STAT 359: Binary Response Data
Final Project Report
John Lee

Using Machine Learning to Predict the 2016 Trump Vote

Abstract

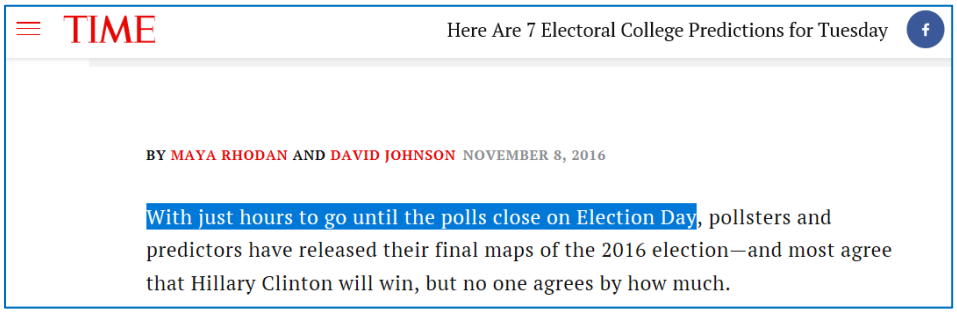
For my final project in STAT 359, I decided to use machine learning methods to predict the 2016 Trump vote. My dependent variable is a binary outcome: 1 = voted for Trump, 0 = voted for someone else. I will use three types of methods: (1) classification tree-based methods (e.g., bagging, random forests, boosted trees); (2) logistic lasso, which is a penalized form of the logistic regression; (3) and support vector machines (linear support vector classifier, SVM with polynomial and radial kernels). We covered method #1 (decision trees) in our STAT 359 class; I taught myself method #2 (logistic lasso) through self-study; and I learned method #3 (SVC/SVM) through in STAT 301-3. After the tuning parameters were optimized using cross-validation, my final models achieved a test prediction accuracy of around 90-92%; this indicates that despite the surprising outcome of the 2016 elections, we can actually predict vote choice at the individual level with a high degree of precision.

Part 1: Introduction

Why use machine learning to predict the 2016 Trump vote? The first reason is in the U.S. political system, the president has many formal powers—and thus it actually matters who wins the presidential election. According to the U.S. constitution, the president has the power to appoint cabinet members, who lead federal agencies (e.g., Department of Defense); appoint ambassadors and federal judges; approve legislation; and make treaties with other nations. Every year, the president also proposes an annual federal budget to Congress; and President Trump has proposed cutting billions of dollars in federal support to the U.S. National Science Foundation, the National Institutes of Health, and Environmental Protection Agency (Ledford et al. 2019).

The second reason is that for the vast majority of professional analysts, survey researchers, and academics, Trump's victory over Hilary Clinton in the 2016 U.S. presidential elections was a huge surprise. Even up to the actual day of the election, many pollsters predicted that Hilary Clinton would win (Figure 1).

Figure 1



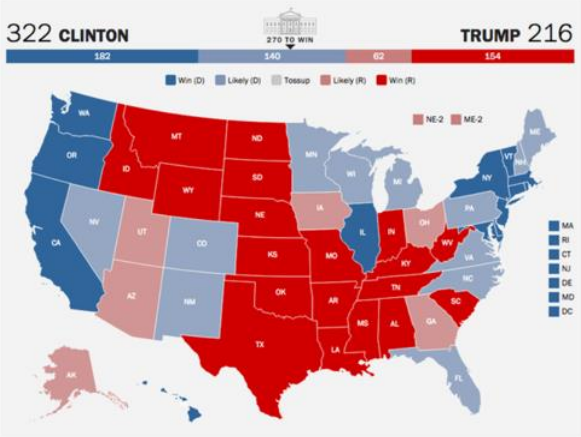
In fact, most believed that she would win by a relatively comfortable margin, with some predicting that she would win by about 100 electoral college votes (e.g., Larry Sabato of UVA), and others such as the *LA Times* predicting a margin of over 150 votes (Rhodan and Johnson 2016). Thus, the first reason for choosing this topic is that it offers an excellent opportunity to use machine learning methods to address an apparent mystery: why did Trump win in 2016?

Key Information on U.S. Presidential Elections

To win the presidency in the U.S. system, the candidate must win a majority of the electoral college votes (which can be thought of as "points"); typically, the candidate who wins the popular vote in a state gets that state's electoral college votes. Thus, to win the presidential election, the candidate has to win a majority of these "points," and not necessarily a majority of the national popular vote. While it may seem counterintuitive, it is possible to lose the popular vote at the national level, but still win a majority of the electoral college votes—and therefore the presidency. That is exactly what happened in 2016: Hilary Clinton won the popular vote by a margin of about 2.9 million votes; however, Trump still won the presidency because he won 306 electoral college votes, compared to Clinton's 232 (*New York Times* 2017).

Figure 2: Predicted v. Actual Results

(1) Predicted Results



Source: Larry Sabato, University of Virginia

(2) Actual Results

Source: *New York Times*

Research Questions

Why were the polls so wrong, even up to the day of the election? Is it because the behaviors of individual voters are difficult to predict? Or, are there other explanations? For example, perhaps the surveys were not representative (e.g., missed voters in rural areas); or, many people may have answered dishonestly when asked by pollsters whom they would vote for. The answers to these questions are important for practical reasons; if democracy is to work properly, voting behavior should not be random. People should generally vote in ways that are consistent with

their self-interest; and if this is the case, then we should be able to predict people's votes by considering their demographic characteristics, policy preferences, and other relevant factors.

My project thus focuses on addressing two important research questions, which are displayed below:

(1) Which models (e.g., SVM, boosted trees) are the best at predicting the 2016 Trump vote?
(2) Which covariates (e.g., race, gender) are the strongest predictors of the 2016 Trump vote?

Part 2: Research Design

Data

My data source is the 2016 survey data published by the American National Election Studies (ANES),[1] which is coordinated by faculty members at Michigan and Stanford. Before every presidential election, the ANES surveys a nationally representative sample of U.S. citizens aged 18 and up. In this initial survey, subjects are asked about their background information (e.g., age, race, employment status), policy preferences, and intended vote choice in that presidential election (Round 1). Then, *after* the election is over, the same respondents are re-interviewed and asked a series of follow-up questions including whom they actually ended up voting for (Round 2). To get to my final dataset, I went through several rounds of filtering. The reason for each step is explained below:

1. In the 2016 study, 4,270 people were initially surveyed before the election; and 3,658 respondents (85-86%) were re-interviewed after the election.[2]
2. Of those who were successfully re-interviewed after the 2016 election, 2,653 respondents (73%) indicated that they had actually voted in the presidential election.
3. Since my study is about predicting the Trump vote among those who actually voted, respondents who did not vote were removed from the dataset. After removing observations with missing data using listwise deletion, I ended up with a **final dataset of 2,394 observations** (or about 90% of those who voted).[3]

The dependent variable (DV) is a binary outcome: $Vote = 1$ if the respondent voted for Trump, and $Vote = 0$ if the respondent voted for someone else (e.g., Hilary Clinton, Gary Johnson, Jill Stein, etc.). In total, 18 predictors were selected. I decided on these predictors after looking through the full dataset, which includes over 1,000 variables; I selected the 18 predictors that I thought were the most relevant for vote choice, based on my substantive domain knowledge as a

[1] ANES website: https://electionstudies.org/
[2] In recent years, the ANES has used a dual-mode sampling strategy. That is, some of the surveys are conducted in face-to-face interviews (mode #1); other selected participants are surveyed over the internet (mode #2). The ANES uses different sampling methods for each mode. For the face-to-face mode, multi-stage stratified cluster sampling is used. For the internet mode, a simple random sample is taken from the U.S. Post Office's list of delivery addresses (residual addresses only).
[3] Listwise deletion can introduce biases if large numbers of observations are deleted and if some subgroups of observations are more likely to be deleted than others; since less than 10% of the otherwise eligible observations were deleted using this procedure, the analysis is unlikely to be biased.

PhD student in sociology. I did not use a larger number of predictors (e.g., 50-60) because that would have resulted in observations with more missing values, due to item non-response; if I had selected a larger number of predictors, doing listwise deletion would have resulted in a much smaller dataset (e.g., 60-70% of the those who voted, instead of 90% of those who voted). Thus, the goal was to strike a balance between choosing enough good predictors and preserving the size of the final dataset. All continuous predictors were standardized (i.e., to have a mean of 0 and a SD of 1). The predictors are listed below, by type (e.g., demographic, socioeconomic); the six continuous predictors are marked by an asterisk (*). All other predictors are binary or multiclass categorical variables.[4] Below is a screenshot of my final dataset (Figure 3).

- Demographic
  - male, race, age*
- Socioeconomic status:
  - education attainment, income band*
- Social, cultural:
  - divorced, religion[5]
- General partisanship, ideology[6]
  - Democratic Party feeling thermometer*, Republican Party feeling thermometer*, conservative scale*, political interest
- Specific policy issues
  - support for Obamacare, support for gun control, opposition to the wall*, pro-choice

Figure 3: Screenshot of the Final Dataset (2,394 x 19)

```
> final_anes2016_dat
# A tibble: 2,394 x 19
   votedfortrump  male race  educ  divorced born_again evangelical religious athiest pol_interest support_obamaca~
   <fct>         <dbl> <fct> <fct>    <dbl>      <dbl>       <dbl>     <dbl>   <dbl>        <dbl>             <dbl>
 1 1                 1 White Non-~        0          0           0         1       0            0                 0
 2 1                 1 White Coll~        0          0           0         0       0            1                 0
 3 0                 0 White Non-~        1          0           0         1       0            0                 0
 4 1                 1 White Grad~        0          0           0         1       0            1                 0
 5 0                 1 Black Non-~        0          0           0         1       0            0                 0
 6 1                 0 White Non-~        0          0           0         1       0            1                 0
 7 0                 0 White Non-~        0          0           0         1       1            0                 1
 8 0                 1 Hisp~ Coll~        0          0           0         1       0            0                 0
 9 1                 1 White Non-~        0          0           0         1       0            0                 0
10 0                 0 White Grad~        1          0           0         1       0            1                 1
# ... with 2,384 more rows, and 8 more variables: support_gcontrol <dbl>, pro_choice <dbl>, age <dbl>,
#   conserv_scale <dbl>, dem_FT <dbl>, income_band <dbl>, oppose_wall <dbl>, repub_FT <dbl>
>
```

[4] All of the categorical predictors are dummy-coded (i.e., 1 or 0) except for educational attainment, which has three categories: no four-year degree, four-year degree, and graduate degree; and race, which has six categories: White, Black, Hispanic/Latino, Asian, Native American, and Other.

[5] There are actually four different religion measures (all are binary predictors): (1) evangelical Christian; (2) "born-again" Christian; (3) atheist; (4) religiosity (1 = religion is important, 0 = religion is not important).

[6] The political party feeling thermometers were originally measured on 0-100 point scales, and they indicate how positively respondents feel about the party (with bigger numbers indicating more positive feelings). The income band was originally measured on a 1-28 point scale (e.g., 1 = under $5,000 per year; 28 = at least $250,000 per year). However, before being used in the models, all continuous predictors were first standardized.

General Analytical Strategy

Below, I provide a brief overview of my general analytical strategy. It involves going through four steps:

(1) Split the data into training and test sets
(2) With just the training set, use 10-fold cross-validation (CV) to perform model selection (i.e., optimize the tuning parameters)
    a. Decision Trees: Bagging, Random Forest, Boosted Trees
    b. Logistic Lasso
    c. SVM: Linear, Polynomial, Radial Kernels
(3) Fit the final optimized version of each model type (e.g., a fully tuned logistic lasso) using the full training set
    a. This step produces the seven "candidate models"
(4) Assess the seven candidate models by using them to generate predictions on the held-out test set – and then compare the test error rates
    a. Which of the candidate models achieved the lowest test error rate?

As indicated in step (1) above, I first randomly assigned 70% of the observations to the training set ($n_1 = 1{,}676$); the 30% remaining observations were assigned to the test set ($n_2 = 718$). Before getting to the results of my analysis, I want to cover two more issues in this section: first, I provide an explanation of why I decided to use CV to perform model selection; second, I also provide a brief overview of the three main model classes used in my paper.

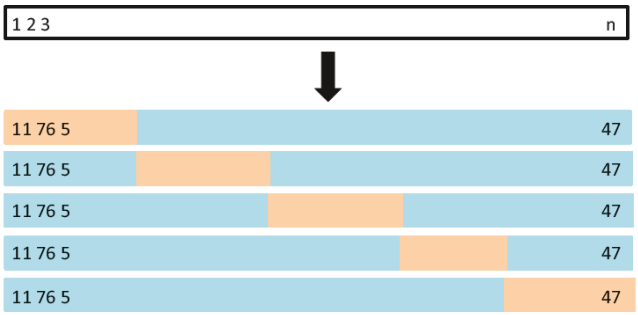
Why K-Fold Cross-Validation?

The performance of machine learning models is often highly dependent on the tuning parameters (e.g., the value of λ for the logistic lasso). For example, let us assume that we want to compare 100 unique values of λ_i for the logistic lasso; how do we know which value will yield the lowest test error rate?

One option is use the "validation set approach." This entails randomly splitting the training set (n_1) into two non-overlapping subsets: a model-building set (n_{1a}) and a validation set (n_{1b}). Next, we fit a model using the model-building set (n_{1a}) and the first value of lambda (λ_1), and then test it against the validation set (n_{1b}). We would then repeat this process 99 more times, so that we have done it once for each λ_i. Afterward, we could compare the 100 validation error rates and identify the value of the parameter that generated the lowest validation error. However, this approach has two important disadvantages (James et al. 2013, pp. 176-178). First, the estimated validation error rate for each λ_i is highly variable, since it is very sensitive to the specific observations that were randomly selected to be in n_{1a} and n_{1b}; if a small percentage of the observations in the n_{1a} were moved to n_{1b} (and vice-versa), the test error rate would likely change. Second, statistical learning models tend to perform more poorly when trained on a smaller dataset; thus, by only training the model on a subset of the full training set (i.e., since $n_{1a} \ll n_1$), we may actually overestimate the test error.

K-fold cross-validation (CV) is a statistically efficient resampling method that addresses both of these problems. The purpose of CV is to provide a more reliable estimate of the test error, which can then be used to compare and evaluate unique values of the tuning parameters (e.g., λ_i). It involves randomly splitting the training set (n_1) into k non-overlapping groups (or "folds") that are equal in size; the first fold is treated as the held-out validation set, and the model is fit using the observations in the remaining $k-1$ folds. This process is repeated until each fold has served as a validation set. Figure 4 below provides an illustration of how this would work $k = 5$ (from James et al. 2013, p. 181).

Figure 4: Example of 5-fold CV



As we can see in Figure 4, since there are 5 folds, there are also 5 estimates of the validation error (which are based on five validation sets displayed in the reddish-brown color); these estimates are averaged to form the CV error rate. The idea is this CV error rate is a more reliable estimate of the validation error since it is based on an average of k estimates (i.e., the CV error rate has a lower variance). Returning to the previous example, if we wanted to test 100 potential values of λ_i, we would perform the k-fold CV procedure for each λ_i; then, we would choose the value of λ_i that yielded the lowest CV error.

Although I will consider seven model types, they really belong to three main types of model classes. Before discussing the actual results of my analysis, I briefly review each of these model classes.

Model Class #1: Tree-based Methods

All three tree-based methods used in this project (i.e., bagged trees, random forests, boosted trees) share some important similarities. Each tree divides the training observations into m non-overlapping regions of the predictor space $\{R_1, R_2, \ldots R_m\}$. Internal nodes refer to the place where the splits are made. The algorithm uses binary recursive splitting, and generally operates in the following way. The splits (i.e., predictor used, values of the predictor) are chosen in order to maximize the purity or homogeneity of the child nodes with respect to outcome class: i.e., in this case, whether or not the respondents voted for Trump (i.e., $Vote = 1$ or $Vote = 0$). As such, the first split has the most explanatory power (i.e., in terms of being able to predict the outcome class of a training observation), the second split has the second most explanatory power, and so on.

Node purity or homogeneity is often measured using the Gini index or entropy. In both cases, the measures are smaller when the nodes (or regions) are more homogeneous; thus, the objective is actually to choose splits that minimize the Gini index or entropy (which is equivalent to maximizing node purity). The Gini index is defined below (James et al. 2013, p. 312). Here, $\hat{p}_{mk}$ represents the proportion of the training observations that are in mth region (R_m) from the kth class. Recall that G is small when the nodes are more homogeneous: e.g., when the proportion of training observations in mth region are closely split between two classes 45%-55%, then $G = (.45)(.55)(2) = 0.495$; in contrast, when the training observations in R_m are more dominated by a single class (and thus R_m is more homogeneous), for instance, 90%-10%, then $G = (.10)(.90)(2) = 0.18$.

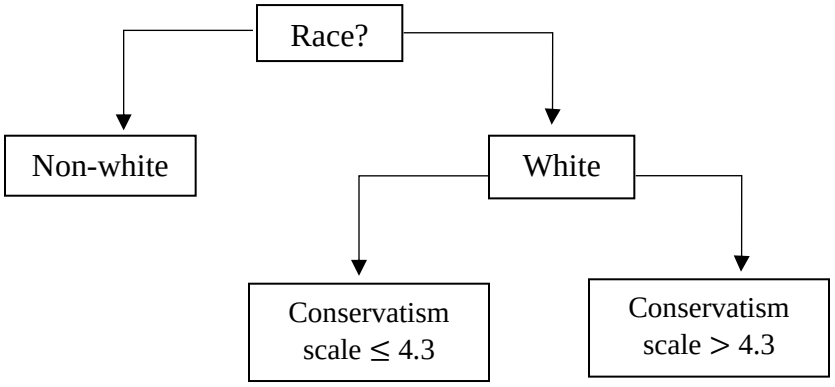
$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

Entropy (D) is defined below. Like with the Gini index, when the training observations in R_m are more dominated by a single class, then D is small. For example, if the class balance in R_m is (e.g., 90%-10%), then $D = -[(.1)\log(.1) + (.9)\log(.9)] = .14$. In contrast, when the class balance is 55%-45%, then $D = -[(.55)\log(.55) + (.45)\log(.45)] = .30$.

$$D = -\sum_{k=1}^{K} \hat{p}_{mk}\log(\hat{p}_{mk})$$

How does this work in practice? For example, let us assume that in the training set, being white v. not being white was the strongest individual predictor of the Trump vote (i.e., it would maximize node purity). If this were the case, then the first split would be based on race: all white respondents would be assigned to the right branch, and all non-white respondents would be assigned to the left branch (see Figure 5 below). Next, let us assume that among white respondents, being above the mean on the 1-7 point political conservatism scale is the strongest predictor of the Trump vote; if so, then the second split would be based on whether the white respondents' conservatism score is $\leq$ 4.3 (left branch) or $>$ 4.3 (right branch).

Figure 5: Example of a Decision Tree (DV: Trump vote)



In this simple example, the observations (or respondents) in the training set are assigned to one of three non-overlapping regions in the predictor space: $R_1 = \{Vote \mid not\ white\}$, $R_2 = \{Vote \mid white, conservatism \leq 4.3\}$, and $R_3 = \{Vote \mid white, conservatism > 4.3\}$. To predict the outcome class of an observation in the validation or test set, we simply look at which region the observation would be assigned to based on its predictor values (e.g., is $x_1 = white$?), and then choose the most common class of that region. For instance, if the test observation would belong to R_3, and 70% of the training observations in R_3 voted for Trump, then the predicted class of that test observation would be $Vote = 1$. In general, of course, there are usually more than three regions (or terminal nodes); the splitting ends once a stopping point is reached: e.g., in order to satisfy the minimum terminal node size.

Model Class #2: Penalized Logistic Regression

Recall that in a general case with p predictors, the logistic function is the following:

$$p\big(Y = 1\big|X_1, \dots, X_p\big) = \frac{e^{\beta_0+\beta_1X_1+\cdots+\beta_pX_p}}{1 + e^{\beta_0+\beta_1X_1+\cdots+\beta_pX_p}}$$

What is appealing about this function is that it guarantees an output between 0 and 1, due to the S-shape of the sigmoid curve (Long 1997, pp. 39-43). By performing some algebraic manipulation, we can rewrite the logistic function as an odds ratio. In the context of this project, the odds ratio refers to the probability of a respondent having voted for Trump over the probability that the respondent did not vote for Trump (conditional on a set of predictor values such as race, gender, religion, etc.).

$$\frac{p\big(Y = 1\big|X_1, \dots, X_p\big)}{1 - p\big(Y = 1\big|X_1, \dots, X_p\big)} = e^{\beta_0+\beta_1X_1+\cdots+\beta_pX_p}$$

After logging both sides, we finally get the logistic regression model. The left-hand side is called the log-odds (i.e., since it is literally the log of the odds ratio). The right-hand side indicates that the log-odds, also called the logit, is linear with respect to X. We can fit the model by using maximum likelihood estimation (e.g., Greene 2012, pp. 509-523).

$$\log\left[\frac{p\big(Y = 1\big|X_1, \dots, X_p\big)}{1 - p\big(Y = 1\big|X_1, \dots, X_p\big)}\right] = \beta_0 + \beta_1X_1 + \cdots + \beta_pX_p$$

Why use a penalized from of the logistic regression? In reality, a subset of the predictors are probably not useful for predicting the outcome; that is, the true parameters for these predictors should be equal to zero or near zero. When the logistic regression model does not employ a shrinkage method, the coefficients that should actually be zero may remain at non-zero values—which increases the likelihood of the model overfitting to the training data (i.e., modeling the random noise, which is not desirable since it increases variance).

I use the implementation of the penalized logistic regression which is implemented in the *glmnet* package in R (Hastie and Qian 2014). The objective function for the penalized logistic regression uses the negative binomial log-likelihood and is displayed below. The goal is to find the vector of coefficients that solves the objective function. According to the authors of the package, the algorithm "use cyclical coordinate descent, which successively optimizes the objective function over each parameter with others fixed"–and repeats the cycle until convergence.

$$\min_{(\beta_0,\beta)\in\mathbb{R}^{p+1}} -\left[\frac{1}{N}\sum_{i=1}^{N} y_i \cdot (\beta_0 + x_i^T\beta) - \log(1+e^{(\beta_0+x_i^T\beta)})\right] + \lambda\left[(1-\alpha)||\beta||_2^2/2 + \alpha||\beta||_1\right]$$

The important part about this optimization process is that it is subject to the constraint on the right: $\lambda[(1-\alpha)\|\beta\|_2^2/2 + \alpha\,\|\beta\|_1]$. In this function, α refers to the elastic-net penalty: by default, $\alpha = 1$ (under the lasso); and under the ridge, $\alpha = 0$. The lasso option is often preferable because by allowing some coefficients to actually be zeroed out, it allows us to automate the variable selection process. Since I set $\alpha = 1$, the constraint simplifies to $\lambda[\|\beta\|_1]$, where $\|\beta\|_1 = \sum_{j=1}^{p}|\beta_j|$ and p is the number of predictors. The size of the penalty is controlled by λ: the bigger the λ, the smaller the coefficients. To find the optimal λ, we can use k-fold cross-validation.

$$\min_{(\beta_0,\beta)\in\mathbb{R}^{p+1}} -\left[\frac{1}{N}\sum_{i=1}^{N} y_i \cdot (\beta_0 + x_i^T\beta) - \log(1+e^{(\beta_0+x_i^T\beta)})\right] + \lambda\left[||\beta||_1\right]$$

Model Class #3: Support Vector Machines

Support Vector Machines (SVMs) are a class of methods that seek to assign observations to the correct outcome by using hyperplanes as decision boundaries. Hyperplanes are a $p-1$ dimensional flat subspace in a p-dimensional space: e.g., if $p = 2$ (there are 2 predictors), then the hyperplane is simply a line; if $p = 3$, then the hyperplane is a 2d plane. For instance, let us assume a simple case in which we are predicting the binary outcome $Vote$ using only two predictors; in this case, the separating hyperplane is a line. If the respondents in the training set who voted for Trump are labeled as $y_i = 1$, and those who did not vote for Trump are labeled as $y_i = -1$, then the separating hyperplane can be formally described as follows:

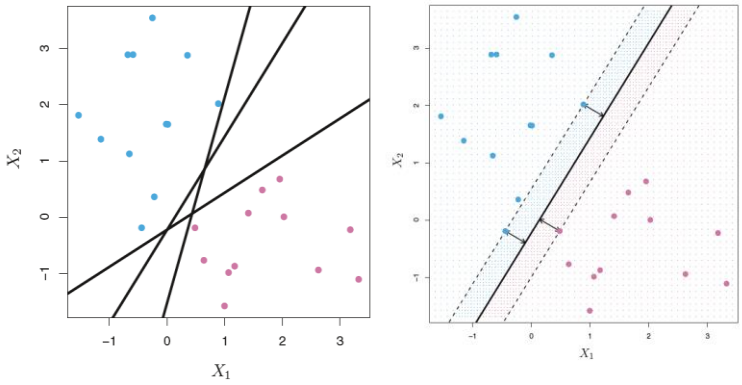
$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2}) > 0$$

In this case, $\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} > 0$, $y_i = 1$ and $\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} < 0$, then $y_i = -1$. That is, we can classify the observations based on whether they are above or below the separating hyperplane.

Next, I will briefly review how SVMs have been designed to address two key practical challenges. The first challenge is that there are often many hyperplanes that can correctly classify

the observations, since the hyperplane can be slightly adjusted in any direction and probably still produce the same classifications. If we return to the equation above, it is easy to see that there are many ways the coefficients β_0, β_1, and β_2 can be slightly adjusted (e.g., a 0.5% increase or decrease) and still perfectly classify the observations. This point is also illustrated in the left-hand graph in Figure 6 (which is from James et al. 2013, pp. 340-342). The solution is to maximize the margins, or the distance between the training data points and the separating hyperplane: this is also known as the maximal margin classifier or MMC (Figure 6, right-hand side graph).

Figure 6: Multiple Hyperplanes (left) and the MMC (right)



Unfortunately, the MMC approach faces its own problems. Sometimes, there is simply no perfectly separating hyperplane; and even if there is, it is highly sensitive to individual observations, which can lead to overfitting (and high variance). The solution is a more flexible form of the MMC that allows some observations to fall on the wrong side of the margin and/or hyperplane: this is also known as the support vector classifier (SVC). The SVC is more robust in that it is less sensitive to individual observations, such as outliers (i.e., since some violations of the margin are allowed); this property allows the SVC to do a better job of classifying *most* of the observations.

Like the MMC, the SVC seeks to maximize the margin, but it is subject to a number of important constraints (James et al. 2013, pp. 346-347). Here, I will specifically focus on the cost parameter (C), since that is what is optimized in my project using cross-validation. Below, ϵ_i indicates how severely the ith observation violates the margin and separating hyperplane. When $\epsilon_i = 0$, the ith observation is located on the correct side of the margin (i.e., no error); when $\epsilon_i > 0$, it is on the wrong side of the margin; and if $\epsilon_i > 1$, it has actually violated the separating hyperplane. That is, C essentially represents the budget for the number and severity of the classification errors across the n observations. When C is small, the SVC will fit more tightly to the training observations, at the cost of higher variance; and when C is large, the opposite can occur. To optimize this bias-variance trade-off, I use 10-fold CV.

$$\sum_{i=1}^{n} \epsilon_i \leq C$$

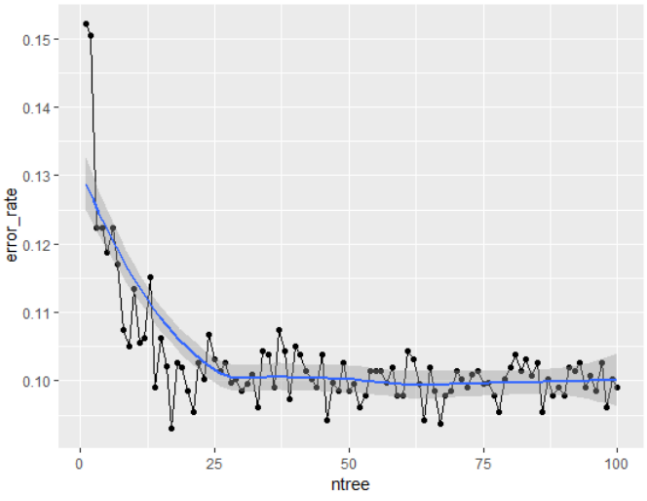
The SVM is an extension of the SVC, which allows us to model nonlinear decision boundaries using kernels. By using kernels, the SVM allows us to expand the feature space (e.g., include polynomial functions of the predictors) in a computationally efficient manner. For more details on this procedure, we can refer to James et al. 2013, pp. 350-353; and Hastie et al. 2017, pp. 423-426).

Part 3: Model Selection using 10-Fold CV

I used 10-fold CV to estimate the optimal values of the main tuning parameter(s) for 7 different types of models.[7] In this section, I will describe how I did this for each of the model type (e.g., boosted trees, logistic lasso, etc.).

(1) 10-Fold CV for Bagged Trees

This entails creating B bootstrap samples by sampling with replacement from the original training set (n_1). A classification tree is fit using each sample, and then the trees are combined. This method is superior to using a single decision tree, because a single decision tree is affected by high variance; in contrast, by averaging the predictions across many B trees, the variance is reduced. The main tuning parameter for bagged trees is B, the number of bootstrapped trees. I fit the bagged trees using the *randomForest* package in R. According to the results of the 10-fold CV, there are diminishing returns to the CV accuracy after B reaches around 40 or so. Just to be safe, I selected $B = 50$ for my candidate model. The estimated CV error rate ~10%.



[7] The choice of k is somewhat arbitrary, but the ISLR textbook noted that $k = 5$ or 10 offers a good trade-off between bias and variance (James et al. 2013, pp. 183-184).

(2) 10-Fold CV for Random Forests

This is the same as bagging, but now the model is only allowed to consider only a random subset m of the p predictors at each split (such that $m \ll p$, e.g., $n \approx \sqrt{p}$). The logic here is that by intentionally restricting the number of predictors that can be considered at each split, the B trees will become less correlated. In many cases, decorrelating the trees in this way can lead to reduced test error. I selected the following two main tuning parameters to optimize during the CV: (a) number of predictors to try at each split (*mtry* in R), (b) minimum node size, which refers to the minimum number of training observations in a terminal node. RF can also be fit using the *randomForest* package in R. According to the CV results, the best *mtry* is 10 and the best minimum node size is 14. Again, the estimated CV error rate ~10%.

```
> estimateTimeTuneRanger(status.task)
Approximated time for tuning: 2M 19S>
> res = tuneRanger(status.task, measure = list(acc), num.trees = 50,
+                  tune.parameters = c("mtry", "min.node.size"), iters = 70)
Computing y column(s) for design. Not provided.
[mbo] 0: mtry=12; min.node.size=4 : y = 0.904 : 0.2 secs : initdesign
[mbo] 0: mtry=16; min.node.size=4 : y = 0.907 : 0.2 secs : initdesign
[mbo] 0: mtry=14; min.node.size=16 : y = 0.903 : 0.2 secs : initdesign
[mbo] 0: mtry=10; min.node.size=42 : y = 0.903 : 0.2 secs : initdesign
[mbo] 0: mtry=6; min.node.size=100 : y = 0.9 : 0.2 secs : initdesign
[mbo] 0: mtry=3; min.node.size=31 : y = 0.906 : 0.2 secs : initdesign
[mbo] 0: mtry=2; min.node.size=2 : y = 0.903 : 0.2 secs : initdesign
[mbo] 0: mtry=13; min.node.size=3 : y = 0.905 : 0.2 secs : initdesign
```

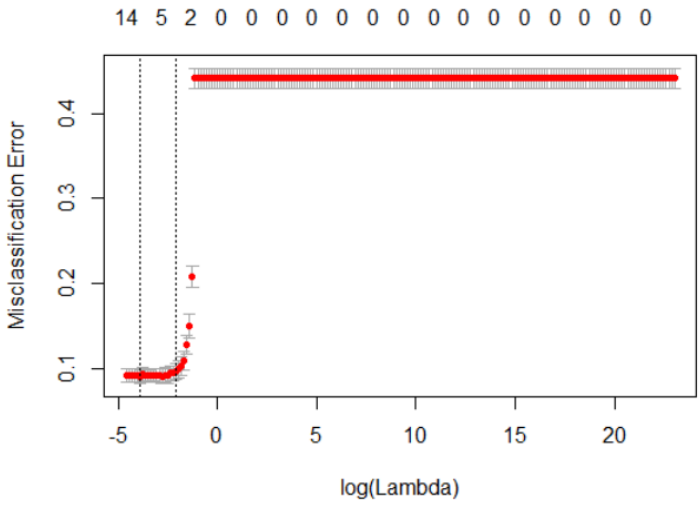
(3) 10-Fold CV for Boosted Trees

With boosted trees, the trees are grown sequentially: each tree is fit to the residuals from the previous model. The logic of this approach is that it allows each successive tree to address the weaknesses of the previous tree. I selected two main tuning parameters to optimizing using CV: (a) the learning rate, which controls how slowly the trees learn (*eta* in R), (b) and max tree depth, which controls the number of splits in each tree. In general, more flexible models (e.g., slower learning rate) tend to perform well. The boosted trees were fit using the *xgboost* package in R. According to the CV results, the best learning rate is .09, and the best max tree depth is 4. The estimated CV error rate is ~9%, which is a little better than the CV error rates for RF and bagged trees.

```
+   summarize(test_error= mea
+   arrange(test_error)
# A tibble: 40 x 2
   param_combo test_error
   <chr>            <dbl>
 1 0.09, 4         0.0907
 2 0.03, 4         0.0913
 3 0.07, 4         0.0919
 4 0.03, 6         0.0925
 5 0.05, 5         0.0931
 6 0.09, 5         0.0937
 7 0.03, 5         0.0937
 8 0.07, 5         0.0948
 9 0.12, 4         0.0949
10 0.14, 5         0.0954
```

(4) 10-Fold CV for Logistic Lasso Regression

As discussed previously, the tuning parameter of the logistic lasso is λ, which controls the size of the penalty. The larger the penalty parameter, the greater the shrinkage in the coefficients. According to the results of the CV, the best lambda value is 0.020022 (or equivalently, the ideal $ln(\lambda) \approx -3.91$). This suggests that a relatively small penalty is favored, which means that many of the covariates are useful predictors of the outcome. The estimated CV error rate is < 10%.



(5) 10-Fold CV for Support Vector Classifier

The main tuning parameter for the SVC is cost, which controls the budget for the number and severity of the classification errors. I fit the SVC (and SVMs) using the *e1071* package in R. The CV suggests that an ideal cost parameter is about 1.83. The estimated CV error rate is about 9.3%.

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
     cost
 1.826364

- best performance: 0.09307813
```

(6) 10-Fold CV for SVM (Polynomial)

I optimized the two main tuning parameters for the polynomial SVM: cost and degree. According to the CV results, the best parameter values are the following: best cost is 1.12, and best degree is 3. The estimated CV error rate about 10.1%.

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost degree
 1.12      3

- best performance: 0.1014293
```

(7) 10-Fold CV for SVM (Radial)

I optimized the two main tuning parameters for the radial SVM: cost and gamma. According to the CV results, the best parameter values are the following: best cost is 1.12, and best gamma is 0.01. The estimated CV error rate is around 9.3%.

```
# A tibble: 50 x 2
   param_combo              test_error
   <chr>                         <dbl>
 1 cost: 1.12, gamma: 0.01      0.0925
 2 cost: 2.23, gamma: 0.01      0.0925
 3 cost: 4.45, gamma: 0.01      0.0925
 4 cost: 3.34, gamma: 0.01      0.0925
```

Part 4: Final Results

Research Question #1: Which models are the best at predicting the 2016 Trump vote?

After completing the 10-fold CV procedure, I fit the final optimized version of each model type (e.g., a fully tuned logistic lasso) using the full training set. Next, I compared the performance of the seven candidate models by using the held-out test set (30% of the data). It appears as though the SVM (radial) performed the best, achieving a test prediction accuracy of about 92.1%. However, most of the candidate models also achieved a test prediction accuracy of at least 91%, which means that once the models have been optimized using CV, the differences in performance across model types is relatively small.

Tree-based Methods

method	test_error
boosting	0.085
bagging	0.089
random_forest	0.089

Logistic Lasso

method	test_error
glm_lasso	0.081

SVC/SVM Methods

method	test_error
radial_svm	0.079
linear_svm	0.082
polynom_svm	0.100

Research Question #2: Which covariates are the strongest predictors of the 2016 Trump vote?

To address my second research question, I use two measures. First, I extract the coefficient estimates from the logistic lasso candidate model. The continuous predictors were standardized, so the lasso coefficients are comparable across the continuous predictors; moreover, the categorical predictors are also comparable with other categorical predictors, because they have all been converted into binary variables (for the list of predictors, see page 4). Second, I used the *importance* function of the *randomForest* package in R, to compute the **mean decrease in accuracy** (MDA) for each predictor for the candidate RF model: the MDA provides a measure of how much the accuracy decreases on average (i.e., across the trees) when a variable is omitted. Thus, in a sense, the absolute value is also an indicator of variable importance.

The results below indicate that the four strongest predictors are consistent across the logistic lasso and RF: the Democratic Party feeling thermometer, opposition to Trump's proposed wall with Mexico, the Republican feeling thermometer, and support for Obamacare.[8]

Logistic Lasso

	name	coefficient
1	dem_ft	-1.211854790
2	oppose_wall	-0.838536415
3	repub_ft	0.831910729
4	support_obamacare	-0.458450381
5	conserv_scale	0.355856002
6	(Intercept)	-0.245824884
7	raceWhite	0.194579659
8	support_gcontrol	-0.184851919
9	pro_choice	-0.109095555
10	age	0.073570854
11	religious	0.004082952
12	male	*NA*
13	raceAsian	*NA*
14	raceBlack	*NA*
15	raceHispanic	*NA*
16	raceNative American	*NA*
17	raceOther	*NA*
18	educNon-BA	*NA*

Random Forest

	names_vec	MeanDecrease_Acc
1	dem_ft	27.9481665
2	repub_ft	19.8671166
3	oppose_wall	18.7075687
4	conserv_scale	6.8170849
5	support_obamacare	6.7242555
6	age	4.3566431
7	race	3.8451837
8	religious	3.2157962
9	support_gcontrol	3.0371243
10	pro_choice	2.5096067
11	athiest	2.4604495
12	male	2.1050311
13	born_again	1.7421471
14	income_band	0.6214337
15	educ	0.3971507
16	evangelical	-0.1857594
17	divorced	-0.7542547
18	pol_interest	-1.2909944

Moreover, by looking at the sign on the lasso coefficients, we can also see the expected direction of the effects; all of these top four predictors except for the Republican feeling thermometer are negatively associated with the likelihood of voting for Trump. While the general importance of partisanship and policy preferences is not surprising, what is surprising is that contrary to popular belief, demographic and social predictors (e.g., race, religion) do not seem to be

[8] There are more coefficients for the logistic lasso, since the model actually converts non-binary categorical predictors such as race into binary predictors. I only showed the first 18 coefficients of the lasso due to space constraints.

strongest predictors of the Trump vote. After the election was over, many media pundits and analysts argued that Trump was propelled to victory by religious conservatives and poor, white working-class voters. While my results do not suggest that factors such as race, gender, and religion are totally inconsequential, they do suggest that these demographic and social predictors may operate more indirectly. For example, they shape the policy preferences and partisanship of voters, which are themselves understood as the more proximate (or direct) causes of the Trump vote in 2016.

Part 5: Implications, Future Work

In sum, at least at the individual level, there is very little evidence that voters are capricious and inconsistent agents who behave in random ways. As evidenced in this project, given a relatively small set of key predictors (e.g., even 10-15), we can actually predict the likelihood of someone voting for Trump with about 90-92% accuracy. Thus, the implication is that pollsters and analysts were "wrong" about the expected outcome of the 2016 election for other reasons. For example, I suspect that social desirability bias was a big contributor: the largely negative media coverage of the Trump campaign in 2015 and 2016 had reduced the willingness of some Trump supporters to actually indicate their true voting intentions when surveyed, especially in swing states. There may have also been issues with the sampling design, such that voters in some rural hard-to-reach areas (who usually lean Republican) were less likely to be surveyed.

Can we use machine learning to predict the outcome of the 2020 elections? Yes, I think we can, and there are at least two ways to do this:

- Method #1: The unit of analysis is the state. We are thus trying to predict whether individual states will vote for Trump. In this case, we would need to add weights for the average state-level characteristics (e.g., average ideology, party support) and the likelihood of people turning out, and so on.[9]
- Method #2: The unit of analysis is the individual. We would need a large nationally representative sample of registered U.S. voters, with a sufficient number of respondents in each state. Next, we would need to add weights for the likelihood that they will turn out and also for the number of electoral college votes assigned to their state (i.e., we need to account for the fact that a vote in some states simply "matters more," since larger states get more electoral votes).

[9] Professor Wenxin Jiang actually suggested something like this (i.e., for Method #1) when we met during his office hours on May 27, 2019.

References

Greene, William H. 2012. "Econometric Analysis." New York: Prentice Hall. 7th Edition.

Hastie, Trevor and Junyang Qian. "Glmnet Vignette" <https://web.stanford.edu/~hastie/glmnet/glmnet_alpha.html>

Hastie, Trevor, Robert Tibshirani, Jerome Friedman. 2017. "The Elements of Statistical Learning Data Mining, Inference, and Prediction." New York: Springer. 2nd Edition.

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2013. "An Introduction to Statistical Learning: with Applications in R." New York: Springer. 7th Edition.

Long, Scott J. "Regression Models for Categorical and Limited Dependent Variables." Thousand Oaks, CA: Sage Publications.

New York Times. 2017. "Presidential Election Results: Donald J. Trump Wins" <https://www.nytimes.com/elections/2016/results/president>

Rhodan, Maya, and David Johnson. 2016. "Here Are 7 Electoral College Predictions for Tuesday." *Time Magazine*. <http://time.com/4561625/electoral-college-predictions/>