PSTAT 131 Final Project: 2020 Election Analysis

Brandon Lee (3352028)

12/17/2020

Data

We will begin our analysis of the 2020 Election by reading in two datasets from the "election.raw" and "census_county.csv" files:

```
## read data and convert candidate names and party names from string to factor
election.raw <- read_csv("candidates_county.csv", col_names = TRUE) %>%
  mutate(candidate = as.factor(candidate), party = as.factor(party))

## remove the word "County" from the county names
words.to.remove = c("County")
remove.words <- function(str, words.to.remove){
  sapply(str, function(str){
    x <- unlist(strsplit(str, " "))
    x <- x[!x %in% words.to.remove]
    return(paste(x, collapse = " "))
  }, simplify = "array", USE.NAMES = FALSE)
}
election.raw$county <- remove.words(election.raw$county, words.to.remove)

## read census data
census <- read_csv("census_county.csv")
```

The first dataset contains election data consisting of county-level election results. (This dataset does not contain the final election results due to the fact that recounting was still taking place in many states. The second dataset contains the 2017 United States county-level census data.

Election Data

1. Report the dimension of election.raw. Are there missing values in the data set? Compute the total number of distinct values in state in election.raw to verify that the data contains all states and a federal district.

```
#find dimension of election.raw
dim(election.raw)
```

```
## [1] 31167     5
```

The dimension of election.raw is is 31,167x5.

```
#calculate number of missing observations
sum(!complete.cases(election.raw))
```

```
## [1] 0
```

```
#calculate number of missing values in each variable
election.raw %>%
  summarise_all(~sum(is.na(.)))
```

```
## # A tibble: 1 x 5
##   state county candidate party votes
##   <int>  <int>     <int> <int> <int>
## 1     0      0         0     0     0
```

There are no missing values in the data set.

```
#compute total number if distinct values in state
election.raw %>%
  group_by(state) %>%
  summarise(n = n(), na.rm=TRUE)
```

```
## # A tibble: 51 x 3
##    state                   n na.rm
##    <chr>               <int> <lgl>
##  1 Alabama               268 TRUE
##  2 Alaska                280 TRUE
##  3 Arizona                47 TRUE
##  4 Arkansas              975 TRUE
##  5 California            355 TRUE
##  6 Colorado             1354 TRUE
##  7 Connecticut           845 TRUE
##  8 Delaware               14 TRUE
##  9 District of Columbia   56 TRUE
## 10 Florida               522 TRUE
## # ... with 41 more rows
```

The total number of distinct values in the state variable in the election.raw data is 51, which confirms that the data includes all states and a federal district.

Census Data

2. Report the dimension of census. Are there missing values in the data set? Compute the total number of distinct values in county in census. Compare the values of total number of distinct county in census with that in election.raw. Comment on your findings.

```r
#find dimension of census
dim(census)
```

```
## [1] 3220   37
```

The dimension of census is 3,220x37.

```r
#calculate number of missing observations
sum(!complete.cases(census))
```

```
## [1] 1
```

```r
#calculate number of missing values in each variable
census %>%
  summarise_all(~sum(is.na(.)))
```

```
## # A tibble: 1 x 37
##   CountyId State County TotalPop   Men Women Hispanic White Black Native Asian
##      <int> <int>  <int>    <int> <int> <int>    <int> <int> <int>  <int> <int>
## 1        0     0      0        0     0     0        0     0     0      0     0
## # ... with 26 more variables: Pacific <int>, VotingAgeCitizen <int>,
## #   Income <int>, IncomeErr <int>, IncomePerCap <int>, IncomePerCapErr <int>,
## #   Poverty <int>, ChildPoverty <int>, Professional <int>, Service <int>,
## #   Office <int>, Construction <int>, Production <int>, Drive <int>,
## #   Carpool <int>, Transit <int>, Walk <int>, OtherTransp <int>,
## #   WorkAtHome <int>, MeanCommute <int>, Employed <int>, PrivateWork <int>,
## #   PublicWork <int>, SelfEmployed <int>, FamilyWork <int>, Unemployment <int>
```

There is 1 missing value in the ChildPoverty variable in the census data.

```r
#compute total number of distinct values in County
census %>%
  group_by(County) %>%
  summarise(n = n(), na.rm=TRUE)
```

```
## # A tibble: 1,955 x 3
##    County                n na.rm
##    <chr>             <int> <lgl>
##  1 Abbeville County      1 TRUE
##  2 Acadia Parish         1 TRUE
##  3 Accomack County       1 TRUE
##  4 Ada County            1 TRUE
##  5 Adair County          4 TRUE
##  6 Adams County         12 TRUE
##  7 Addison County        1 TRUE
##  8 Adjuntas Municipio    1 TRUE
##  9 Aguada Municipio      1 TRUE
## 10 Aguadilla Municipio   1 TRUE
## # ... with 1,945 more rows
```

```r
#compute total number of distinct values in county
election.raw %>%
  group_by(county) %>%
  summarise(n = n(), na.rm=TRUE)
```

```
## # A tibble: 2,825 x 3
##    county            n na.rm
##    <chr>         <int> <lgl>
##  1 Abbeville         5 TRUE
##  2 Abbot             6 TRUE
##  3 Abington          5 TRUE
##  4 Acadia Parish    13 TRUE
##  5 Accomack          3 TRUE
##  6 Acton            11 TRUE
##  7 Acushnet          5 TRUE
##  8 Acworth           4 TRUE
##  9 Ada               7 TRUE
## 10 Adair            28 TRUE
## # ... with 2,815 more rows
```

The total number of distinct values in the County variable in the census data is 1,955, while the the total number of distinct values in the county variable in the election.raw data is 2,825. This suggests that the census data demographics were gathered with a more general, broader county location than the election.raw data.

Data Wrangling

3. Construct aggregated data sets from election.raw data: i.e.,

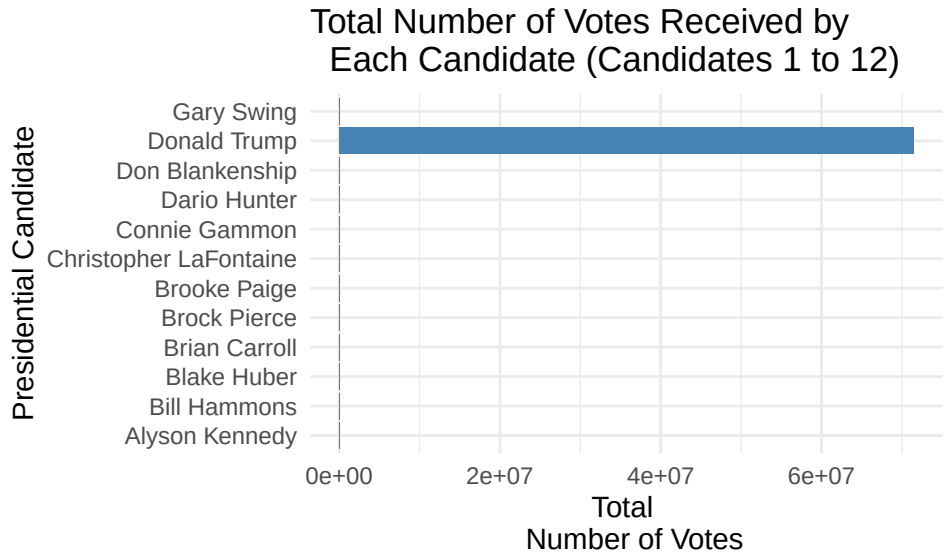
- Keep the county-level data as it is in election.raw.
- Create a state-level summary into a election.state.
- Create a federal-level summary into a election.total.

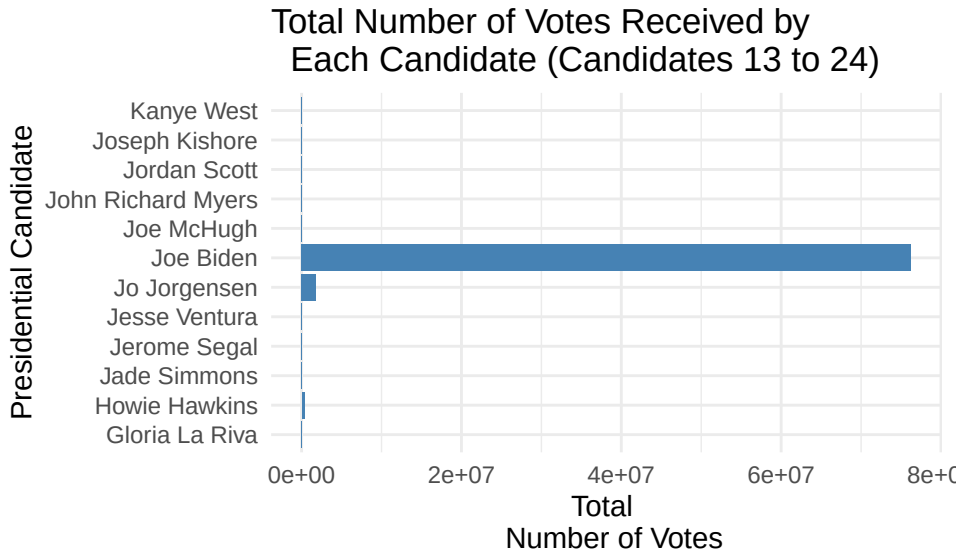
```r
#create a state-level summary for election.raw
election.state = election.raw %>%
  group_by(state,candidate) %>%
  summarise(votes = sum(votes), na.rm=TRUE)

#create a federal-level summary
election.total = election.raw %>%
  group_by(candidate) %>%
  summarise(votes = sum(votes), na.rm=TRUE)
```

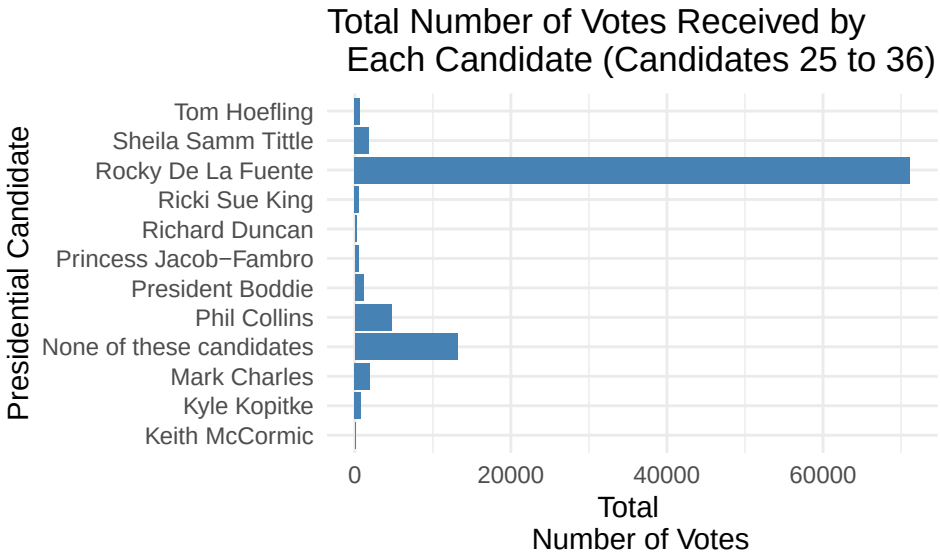
4. How many named presidential candidates were there in the 2020 election? Draw a bar chart of all votes received by each candidate. You can split this into multiple plots or may prefer to plot the results on a log scale. Either way, the results should be clear and legible!

After excluding "None of these candidates" and "Write-ins," we see that there was a total of 36 named presidential candidates in the 2020 election. We proceed to create bar charts that display the number of all

votes received by each candidate in groups of 12 per bar chart (Note that the scaling of "Total Number of Votes" changes between plots):





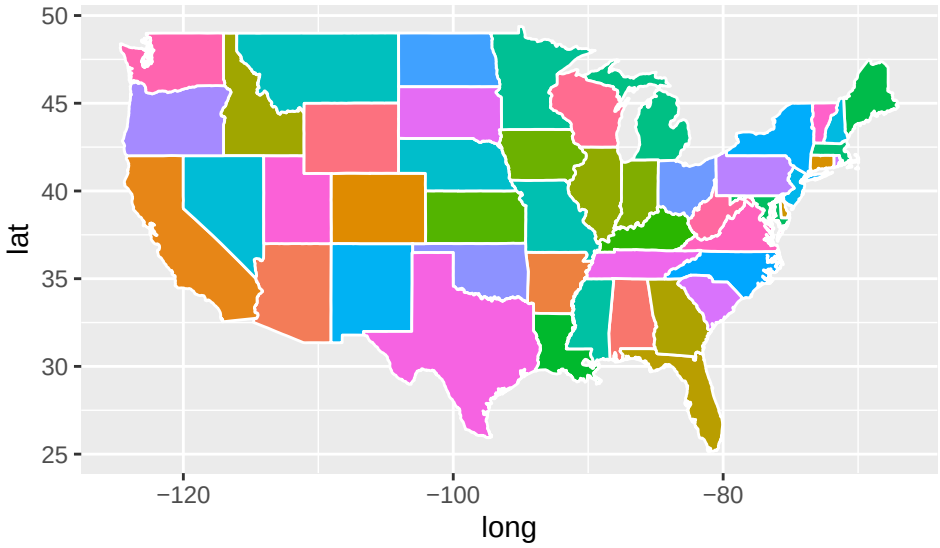


5. Create data sets county.winner and state.winner by taking the candidate with the highest proportion of votes in both county level and state level.

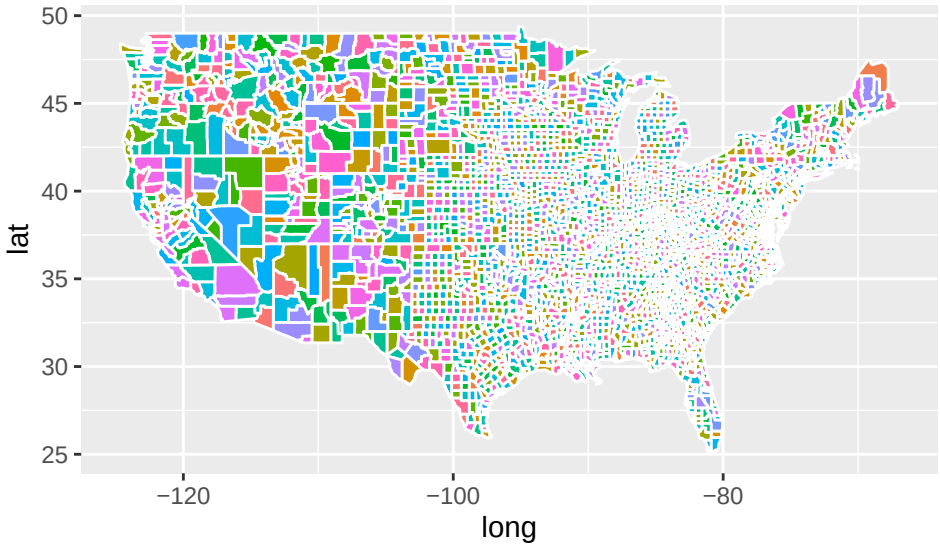
```
#create county.winner dataset
county.winner = election.raw %>%
  group_by(county) %>%
  mutate(total = sum(votes), pct = votes/total) %>%
  top_n(1, pct)

#create state.winner dataset
state.winner = election.state %>%
  group_by(state) %>%
  mutate(total = sum(votes), pct = votes/total) %>%
  top_n(1, pct)
```

Visualization

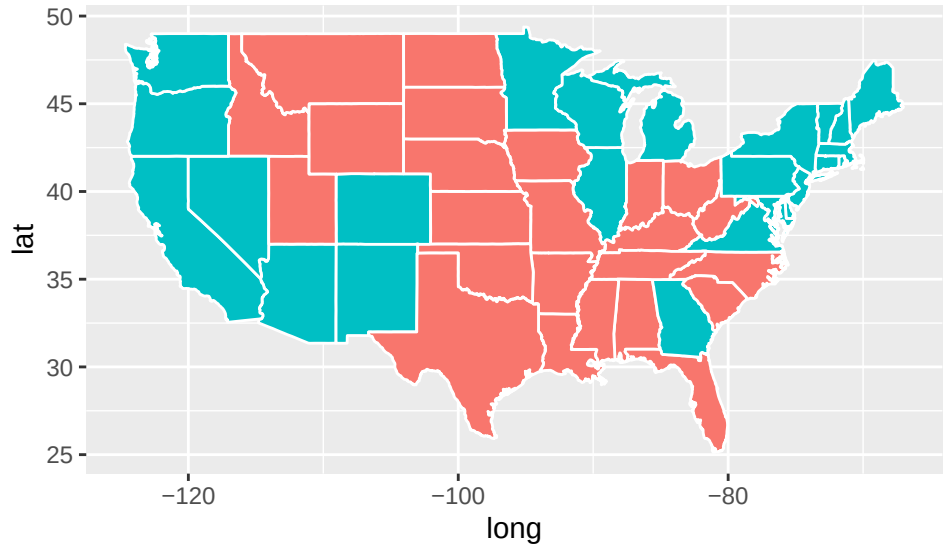
Since visualization is a very powerful tool in data mining due its importance in insight and intuition gathering, we will begin to literally map our data onto maps. We start with a state-level map:



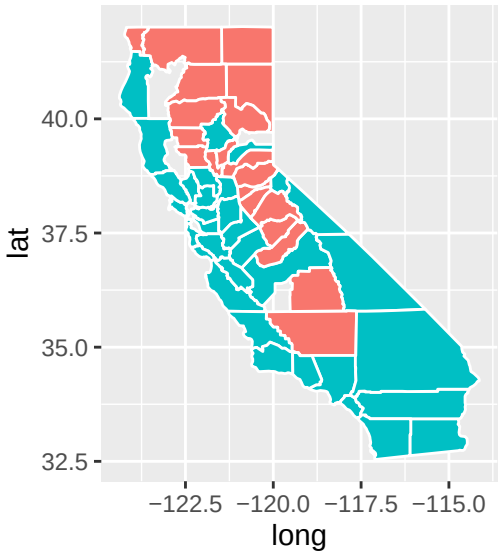
6. Draw a county-level map by creating counties = map_data("county"). Color by county.



7. Now color the map by the winning candidate for each state.

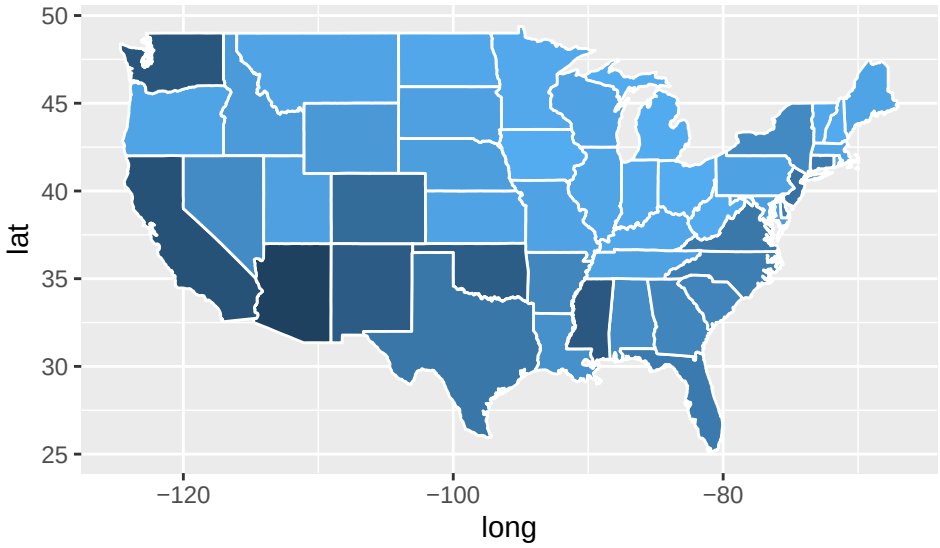


8. Color the map of the state of California by the winning candidate for each county. Note that some county have not finished counting the votes, and thus do not have a winner. Leave these counties uncolored.



9. (Open-ended) Create a visualization of your choice using census data.

We choose to color the map of the US by the percentage of the White population in each state to have an idea on whether or not there are differences between the voting preferences of White people and people of color for this election specifically:



When comparing this map colored by the percentage of White people in each state and the map colored by winning candidate, we see that in the bottom left area of the US there seems to be at least some correlation of states with a higher percentage of people of color voting for Biden. However, this relationship cannot be clearly seen in different areas of the US map.

10. The census data contains county-level census information. In this problem, we clean and aggregate the information as follows.

```
#clean county-level census data
census.clean = census %>%
  filter(complete.cases(census)) %>%
  mutate(Men=Men/TotalPop, Employed=Employed/TotalPop,
         VotingAgeCitizen=VotingAgeCitizen/TotalPop) %>%
  mutate(Minority = Hispanic+Black+Native+Asian+Pacific) %>%
  select(-c(Hispanic, Black, Native, Asian, Pacific)) %>%
  select(-c(IncomeErr, IncomePerCap, IncomePerCapErr, Walk, PublicWork, Construction))
```

The last step of the census data cleaning process is determining which of the columns are perfectly colineared pairs. To accomplish this, we use the function findLinearCombos from the caret package to find all linear combinations in census.clean:

```
#use findLinearCombos from caret package to find perfect colineared columns
findLinearCombos(census.clean[-c(2,3)])
```

```
## $linearCombos
## list()
##
## $remove
## NULL
```

Based on the findLinearCombos output, however, there does not seem to be any perfectly colineared pairs of columns. As such, we instead attempt to find columns that are very highly correlated:

```
#find columns that are highly correlated
high.cor = subset(as.data.frame(which(cor(census.clean[-c(2,3)]) > 0.9, arr.ind=TRUE)), row < col)
high.cor
```

```
##            row col
## TotalPop.1   2   4
## Poverty.1    8   9
```

From the given output it seems that TotalPop and Poverty seem to be highly correlated. And although the columns Men and Women or Employment and Unemployment or White and Minority were not output, intuitively the proportion of these pairs of columns are closely related to each other. As such, for our final step in data cleaning, we decide to remove Poverty, TotalPop, Women, Unemployment, and White from the census.clean dataset:

```
#remove Poverty from census.clean
census.clean = census.clean %>%
  select(-c(TotalPop, Poverty, Women, Unemployment, White))

#print first 5 rows of census.clean
head(census.clean, n=5)
```

```
## # A tibble: 5 x 22
##   CountyId State County   Men VotingAgeCitizen Income ChildPoverty Professional
##      <dbl> <chr> <chr>  <dbl>            <dbl>  <dbl>        <dbl>        <dbl>
## 1     1001 Alab~ Autau~ 0.489            0.745  55317         20.1         35.3
## 2     1003 Alab~ Baldw~ 0.489            0.764  52562         16.1         35.7
## 3     1005 Alab~ Barbo~ 0.533            0.774  33368         44.9         25
## 4     1007 Alab~ Bibb ~ 0.543            0.782  43404         26.6         24.4
## 5     1009 Alab~ Bloun~ 0.494            0.737  47412         25.4         28.5
## # ... with 14 more variables: 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>, Minority <dbl>
```

Dimensionality reduction

11. Run PCA for the cleaned county level census data (with State and County excluded).

```
#run PAC
pr.out = prcomp(census.clean[-c(2,3)], scale=TRUE, center=TRUE)

#save first two principal components as two column data frame
pc.county = as.data.frame(pr.out$x[,1:2])

#calculate the 3 highest absolute loadings for PC1
abs.PC1 = as.data.frame(abs(pr.out$rotation[,1]))
HAL = abs.PC1[order(-abs.PC1$`abs(pr.out$rotation[, 1])`), , drop = FALSE]
head(HAL, n=3)
```

```
##              abs(pr.out$rotation[, 1])
## ChildPoverty                    0.3863
## Employed                        0.3805
## Income                          0.3681
```

```
#output first two PC features
pc.features = as.data.frame(pr.out$rotation[,1:2])
pc.features
```

```
##                       PC1      PC2
## CountyId          0.05250 -0.04335
## Men              -0.03441 -0.18422
## VotingAgeCitizen  0.02510 -0.02943
## Income           -0.36807  0.26306
## ChildPoverty      0.38626 -0.21945
## Professional     -0.36256  0.04322
## Service           0.19441 -0.23366
## Office            0.10021  0.20857
## Production        0.21888  0.14123
## Drive             0.26129  0.32087
## Carpool           0.07240 -0.12755
## Transit          -0.12072  0.02194
## OtherTransp      -0.05940 -0.18388
## WorkAtHome       -0.33669 -0.24824
## MeanCommute       0.12373  0.17498
## Employed         -0.38048  0.24026
## PrivateWork       0.06164  0.47897
## SelfEmployed     -0.22020 -0.32650
## FamilyWork       -0.13062 -0.23017
## Minority          0.22175 -0.18324
```

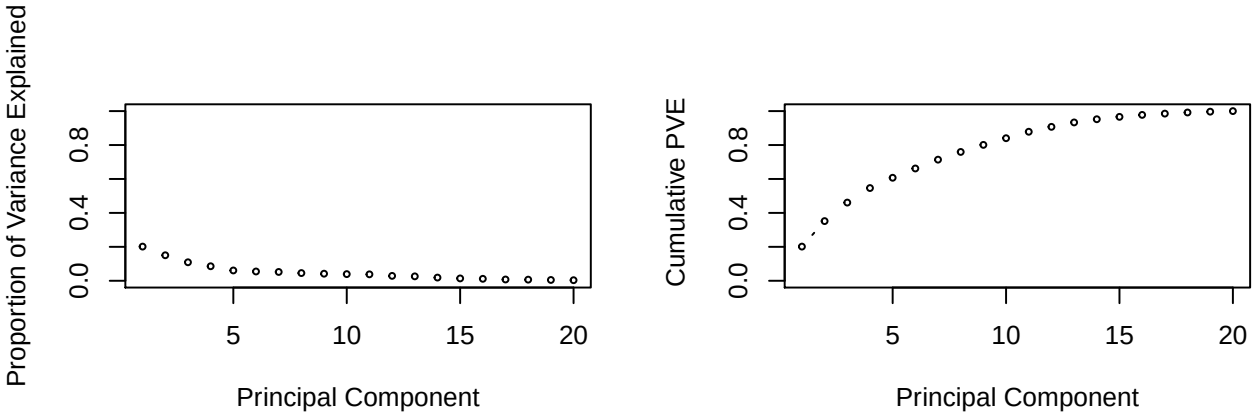
We center the data before performing PCA by centering the PCA scores at zero, zero meaning that the observation we are interested in has an average score in the PCA. We also scale the data to normalize it, since PCA is a variance maximizing exercise and thus normalizing data is important. The three features with the largest absolute values of the first principal component are ChildPoverty, Employed, and Income. The features CountryId, VotingAgeCitizen, Income, ChildPoverty, Professional, Service, Carpool, Transit, Employed, and Minority have opposite signs between PC1 and PC2, meaning that their correlations with PC1 and their correlations with PC2 are very different.

12. Determine the number of minimum number of PCs needed to capture 90% of the variance for the analysis.

```
#calculate PVE
pr.var = pr.out$sdev^2
pve = pr.var/sum(pr.var)
#calculate cumulative PVE
cum.pve = cumsum(pve)

#find how many PCs we need to explain 90% of total variation
needed.PCs = data.frame(index=1:as.numeric(length(cum.pve)),cumpve=cum.pve)
needed.PCs = subset(needed.PCs, cumpve >= 0.9)
head(needed.PCs, n=1)
```

```
##    index cumpve
## 12    12 0.9071
```

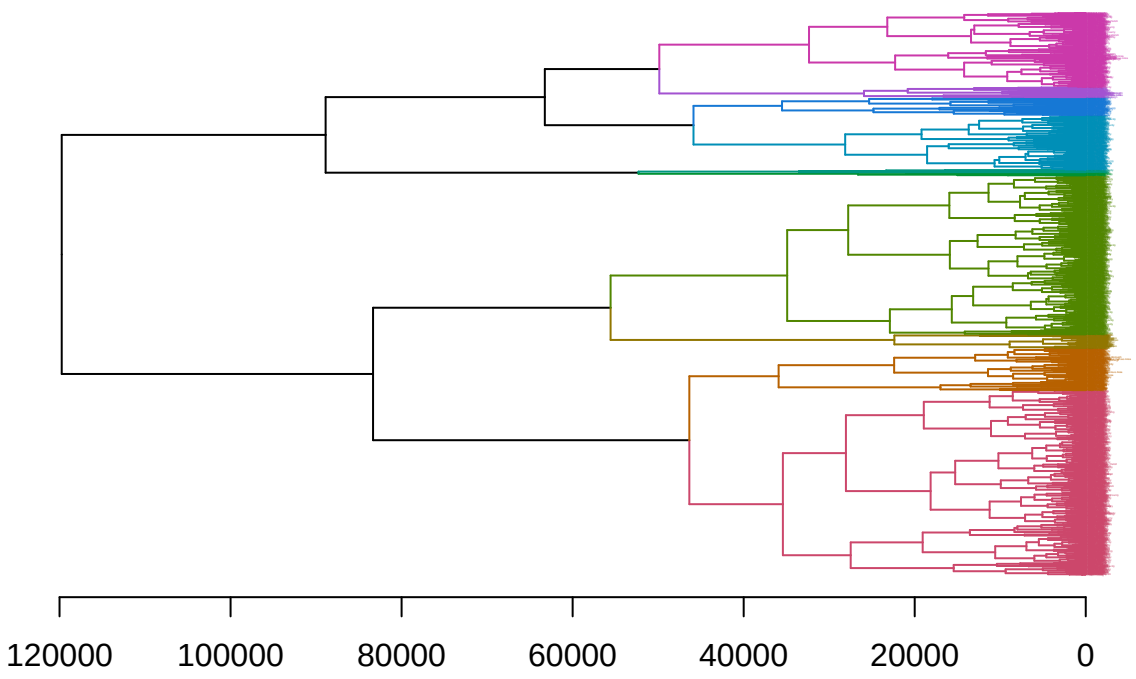
The minimum number of PCs needed to capture 90% of the variance for the analysis is 12.



Clustering

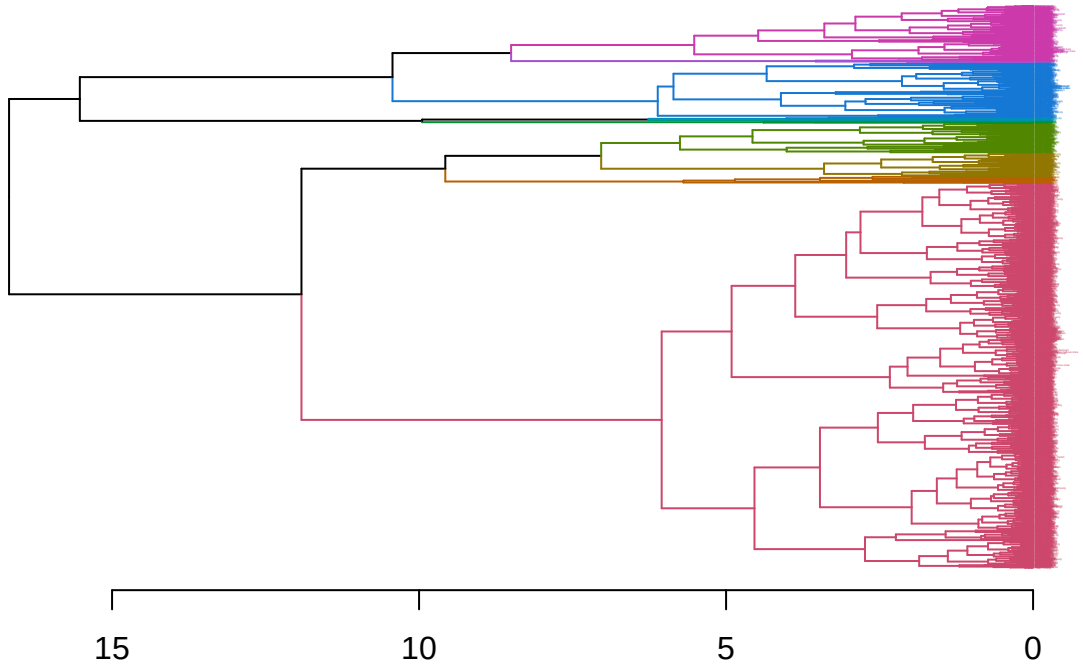
13. With census.clean (with State and County excluded), perform hierarchical clustering with complete linkage.

```
#compute euclidean distance matrix
EDM = dist(census.clean[-c(2,3)], method="euclidean")
#run hierarchical clustering using complete linkage
set.seed(20)
census.hclust = hclust(EDM, method="complete")
#assign County labels
census.hclust$labels = census.clean$County
#cut tree into 10 clusters
cut.census.hclust = cutree(census.hclust, k=10)
#plot first dendrogram
dend1 = as.dendrogram(census.hclust)
dend1 = color_branches(dend1, k=10)
dend1 = color_labels(dend1, k=10)
dend1 = set(dend1, "labels_cex", .1)
dend1 = set_labels(dend1, labels=census.clean$County[order.dendrogram(dend1)])
plot(dend1, horiz=T, main = "Census: Features as Inputs")
```




```
#compute euclidean distance matrix
EDM2 = dist(pc.county, method="euclidean")
#run hierarchical clustering using complete linkage
set.seed(20)
census.hclust2 = hclust(EDM2, method="complete")
#assign County labels
census.hclust2$labels = census.clean$County
#cut tree into 10 clusters
cut.census.hclust2 = cutree(census.hclust2, k=10)
#plot second dendrogram
dend2 = as.dendrogram(census.hclust2)
dend2 = color_branches(dend2, k=10)
dend2 = color_labels(dend2, k=10)
dend2 = set(dend2, "labels_cex", .1)
dend2 = set_labels(dend2, labels=census.clean$County[order.dendrogram(dend1)])
plot(dend2, horiz=T, main = "Census: First 2 PCs as Inputs")
```

Census: First 2 PCs as Inputs




```
#find cluster for Santa Barbary County in first hclust
subset(cut.census.hclust, names(cut.census.hclust)=="Santa Barbara County")
```

```
## Santa Barbara County
##                    2
```

```
#output cluster 2 in first hclust
head(split(names(cut.census.hclust),cut.census.hclust)$'2')
```

```
## [1] "Madison County"            "Aleutians East Borough"
## [3] "Dillingham Census Area"    "Hoonah-Angoon Census Area"
## [5] "Kenai Peninsula Borough"   "Ketchikan Gateway Borough"
```

```
#analyze first 6 counties in cluster 2
cluster2 = census.clean %>%
  filter(County == c("Madison County","Aleutians East Borough","Dillingham Census Area",
                     "Hoonah-Angoon Census Area", "Kenai Peninsula Borough",
                     "Ketchikan Gateway Borough", "Santa Barbara"))
cluster2
```

```
## # A tibble: 3 x 22
##   CountyId State County   Men VotingAgeCitizen Income ChildPoverty Professional
##      <dbl> <chr> <chr>  <dbl>            <dbl>  <dbl>        <dbl>        <dbl>
## 1     5087 Arka~ Madis~ 0.503            0.745  42894         22.6         29.3
## 2    13195 Geor~ Madis~ 0.502            0.748  47653         24.1         29.7
## 3    37115 Nort~ Madis~ 0.495            0.804  40563         24.1         31.1
## # ... with 14 more variables: 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>, Minority <dbl>
```

```
#find cluster for Santa Barbary County in second hclust
subset(cut.census.hclust2, names(cut.census.hclust)=="Santa Barbara County")
```

```
## Santa Barbara County
##                    4
```

```
#output cluster 4 in second hclust
head(split(names(cut.census.hclust),cut.census.hclust2)$'4')
```

```
## [1] "Aleutians East Borough"    "Bristol Bay Borough"
## [3] "Denali Borough"            "Dillingham Census Area"
## [5] "Haines Borough"            "Hoonah-Angoon Census Area"
```

```
cluster4 = census.clean %>%
  filter(County == c("Aleutians East Borough","Bristol Bay Borough",
                     "Denali Borough","Dillingham Census Area","Haines Borough",
                     "Hoonah-Angoon Census Area","Santa Barbara"))
cluster4
```

```
## # A tibble: 3 x 22
##   CountyId State County   Men VotingAgeCitizen Income ChildPoverty Professional
##      <dbl> <chr> <chr>  <dbl>            <dbl>  <dbl>        <dbl>        <dbl>
## 1     2060 Alas~ Brist~ 0.580            0.757  79500         11.6         30.5
## 2     2068 Alas~ Denal~ 0.507            0.785  83295          4.8         23.1
## 3     2070 Alas~ Dilli~ 0.520            0.682  58708         24.2         42.8
## # ... with 14 more variables: 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>, Minority <dbl>
```

From constructing dendrograms for both hierarchical clusterings, we see that while the hierarchical clustering using features as inputs has a fairly spread out distribution of clusters, the hierarchical clustering using the first two PCs as inputs has a very imbalanced and skewed distribution for its clusters, with the red cluster taking up a majority of the graph. Based on this extreme skewing in the hierarchical clustering using PCs, the hierarchical clustering using features seems to have placed Santa Barbara County in a more appropriate cluster.

Classification

We start considering supervised learning tasks now. The most interesting/important question to ask is: can we use census information in a county to predict the winner in that county?

In order to build classification models, we first need to combine county.winner and census.clean data. This seemingly straightforward task is harder than it sounds. For simplicity, the following code makes necessary changes to merge them into election.cl for classification.

```
# we move all state and county names into lower-case
tmpwinner <- county.winner %>% ungroup %>%
  mutate_at(vars(state, county), tolower)

# we move all state and county names into lower-case
# we further remove suffixes of "county" and "parish"
tmpcensus <- census.clean %>% mutate_at(vars(State, County), tolower) %>%
  mutate(County = gsub(" county|  parish", "", County))

# we join the two datasets
election.cl <- tmpwinner %>%
  left_join(tmpcensus, by = c("state"="State", "county"="County")) %>%
  na.omit

# drop levels of county winners if you haven't done so in previous parts
election.cl$candidate <- droplevels(election.cl$candidate)

## save meta information
election.meta <- election.cl %>% select(c(county, party, CountyId, state, votes, pct, total))

## save predictors and class labels
election.cl = election.cl %>% select(-c(county, party, CountyId, state, votes, pct, total))
```

14. Understand the code above. Why do we need to exclude the predictor party from election.cl?

We need to exclude the predictor party from election.cl due to the fact that the party variable only describes the party affiliation of the presidential candidates and, as such, does not serve as a demographic or predictive feature like the other predictors that remain in election.cl.

Using the following code, partition data into 80% training and 20% testing:

```
set.seed(10)
n <- nrow(election.cl)
idx.tr <- sample.int(n, 0.8*n)
election.tr <- election.cl[idx.tr, ]
election.te <- election.cl[-idx.tr, ]
```

Use the following code to define 10 cross-validation folds:

```
set.seed(20)
nfold <- 10
folds <- sample(cut(1:nrow(election.tr), breaks=nfold, labels=FALSE))
```

Use the following error rate function. And the object records is used to record the classification performance of each method in the subsequent problems.

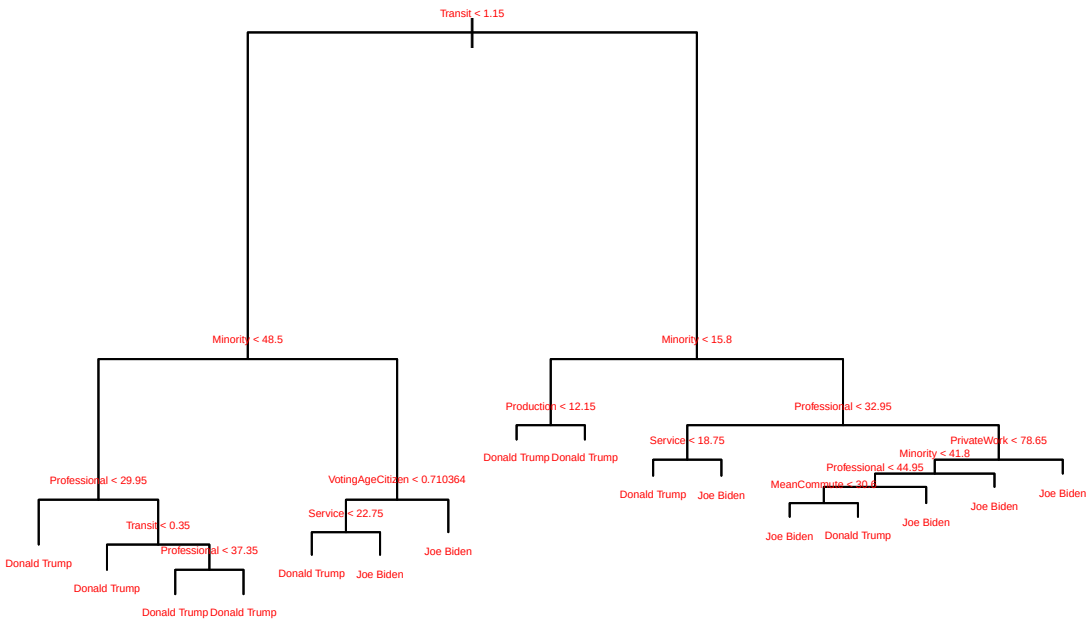
```
calc_error_rate = function(predicted.value, true.value){
  return(mean(true.value!=predicted.value))
}
records = matrix(NA, nrow=3, ncol=2)
```

```r
colnames(records) = c("train.error","test.error")
rownames(records) = c("tree","logistic","lasso")
```

Classification

15. Decision tree: train a decision tree by cv.tree().

```r
#construct single decision tree
tree.election = tree(candidate~.,data=election.tr)
plot(tree.election)
text(tree.election, pretty = 0, cex = 0.3, col = "red")
title("Decision Tree on election.tr", cex = 0.25)
```

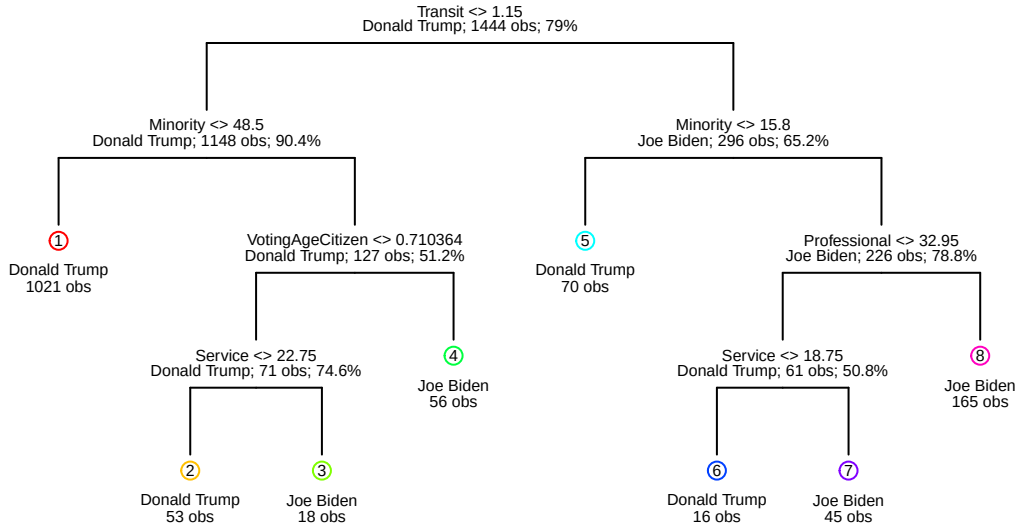



```r
#k-fold cross validation
cv = cv.tree(tree.election, FUN=prune.misclass, k=folds, rand=folds)
#best size
best_size = min(cv$size[cv$dev == min(cv$dev)])
best_size
```

```
## [1] 8
```

```r
#prune tree.
pt.cv = prune.misclass(tree.election, best=best_size)
#plot pruned tree
draw.tree(pt.cv, nodeinfo=TRUE, cex = 0.5)
title("Pruned Tree", cex = 0.25)
```

Pruned Tree




```
#prediction on training set
pred.tree.tr = predict(pt.cv, election.tr, type="class")

#prediction on test set
pred.tree.te = predict(pt.cv, election.te, type="class")

#calculate training and test error rates and add to records
tree.tr.error = calc_error_rate(pred.tree.tr, election.tr$candidate)
tree.te.error = calc_error_rate(pred.tree.te, election.te$candidate)
records[1, c("train.error", "test.error")] <- c(tree.tr.error, tree.te.error)
```

After pruning the decision tree to have 8 terminal nodes so that it results in the lowest misclassification error, we notice that the variable Transit is split first in this pruned decision tree, which is the same variable that was split first in the decision tree before pruning. Looking at the next splits, we see that once again both the original tree and the prune tree are split by the Minority variable. The first difference between the two trees occurs at this next split where the original decision tree is split at the Production and Professional variables, while the pruned tree is split at the VotingAgeCitizen and Professional variables. Beyond these first few splits the two trees differ quite significantly. In the pruned tree, however, the most important variables (in order) seem to be Transit, Minority, Professional, VotingAgeCitizen, and then Service. This signifies that there are notable differences in the voting preferences between counties in which a larger percentage of the population that use public transportation, are minorities, are in professional work fields, are able to vote, or are in service occupations than counties where these percentages are lower.

16. Run a logistic regression to predict the winning candidate in each county.

```
#perform logistic regression on election.tr
glm.fit = glm(candidate ~ ., data=election.tr, family=binomial)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
#summarize the logistic regression model
summary(glm.fit)
```

```
##
## Call:
## glm(formula = candidate ~ ., family = binomial, data = election.tr)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -3.801  -0.267  -0.104  -0.021   3.647
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -4.57e+01   8.29e+00   -5.51  3.6e-08 ***
## Men              -3.69e-01   6.16e+00   -0.06  0.95228
## VotingAgeCitizen  1.49e+01   2.87e+00    5.19  2.1e-07 ***
## Income           -1.79e-05   1.79e-05   -1.00  0.31736
## ChildPoverty      1.38e-02   2.11e-02    0.65  0.51326
## Professional      3.95e-01   4.73e-02    8.35  < 2e-16 ***
## Service           4.07e-01   5.98e-02    6.80  1.0e-11 ***
## Office            2.82e-01   6.04e-02    4.66  3.2e-06 ***
## Production        3.02e-01   5.24e-02    5.75  8.8e-09 ***
## Drive            -1.99e-01   5.41e-02   -3.69  0.00023 ***
## Carpool          -1.97e-01   7.06e-02   -2.79  0.00522 **
## Transit           1.71e-01   1.14e-01    1.51  0.13124
## OtherTransp       1.88e-01   1.10e-01    1.71  0.08638 .
## WorkAtHome       -8.25e-02   8.32e-02   -0.99  0.32159
## MeanCommute       4.91e-02   3.09e-02    1.59  0.11203
## Employed          2.08e+01   3.84e+00    5.42  5.9e-08 ***
## PrivateWork       7.06e-02   2.41e-02    2.93  0.00343 **
## SelfEmployed      1.97e-02   5.65e-02    0.35  0.72817
## FamilyWork       -1.84e-01   3.06e-01   -0.60  0.54728
## Minority          1.47e-01   1.26e-02   11.69  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1483.67  on 1443  degrees of freedom
## Residual deviance:  548.67  on 1424  degrees of freedom
## AIC: 588.7
##
## Number of Fisher Scoring iterations: 7
```

```r
#predictions
pred.glm.tr = predict(glm.fit, election.tr, type="response")
pred.glm.te = predict(glm.fit, election.te, type="response")
election.tr = election.tr %>%
  mutate(PREDcandidate=as.factor(ifelse(pred.glm.tr<=0.5, "Donald Trump", "Joe Biden")))
election.te = election.te %>%
  mutate(PREDcandidate=as.factor(ifelse(pred.glm.te<=0.5, "Donald Trump", "Joe Biden")))

#calculate training and test error rates and add to records
```

```
glm.tr.error = calc_error_rate(election.tr$PREDcandidate, election.tr$candidate)
glm.te.error = calc_error_rate(election.te$PREDcandidate, election.te$candidate)
records[2, c("train.error", "test.error")] <- c(glm.tr.error, glm.te.error)

#remove PREDcandidate from datasets
election.tr = election.tr %>%
  select(-PREDcandidate)
election.te = election.te %>%
  select(-PREDcandidate)
```

The significant variables in this logistic regression are VotingAgeCitizen, Professional, Service, Office, Production, Drive, Employed, PrivateWork, and Minority. These signficant variablesn are only somewhat consistent with our pruned decision tree results, since the logistic regression found Transit (the most significant variable in the pruned decision tree) to be insignificant. However, the rest of the variables that were found to be significant in the pruned decision tree (Minority, Professional, VotingAgeCitizen, and Service) were found to be significant in the logistic regression. Tn terms of a unit change in the variables, as the populations or percentages of Minority, Professional, VotingAgeCitizen, and Service increase by 1 unit, the effect of the variable increases by 0.147, 0.395, 10.49, and 0.407, respectively.

17. You may notice that you get a warning glm.fit: fitted probabilities numerically 0 or 1 occurred.

```
#perform CV to choose best lambda from list
set.seed(20)
lasso.tr = as.matrix(election.tr %>% select(-candidate))
cv.out.lasso = cv.glmnet(lasso.tr, election.tr$candidate, alpha=1,
                         lambda=seq(1,50)*1e-4, family=binomial)

#determine optimal value of lambda
bestlam = cv.out.lasso$lambda.min
bestlam
```

```
## [1] 6e-04
```

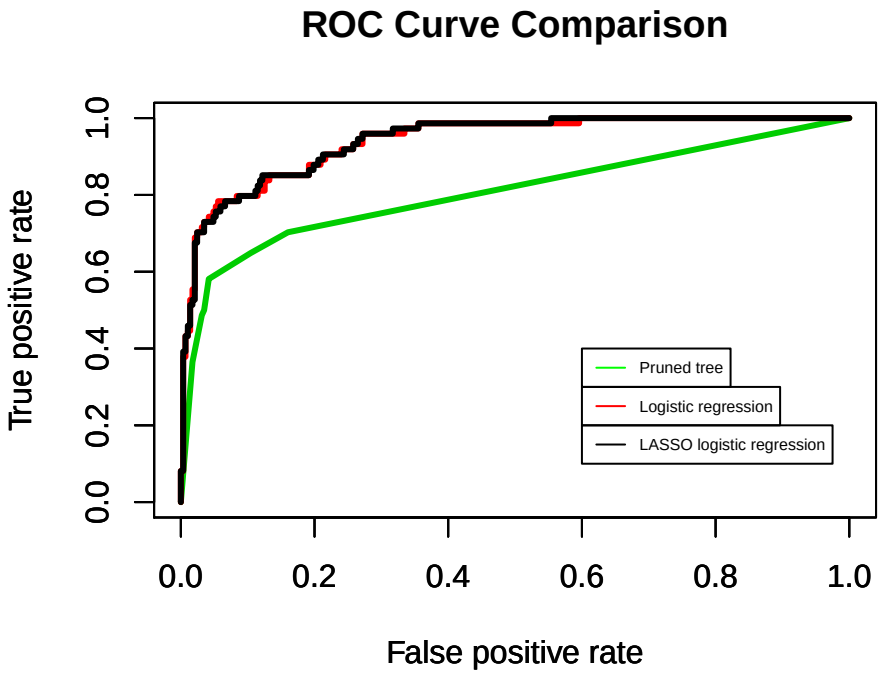
```
#find lasso coefficient estimates using best lambda
lasso.cl = as.matrix(election.cl %>% select(-candidate))
out=glmnet(lasso.cl, election.cl$candidate, alpha=1, family=binomial)
lasso.coef=predict(out,type="coefficients",s=bestlam)[1:20,]
lasso.coef
```

```
##      (Intercept)              Men VotingAgeCitizen           Income
##       -4.305e+01       -3.943e+00        1.519e+01       -5.157e-06
##     ChildPoverty     Professional          Service           Office
##        2.118e-02        3.472e-01        3.728e-01        2.343e-01
##       Production            Drive          Carpool          Transit
##        2.379e-01       -1.446e-01       -1.215e-01        7.075e-02
##      OtherTransp       WorkAtHome      MeanCommute         Employed
##        1.990e-01        0.000e+00        2.658e-02        1.932e+01
##      PrivateWork     SelfEmployed       FamilyWork         Minority
##        5.581e-02       -1.722e-02       -2.626e-01        1.329e-01
```

```r
#lasso model
lasso.mod = glmnet(lasso.tr, election.tr$candidate, alpha=1,
                         lambda=seq(1,50)*1e-4, family=binomial)
lasso.te = as.matrix(election.te %>% select(-candidate))

#predictions
pred.lasso.tr = predict(lasso.mod, s=bestlam, newx=lasso.tr)
pred.lasso.te = predict(lasso.mod, s=bestlam, newx=lasso.te)
election.tr = election.tr %>%
  mutate(PREDcandidate=as.factor(ifelse(pred.lasso.tr<=0.5, "Donald Trump", "Joe Biden")))
election.te = election.te %>%
  mutate(PREDcandidate=as.factor(ifelse(pred.lasso.te<=0.5, "Donald Trump", "Joe Biden")))

#calculate training and test error rates and add to records
lasso.tr.error = calc_error_rate(election.tr$PREDcandidate, election.tr$candidate)
lasso.te.error = calc_error_rate(election.te$PREDcandidate, election.te$candidate)
records[3, c("train.error", "test.error")] <- c(lasso.tr.error, lasso.te.error)

#remove PREDcandidate from datasets
election.tr = election.tr %>%
  select(-PREDcandidate)
election.te = election.te %>%
  select(-PREDcandidate)
```

The optimal value of λ in cross validation is 0.0006. The non-zero coefficients in the LASSO regression for the optimal value of λ are Men, VotingAgeCitizen, Income, ChildPoverty, Professional, Service, Office, Production, Drive, Carpool, Transit, OtherTransp, MeanCommute, Employed, PrivateWork, SelfEmployed, FamilyWork, and Minority. In other words, the LASSO regression only considers the WorkAtHome variable to be completely insignificant. The rest of the coefficients of the LASSO regression are fairly similar to the unpenalized logistic regression, with, for example, the coefficients of Minority, Professional, VotingAgeCitizen, and Service for the LASSO regression (0.1329, 0.3472, 15.19, and 0.3728) being pretty close to the coefficients for the same variables in the unpenalized logistic regression (0.147, 0.395, 10.49, and 0.407). The biggest difference between these two models is, of course, the fact that the LASSO regression effectively removes the WorkAtHome variable after performing its feature selection.

18. Compute ROC curves for the decision tree, logistic regression and LASSO logistic regression using predictions on the test data.




```
#best pruned decision tree AUC
pt.auc = performance(pt.pred, "auc")@y.values
pt.auc
```

```
## [[1]]
## [1] 0.8057
```

```
#logistic regression fit AUC
glm.auc = performance(glm.pred, "auc")@y.values
glm.auc
```

```
## [[1]]
## [1] 0.9399
```

```
#LASSO logistic regression fit AUC
lasso.auc = performance(lasso.pred, "auc")@y.values
lasso.auc
```

```
## [[1]]
## [1] 0.9406
```

```
#output records for training and test errors
records
```

```
##          train.error test.error
## tree         0.08241    0.11911
## logistic     0.06994    0.08310
## lasso        0.08172    0.09972
```

From the plot comparing the ROC curves and the AUC values for each approach, we see that both the unpenalized logistic regression fit and LASSO regression fit perform significantly better than the pruned decision tree fit. Between the unpenalized logistic regression fit and LASSO logistic regression fit, however, there is barely any noticeable difference besides the AUC value of the lasso regression fit being higher by 0.0007. In the end, it seems that, out of the three approaches, the LASSO logistic fit performs the best in terms of minimizing classification error. The LASSO logistic regression fit also controls for the overfitting issue that is experienced by the unpenalized logistic regression fit, so the LASSO logistic regression fit is usually going to be the better choice out of the two. However, although it seems to perform the worst, the pruned tree may have its uses in answering questions that the logistic regression fits cannot, such as when we want to analyze parts of the election data that are not linearly seperable, since logistic regression typically requires data to be linearly separable and is usually used for data with binary outputs. The training and test errors seem reasonable with the logistic regression fit training and test errors probably being the lowest due to overfitting.

Taking it further

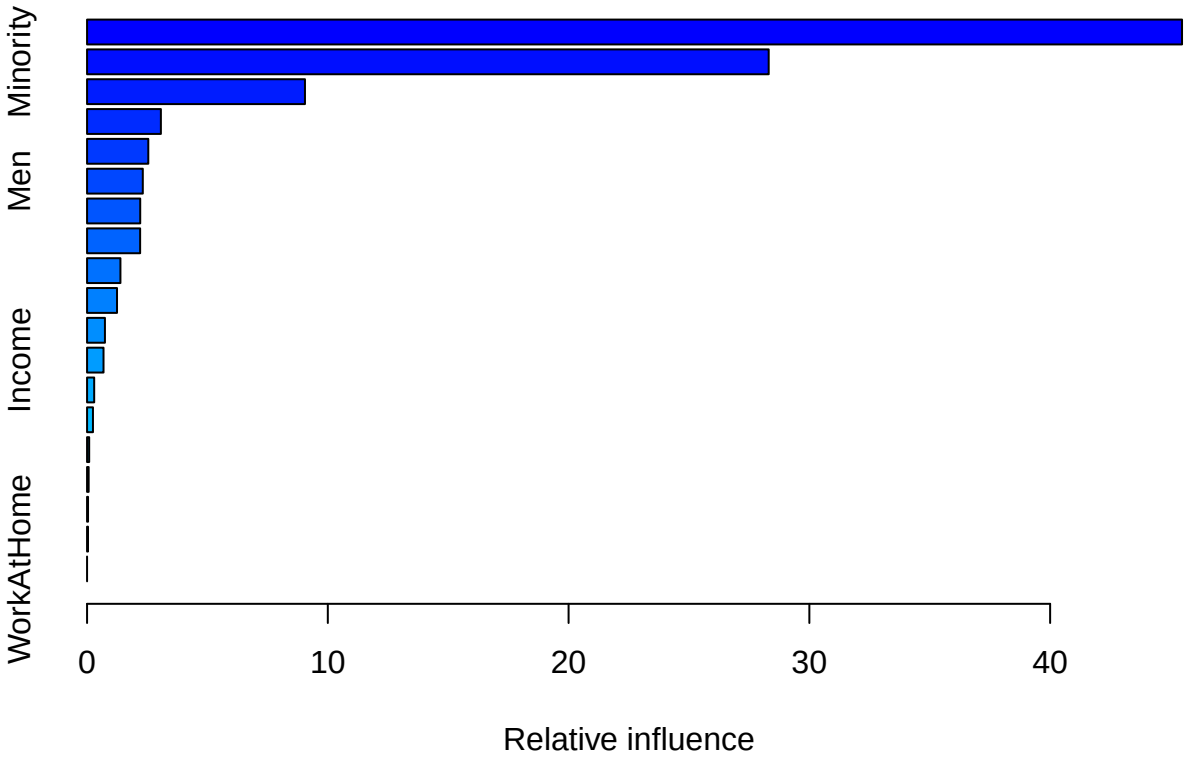
19. Explore additional classification methods.

For our exploration of additional classification methods, we will fit a boosting model and a random forest model to our election training set and compare these methods to the methods performed above.

Boosting Model:

For our first additional classification method we will fit a 1,000 tree boosted model with a shrinkage value of 0.01:

```
set.seed(20)
#fit boosting model to training set
boost = gbm(ifelse(candidate=="Joe Biden",1,0)~.,
                  data=election.tr,
                  distribution="bernoulli",
                  n.trees=1000,
                  shrinkage=0.01)
summary(boost)
```




```
##                                  var  rel.inf
## Transit                      Transit 45.47814
## Minority                    Minority 28.31031
## Professional           Professional  9.05007
## ChildPoverty           ChildPoverty  3.06596
## Employed                   Employed  2.54244
## Men                             Men  2.31466
## Production               Production  2.20699
## SelfEmployed           SelfEmployed  2.20448
## OtherTransp             OtherTransp  1.38517
## Service                     Service  1.24434
## VotingAgeCitizen VotingAgeCitizen  0.74219
## Income                       Income  0.68326
## PrivateWork             PrivateWork  0.29585
## Office                       Office  0.24462
## Carpool                     Carpool  0.08746
## MeanCommute             MeanCommute  0.06548
## FamilyWork               FamilyWork  0.03998
## Drive                         Drive  0.03860
## WorkAtHome               WorkAtHome  0.00000
```

```r
#predictions
pred.boost.tr = predict(boost, election.tr, type="response")
pred.boost.te = predict(boost, election.te, type="response")
election.tr = election.tr %>%
  mutate(PREDcandidate=as.factor(ifelse(pred.boost.tr<=0.5, "Donald Trump", "Joe Biden")))
election.te = election.te %>%
  mutate(PREDcandidate=as.factor(ifelse(pred.boost.te<=0.5, "Donald Trump", "Joe Biden")))
```

```
#calculate training and test error rates
boost.tr.error = calc_error_rate(election.tr$PREDcandidate, election.tr$candidate)
boost.te.error = calc_error_rate(election.te$PREDcandidate, election.te$candidate)
#create records2
records2 = matrix(NA, nrow=2, ncol=2)
colnames(records2) = c("train.error","test.error")
rownames(records2) = c("boosting","r.forest")
#add training and test error to records 2
records2[1, c("train.error", "test.error")] <- c(boost.tr.error, boost.te.error)

#remove PREDcandidate from datasets
election.tr = election.tr %>%
  select(-PREDcandidate)
election.te = election.te %>%
  select(-PREDcandidate)
```

Based on the boosting model the Transit variable seems to be the most important predictor by far with only Minority coming somewhat close. Besides the Transit and Minority variables that are very significant, however, the predictor Professional also seems to be at least somewhat noteworthy. This boosting model is more consistent with the pruned tree fit due to the fact that both of these approaches considered Transit, Minority, and Professional to be the most signficant in the exact same order. We will compare the boosting model ROC curve, AUC values, and training/test errors with the other approaches in a later step.

Random Forest Model:

Next, for our second additional classification, we will fit a random forest model:

```
set.seed(20)
#fit random forest model
rf = randomForest(candidate ~ ., data=election.tr,importance=TRUE)
rf
```

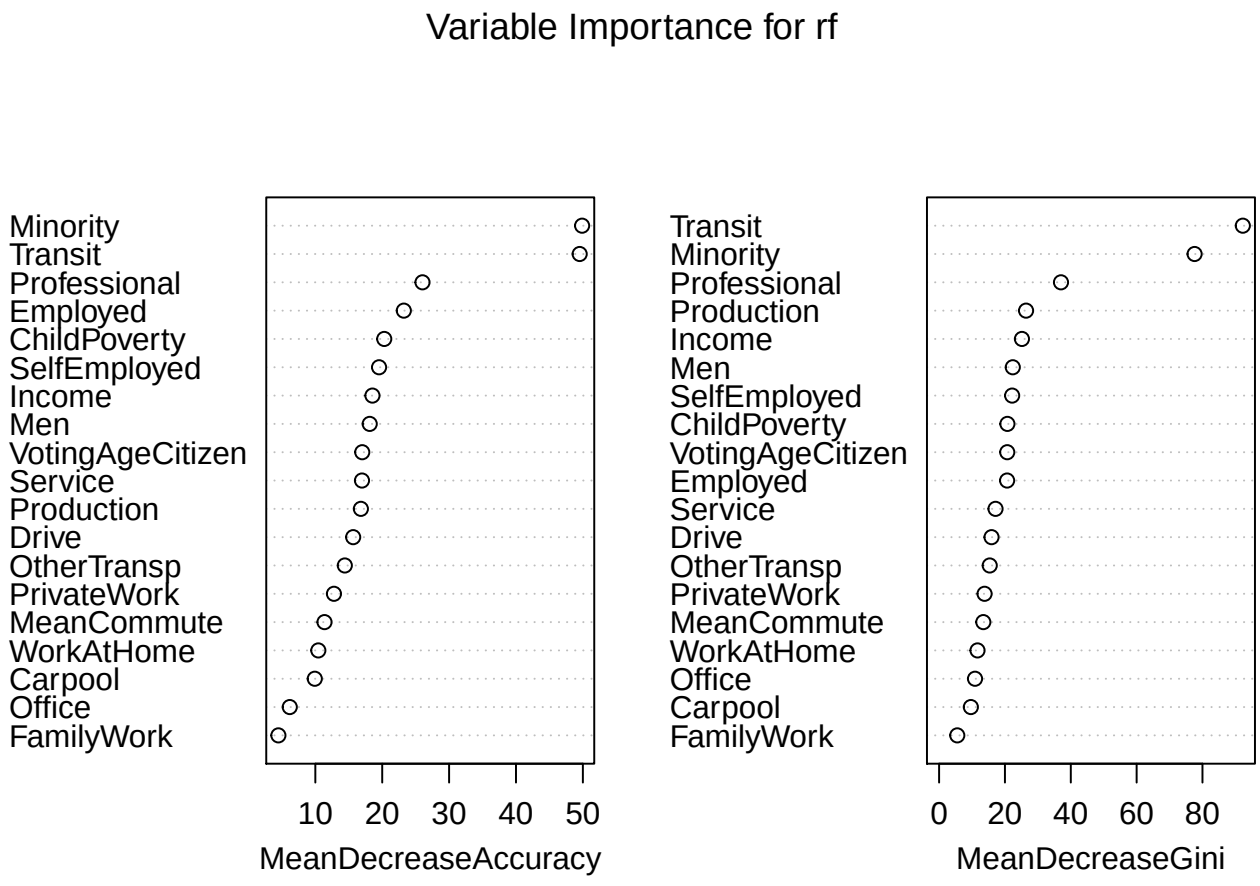
```
##
## Call:
##  randomForest(formula = candidate ~ ., data = election.tr, importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 7.62%
## Confusion matrix:
##              Donald Trump Joe Biden class.error
## Donald Trump         1109        32     0.02805
## Joe Biden              78       225     0.25743
```

```
#look at variable importance
importance(rf)
```

```
##                   Donald Trump Joe Biden MeanDecreaseAccuracy MeanDecreaseGini
## Men                      7.965    17.195               18.130           22.377
## VotingAgeCitizen        13.237     9.847               17.011           20.700
```

```
## Income              11.957    12.610    18.537    25.159
## ChildPoverty        12.087    15.455    20.302    20.746
## Professional        16.972    22.415    26.020    37.022
## Service             13.553    10.142    16.973    17.141
## Office               1.507     7.183     6.177    10.906
## Production           6.605    15.339    16.810    26.413
## Drive                9.719    13.036    15.656    15.929
## Carpool              7.592     5.880     9.917     9.645
## Transit             25.266    43.431    49.522    92.258
## OtherTransp          3.025    17.840    14.406    15.380
## WorkAtHome           6.358     7.531    10.448    11.669
## MeanCommute         10.300     5.821    11.370    13.427
## Employed            11.451    18.163    23.228    20.658
## PrivateWork          9.965     8.733    12.783    13.807
## SelfEmployed        13.851    15.357    19.529    22.187
## FamilyWork          -2.584     8.252     4.478     5.519
## Minority            28.613    51.125    49.884    77.610
```

```r
#plot variable importance
varImpPlot(rf, sort=T, main="Variable Importance for rf")
```



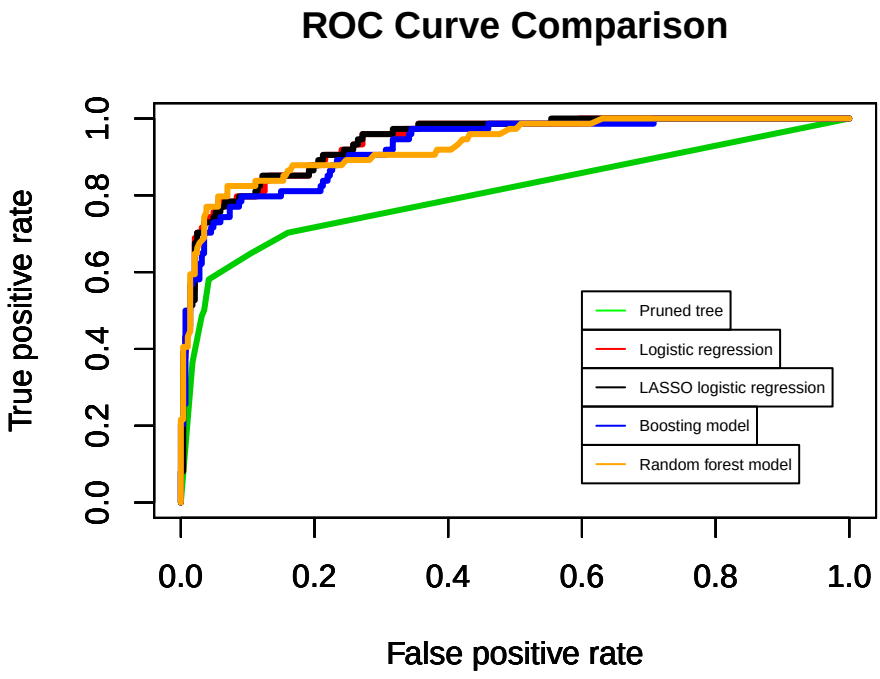

```r
#predictions
pred.rf.tr = predict(rf, election.tr ,type="prob")[,"Joe Biden"]
pred.rf.te = predict(rf, election.te ,type="prob")[,"Joe Biden"]
```

```
election.tr = election.tr %>%
  mutate(PREDcandidate=as.factor(ifelse(pred.rf.tr<=0.5, "Donald Trump", "Joe Biden")))
election.te = election.te %>%
  mutate(PREDcandidate=as.factor(ifelse(pred.rf.te<=0.5, "Donald Trump", "Joe Biden")))

#calculate training and test error rates and add to records
rf.tr.error = calc_error_rate(election.tr$PREDcandidate, election.tr$candidate)
rf.te.error = calc_error_rate(election.te$PREDcandidate, election.te$candidate)
records2[2, c("train.error", "test.error")] <- c(rf.tr.error, rf.te.error)

#remove PREDcandidate from datasets
election.tr = election.tr %>%
  select(-PREDcandidate)
election.te = election.te %>%
  select(-PREDcandidate)
```

Some details we notice about our random forest model are that the out-of-bag estimate of error is 7.62%, that 4 variables were randomly considered at each split in the trees, and 500 trees were used to fit the data.

Based on the Model Accuracy plot of variable importance, the order of important variables is fairly similar for the boosting model and random forest model with Minority and Transit being the top two most significant variables for both models. Professional being the third most significant variable in the random forest Model Accuracy plot is also consistent with the boosting model. Based on the Gini index plot of variable importance, the order of important variables also seems to be even more similar for the boosting and random forest models since Transit is the most important predictor, followed by Minority and Professional in both models.

Now we proceed to comparing the ROC curves, AUC values, and training/test errors between each approach we have considered so far.




```
#best pruned decision tree AUC
pt.auc = performance(pt.pred, "auc")@y.values
pt.auc
```

```
## [[1]]
## [1] 0.8057
```

```
#logistic regression fit AUC
glm.auc = performance(glm.pred, "auc")@y.values
glm.auc
```

```
## [[1]]
## [1] 0.9399
```

```
#LASSO logistic regression fit AUC
lasso.auc = performance(lasso.pred, "auc")@y.values
lasso.auc
```

```
## [[1]]
## [1] 0.9406
```

```
#boosting model fit AUC
boost.auc = performance(boost.pred, "auc")@y.values
boost.auc
```

```
## [[1]]
## [1] 0.9256
```

```
#random forest model fit AUC
rf.auc = performance(rf.pred, "auc")@y.values
rf.auc
```

```
## [[1]]
## [1] 0.9293
```

```
#create totalrecords and output for training and test errors
totalrecords = rbind(records, records2)
totalrecords
```

```
##          train.error test.error
## tree         0.08241    0.11911
## logistic     0.06994    0.08310
## lasso        0.08172    0.09972
## boosting     0.08518    0.11357
## r.forest     0.00000    0.09418
```

From the plot comparing the ROC curves and the AUC values for each approach, we see that while the boosting model seems to perform slightly better than the random tree model based on the ROC curves, the boosting model seems to perform slightly worse than the random tree model based on the boosting model having a lower AUC value than the random forest model. We also see that the boosting model and random forest model do not perform as well as the penalized logistic regression or LASSO logistic regression models. However, they do perform better than the pruned decision tree model. This means that, out of the five approaches we have considered, the LASSO logistic fit still performs the best in terms of minimizing classification error. Looking at the training and test errors we see that all but one of the errors seems reasonable due to the training error for the random forest model being 0, which seems to suggest that the random forest model is experiencing overfitting. In the end, we conclude that the LASSO logistic regression fit seems to be the best overall model for classification.

20. Tackle at least one more interesting question. Creative and thoughtful analysis will be rewarded!

Would any of the models we have fitted be as effective on a dataset that combines state.winner and census.clean, instead of county.winner and census.clean?

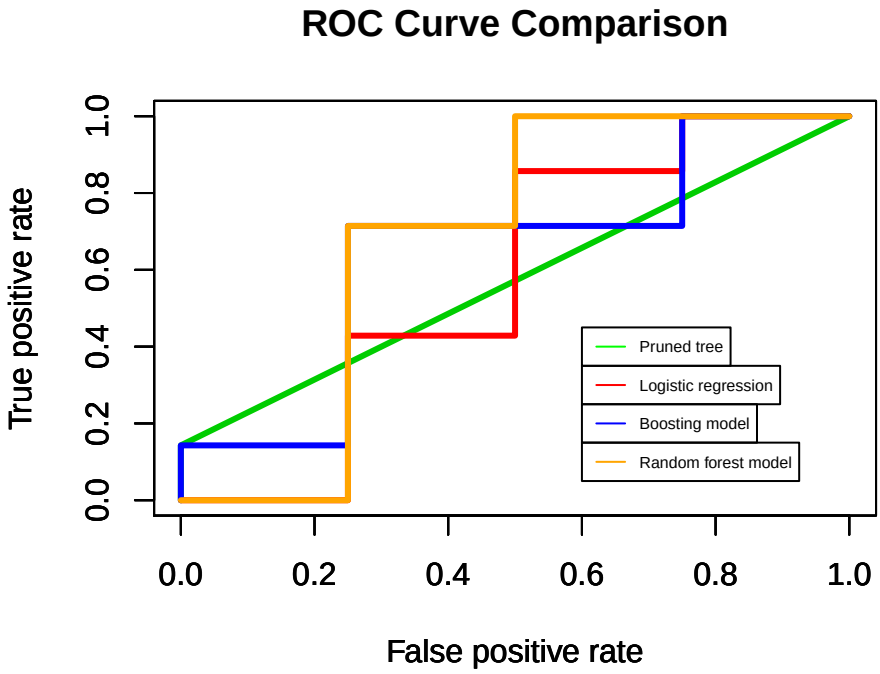
An topic of interest if any of the models we have fitted using county-level data would be able to adequately perform on a state-level of our election data to predict the winner of that state using census data that has been altered to be state-level, since we created these models under larger number of data values that make the predictability transferable to a smaller dataset. To determine whether any of our models behave similarly on both county-level and state-level data, we begin by creating the dataset that will be predicted on: election.cl2.

```
#move all state names into lower-case
tmpwinner <- state.winner %>% ungroup %>%
  mutate_at(vars(state), tolower)
#move all state names into lower-case, remove County
tmpcensus <- census.clean %>% mutate_at(vars(State, County), tolower) %>%
  select(-County)
#join the two datasets
election.cl2 <- tmpwinner %>%
  left_join(tmpcensus, by = c("state"="State")) %>%
  na.omit
#drop levels of state winners if you haven't done so in previous parts
election.cl2$candidate <- droplevels(election.cl2$candidate)
#save meta information
election.meta2 <- election.cl2 %>% select(c(CountyId, votes, pct, total))
#save predictors and class labels
election.cl2 = election.cl2 %>% select(-c(CountyId, votes, pct, total))
#group by state,candidate and update variables accordingly
election.cl2 = election.cl2 %>%
  group_by(state,candidate) %>%
  summarise(Men = mean(Men),
            VotingAgeCitizen = mean(VotingAgeCitizen),
            Income = mean(Income),
            ChildPoverty = mean(ChildPoverty),
            Professional = mean(Professional),
            Service = mean(Service),
            Office = mean(Office),
            Production = mean(Production),
            Drive = mean(Drive),
            Carpool = mean(Carpool),
            Transit = mean(Transit),
            OtherTransp = mean(OtherTransp),
            WorkAtHome = mean(WorkAtHome),
            MeanCommute = mean(MeanCommute),
            Employed = mean(Employed),
            PrivateWork = mean(PrivateWork),
            SelfEmployed = mean(SelfEmployed),
            FamilyWork = mean(FamilyWork),
            Minority = mean(Minority))
```

After creating our dataset, we partition the data into 80% training and 20% testing:

```
set.seed(10)
n <- nrow(election.cl2)
idx.tr <- sample.int(n, 0.8*n)
election2.tr <- election.cl2[idx.tr, ]
election2.te <- election.cl2[-idx.tr, ]
```

We then move on to the prediction testing using our already existing models, since we simply want to check how effective our county-level models will be on the state-level data. To accomplish this, we will be looking at the ROC curves and AUC values.




```
#best pruned decision tree AUC
pt.auc = performance(pt.pred, "auc")@y.values
pt.auc
```

```
## [[1]]
## [1] 0.5714
```

```
#logistic regression fit AUC
glm.auc = performance(glm.pred, "auc")@y.values
glm.auc
```

```
## [[1]]
## [1] 0.5714
```

```
#boosting model fit AUC
boost.auc = performance(boost.pred, "auc")@y.values
boost.auc
```

```
## [[1]]
## [1] 0.6429
```

```r
#random forest model fit AUC
rf.auc = performance(rf.pred, "auc")@y.values
rf.auc
```

```
## [[1]]
## [1] 0.6786
```

Since the grouping variable of state for the dataset we created is irremovable, testing the predictability of the LASSO logistic regression fit was not possible and was thus not tested. Moving on, looking at the resulting ROC curves and AUC values, it is safe to say that the classification models fitted on county-level data are not adequate in predicting the state-winning candidates based on state-level census data. The best model based on both the ROC curves and AUC values was the random forest model with an AUC value of 0.6786. However, although it is the "best," this model is obviously an unsatisfactory classification model. Thus, we conclude that none of the models we have fitted using county-level data is able to adequately perform on a state-level of our election data to predict the winning candidate.

21. (Open ended) Interpret and discuss any overall insights gained in this analysis and possible explanations.

This analysis of the 2020 US Presidential Election has generated some key insights regarding factors that influence the overall voting tendencies of populations at a county-level or state-level and regarding how well different approaches to supervised learning and classification techniques can be used to predict the presidential candidate winner in a county based on census information. We learned that the factors that were the most influential overall throughout each of the five fitted models were the Transit, Minority, Professional, VotingAgeCitizen, and Service predictors. We also determined that it seems thatthe LASSO logistic fit performs the best in terms of minimizing classification error, especially since it also controls for the overfitting issue that may be experienced by other models. We also concluded that, although it seems to perform the worst, the pruned tree may be useful when we want to analyze parts of the election data that are not linearly seperable, since logistic regression typically requires data to be linearly separable and is usually used for data with binary outputs. Lastly, although these county-level models were fitted on a dataset that had much more data, these models did not perform well when we downscaled our data to state-level, indicating that the predictability of county-level winning candidates is not transferable to the prediction of state-level winning candidates.