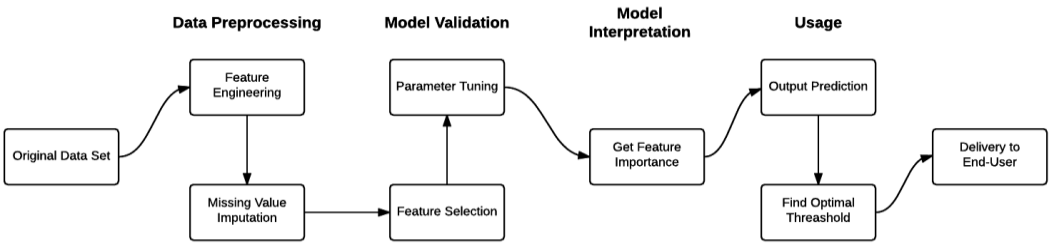
Summary

In this project, I predicted the probability of supporting Democratic candidate versus Republican candidate for all given individuals based on survey and census data. The model I used is an ensemble of gradient boosted trees trained on different imputation data sets. Feature selection and parameter tuning are done by measuring model performance on 10-fold cross validation. At the end, an optimal threshold is found to help end-users to distinguish the voting preference.

Work Flow



I. Data Preprocessing

Before building the predictive model, we need to preprocess the data to deal with the missing values. Also, new features are generated based on existing features.

1. Missing Values

The modeling dataset has 5,511 records, of which only 3,819 are complete. Considering the amount of incomplete records, a lot of information (30% of total) will be lost if we simply drop them. Therefore, we should find ways to fill the missing part. The process of filling the missing values is called imputation. Three imputation methods are considered in this project:

1) Mean/Median/Mode imputation

   Fill the missing values with column mean/median/mode. It is simple and fast, but it corrupted the original distribution of the data, so I only used this method in columns with very low percentage of miss values.

2) Filling with -1

   -1 is filled as the indicator of missing values. The advantage of this method is it doesn't change the original order of the data. We use it in tree based method. However, it is not a good practice in linear models or SVM, because the magnitude of the imputed values matters in these models.

3) K Nearest Neighbor Impute

   The missing values is replaced by the mean of the corresponding column of its k nearest neighbors.

To make the prediction more stable, the model will be built based on data sets filled different imputation methods and then we take the average as final output.

2. Feature Engineering

New features are extracted from existing features or generated based on the interaction of existing features.

1) Year/Month/Day/Day_of_Week

Seasonal features are extracted from the feature "earliest_reg_date", trying to catch seasonal pattern in registration time.

2) Difference/Ratio of Democratic vs. Liberal

Define the difference and ratio between democratic index and liberal index, working as indicator of personal political preference.

3) Low Income

A binary variable indicating if the person's income is lower than the census median household income.

4) Total number of vote from 2004 – 2012

Sum up the vote in Democratic/Republican non-presidential primary across all years.

5) Difference/ratio: dem_performance_pct vs. random_var

Although in data dictionary, random_var is explained as a column of random numbers, it is found that the column "dem_performance_pct" is highly correlated with the "random_var" column (correlation coefficient 0.97). As a guess, we mark it as the Republican performance percentage. Based on this guess, we define the interaction between these two features.

6) Convert Categorical Features to Numeric

Two approaches are used to convert levels in categorical features (sex, state_region, and ethnicity) to numeric values:

a) One-Hot Encoding

   Creating dummy variables for each level in the categorical feature.

b) Label Encoding

   Mapping the levels to integers. This approach works well in tree based method. However, it's not a good practice in linear models and SVM because magnitude of the mapped values matters.

II. Feature Selection and Parameter Tuning

At the feature selection and parameter tuning stage, we look at the AUC score as the measure of model performance. We want to find the feature set and parameter set that maximizes the AUC score.

To avoid getting a misleading high score resulted from overfitting, we evaluate all model performance based on cross-validation.

1. Modeling Approach

In this project, the only model used is gradient boosted tree with xgboost realization. The reason I solely based my prediction on this one model is:

1) The gradient boosted tree method achieve significantly better accuracy than other type of classifier in general.

2) Considerably fast to train compared to random forest and SVM.

The model is built on data sets imputed by different methods. The final result is an average of output from both imputed data sets, which makes the prediction more robust.

Overfitting is controlled mainly by cross-validation. The process of generating new features also makes sure there is no information leakage and therefore avoided overfitting.

2. Feature Selection

In the first step, we will test different feature subsets on a gradient boosted tree classifier with fixed parameters and compare the AUC scores back and forth.

1) Original Features

First, we train a gradient boosted tree classifier on all original features (non-generated features). Here are the performances:

Neg_1 imputed: AUC 0.919689

KNN imputed: AUC 0.919945

2) Original + Engineered Features

Then we train gradient boosted tree classifier features we have (original + engineered):

Neg_1 imputed: AUC 0.919747

KNN imputed: AUC 0.920042

We can see with the engineered features, model performance on both train sets are improved.

3) All features excludes features w/ near zero variance

When calculating the correlations between features, it is found that some columns has almost zero variance. Features w/ zero or near zero variance contain very little information. Let's see how AUC score change if we take these features out.

Neg_1 imputed: AUC 0.920116

KNN imputed: AUC 0.919756

We can see, by taking out the near zero columns, we got the best performance so far with a neg_1 imputed data sets.

4) Filter features based on feature importance

In this step, we trained gradient boosted tree models on the data sets we obtained in 3) and check the feature importance.

The xgboost package in R gives feature importance based on information gain. By listing the least contributed features, we have list of features to remove:

"smarstat_s","bookmusc_1","vote_g2008",

"state_region_Region 2","state_region_Region 3",

"state_region_Region 4","state_region_Region 5",

"state_region_Region 6"

The list contains all dummy variables we created for the feature “state_region”.

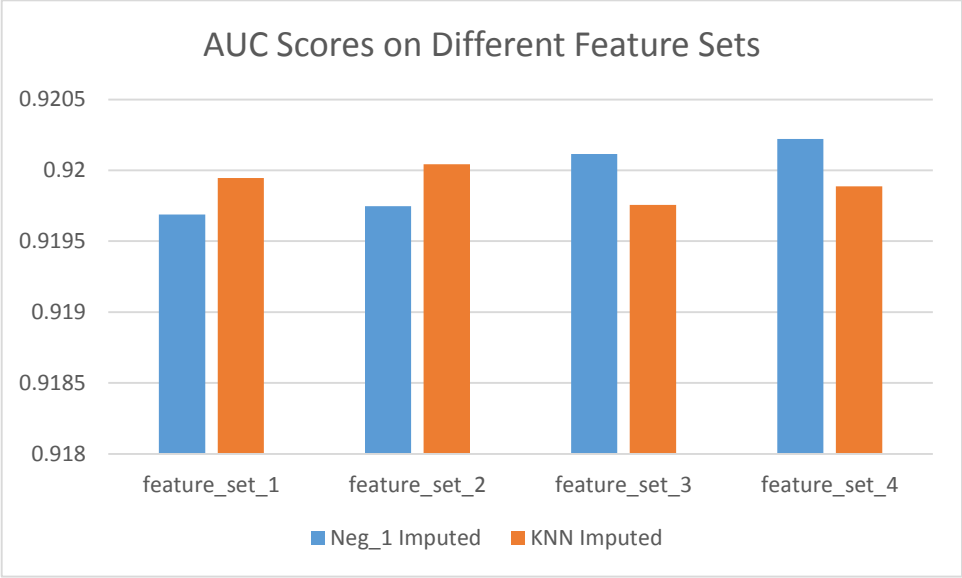
After removing the above features, we train out model on a 10-fold cross-validation again. This time, the scores we got are:

Neg_1 imputed: AUC 0.920222

KNN imputed: AUC 0. 919887

Both scores got improved. Also, the model trained on neg_1 imputed data sets are the best single model we trained so far.

Below is the bar graph comparing AUC scores on different feature sets:



As we can see, the KNN imputed datasets performs better with more features, while the Neg_1 imputed datasets performs better with the filtered data sets. Therefore, in the next step, we’ll train and fine-tune models on two different sets, namely:

a. KNN Imputed data sets w/ all features (feature set 2)

b. Neg_1 Imputed data sets w/ filtered features (feature set 4)

3. Parameter Tuning

For this part, we will use grid search as our search technique. The grid search conducts exhaustive search based on the performance feedback returned by cross-validation.

The optimal parameter sets are:

Model 1: gboost on KNN Imputed data sets w/ all features (feature set 2):

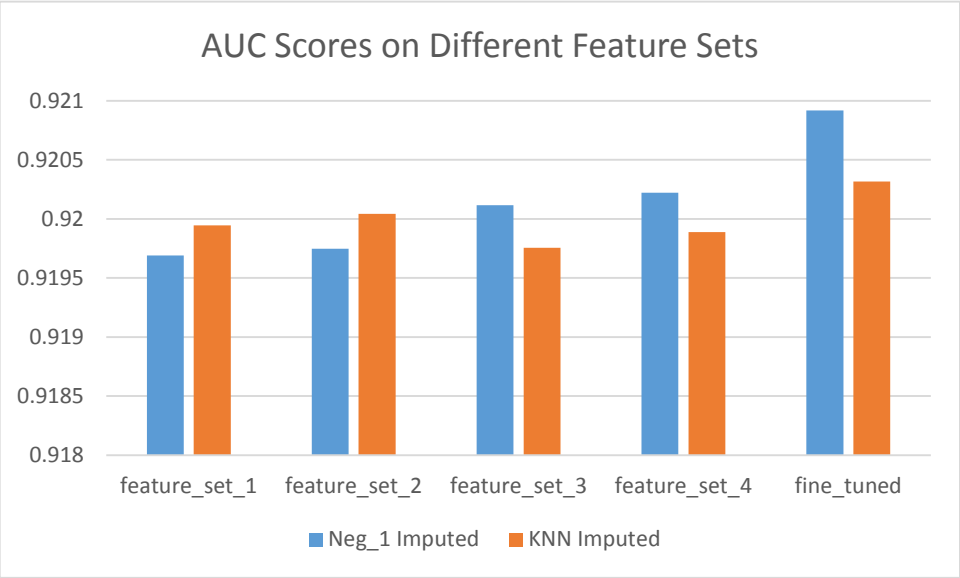
eta = 0.005, max_depth = 3 , gamma = 0.01
colsample_bytree = 0.5, min_child_weight = 3, nrounds = 1500

The AUC score achieved is 0.9209165.

Model 2: xgboost on Neg_1 Imputed data sets w/ filtered features (feature set 4):

eta = 0.005, max_depth = 4 , gamma = 0.01
colsample_bytree = 0.5, min_child_weight = 5, nrounds = 1500

The AUC score achieved is 0.9203164

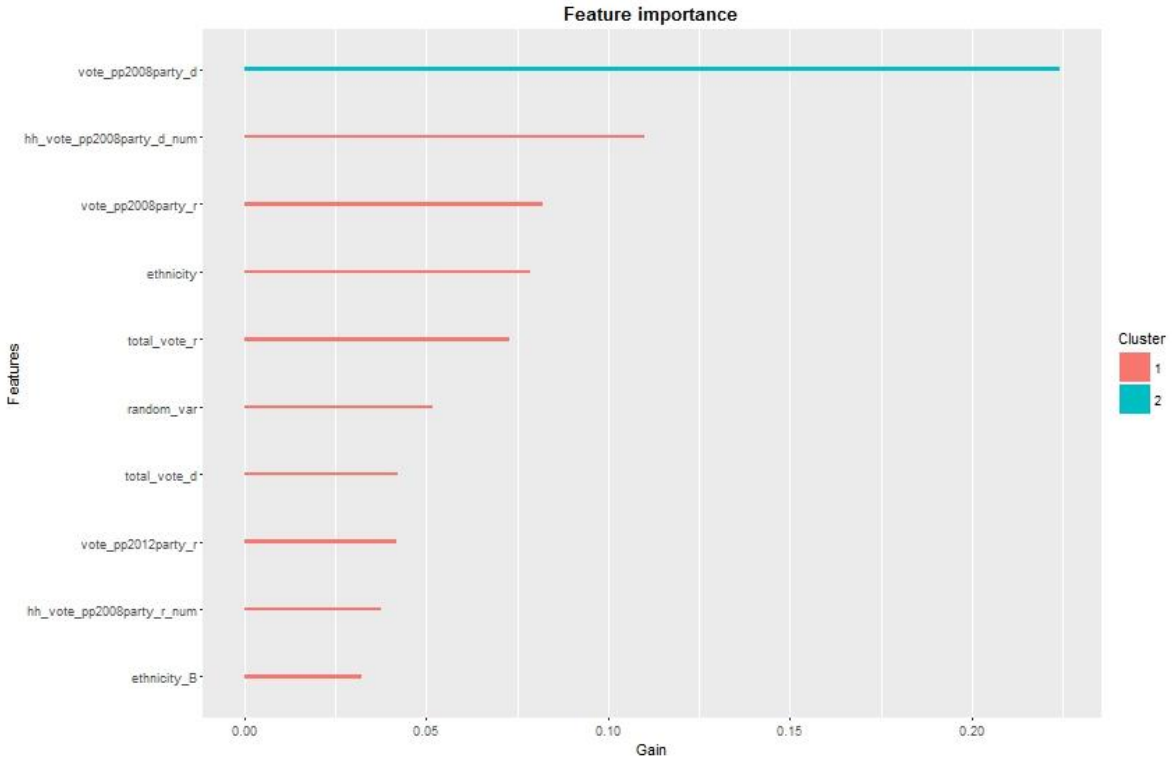


We can see, after grid search parameter tuning, both model get improved.

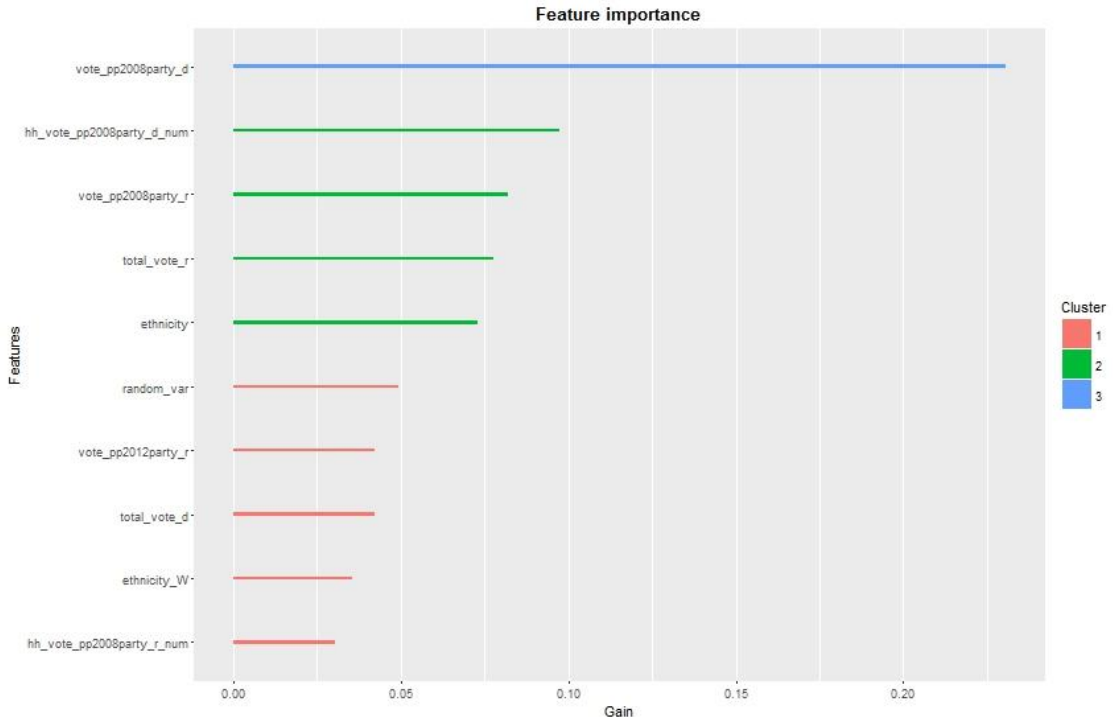
III. Model Interpretation

Now let's take a look at the feature importance:

Feature Importance by Model 1:



Feature Importance by Model 2:



To make it more visual, below is the list of the top 10 from both models:

Order	Feature Importance	
	Model 1	Model 2
1	vote_pp2008party_d	vote_pp2008party_d
2	hh_vote_p2008party_d_num	hh_vote_p2008party_d_num
3	vote_pp2008party_r	vote_pp2008party_r
4	ethnicity	total_vote_r
5	total_vote_r	ethnicity
6	random_var	random_var
7	total_vote_d	vote_pp2012party_r
8	vote_pp2012party_r	total_vote_d
9	hh_vote_p2008party_r_num	ethnicity_W
10	ethnicity_B	hh_vote_p2008party_r_num

We can see order and features in the top 10 important from two models are basics same.

Conclusion:

1) Result of ***2008 presidential primary*** is the most important information. All four features about 2008 presidential primary are among the top 10.

2) **Ethnicity** is considered to be the second most predictive feature. Among all ethnicities, Black and White are the most predictive two.

3) **Random_var** is not real random. It contains important information. Actually, the other feature “dem_performance_pct” (which is highly correlated with random_var) is also among the top 15 important features.

IV. Usage

With the well-tuned model, we can generate predictions and we should also know how to use the predictions.

The prediction is an unweighted average of the output predictions from both models. The ensemble lowers the variance and gives more robust prediction. The result can be found in the csv file “pred_prob.csv”

1. Prediction Output

The final output is the probability to support Democratic candidate. The higher the score is, the higher the chance the person to support Democratic.

2. Find Optimal Threshold

In real life use, we need to find a threshold value for the end-user to determine whether we should classify an individual as support or not given the predicted probabilities.

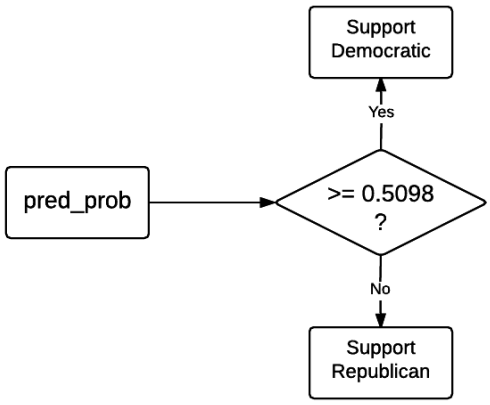
Usually, we have an estimate of gain and loss for correctly and wrongly classify the support. With this information, we can find the threshold that maximize the gain or minimize the loss.

However, without the gain and loss information, we can still use other method to determine the threshold. Here, we will find the threshold that maximizes the sum of specificity and sensitivity.

Specificity = When a person supports Republican candidate, how often do we predict he/she supports Republican candidate?

Sensitivity = When a person supports Democratic candidate, how often do we predict he/she supports Democratic candidate?

After calculation, the thresh hold that maximize the sum of specificity and sensitivity is 0.5098. Therefore, the take away from this analysis for end users is: when the predicted probability is greater than 0.5098, predict as support the democratic candidate, otherwise predict not support, which can be shown as:



V. Potential Improvement

Here are some questions I have about the data:

1. Result for the primary presidential vote seems the best predictor, especially the vote result in 2008 (i.e. “vote_pp2008party_r”, “vote_pp2008party_d”). It is strange that variable

"vote_pp2012party_d" and "vote_pp2004party_r" both have all 0 in the entire column. They should have been very important predictors like votes in 2008. If possible, I would definitely have my people check on that.

2. Check if there is an variable for the republican average performance percentage and check how the random_var is put into the data set.

Given more time, here are the thing I would try:

1. I would try other imputation method, for example random forest can be applied for imputation.
2. An encoding method called "Leave-One-Out" Encoding, which uses the mean response of levels in categorical feature to present the level. However, it is very prone to overfitting and needs carefully stratified cross-validation. It is discussed in this post:

   https://www.kaggle.com/c/caterpillar-tube-pricing/forums/t/15748/strategies-to-encode-categorical-variables-with-many-categories/88916
3. In this challenge, the only model used is gradient boosted tree models realized by xgboost package. It is chosen based on my previous experience that it is superior to other classifier for example random forest, logistic regression and SVM in general. However, given more time, I would try other types of classifier as well.
4. I believe there is more room for improvement using grid search. It is computationally expensive, but given more time, it is definitely worth trying.