Predicting winner of US general election 2016 at County Level Using Selected Demographic-socio-economic Variables

Aritra Ghosal, Harleen Kaur, Evan Mouchard, Elizabeth Forney

Introduction

The results of US general election 2012 was predicted with considerable high accuracy, especially by Nate Silver's approach. The success of the polling and analysis methods can be attibuted primarily to the fact that the voters who were sampled for survey were honest with their opinion and that whatever change occurred over time was tractable with high accuracy. As we have seen that the voting preferences may change with a change in employment or a shift in federal income tax schemes or an impactful campaign ad. Other sources of variation are sampling error and "house effect". The former implies overrepresentation of voters of one candidate in the sample, and the corrections to such errors by pollsters may be biased due to "house effect". Both these error can be estimated to great degree and adjusted for but the accuracy of polling survey remains the most important and sensitive part of this process.

However, for the General election of 2016, polls and forecasts were generally wrong about the election due to systematic polling challenges. Clinton's projected voteshare was overestimated in most cases, particularly within swing states. Experts have speculated that Trump supporters were reluctant to participate and/or answer honestly to polling questions. This is probably due to the controversies surrounding Donald Trump. On the other hand Hillary was a seasoned politician with well defined agendas and ideas for presidency.

In this project, we strove to analyze the results of the 2016 federal election at the county level. Our goal was to construct a model that accurately predicts which candidate will receive the most votes in a particular county. A variety of models were trained and tested before arriving at our final selection. The methods that were compared include Logistic Regression, K-Nearest Neighbors, Classification Trees, Random Forest, LDA, and QDA. Principal Component Analysis was implemented to produce a set of principal components that were used as covariates for the models. We trained the models, and the principal components, on a 70% training partition of the data. Models were compared based on their misclassification of the test data. Ultimately, we chose the Random Forest method. We then conducted further analysis to optimize its decision threshold to meet certain criteria, allowing us to arrive at a final model.

Materials and methods

Some initial analysis showed that every county in US were won either by Donald Trump or Hillary Clinton. Since we perform our analysis retrospectively with the final winner of the election known, we construct the response as a binary variable with two outcomes 'Don' and 'Hill' respectively meaning that Donald Trump or Hillary Clinton won that particular county.

Let $i = 1, 2, 3, ..., n$ be the indicator for counties, and n be total number of counties in US for our analysis. Define,

$$y_i = \begin{cases} \text{'Don'} & , \quad \text{Donald Trump won in i-th county} \\ \text{'Hill'} & , \quad \text{Hillary Clinton won in i-th county} \end{cases}$$

This particular construction of our response variable was done to facilitate the use of classification models under Supervised learning methods we discussed in our PSTAT 131 course. These models are:

1) Logistic Regression
2) K- nearest neighbors
3) Linear Discriminant Analysis
4) Quadratic Discriminant Analysis
5) Regression Tree
6) Random Forest

In each model above we predict the probability of win for Donald Trump, $p_i = P(y_i = 1)$, $\forall i$. Then we consider a threshold $\alpha \in (0, 1)$. If the predicted $p_i > \alpha$, then we predict that that Trump won, i.e. $\hat{y}_i =' Don'$, otherwise $\hat{y}_i =' Hill'$. Here $\hat{y}_i^M$ is the predicted response for i^{th} county from the model M above.

Then we compute the misclassification for i^{th} county:

$$z_i^M = \begin{cases} 0 & , \quad y_i = \hat{y}_i^M \\ 1 & , \quad y_i \neq \hat{y}_i^M \end{cases}$$

Hence $z_i^M(\alpha) = 1$ means a misclassification for the i^{th} county and model M. Then compute $TME^M(\alpha) = \frac{1}{n}\sum_i^n z_i^M(\alpha)$, the total misclassification error for each model and choose the model which gives the minimum of $TME^M(\alpha)$. Here the parameter $\alpha = 0.5$ is chosen for our analysis.

However which model turned out the winner optimize for α over a grid of $(0, 1)$.

Datasets

There were two raw data sources used in this project: election and census. These datasets required cleaning and processing which we executed in Stage 1 of this project. The raw election dataset stored federal, state, and county level vote tallies which were then divided into separate datasets. The raw census dataset consisted of higher resolution, demographic information. This data consisted of racial, employment, income, transportation, and location variables. We aggregated the raw census data to the county level. This allowed us to merge the county level election data set with the aggregated census data. The result was a single dataset containing vote information for the winning candidate and runner-up in each county. Table 1 shows a few rows and columns of the dataset.

Table 1: Example rows and columns of the dataset.

county	fips	candidate	state	votes	total	pct
autauga	1001	Donald Trump	alabama	18172	24759	0.734
autauga	1001	Hillary Clinton	alabama	5936	24759	0.2398
baldwin	1003	Donald Trump	alabama	72883	94261	0.7732
baldwin	1003	Hillary Clinton	alabama	18458	94261	0.1958
barbour	1005	Donald Trump	alabama	5454	10436	0.5226
barbour	1005	Hillary Clinton	alabama	4871	10436	0.4667

Methods

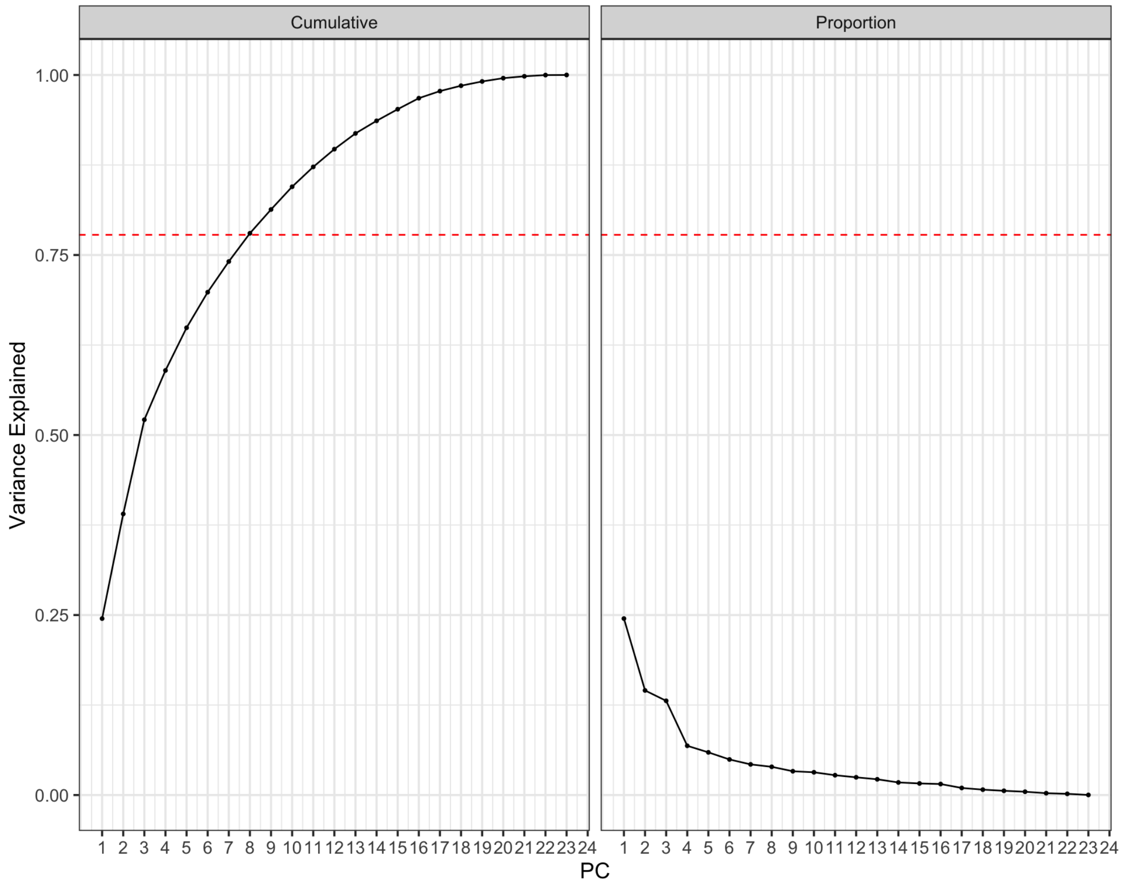
Principal Component Analysis

We began by conducting Principal Component Analysis on the training data. to use as inputs into our supervised methods in determined the winner of each county. This required scaling and centering of the merged data. We then computed loadings for the principal components and plotted them. This allowed us to visualize the variables that were the most influential in determining the value of the principal component. To figure out the correct number of PC's to use in the supervised models we constructed a scree and cumulative variance plot.

$$\text{cumulative variance explained}(q) = \frac{\sum_{j=1}^{q} \lambda_j}{\sum_{j=1}^{p} \lambda_j}$$

We found that the first 8 PC's captured 77.8% of the total variation which was sufficient cumulative variance explained threshold to implement onto our regression models.

Table 1: Scree plot of cumulative and proportion of variance explained by the principal components.



Logistic Regression

The first model we fit was a logistic regression model to predict the the winning candidate in each county during the 2016 presidential election. We trained the model on our principal component training partition (70% of the data), regressing the winning candidate on the first 8 principal components. Moreover, The optimal threshold was computed using Youden's statistic. The accuracy was then assessed on the remaining 30% of the data and the total misclassification rate was calculated in order to compare this model to other models.

k-Nearest Neighbors

The next model we wanted to compare was a *k*-nearest neighbors model. The class labels we used were the winning candidates (either Donald Trump or Hillary Clinton) and leave one out cross validation was performed in order to select the best *k* that would minimize the error for our model. Like with the logistic regression model, we trained the *k*-nearest neighbors model on a 70% partition of the first 8 principal components data and measured the predictive accuracy based on the remaining 30% by computing the total misclassification error rate.

Classification Tree

A classification tree was trained, at first, by overfitting a very large tree to the training PC data, then pruning it according to an optimal tuning parameter (k = 21.9) calculated via 25-fold cross validation. We ran the training and test sets through this tree to generate predictions and compute misclassifications. Youden-optimized threshold was also tested on the model.

Random Forest

A random forest – which is an ensemble decision-tree-based method – was constructed according to the "Adaptive Boosting" scheme. We used 100 trees with 3 interaction terms and 5-fold cross validation. This method produced the best candidates for our final model. The Youden-optimized threshold was tested, as well as two more thresholds that minimized total misclassification and balanced false positives with false negatives to arrive at our preferred model.

Linear and Quadratic Analysis

For categorical response, and and covariates being Principal components are in euclidean space of p-dimension, we follow similar steps as earlier models and train both the LDA and QDA models on 70% data and test the predictive performance on the rest. The total misclassification error rates turning out to be 0.0837 for the LDA and 0.08913043 for the QDA. Hence LDA performs slightly better. But their performances are not at par with the random forest model discussed above.

Results

The main metric we used to compare our models was the total misclassification rate. Table 2 shows the total misclassification rate for each model.

Table 2: Total misclassification rate for each model we fit.

Regression.Model	Misclassification.Rate
Random Forest	0.0599
K Nearest Neighbors	0.07283
Linear Discriminant Analysis	0.0837
Logistic Regression	0.08804
Quadratic Discriminant Analysis	0.08913
Random Tree	0.09348

Strongest candidate: Random Forest

The Random Forest model appears to be the strongest candidate because it has the lowest total misclassification rate of any model at the 50% threshold (7.02%) as well as the Youden-optimized threshold (7.39%). Examining these thresholds more closely, we can see that there is room for improvement. Looking at the misclassifiation table for the 50% threshold:

Table 3: Misclassifications for the random forest model at 50% threshold.

class	Don	Hill
Don	766	11
Hill	57	86

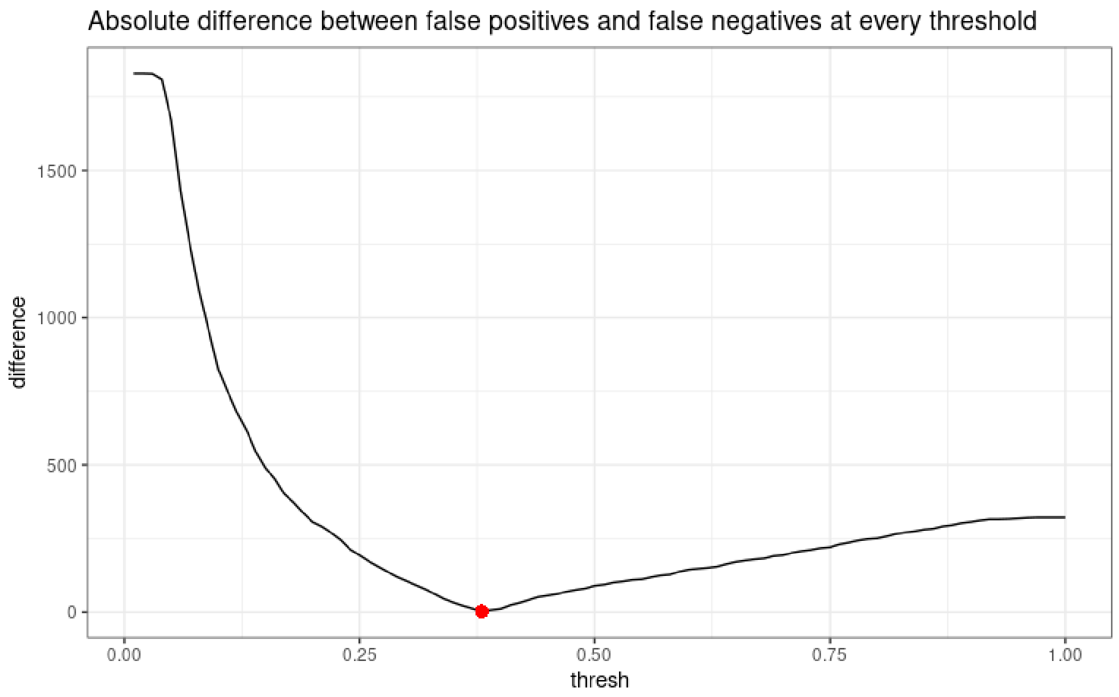
The total misclassifications are low, but 57 out of the 68 false predictions predict in favor of Donald Trump. This bias can be an effective way of minimizing total misclassification error, but it also ensures that our model will under-represent the counties won by Hillary Clinton. The Youden threshold, which maximizes the difference between True Positive Rate an False Positive Rate, might help to balance this out. Looking at the misclassification table for the Youden threshold of 26%:

Table 4: Misclassifications for the random forest model at 26% threshold.

class	Don	Hill
Don	695	82
Hill	19	124

The same problem exists with this set of predictions, but this time in favor of Hillary Clinton. We found that all of our models had this problem. The 50% threshold tended to over-predict Donald Trump victories, so we tried using Youden's threshold to shift our predictions in the direction of Clinton, but this resulted in over-prediction of Clinton victories (and with higher total misclassification error). Our revised target is for our prediction errors to be split evenly between both candidates while maintaining low total misclassifications. With this target in mind, we looked for a new optimal threshold.

Figure 2: Line plot of absolute difference between false positives and false negatives against thresholds.



For each threshold in the range (0,1), we calculated the resultant absolute difference between false 'Clinton' predictions and false 'Trump' predictions. The threshold of 38% was found to yield the smallest difference. It also yielded a very low total misclassification rate of the training data (6.2%) Applying this threshold to random forest predictions of the test data yielded these misclassifications:

Table 5: Misclassifications for the random forest model at 38% threshold.

class	Don	Hill
Don	753	24
Hill	33	110

These are great results for multiple reasons. The total misclassification error is 6.2% (same as the training data), which is the lowest of any model tested. Additionally, the false 'Clinton' predictions and false 'Trump' predictions are much more balanced, with a ratio of 24:33. Thus, the sums of the predictions are more accurate than any other model. The projected 134 Hillary counties and 786 Trump counties are quite close to the actual totals of the test data which are 143 and 777. This model meets our optimal criteria: yield a low total misclassification rate, and have misclassifications that are split evenly between the candidates. Therefore, it is one of our choices for the final model.

To be thorough in our analysis, we also calculated the model that truly minimizes total misclassification error. We calculated the total training misclassifications at each threshold between 0 and 1, and found that the optimal threshold is 46%. The misclassification table of the test data looks like this:

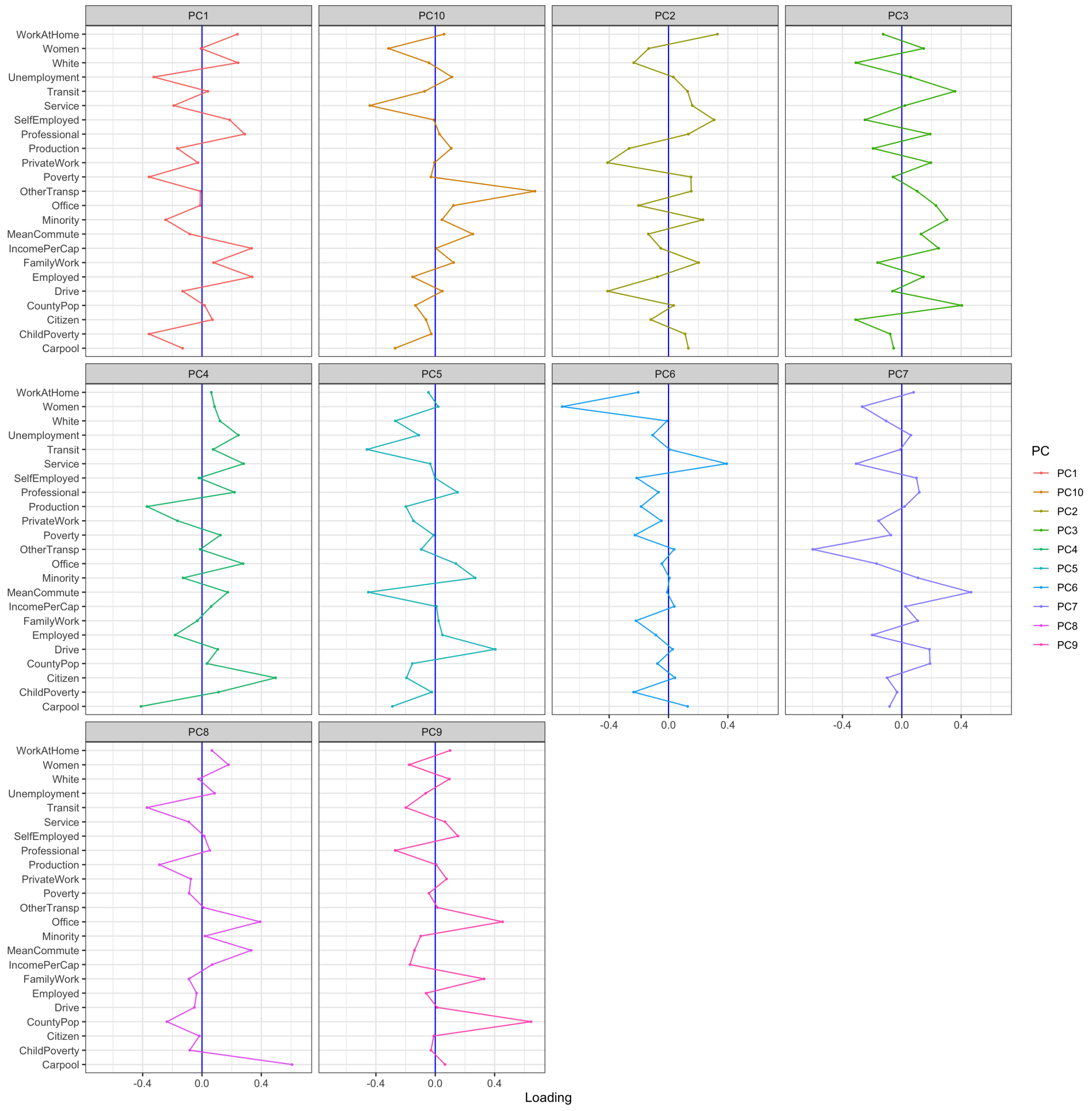
Table 6: Misclassifications for the random forest model at 46% threshold.

class	Don	Hill
Don	759	18
Hill	40	103

The predictions slightly over-represent Donald trump, with a mistake ratio of 40:18, but the total misclassification rate of 5.99% is the absolute lowest of any model we found. Therefore, we have included it as one of our choices for the best model.

Furthermore, we analyzed the loading plots for thee first ten principal components to identify the most important covariates.

Figure 3: Loadings plot for the first 10 principal components.



The variables with the largest magnitudes suggest that they are most influential. Thus, variables pertaining to transportation, such as `Carpool`, `OtherTransp`, `MeanCommute`, and `Transit`, seem to be the most important. `Women` and `CountyPop` also appear to be important covariates due to their large loading values in principal components 6 and 9, respectively.

Discussion

By using principal component analysis in our regression models, we were able to build more predictive models. However, this reduced the interpretability of our results. Subsequently, we were unable to identify the demographic variables that were predictive of the election outcome through our models. However, we investigated the loadings for each variable for each principal component to identify important covariates. In our analysis, we fit several different models to the data and computed the total misclassification error rate in order to compare them all. Optimal probability thresholds were also computed using Youden's statistic in order to balance the trade-off between true positive rates and false positive rates. Additionally, we further focused on the random forest model which we ultimately chose to be a candidate model. This model had the lowest total misclassification rate and fit a reasonable criteria of balancing misclassifications between the two candidates.

Codes

```r
# load data
load('merged_data2.RData')

county_win_data <- merged_data2 %>%
  group_by(fips) %>%
  slice_max(votes) %>%
  ungroup()

#center and scale
x_mx <- county_win_data %>%
  select(-c('county':'pct')) %>%
  scale(center = T, scale = T) #centers and scales data

# set rng
set.seed(20121)

# partition data
county_part <- resample_partition(as_tibble(x_mx), c(test = .3, train = .7))
test <- county_part$test
train <- county_part$train
testdata <- as_tibble(x_mx) %>% slice(test$idx) #test dataframe
traindata <- as_tibble(x_mx) %>% slice(train$idx) #training dataframe

# compute loadings for PC1 and PC2
x_svd <- svd(traindata)
d_sq <- x_svd$d^2/(nrow(traindata) - 1)
loadings <- x_svd$v

# Loadings plot
loadings[, 1:10] %>%
  as.data.frame() %>%
  rename(PC1 = V1, PC2 = V2, PC3 = V3, PC4 = V4, PC5 = V5,
         PC6 = V6, PC7 = V7, PC8 = V8, PC9 = V9, PC10 = V10) %>%
  mutate(variable = colnames(traindata)) %>%
  gather(key = 'PC', value = 'Loading', 1:10) %>%
  arrange(variable) %>%
  ggplot(aes(x = Loading, y = variable)) +
  geom_point(aes(color = PC), size = 0.5) +
  facet_wrap(~ PC) +
  theme_bw(base_size = 12) +
  geom_vline(xintercept = 0, color = 'blue') +
  geom_path(aes(group = PC, color = PC)) +
  labs(y = '')

## scree and cumulative variance plots
tibble(PC = 1:min(dim(traindata)),
       Proportion = d_sq/sum(d_sq),
       Cumulative = cumsum(Proportion)) %>%
  gather(key = 'measure', value = 'Variance Explained', 2:3) %>%
  ggplot(aes(x = PC, y = `Variance Explained`)) +
  geom_point(size = 0.75) +
  geom_path() +
  facet_wrap(~ measure) +
  theme_bw(base_size = 6) +
  scale_x_continuous(breaks = 1:31, labels = as.character(1:31)) +
  geom_hline(yintercept = 0.778,
             color = 'red',
             linetype = 2)
sum(d_sq[1:8])/sum(d_sq)

# get training PC data for first 8 PCs
v_q <- loadings[, 1:8]
Z_train <- as.matrix(traindata) %*% v_q #training PC's
Z_test <- as.matrix(testdata) %*% v_q #test PC's, *calculated on training loadings*

#construct testing and training dataframes
colnames(v_q) <- colnames(Z_train) <- colnames(Z_test) <- paste('PC', 1:8, sep = '')
pc_train <- tibble(winner = factor(county_win_data$candidate[train$idx])) %>%
  bind_cols(as_data_frame(Z_train))
pc_train$winner <- fct_collapse(pc_train$winner,
                               Don = 'Donald Trump',
                               Hill = 'Hillary Clinton')

pc_test <- tibble(winner = factor(county_win_data$candidate[test$idx])) %>%
  bind_cols(as_data_frame(Z_test))
pc_test$winner <- fct_collapse(pc_test$winner,
                               Don = 'Donald Trump',
                               Hill = 'Hillary Clinton')

# Logistic regression model on training data using first 8 PCs
glm_model <- glm(winner ~ ., family = 'binomial', data = pc_train)

# compute estimated probabilities
p_hat_glm <- predict(glm_model, pc_test, type = 'response')

# bayes classifier
y_hat_glm <- factor(p_hat_glm > 0.5, labels = c('Don', 'Hill'))

# errors
error_glm <- table(y = pc_test$winner, y_hat_glm)
error_glm

# total misclassification rate
tot_misclass_glm <- 1 - sum(diag(error_glm))/nrow(pc_test)
tot_misclass_glm

# roc curve for glm
prediction_glm <- prediction(predictions = p_hat_glm, labels = pc_test$winner)

# compute error rates as a function of probability threshhold
```

```r
perf_glm <- performance(prediction.obj = prediction_glm, 'tpr', 'fpr')

# extract error rates as a tibble
rates_glm <- tibble(fpr = slot(perf_glm, 'x.values'),
                  tpr = slot(perf_glm, 'y.values'),
                  thresh = slot(perf_glm, 'alpha.values')) %>%
  unnest(everything()) %>%
  mutate(method = 'glm',
         youden = tpr - fpr)

# plot roc curve and optimal threshold
rates_glm %>%
  ggplot(aes(x = fpr, y = tpr)) +
  geom_path(aes(color = thresh), size = 1) +
  scale_color_binned(type = 'viridis') +
  guides(color = guide_bins()) +
  theme_bw() +
  geom_point(data = slice_max(rates_glm, youden),
             shape = 16, color = 'red')

# store optimal threshold
optimal_thresh <- slice_max(rates_glm, youden)

# recalibrate qda with different probability threshold
preds_glm_adj <- factor(p_hat_glm > optimal_thresh$thresh,
                        labels = c(0, 1))

# cross-tabulate estimated and true classes with adjusted threshold
errors_glm_adj <- table(class = pc_test$winner, pred = preds_glm_adj)
errors_glm_adj

misclass_glm_adj = 1-sum(diag(errors_glm_adj))/nrow(pc_test)
misclass_glm_adj

y <- (pc_train %>% pull(winner))

# leave one out cross validation
cv_out <- tibble(k = seq_range(5:50, n = 20, pretty = T)) %>%
  mutate(loocv_preds = map(k, ~ knn.cv(pc_train[-1], y, .x)),
         class = map(k, ~ y)) %>%
  mutate(misclass = map2(loocv_preds, class,
                         ~ as.numeric(.x) - as.numeric(.y))) %>%
  mutate(error = map(misclass, ~ mean(abs(.x))))

# error rates for each k
cv_errors <- cv_out %>%
  select(k, error) %>%
  unnest(everything())

# plot errors against k
cv_errors %>%
  ggplot() +
  geom_line(aes(x = k, y = error), color = 'cornflowerblue') +
  theme_bw()

# select k
best_k <- cv_errors$k[which.min(cv_errors$error)]

# re-train with best k
pred_knn <- knn(train = pc_train[-1], test = pc_test[-1], cl = y, k = best_k)

# misclassifications
error_knn <- table(pc_test$winner, pred_knn)
error_knn

# total misclassification rate
tot_misclass_knn <- 1 - sum(diag(error_knn))/nrow(pc_test)
tot_misclass_knn

set.seed(20121)

#initial tree
nmin <- 5
tree_opts <- tree.control(nobs = nrow(pc_train),
                          minsize = nmin,
                          mindev = 0)
t_0 <- tree(winner ~ ., data = pc_train,
                control = tree_opts, split = 'deviance')

nfolds <- 25
cv_out <- cv.tree(t_0, K = nfolds)
cv_df <- tibble(alpha = cv_out$k,
                impurity = cv_out$dev,
                size = cv_out$size)
best_alpha <- slice_min(cv_df, impurity) %>%
  slice_min(size)
best_alpha
# prune initial tree
t_opt <- prune.tree(t_0, k = best_alpha$alpha)

# plot
draw.tree(t_opt, cex = 0.8, size = 2, digits = 1)

# class probabilities on training partition
probs_tree <- predict(t_opt, pc_train)

# predicted training results
preds_tree <- factor(probs_tree[, 2] > 0.5, labels = c('Don', 'Hill'))

# training misclassification error rates
training_error_tree <- table(class = pc_train$winner, pred = preds_tree)
training_misclass_tree = 1-sum(diag(training_error_tree))/nrow(pc_train)

# roc curve for classification tree
prediction_tree <- prediction(predictions = probs_tree[,2], labels = pc_train$winner)
```

```r
# compute error rates as a function of probability threshhold
perf_tree <- performance(prediction.obj = prediction_tree, 'tpr', 'fpr')

# extract error rates as a tibble
rates_tree <- tibble(fpr = slot(perf_tree, 'x.values'),
                    tpr = slot(perf_tree, 'y.values'),
                    thresh = slot(perf_tree, 'alpha.values')) %>%
  unnest(everything()) %>%
  mutate(method = 'lda',
         youden = tpr - fpr)

# plot roc curve and optimal threshold
rates_tree %>%
  ggplot(aes(x = fpr, y = tpr)) +
  geom_path(aes(color = thresh), size = 1) +
  scale_color_binned(type = 'viridis') +
  guides(color = guide_bins()) +
  theme_bw() +
  geom_point(data = slice_max(rates_tree, youden),
             shape = 16, color = 'red')

# store optimal threshold
optimal_thresh <- slice_max(rates_tree, youden)

# recalibrate qda with different probability threshold
preds_tree_adj <- factor(probs_tree[,2] > optimal_thresh$thresh,
                        labels = c('Don', 'Hill'))

# cross-tabulate estimated and true classes with adjusted threshold
training_error_tree_adj <- table(class = pc_train$winner, pred = preds_tree_adj)
training_misclass_tree_adj = 1-sum(diag(training_error_tree_adj))/nrow(pc_train)

# class probabilities on test partition
probs_tree <- predict(t_opt, pc_test)

# predict test results
preds_tree <- factor(probs_tree[,2] >0.5,
                     label = c('Don', 'Hill'))

# compute misclassifications
error_tree <- table(class = pc_test$winner, pred = preds_tree)
misclass_tree = 1-sum(diag(error_tree))/nrow(pc_test)

#predict test results with youden threshold
preds_tree <- factor(probs_tree[,2] > optimal_thresh$thresh, label = c("Don", "Hill"))

#compute adjusted misclassifications
error_tree_adj <-table(class = pc_test$winner, pred = preds_tree)
misclass_tree_adj = 1-sum(diag(error_tree_adj))/nrow(pc_test)

error_tree
misclass_tree
```

```
error_tree_adj
misclass_tree_adj

pc_train_rf <- pc_train %>%
  mutate(winner = ifelse(winner == "Hill",1,0))
pc_test_rf <- pc_test %>%
      mutate(winner = ifelse(winner == "Hill",1,0))

fit_gbm <- gbm(winner ~ ., data = pc_train_rf, distribution ='adaboost', interaction.depth =3, n.trees =

# select boosting iterations
best_m <- gbm.perf(fit_gbm, method ='cv')

# compute predictions on training set
preds_rf <- predict(fit_gbm, pc_train_rf, n.trees = best_m)
probs_rf <-1/(1+ exp(-preds_rf))
y_hat <- factor(probs_rf >0.5, labels = c('Don','Hill'))
y <- factor(pc_train_rf$winner, labels = c('Don','Hill'))

# compute training misclassification errors
rf_errors <- table(class = y, pred = y_hat)
training_misclass_rf = 1-sum(diag(rf_errors))/nrow(pc_train)
training_misclass_rf

# roc curve for random forest
prediction_rf <- prediction(predictions = probs_rf, labels = pc_train_rf$winner)

# compute error rates as a function of probability threshhold
perf_rf <- performance(prediction.obj = prediction_rf, 'tpr', 'fpr')

# extract error rates as a tibble
rates_rf <- tibble(fpr = slot(perf_rf, 'x.values'),
                  tpr = slot(perf_rf, 'y.values'),
                  thresh = slot(perf_rf, 'alpha.values')) %>%
  unnest(everything()) %>%
  mutate(method = 'lda',
         youden = tpr - fpr)

# plot roc curve and optimal threshold
rates_rf %>%
  ggplot(aes(x = fpr, y = tpr)) +
  geom_path(aes(color = thresh), size = 1) +
  scale_color_binned(type = 'viridis') +
  guides(color = guide_bins()) +
  theme_bw() +
  geom_point(data = slice_max(rates_rf, youden),
             shape = 16, color = 'red')

# store optimal threshold
optimal_thresh <- slice_max(rates_rf, youden)

# recalibrate qda with different probability threshold
preds_rf_adj <- factor(probs_rf > optimal_thresh$thresh,
```

```r
                        labels = c(0, 1))

# cross-tabulate estimated and true classes with adjusted threshold
errors_rf_adj <- table(class = pc_train_rf$winner, pred = preds_rf_adj)

training_misclass_rf_adj = 1-sum(diag(errors_rf_adj))/nrow(pc_train)
training_misclass_rf_adj

# compute test predictions
preds_rf <- predict(fit_gbm, pc_test_rf, n.trees = best_m)
probs_rf <-1/(1+ exp(-preds_rf))
y_hat <- factor(probs_rf >0.5, labels = c('Don','Hill'))
y <- factor(pc_test_rf$winner, labels = c('Don','Hill'))

# compute test misclassification errors
rf_errors <- table(class = y, pred = y_hat)

misclass_rf = 1-sum(diag(rf_errors))/nrow(pc_test)

# compute test predictions with optimal threshold
preds_rf <- predict(fit_gbm, pc_test_rf, n.trees = best_m)
probs_rf <-1/(1+ exp(-preds_rf))
y_hat <- factor(probs_rf >optimal_thresh$thresh, labels = c('Don','Hill'))
y <- factor(pc_test_rf$winner, labels = c('Don','Hill'))

#compute adjusted test misclassification errors
rf_adj_errors <- table(class = y, pred = y_hat)
misclass_rf_adj = 1-sum(diag(rf_errors))/nrow(pc_test)


rf_errors
misclass_rf

rf_adj_errors
misclass_rf_adj

# filter out losing candidates by county => 1 row per county, 'candidate' is winner of the county
county_win_data <- merged_data2 %>%
  group_by(fips) %>%
  slice_max(votes) %>%
  ungroup()

#center and scale
x_mx <- scale(na.omit(county_win_data[,-c(1:7)]))
#install.packages('hmeasure')
library(hmeasure)
#goal: fp = fn; fp-fn = 0
preds_rf <- predict(fit_gbm, pc_train_rf, n.trees = best_m)
probs_rf <-1/(1+ exp(-preds_rf))


difs = vector('list', 100)
for(i in seq_along(x)){
  e = misclassCounts(probs_rf > i*0.01, pc_train_rf$winner)
  dif <- abs(e$conf.matrix[1,2] - e$conf.matrix[2,1])
  difs[[i]] <- dif
}

opt_search <- data.frame('thresh' = 0.01*1:100, 'difference'= unlist(difs))
min_index = which.min(unlist(difs))
min = min(unlist(difs))
optimum <- data.frame(min_index, min)

opt_search %>%
  ggplot(aes(x = thresh, y = difference)) +
  geom_path() +
  theme_bw() +
  geom_point(data = optimum, aes(x = 0.01*min_index, min),
             shape = 16, size = 3, color = 'red') +
  ggtitle("Absolute difference between false positives and false negatives at every threshold")

balanced_thresh <- 0.01*min_index
balanced_thresh

preds_rf <- predict(fit_gbm, pc_train_rf, n.trees = best_m)
probs_rf <-1/(1+ exp(-preds_rf))

y_hat <- factor(probs_rf > balanced_thresh, labels = c('Don','Hill'))
y <- factor(pc_train_rf$winner, labels = c('Don','Hill'))

#training misclassification
training_rf_balanced_errors <- table(class = y, pred = y_hat)
training_misclass_rf_balanced = 1-sum(diag(rf_balanced_errors))/nrow(pc_test)
training_rf_balanced_errors
training_misclass_rf_balanced

#compute adjusted test misclassification errors
rf_balanced_errors <- table(class = y, pred = y_hat)
misclass_rf_balanced = 1-sum(diag(rf_balanced_errors))/nrow(pc_test)
rf_balanced_errors
misclass_rf_balanced

#calculate minimum training misclassification error across all thresholds
preds_rf <- predict(fit_gbm, pc_train_rf, n.trees = best_m)
probs_rf <-1/(1+ exp(-preds_rf))
ME = vector('list', 100)
for(i in seq_along(x)){
  me = misclassCounts(probs_rf > i * 0.01, pc_train_rf$winner)$metrics$ER #calculates total misclassifi
  ME[[i]] = me
}
min_thresh = 0.01*which.min(unlist(ME)) # threshold corresponding to smallest error

training_rf_min_errors <- table(class = factor(pc_train_rf$winner, labels = c('Don', 'Hill')),
                                pred = factor(probs_rf>min_thresh, labels = c('Don','Hill')))
training_rf_min_misclass = 1 - sum(diag(training_rf_min_errors))/nrow(pc_train)
```

```r
#calculate test misclassifications based on training threshold for smallest error
preds_rf <- predict(fit_gbm, pc_test_rf, n.trees = best_m)
probs_rf <-1/(1+ exp(-preds_rf))
rf_min_errors <- table( class = factor(pc_test_rf$winner, labels = c('Don', 'Hill')),
                                pred = factor(probs_rf>min_thresh, labels = c('Don','Hill')))
rf_min_misclass <- 1 - sum(diag(rf_min_errors))/nrow(pc_train)

rf_min_errors
rf_min_misclass

# fit lda model
fit_lda <- MASS::lda(winner ~ ., method = 'mle', data = pc_train_rf)

# compute estimated classes
preds_lda <- predict(fit_lda, pc_train_rf)

# cross-tabulate estimated and true classes
errors_lda <- table(class = pc_train_rf$winner, pred = preds_lda$class)
misclass_lda = 1-sum(diag(errors_lda))/nrow(pc_train)
misclass_lda

# roc curve for lda
prediction_lda <- prediction(predictions = preds_lda$posterior[, 2], labels = pc_train_rf$winner)

# compute error rates as a function of probability threshhold
perf_lda <- performance(prediction.obj = prediction_lda, 'tpr', 'fpr')

# extract error rates as a tibble
rates_lda <- tibble(fpr = slot(perf_lda, 'x.values'),
                  tpr = slot(perf_lda, 'y.values'),
                  thresh = slot(perf_lda, 'alpha.values')) %>%
  unnest(everything()) %>%
  mutate(method = 'lda',
         youden = tpr - fpr)

# plot roc curve and optimal threshold
rates_lda %>%
  ggplot(aes(x = fpr, y = tpr)) +
  geom_path(aes(color = thresh), size = 1) +
  scale_color_binned(type = 'viridis') +
  guides(color = guide_bins()) +
  theme_bw() +
  geom_point(data = slice_max(rates_lda, youden),
             shape = 16, color = 'red')

# store optimal threshold
optimal_thresh <- slice_max(rates_lda, youden)

# recalibrate qda with different probability threshold
preds_lda_adj <- factor(preds_lda$posterior[, 2] > optimal_thresh$thresh,
                        labels = c(0, 1))

# cross-tabulate estimated and true classes with adjusted threshold
```

```r
errors_lda_adj <- table(class = pc_train_rf$winner, pred = preds_lda_adj)

misclass_lda_adj = 1-sum(diag(errors_lda_adj))/nrow(pc_train)
misclass_lda_adj

# fit qda model
fit_qda <- MASS::qda(winner ~ ., method = 'mle', data = pc_train_rf)

# compute estimated classes
preds_qda <- predict(fit_qda, pc_train_rf)

# cross-tabulate estimated and true classes
errors_qda <- table(class = pc_train_rf$winner, pred = preds_qda$class)
misclass_qda = 1-sum(diag(errors_qda))/nrow(pc_train)
misclass_qda

# roc curve for qda
prediction_qda <- prediction(predictions = preds_qda$posterior[, 2], labels = pc_train_rf$winner)

# compute error rates as a function of probability threshhold
perf_qda <- performance(prediction.obj = prediction_qda, 'tpr', 'fpr')

# extract error rates as a tibble
rates_qda <- tibble(fpr = slot(perf_qda, 'x.values'),
                    tpr = slot(perf_qda, 'y.values'),
                    thresh = slot(perf_qda, 'alpha.values')) %>%
  unnest(everything()) %>%
  mutate(method = 'qda',
         youden = tpr - fpr)

# plot roc curve and optimal threshold
rates_qda %>%
  ggplot(aes(x = fpr, y = tpr)) +
  geom_path(aes(color = thresh), size = 1) +
  scale_color_binned(type = 'viridis') +
  guides(color = guide_bins()) +
  theme_bw() +
  geom_point(data = slice_max(rates_qda, youden),
             shape = 16, color = 'red') +
  xlab('FPR (False "Hillary" Rate)') +
  ylab('TPR (True "Hillary" Rate)')

# store optimal threshold
optimal_thresh <- slice_max(rates_qda, youden)

# recalibrate qda with different probability threshold
preds_qda_adj <- factor(preds_qda$posterior[, 2] > optimal_thresh$thresh,
                        labels = c(1, 0))

# cross-tabulate estimated and true classes with adjusted threshold
errors_qda_adj <- table(class = pc_train_rf$winner, pred = preds_qda_adj)
misclass_qda_adj = 1-sum(diag(errors_qda_adj))/nrow(pc_train)
misclass_qda_adj
```