 layers, of 250 neruons each, a learning rate of 0.01, a hyperbolic tangent activation function, and a batch size of 75. This resulted in a final test set accuracy of 56.9%, and had an area under curve score of 0.727. The plot of the ROC curve is shown below (figure 15), and the final result summary is shown in figure 16.

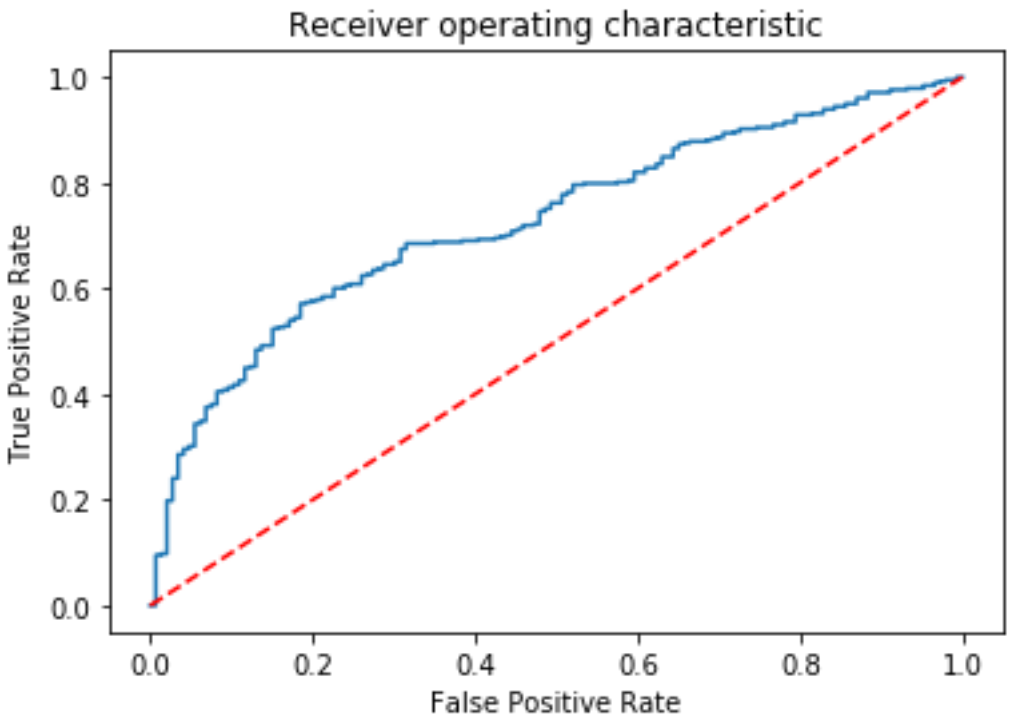

Figure 15

Model	Parameter 1	Parameter 2	Parameter 3	Parameter 4	Parameter 5	Test Accuracy	AUC
SVC	C = 98	Gamma = .0001	NA	NA	NA	.643	.792

DNN	Neurons per layer = 250	Number of layers = 5	Activation function = tanh()	Learning rate = 0.01	Batch Size = 75	.569	.727

Figure 16

Conclusion

Unfortunately, our models failed to obtain any kind of noteworthy performance. While both models performed better than a random classifier, the neural net managed to do so only barely. The support vector classifier performed much better, but still managed to misclassify the test data over a third of the time. The neural net performance could be improved by searching a wider array of possible parameters, such as different activating functions, optimizers, and initializing strategies, but ultimately gains in performance would likely be marginal. The failure of larger layered networks to perform better suggests that a more complicated architecture does not improve performance, as networks with larger numbers of layers were rejected in favor of simpler ones.

Ultimately, the neural network and support vector classifier are only as good as the data that they received. Here-in lies the problem. In a post-hoc analysis of their modeling, the *Huffington Post* concluded that the problem was not in the particular model that they were using in 2016, but rather in the data itself[3]. The polls in the key states in 2016 were consistently biased towards Clinton and Senate Democrats, something that our model likely also inferred. The same analysis also included an interesting bit of information about the polls used by the *Huffington Post*, and by extension our data set, since we drew polls from their API. They tended to sample more online polls and excluded polls that used solely landlines (the phone polls included used both landlines and mobile phones). However, models that used landline-based polls tended to perform better in 2016, suggesting another possible source of bias that was built into our data set.

However, despite this, the *Huffington Post* and 538 model performed much better than our model. Part of that is likely due to the sophistication of those models, but also, in part, to the nature of the questions that we asked. Our question, while directly about predicting election outcomes accurately, also focused

[3] https://www.huffingtonpost.com/entry/pollster-forecast-donald-trump-wrong_us_5823e1e5e4b0e80b02ceca15

in part on what polls we should look at. We wanted our model to be given any poll within a year, and accurately predict the outcome of the election from that poll. In other words, we were predicting response variables from *each poll*, but many of those polls (due to the days from the election they were conducted, as well as the above biases) were simply bad data, that told us little about the eventual outcome of the election. The *Huffington Post* and 538 models used a time series analysis, where the task was to predict an outcome for a given state, after seeing a series of polls. This allowed those models to discount older polls for their final predictions, something our model was not able to do. It would be interesting to assess the performance of those models 100 days before the election instead of the night before and see if they obtain a similar accuracy (the answer is almost certainly no).

The rather limited nature of our training data also prevented our models from learning biases associated with polling in each state over previous election cycles, something most well-regarded polling models include. Also, information about whether a given sample of voters were primarily in urban, suburban, or rural areas, could have helped develop a better model, as each of those populations have well studied voting behavior that could be brought to bear.

Ultimately, election forecasting is difficult work, and even the best models get it wrong quite a bit of the time. The performance of our support vector classifier, although not exceptionally noteworthy, was surprisingly good given the conditions it was tested under (asked to accurately predict the outcome of any poll conducted at any point in the year preceding an election). This suggests that future work could be put into integrating such a classifier into a more complicated model that includes a timeseries approach, and built in state biases, which could hopefully improve its performance.