

2016 Election Prediction

Micheal Alexander, Sriharsha Addepalli

3/23/2018

Background

Voting behavior prediction is a difficult task because the data is based off of human behavior. Human behavior is complex and there is no simple way to predict what decisions humans are going to make. Additionally, there are many factors that influence a person's voting behavior such as race, gender, political views, location, etc. In addition, there are some factors that are hard to collect information about such as what media they consume, how they are influenced by media, opinions affected by friends, family, and others, etc. People also tend to change their mind about who they are voting for, which adds another factor of unpredictability.

In 2012, Nate Silver, an American Statistician, was able to predict the voting outcome of each state despite what the media covered. Silver was able to make accurate predictions because he made a model that was very adjustable. He took the variables that could be easily affected just by random, and made new predictions based on that so his model adjusts by time. He also made it so that he can predict what the future voting behavior is going to look like by measuring uncertainty as a random factor, making it possible to generate varied predictions for voting behavior in each state with some accuracy. Silver also measured variance and looked at the range of variance to compute probability that a candidate would win each day. Using this, he was able to predict what the results would be at the actual election.

The 2012 election results were not a big surprise, but the 2016 election was. This showed us that even current technology was not accurate at making these predictions. Several things went wrong in the 2016 election polling results. First of all, there is polling error such as nonresponse bias and statistical noise. Also, response bias was also prevalent as a lot of Trump supporters did not want to admit to the fact that they are voting for Trump, undermining the polling results, and there were many people who were undecided which led to a large error rate in polling results as well. Future predictions can be made better by improving the bias-variance tradeoff and decreasing nonresponse and response biases. Additionally, predictions have to take into account all voters stances in order to reduce undermining of certain presidential candidates.

Data

Importation of Data.

```
election.raw =
  read.csv("/Users/sriadd/Desktop/PSTAT 131/Final Project/final-project 3/data/election/election.csv")
census_meta =
  read.csv("/Users/sriadd/Desktop/PSTAT 131/Final Project/final-project 3/data/census/metadata.csv", sep = ", ")
census =
  read.csv("/Users/sriadd/Desktop/PSTAT 131/Final Project/final-project 3/data/census/census.csv") %>%
  mutate(CensusTract = as.factor(census$CensusTract))
```

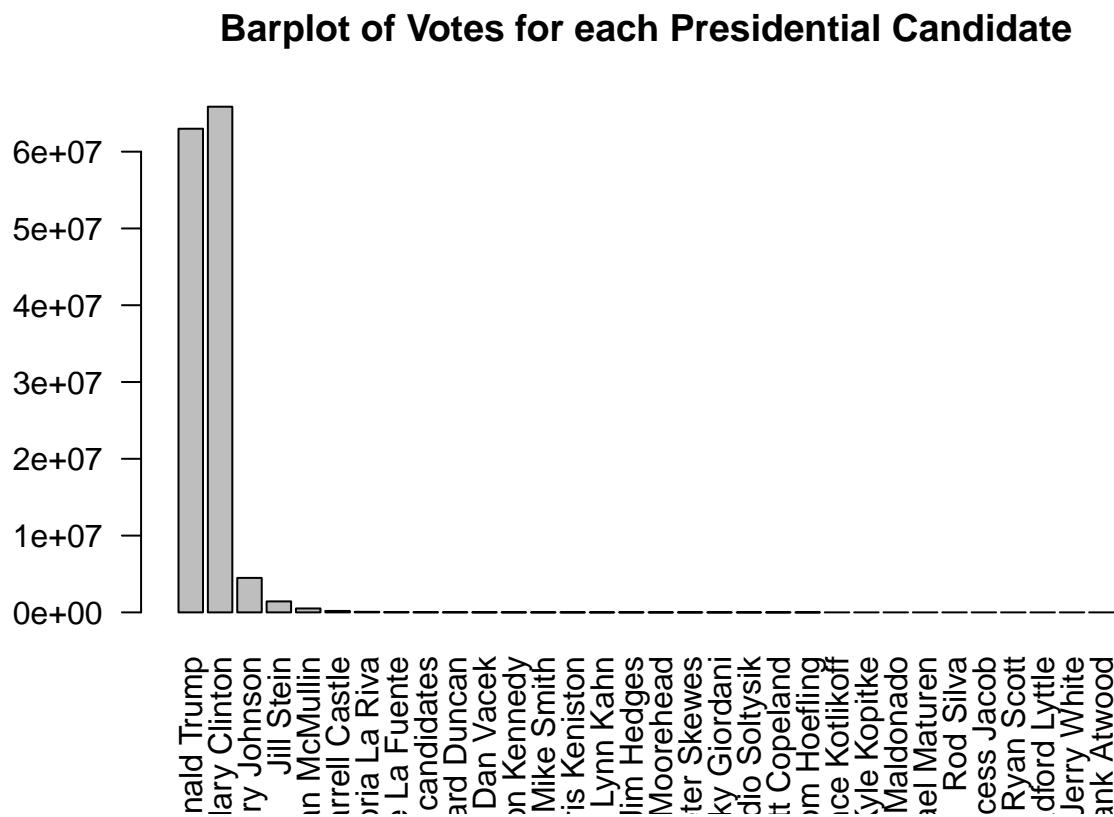
Data Wrangling

We removed summary rows from election.raw and placed federal-level summary and state-level summary in election_federal and election_state respectively. Then we included only county-level data in election.

```
election_federal <- filter(election.raw, state == "US")
election_state_no <- filter(election.raw[33:18351,], is.na(county))
election_no <- filter(election.raw, county != 'NA')
election_state <- election_state_no[1:286,] %>% rbind(election_state_no[293:308,])
election <- election_no %>% rbind(election_state_no[287:292,]) %>%
  rbind(election_state_no[309:312,])
```

In total, there were 31 presidential candidates in the 2016 election. Below is a bar plot showing all the votes received by each candidate.

```
barplot(names.arg = election_federal$candidate, height = election_federal$votes, las = 2,  
        main = "Barplot of Votes for each Presidential Candidate")
```



We created variables county_winner and state_winner which are the candidates with the highest proportion of votes in the county.

```
county_winner <- election %>% group_by(fips) %>% mutate(pct = votes/sum(votes)) %>%  
  top_n(n = 1) #creates winner by each county  
  
## Selecting by pct  
  
state_winner <- election_state %>% group_by(fips) %>% mutate(pct = votes/sum(votes)) %>%  
  top_n(n = 1) #creates winner by each state  
  
## Selecting by pct
```

Visualization

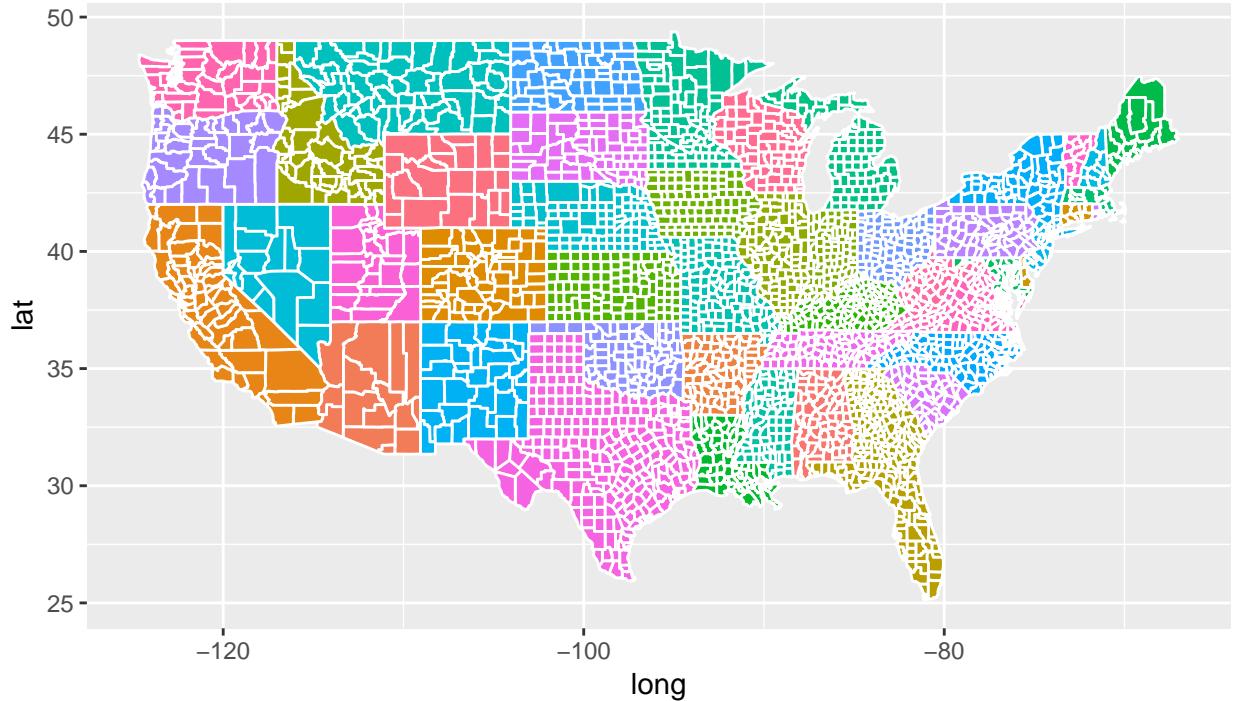
This first map is a county level map. This is a just a map created to visualize all the counties in each state.

```

counties = map_data("county")

ggplot(data = counties) +
  geom_polygon(aes(x = long, y = lat, fill = region, group = group), color = "white") +
  coord_fixed(1.3) +
  guides(fill=FALSE)

```



This map is colored based on the winning candidate for each state. This is done by using the previous variables, states, and left joining it with our variable, state_winner.

```

states = map_data("state")
states <- states %>% mutate(fips = state.abb[match(region, tolower(state.name))])
statesplot <- left_join(states,state_winner)

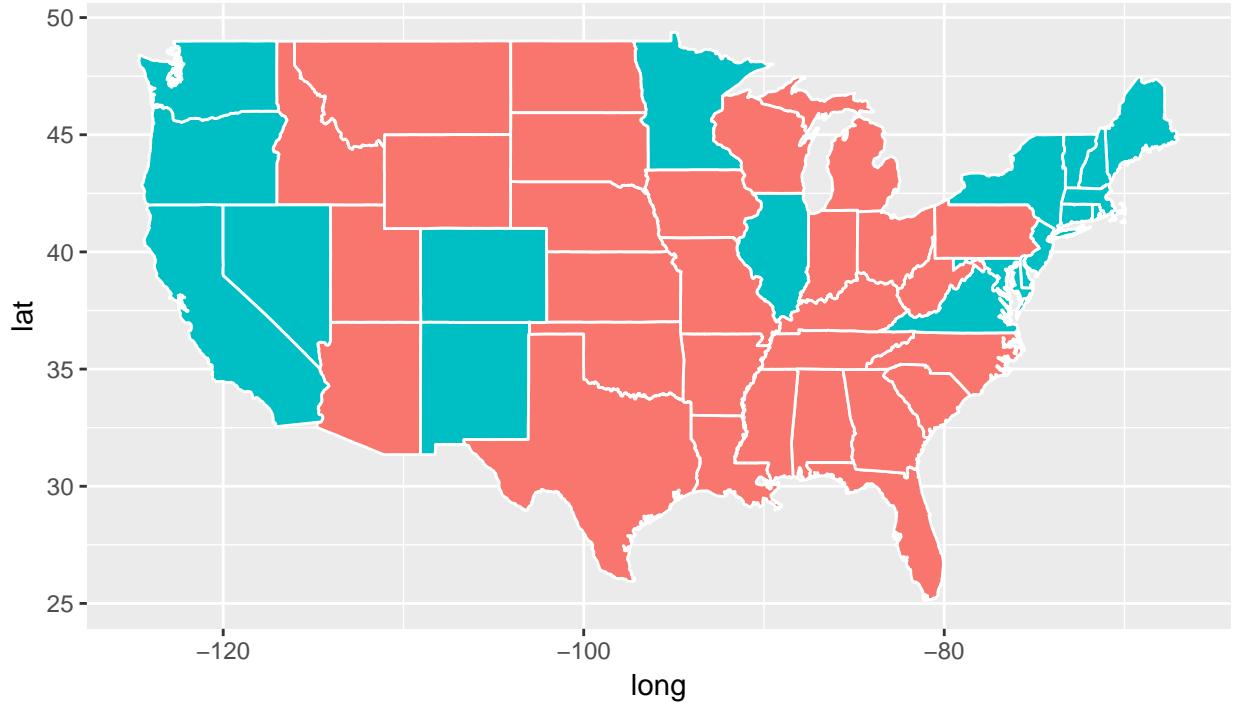
## Joining, by = "fips"

## Warning: Column `fips` joining character vector and factor, coercing into
## character vector

ggplot(data = statesplot) +
  geom_polygon(aes(x = long, y = lat, fill = candidate, group = group), color = "white") +
  coord_fixed(1.3) +
  guides(fill=FALSE) +
  ggtitle("Map of Winning Candidate for each State")

```

Map of Winning Candidate for each State



This map is colored based on the winning candidate for each county.

```
region <- c()
for (i in 1:3085){
  region = c(region,strsplit(maps::county.fips$polyname, ',')[[i]][1])
}
subregion <- c()
for (i in 1:3085){
  subregion = c(subregion,strsplit(maps::county.fips$polyname, ',')[[i]][2])
}
countyregion <- maps::county.fips
countyregion <- countyregion %>% cbind(region) %>% cbind(subregion)
countyleft <- left_join(countyregion,counties)

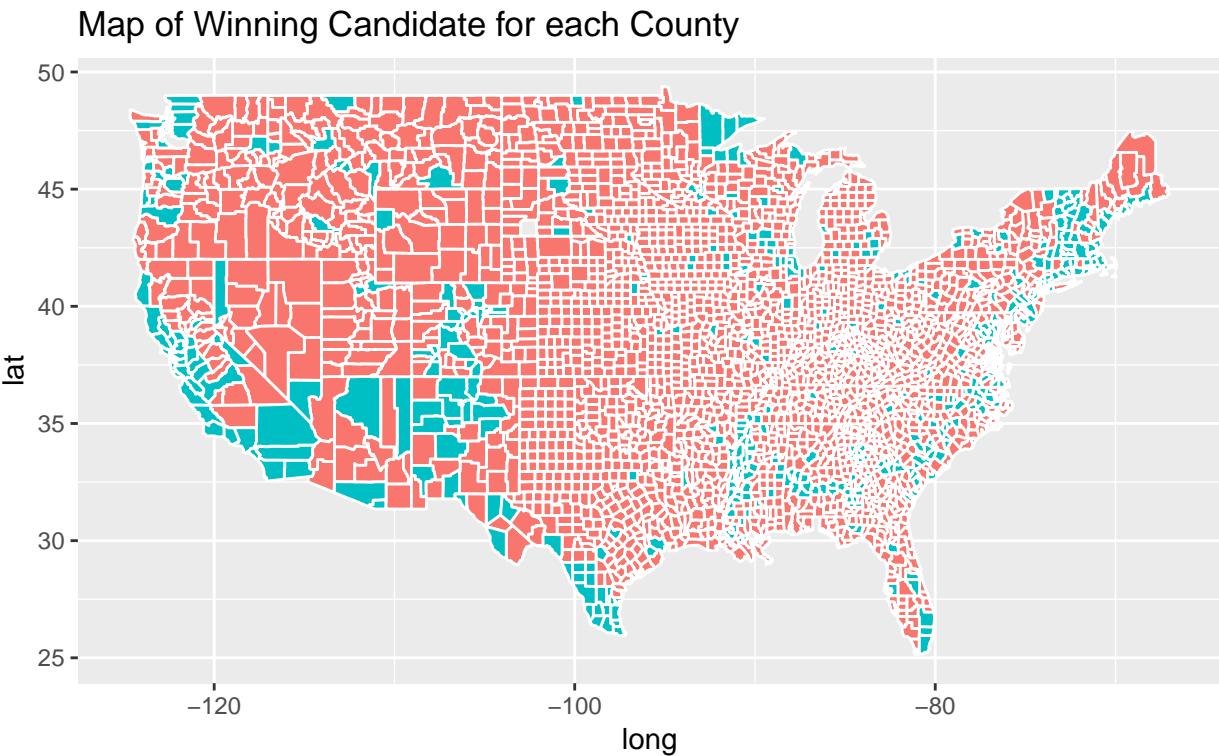
## Joining, by = c("region", "subregion")
## Warning: Column `region` joining factor and character vector, coercing into
## character vector

## Warning: Column `subregion` joining factor and character vector, coercing
## into character vector

countywinnerstrings <- county_winner
countyleftstrings <- countyleft
countywinnerstrings$fips <- as.character(countywinnerstrings$fips)
countyleftstrings$fips <- as.character(countyleftstrings$fips)
countyleft2 <- left_join(countywinnerstrings,countyleftstrings)

## Joining, by = "fips"
ggplot(data = countyleft2) +
  geom_polygon(aes(x = long, y = lat, fill = candidate, group = group), color = "white") +
  coord_fixed(1.3) +
```

```
guides(fill=FALSE) +
ggtitle("Map of Winning Candidate for each County")
```



We then cleaned out the census data from any missing values and converted Men, Employed, and Citizen into percentages, created a Minority attribute by combining Hispanic, Black, Native, Asian, and Pacific. We also removed Walk, PublicWork, and Construction. We also created subcounty census data (census.subct) and computed a CountyTotal and weight by TotalPop/CountyTotal. Then we created county census data (census.ct) using census.subct to compute weighted sum. the following table displays a few rows of census.ct.

```
census.del <- na.omit(census)
census.del$Men <- census.del$Men/sum(census.del$Men)
census.del$Citizen <- census.del$Citizen/sum(census.del$Citizen)
census.del$Employed <- census.del$Employed/sum(census.del$Employed)
census.del <- mutate(census.del, Minority = Hispanic + Black + Native + Asian + Pacific)
census.del <- select(census.del, -c(Walk, PublicWork, Construction))

census.subct <- census.del %>% group_by(State, County) %>% add_tally() %>% mutate(weight = TotalPop/n)

census.ct <- census.subct %>% summarize_at(vars(TotalPop:weight), sum)

tmpwinner = county_winner %>% ungroup %>%
  mutate(state = state.name[match(state, state.abb)]) %>% ## state abbreviations
  mutate_at(vars(state, county), tolower) %>% ## to all lowercase
  mutate(county = gsub(" county| columbia| city| parish", "", county)) ## remove suffixes
tmpcensus = census.ct %>% ungroup %>% mutate_at(vars(State, County), tolower)

election.cl = tmpwinner %>%
  left_join(tmpcensus, by = c("state"="State", "county"="County")) %>%
  na.omit
```

```

## saves meta information to attributes
attr(election.cl, "location") = election.cl %>% select(c(county, fips, state, votes, pct))
election.cl = election.cl %>% select(-c(county, fips, state, votes, pct))

head(census.ct)

## # A tibble: 6 x 36
## # Groups:   State [1]
##   State County TotalPop    Men Women Hispanic White Black Native Asian
##   <fct> <fct>    <int>  <dbl> <int>    <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 Alab~ Autau~    55221 1.71e-4 28476     34.2  878.  250.    6.8   8.4
## 2 Alab~ Baldw~   195121 6.08e-4 99807    126   2587.  298.   18.6  20.4
## 3 Alab~ Barbo~   26932 9.25e-5 12435     43.2   420.  418.    1.3    3
## 4 Alab~ Bibb     22604 7.71e-5 10531      9.9   310   70.9    1.8   0.6
## 5 Alab~ Blount   57710 1.82e-4 29198     80    790.  12.2    2.5   1.1
## 6 Alab~ Bullo~   10678 3.61e-5 5018      8.3    66   217.    4.1   0.7
## # ... with 26 more variables: Pacific <dbl>, Citizen <dbl>, Income <dbl>,
## #   IncomeErr <int>, IncomePerCap <int>, IncomePerCapErr <int>,
## #   Poverty <dbl>, ChildPoverty <dbl>, Professional <dbl>, Service <dbl>,
## #   Office <dbl>, Production <dbl>, Drive <dbl>, Carpool <dbl>,
## #   Transit <dbl>, OtherTransp <dbl>, WorkAtHome <dbl>, MeanCommute <dbl>,
## #   Employed <dbl>, PrivateWork <dbl>, SelfEmployed <dbl>,
## #   FamilyWork <dbl>, Unemployment <dbl>, Minority <dbl>, n <int>,
## #   weight <dbl>

```

The following was done to create interesting visualizations between winning candidate and profession.

```

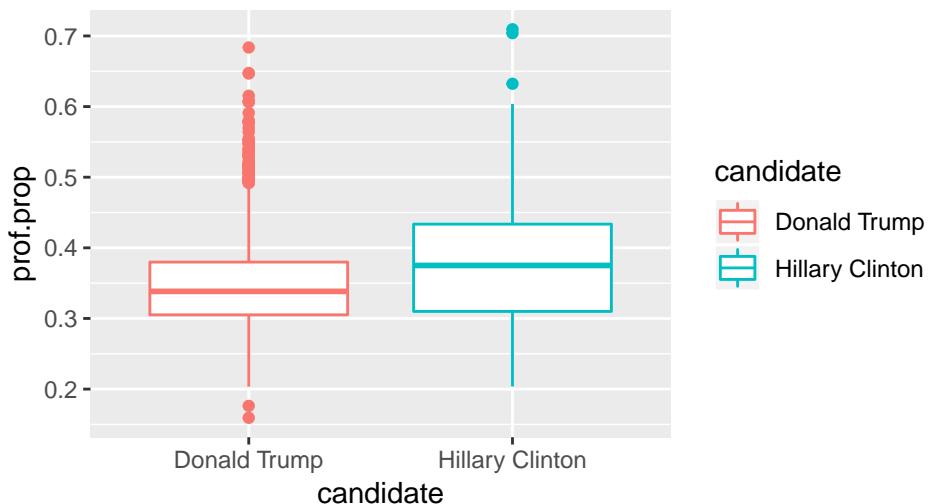
election.cl.jobs <- election.cl %>%
  mutate(prof.prop = Professional/(Professional+Service+Office+Production)) %>%
  mutate(serv.prop = Service/(Professional+Service+Office+Production)) %>%
  mutate(office.prop = Office/(Professional+Service+Office+Production)) %>%
  mutate(prod.prop = Production/(Professional+Service+Office+Production))

p <- (ggplot(election.cl.jobs,aes(candidate,prof.prop,color=candidate)) + geom_boxplot() +
       ggtitle("Boxplot of Winner vs. Percentage of Professional Jobs"))
s <- (ggplot(election.cl.jobs,aes(candidate,serv.prop,color=candidate)) + geom_boxplot() +
       ggtitle("Boxplot of Winner vs. Percentage of Service Jobs"))
o <- (ggplot(election.cl.jobs,aes(candidate,office.prop,color=candidate)) + geom_boxplot() +
       ggtitle("Boxplot of Winner vs. Percentage of Office Jobs"))
pro <- (ggplot(election.cl.jobs,aes(candidate,prod.prop,color=candidate)) + geom_boxplot() +
       ggtitle("Boxplot of Winner vs. Percentage of Production Jobs"))

p

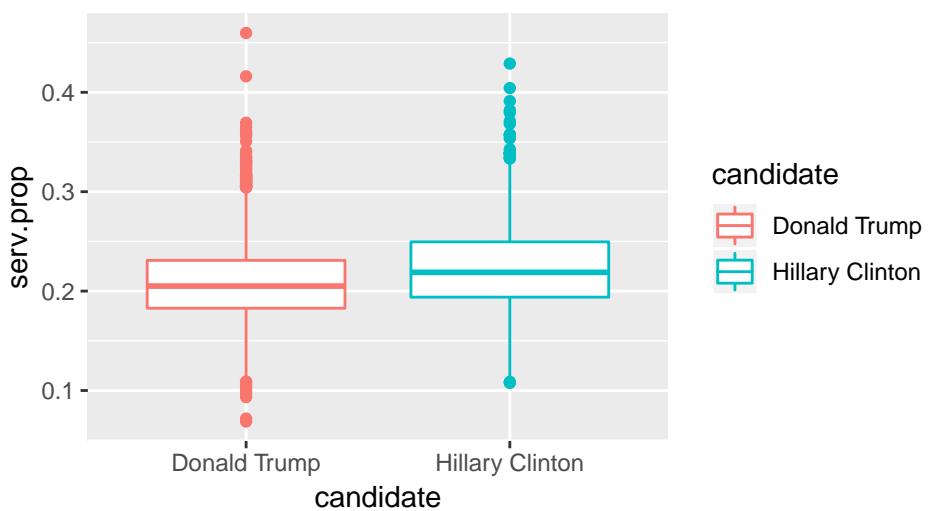
```

Boxplot of Winner vs. Percentage of Professional Jobs



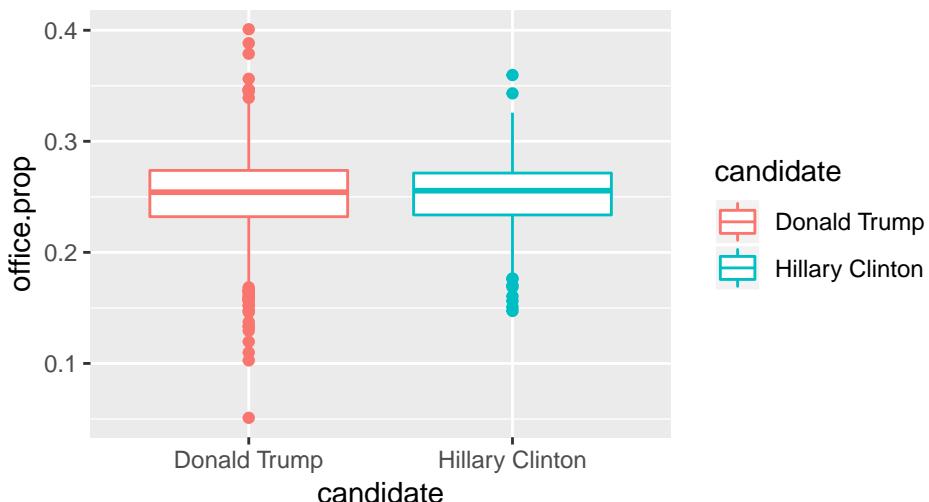
s

Boxplot of Winner vs. Percentage of Service Jobs



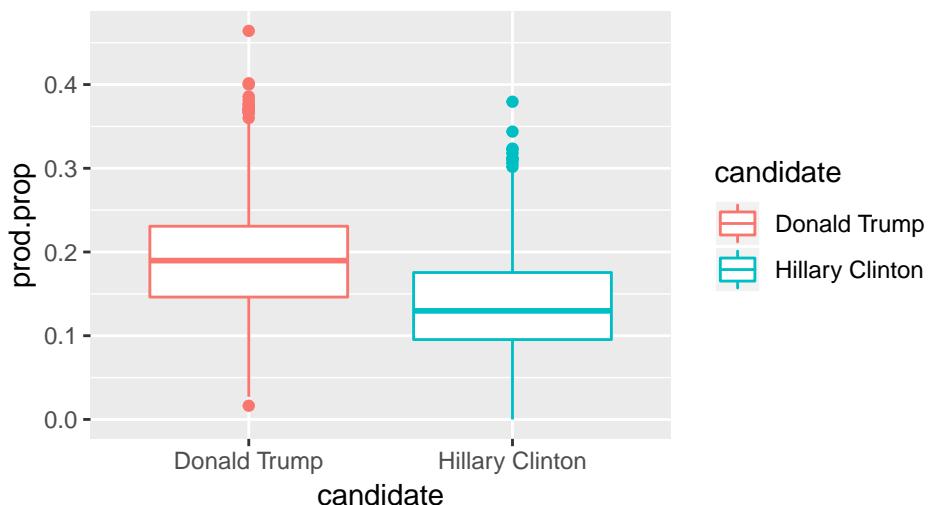
o

Boxplot of Winner vs. Percentage of Office Jobs



pro

Boxplot of Winner vs. Percentage of Production Jobs



These boxplots show the winning candidate for each county based on the profession (professional, service, office, and production). For the boxplot of candidate versus professional jobs and candidate versus service jobs, we observed that the quantiles are slightly higher for Clinton than Trump. In the boxplot of candidate versus office jobs, we observed that the quantiles are similar but Clinton seems to have a smaller variance than Trump. In the boxplot of candidate versus production jobs, the quantiles are slightly higher for Trump than Clinton.

Dimensionality Reduction

We decided to use dimensionality reduction using principal component analysis.

```
censuscounty.pca <- prcomp(census.ct[, 3:36], scale = T)
censussubcounty.pca <- prcomp(census.subct[, 4:37], scale = T)
ct.pc <- data.frame(censuscounty.pca$x)
subct.pc <- data.frame(censussubcounty.pca$x)
```

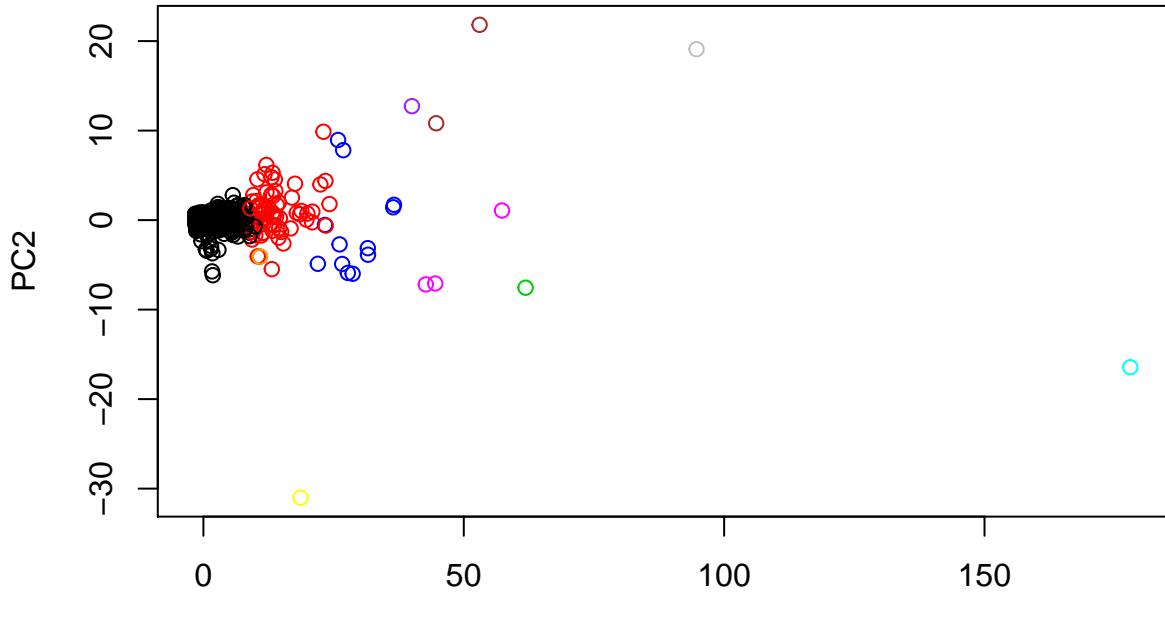
The most prominent loadings are from PC1. This is because PC1 explains the most variance, thus has the more prominent loadings.

Clustering

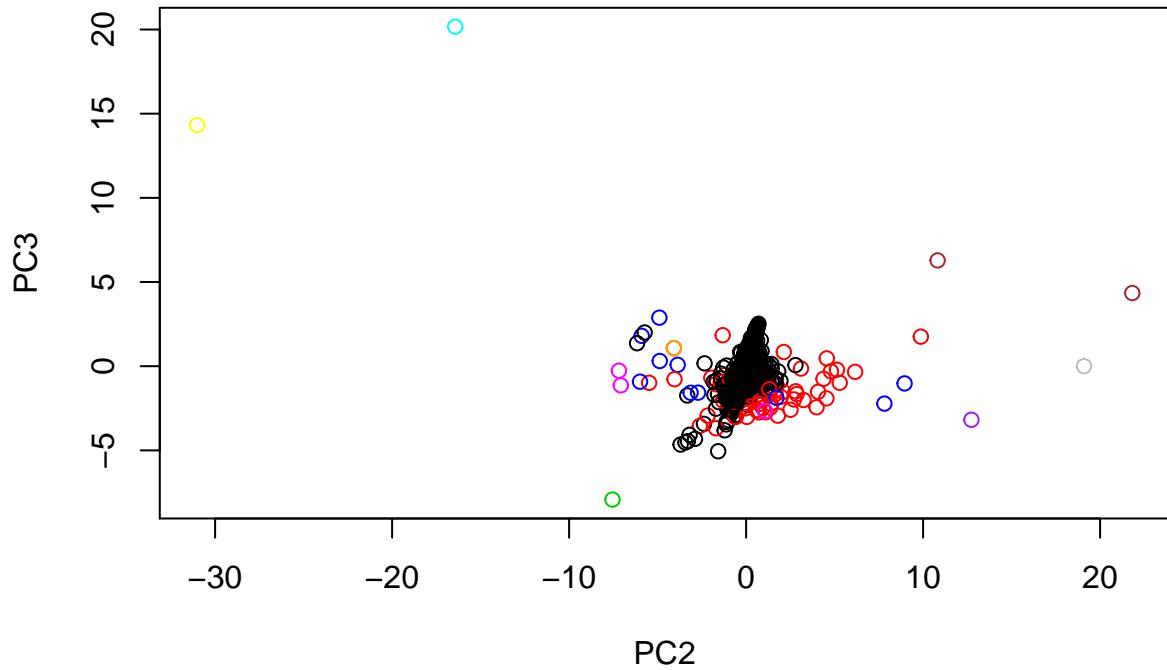
We then performed Hierarchical clustering using the Euclidean distance metric to find 10 clusters. We did so with multiple sets of principal components.

```
scaled_census <- scale(census.ct[,-c(1,2)],center = T,scale = T)
dist.ct <- dist(scaled_census)
hc.ct <- hclust(dist.ct, method = "complete")

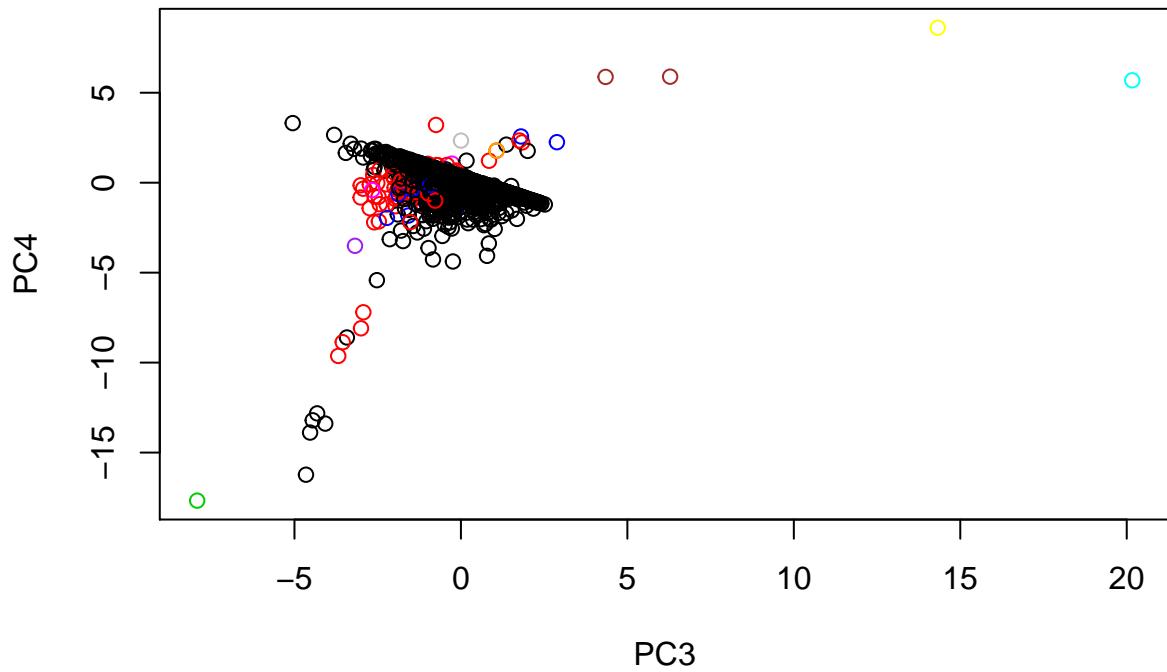
ct.pc2 <- data.frame(censuscounty.pca$x[,1:5])
cc <- palette()
palette(c(cc,"purple","brown"))
plot(ct.pc2[,1:2], col = cutree(hc.ct, k = 10))
points(ct.pc[227,1],ct.pc[227,2], col = "orange")
```



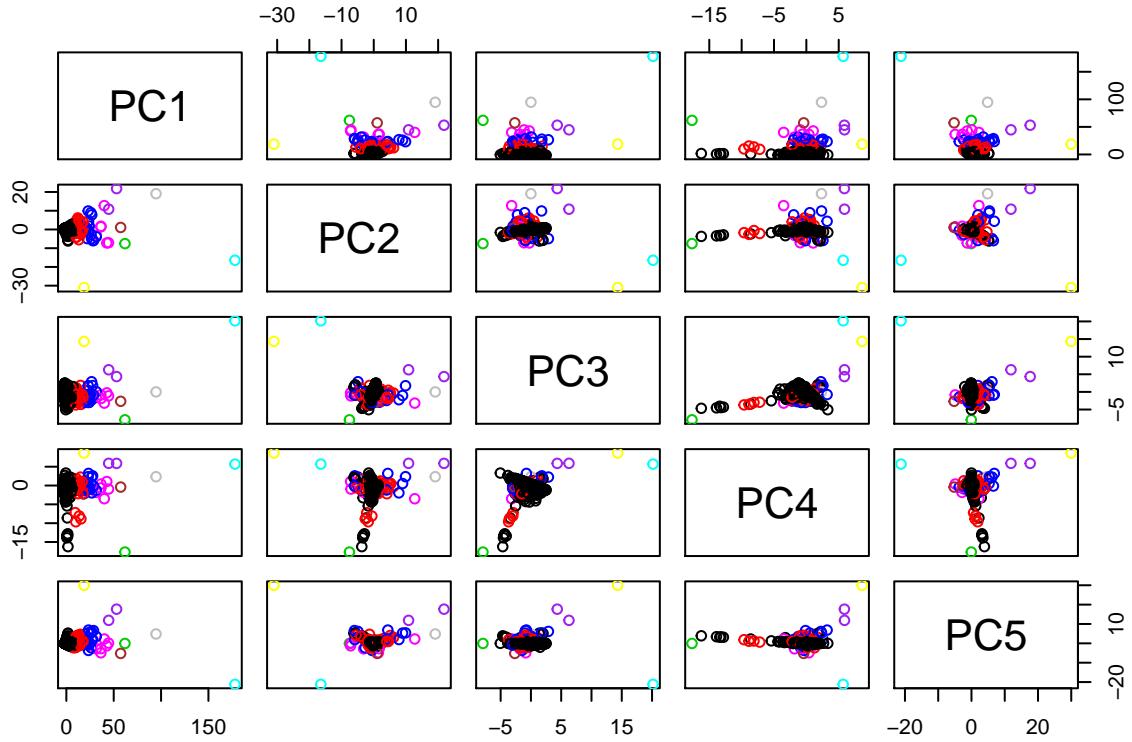
```
plot(ct.pc2[,2:3], col = cutree(hc.ct, k = 10))
points(ct.pc2[227,2],ct.pc2[227,3], col = "orange")
```



```
plot(ct.pc2[,3:4], col = cutree(hc.ct, k = 10))
points(ct.pc2[227,3],ct.pc2[227,4], col = "orange")
```



```
dist.pc <- dist(ct.pc2)
hc.pc <- hclust(dist.pc, method = "complete")
plot(ct.pc2, col = cutree(hc.pc, k=10))
```



The San Mateo county point is indicated on the plots by the orange point. Above are three plots of clusters with different PC loadings, and the overall scatterplot matrix of all the principal components with one another. San Mateo county is part of one of the big clusters but deviates from the rest of the points in that cluster. This may be because Income seems to be much higher than other counties along with many more professionals compared to service and office.

Classification Setup

We partitioned the data into 80% training and 20% test. We also defined 10 cross-validation folds. We also defined an error rate function. We also created a vector matrix where we record the error from training and test and each of the classification methods we use.

```

set.seed(10)
n = nrow(election.cl)
in.trn= sample.int(n, 0.8*n)
trn.cl = election.cl[ in.trn,]
tst.cl = election.cl[-in.trn,]

set.seed(20)
nfold = 10
folds = sample(cut(1:nrow(trn.cl), breaks=nfold, labels=FALSE))

calc_error_rate = function(predicted.value, true.value){
  return(mean(true.value!=predicted.value))
}
records1 = matrix(NA, nrow=3, ncol=2)
colnames(records1) = c("train.error","test.error")
rownames(records1) = c("tree","knn","logistic regression")

```

Classification native-attributes

The first classification method that we used was decision trees. We pruned the tree to minimize missclassification.

```
set.seed(20)
electiontree <- tree(candidate ~ ., data = trn.cl, control = tree.control(nobs = nrow(trn.cl)))
cv <- cv.tree(electiontree, FUN = prune.misclass, K = nfold, rand = folds)
cv

## $size
## [1] 15 14 10 9 7 3 2 1
##
## $dev
## [1] 246 246 242 247 252 262 281 363
##
## $k
## [1] -Inf 0.00 4.00 5.00 9.00 10.75 24.00 86.00
##
## $method
## [1] "misclass"
##
## attr(),"class")
## [1] "prune"      "tree.sequence"
```

We can see that the best size of the cv is going to be the one with the least deviation, and if multiple sizes have the same deviation, we prefer the one with the smallest size. In our case, the size with the smallest deviation would be 10.

The following trees are for both before pruning and after pruning.

```
best.size.cv <- 10
draw.tree(electiontree, cex=0.6, nodeinfo=TRUE)
```

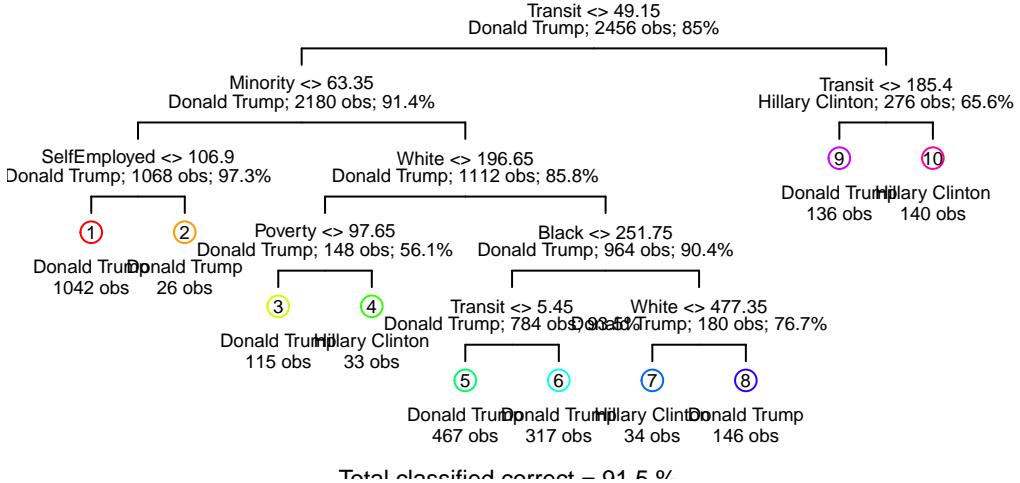
```

graph TD
    0[0] --> 1[1]
    0 --> 14[14]
    1 --> 2[2]
    1 --> 15[15]
    2 --> 3[3]
    2 --> 4[4]
    3 --> 5[5]
    3 --> 6[6]
    4 --> 7[7]
    4 --> 8[8]
    5 --> 9[9]
    5 --> 10[10]
    6 --> 7
    6 --> 12[12]
    7 --> 13[13]
    8 --> 9
    8 --> 11[11]
    9 --> 10
    10 --> 11
    11 --> 12
    11 --> 13
    12 --> 14
    12 --> 15
    13 --> 14
    13 --> 15
    14 --> 15

    0 --- S1[Donald Trump; 2456 obs; 85%]
    1 --- S2[Donald Trump; 2180 obs; 91.4%]
    2 --- S3[Donald Trump; 1068 obs; 97.3%]
    3 --- S4[Donald Trump; 148 obs; 56.1%]
    4 --- S5[Donald Trump; 115 obs; 71.3%]
    5 --- S6[Donald Trump; 83 obs; 86.7%]
    6 --- S7[Donald Trump; 784 obs; 93.5%]
    7 --- S8[Donald Trump; 317 obs; 85.5%]
    8 --- S9[Donald Trump; 467 obs; 88.4%]
    9 --- S10[Donald Trump; 302 obs; 88.4%]
    10 --- S11[Donald Trump; 250 obs; 67.3%]
    11 --- S12[Donald Trump; 52 obs; 67.3%]
    12 --- S13[Hillary Clinton; 136 obs; 40%]
    13 --- S14[Hillary Clinton; 146 obs; 34%]
    14 --- S15[Hillary Clinton; 276 obs; 65.6%]
    15 --- S16[Hillary Clinton; 15 obs; 37%]

    0 --- T1[Transit > 49.15]
    1 --- T2[Transit > 185.4]
    2 --- T3[Minority <= 63.35]
    3 --- T4[SelfEmployed <= 106.9]
    4 --- T5[Poverty <= 97.65]
    5 --- T6[White <= 196.65]
    6 --- T7[Black <= 251.75]
    7 --- T8[Transit <= 5.45]
    8 --- T9[WorkAtHome <= 145.75]
    9 --- T10[Office <= 647.55]
    10 --- T11[Total classified correct = 82.0%]
```

```
electiontree.pruned <- prune.tree(electiontree, best = best.size.cv)
draw.tree(electiontree.pruned, cex = 0.6, nodeinfo = TRUE)
```



```
set.seed(20)
electiontree.pruned.predvl <- predict(electiontree.pruned, tst.cl, type = 'class')
electiontree.pruned.predtr <- predict(electiontree.pruned, trn.cl, type = 'class')
records1[1,1] = calc_error_rate(electiontree.pruned.predtr, trn.cl$candidate)
records1[1,2] = calc_error_rate(electiontree.pruned.predvl, tst.cl$candidate)
```

The next model we trained is the KNN model. We used cross-validation to determine the best number of neighbors.

```
kvec = 29:32 # we evaluated 1:50 but we reduced it to this to speed up knitting time
do.chunk <- function(chunkid, folddef, Xdat, Ydat, k){
  train = (folddef != chunkid)
  Xtr = Xdat[train,]
  Ytr = Ydat[train]
  Xvl = Xdat[!train,]
  Yvl = Ydat[!train]
  ## get classifications for current training chunks
  predYtr = knn(train = Xtr, test = Xtr, cl = Ytr, k = k)
  ## get classifications for current test chunk
  predYvl = knn(train = Xtr, test = Xvl, cl = Ytr, k = k)
  data.frame(train.error = calc_error_rate(predYtr, Ytr),
  val.error = calc_error_rate(predYvl, Yvl))
}

error.folds = NULL
for (j in kvec){
  tmp = lapply(1:9, do.chunk, folds, election.cl %>% select(-candidate), election.cl$candidate, j)
  tmp$neighbors = j
  error.folds = rbind(error.folds, tmp)
}

#errors = melt(error.folds, id.vars='neighbors', value.name='error')
#val.error.means = errors %>%
#filter(variable=='val.error') %>%
#group_by(neighbors, variable) %>%
#summarise_each(funs(mean), error) %>%
#ungroup() %>%
#filter(error==min(error))
```

```
#val.error.means <-- this says 31 neighbors with error 0.1209213
```

The best number of neighbors is 31.

```
best.kfold <- 31
set.seed(20)
Ytrain <- trn.cl$candidate
Xtrain <- trn.cl %>% select(-candidate)
Xtest <- tst.cl %>% select(-candidate)
Ytest <- tst.cl$candidate

pred.YTrain = knn(train=Xtrain, test=Xtrain, cl=Ytrain, k=31)
pred.YTest = knn(train=Xtrain, test=Xtest, cl=Ytrain, k=31)
records1[2,1] = calc_error_rate(pred.YTrain, trn.cl$candidate)
records1[2,2] = calc_error_rate(pred.YTest, tst.cl$candidate)
```

Classification: principal components

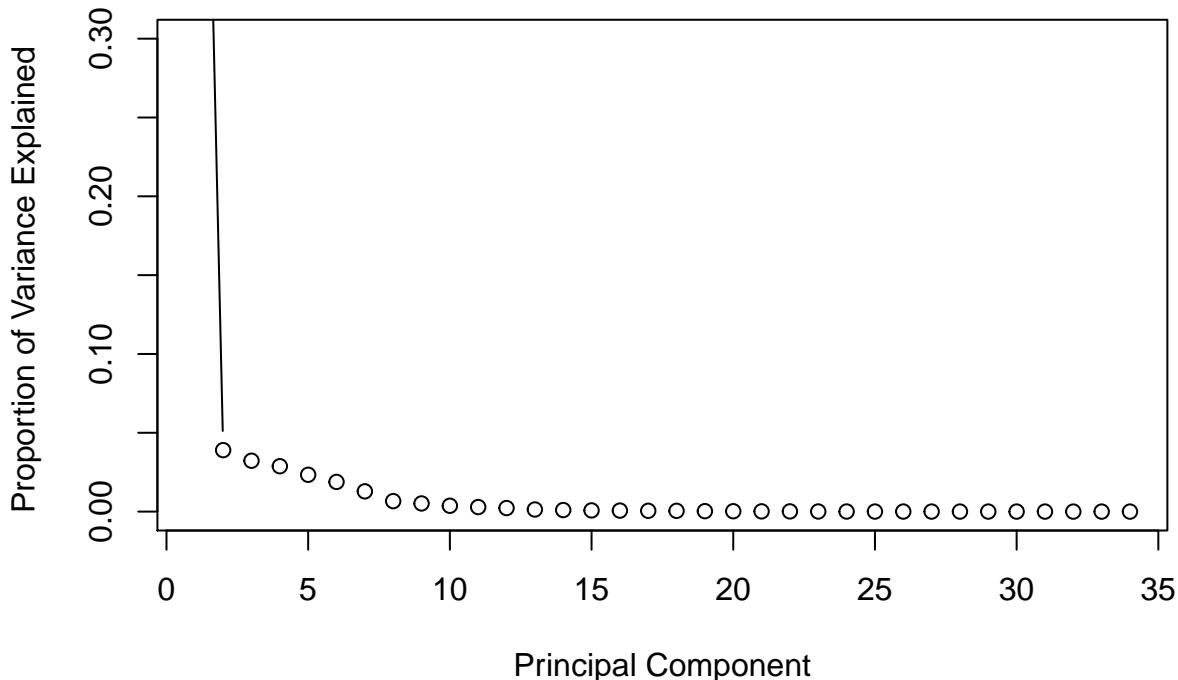
We also used principal components to train our classification models.

```
pca.records = matrix(NA, nrow=2, ncol=2)
colnames(pca.records) = c("train.error", "test.error")
rownames(pca.records) = c("tree", "knn")
```

We computed the principal components form the independent variables in our dataset. The following plot shows the proportion of the variance explained based on the number of principal components.

```
trn.cl.indp <- trn.cl %>% select(-candidate)
trn.cl.indp.pc <- prcomp(trn.cl.indp, scale = T)

pr.var = trn.cl.indp.pc$sdev ^2
pve = pr.var/sum(pr.var)
plot(pve, xlab="Principal Component", ylab="Proportion of Variance Explained ", ylim=c(0,0.3), type='b')
```



The minimum number of PC's required to capture 90% of the variance is 4.

We created a new training and test data using principal components.

```
tr.pca <- data.frame(trn.cl.indp.pc$x[,1:4])
tr.pca <- cbind(trn.cl$candidate, tr.pca)

tst.cl.indp <- tst.cl %>% select(-candidate)
tst.cl.indp.pc <- prcomp(tst.cl.indp, scale = T)

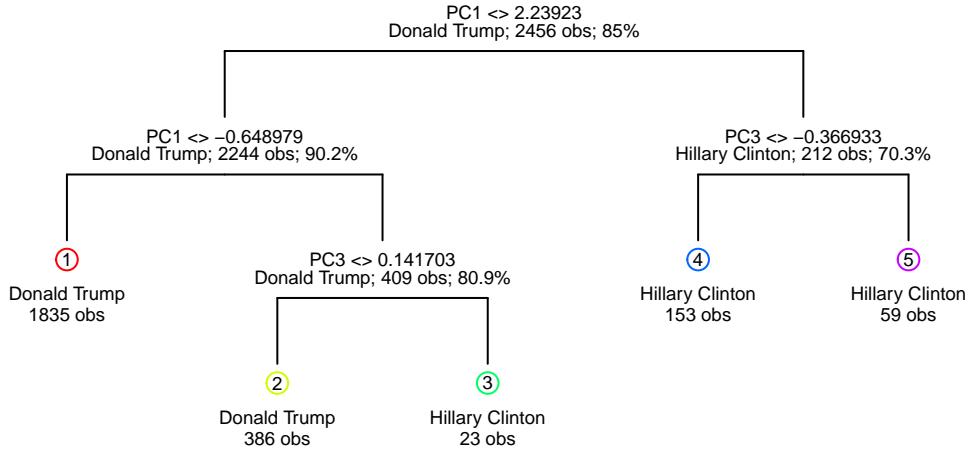
test.pca <- data.frame(tst.cl.indp.pc$x[,1:4])
test.pca <- cbind(tst.cl$candidate, test.pca)
colnames(tr.pca)[1] <- "candidate"
colnames(test.pca)[1] <- "candidate"
```

We then repeated decision trees using the principal components and independent variables.

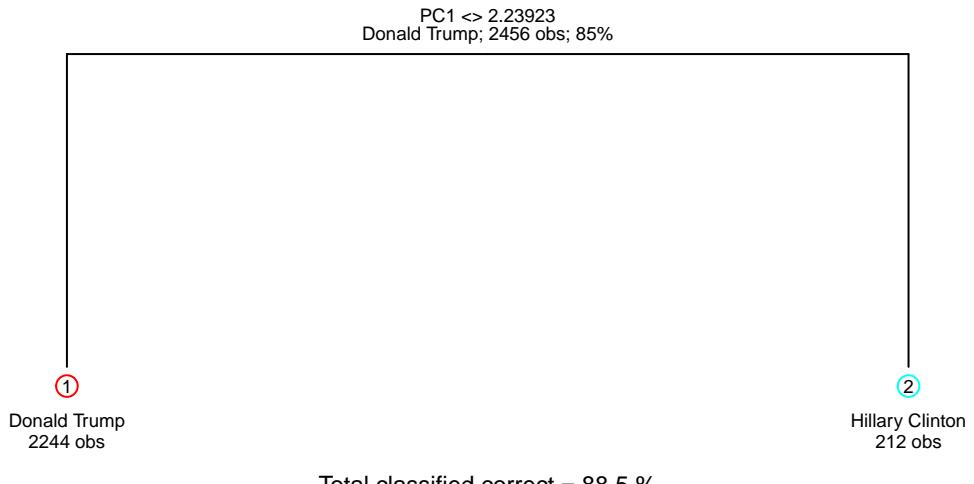
```
set.seed(20)
pcatree <- tree(candidate ~ ., data = tr.pca, control = tree.control(nobs = nrow(tr.pca)))
cvtree <- cv.tree(pcatree, FUN = prune.misclass, K = nfold, rand = folds)
cvtree #best size is 2

## $size
## [1] 5 4 2 1
##
## $dev
## [1] 293 293 286 368
##
## $k
## [1] -Inf  0.0  3.5 86.0
##
## $method
## [1] "misclass"
##
```

```
## attr(,"class")
## [1] "prune"           "tree.sequence"
best.size.cvtree <- 2
draw.tree(pcatree, cex=0.6, nodeinfo=TRUE)
```



```
pcatree.pruned <- prune.tree(pcatree, best = best.size.cvtree)
draw.tree(pcatree.pruned, cex = 0.6, nodeinfo = TRUE)
```



```
set.seed(20)
pcatree.pruned.predvl <- predict(pcatree.pruned, test.pca, type = 'class')
pcatree.pruned.predtr <- predict(pcatree.pruned, tr.pca, type = 'class')
pca.records[1,1] = calc_error_rate(pcatree.pruned.predtr, tr.pca$candidate)
pca.records[1,2] = calc_error_rate(pcatree.pruned.predvl, test.pca$candidate)
```

We also repeated KNN classification using the principal components and independent variables

```
kvec1 <- 17:19 # we evaluated 1:50 but we reduced it to this to speed up knitting time
election.cl.indp <- election.cl %>% select(-candidate)
election.cl.indp.pc <- prcomp(election.cl.indp, scale = T)

election.pca <- data.frame(election.cl.indp.pc$x[,1:4])
election.pca <- cbind(election.cl$candidate, election.pca)
colnames(election.pca)[1] <- "candidate"
```

```

error.folds = NULL
for (j in kvec1){
  tmp = lapply(1:9, do.chunk, folds, election.pca %>% select(-candidate), election.pca$candidate, j)
  tmp$neighbors = j
  error.folds = rbind(error.folds, tmp)
}

#errors = melt(error.folds, id.vars='neighbors', value.name='error')
#val.error.means.pca = errors %>%
#filter(variable=='val.error') %>%
#summarise_each(funs(mean), error) %>%
#ungroup() %>%
#filter(error==min(error))

#val.error.means.pca <-- give neighbors of 18 and error 0.117608

```

The best choice of neighbors is 18.

```

best.kfold <- 18
set.seed(20)
Ytrainpca <- tr.pca$candidate
Xtrainpca <- tr.pca %>% select(-candidate)
Xtestpca <- test.pca %>% select(-candidate)
Ytestpca <- test.pca$candidate

pred.YTrainpca = knn(train=Xtrainpca, test=Xtrainpca, cl=Ytrainpca, k=18)
pred.YTestpca = knn(train=Xtrainpca, test=Xtestpca, cl=Ytrainpca, k=18)
pca.records[2,1] = calc_error_rate(pred.YTrainpca, tr.pca$candidate)
pca.records[2,2] = calc_error_rate(pred.YTestpca, test.pca$candidate)

```

Interpretation and Discussion

```

records1

##          train.error test.error
## tree      0.08509772 0.08794788
## knn       0.11970684 0.12703583
## logistic regression     NA      NA
pca.records

##          train.error test.error
## tree    0.1148208  0.1563518
## knn     0.1123779  0.2052117

```

Our classification methods using decision trees and KNN show how the polls did not predict the outcome of the election. We noticed that in the classification of native attributes, our error rate for decision trees was 8.8% while error for KNN was 12.7%. Normally, it is said that polling errors are usually around 3%, but our error rates show that this is not true. These error rates show that 8.8% (decision trees) and 12.7% (KNN) of predictions for winner of the county was classified incorrectly.

Our classification methods with pca yielded higher test error rates. This may be because of some relationship between the variables that PCA is not accounting for properly.

If there is additional data collection, one thing that may be asked is whether a person is a democrat, republican,

or affiliated to any other political party. This may reduce the bias and undermining of polls as people may be more comfortable to answer what their political affiliation in addition to who they are going to vote for.

Taking it Further

We also wanted to perform logistic regression to see if it will produce less test error:

```
set.seed(20)
trn.cl факт <- as.factor(trn.cl$candidate)
trn.cl факт <- cbind(trn.cl факт, trn.cl[, 2:35])
colnames(trn.cl факт)[1] <- "candidate"

glm.fit <- glm(candidate ~ ., data = trn.cl, family = binomial)

## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
probtraining <- round(predict(glm.fit, type='response'))

prob.cl.training3 = trn.cl %>%
  mutate(predy = as.factor(ifelse(probtraining<=0.5, "Donald Trump", "Hillary Clinton")))

prob.cl.training3$candidate <- factor(prob.cl.training3$candidate)

pred_prob_training <- calc_error_rate(prob.cl.training3$predy, prob.cl.training3$candidate)
pred_prob_training

## [1] 0.1119707

probtest <- round(predict(glm.fit, tst.cl, type='response'))

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading
prob.cl.test = tst.cl %>%
  mutate(predy = as.factor(ifelse(probtest<=0.5, "Donald Trump", "Hillary Clinton")))

prob.cl.test$candidate <- factor(prob.cl.test$candidate)

pred_prob_test <- calc_error_rate(prob.cl.test$predy, prob.cl.test$candidate)
pred_prob_test

## [1] 0.1368078

records1[3,1] <- pred_prob_training
records1[3,2] <- pred_prob_test
records1

##          train.error test.error
## tree           0.08509772 0.08794788
## knn            0.11970684 0.12703583
## logistic regression 0.11197068 0.13680782
```

With logistic regression, we got a smaller test error than KNN gave, but a higher test error than decision trees. If the decision boundary is linear, than logistic regression outperforms KNN, which was the case for

this data. Decision trees most likely had the least test error because decision trees are not sensitive to scale invariants and transformations of the predictors. In our case, the predictors did have many different scales, so this follows the intuition.