PSTAT 131 Final

Sam Leathers: 6630495

11/30/2020

Abstract:

Many Statistical Machine Learning models seek to use readily available data to predict an outcome. An example of this which we will explore will be predicting election results, a notoriously difficult thing to accomplish. We will be using census data and election data in order to create predictive models for the result of the 2020 election. We will explore various methods for classification and model building.

Introduction:

We will first begin by importing our data and assessing various metrics. Note that our data has been imported as census, for the most recent census data, and election.raw for the most recently available voting data as of November 2020. We will then clean this data and build models attempting to classify which state or county would vote in what ways.

Election Data

1-2 data exploration

```
## Election Data
# dimensions of our data
dim(election.raw) # rows: 31167, columns: 5
```

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

```
# completeness of data
sum(is.na.data.frame(election.raw)==TRUE) # 0 NA in data
```

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

```
## Census Data
# dimensions
dim(census)# rows: 3220 , columns: 37
```

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

```
# completeness
sum(is.na.data.frame(census) == TRUE) # 1 NA in data
```

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

```r
# Total number of distinct states
length(unique(election.raw$state)) # 51, 50 states 1 Federal district (DC)
```

```
## [1] 51
```

```r
length(unique(census$State)) # 52, 50 states, DC, Puerto Rico (territory)
```

```
## [1] 52
```

```r
# compare our unique counties in the census to our unique counties in election.raw
length(unique(census$County)) # 1955| Note counties can share names from other states
```

```
## [1] 1955
```

```r
length(unique(election.raw$county)) # 2825
```

```
## [1] 2825
```

```r
length(unique(census$CountyId)) # 3220 ( All counties Unique in census)
```

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

```r
table(census$State)[40]
```

```
## Puerto Rico
##          78
```

For the sake of continuity it is important to note that our census data includes an extra territory, and has extra counties. Two reasons for this are the existence of the extra territory and the fact that not all counties had reported their election outcomes at the time our data was pooled. From the table of our state values we can see that Puerto Rico accoutned for onlly 78 counties and does not fully explain the 395 county discrepancy.

Data wrangling:

3

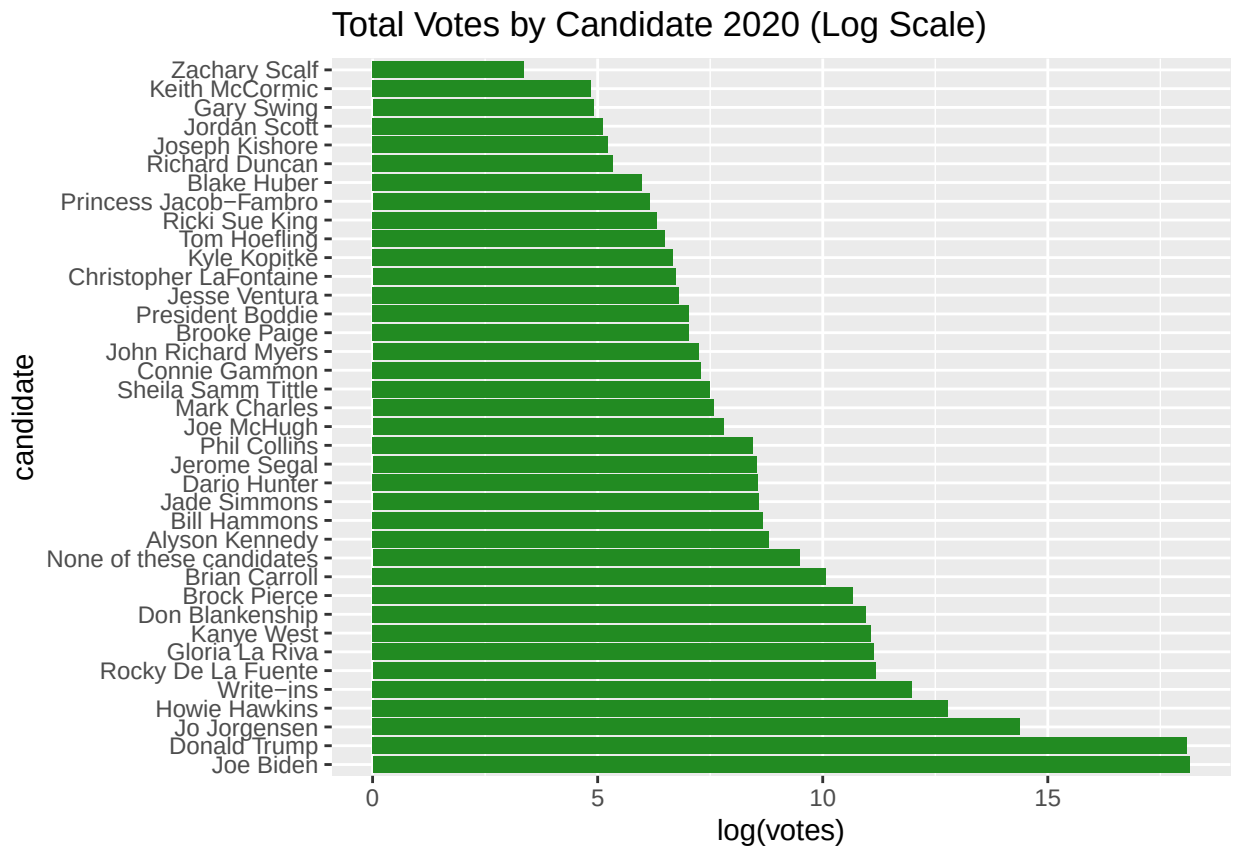
```r
# 3: aggregate Data: State and federal summary

# state summary: ordered by State
election.state <- aggregate(votes~candidate + state, election.raw, sum)


# Federal summary
election.total <- aggregate(votes~candidate + party, election.raw, sum)
# sort by highest number of votes
election.total <- election.total[order(-election.total$votes),]
```

4]

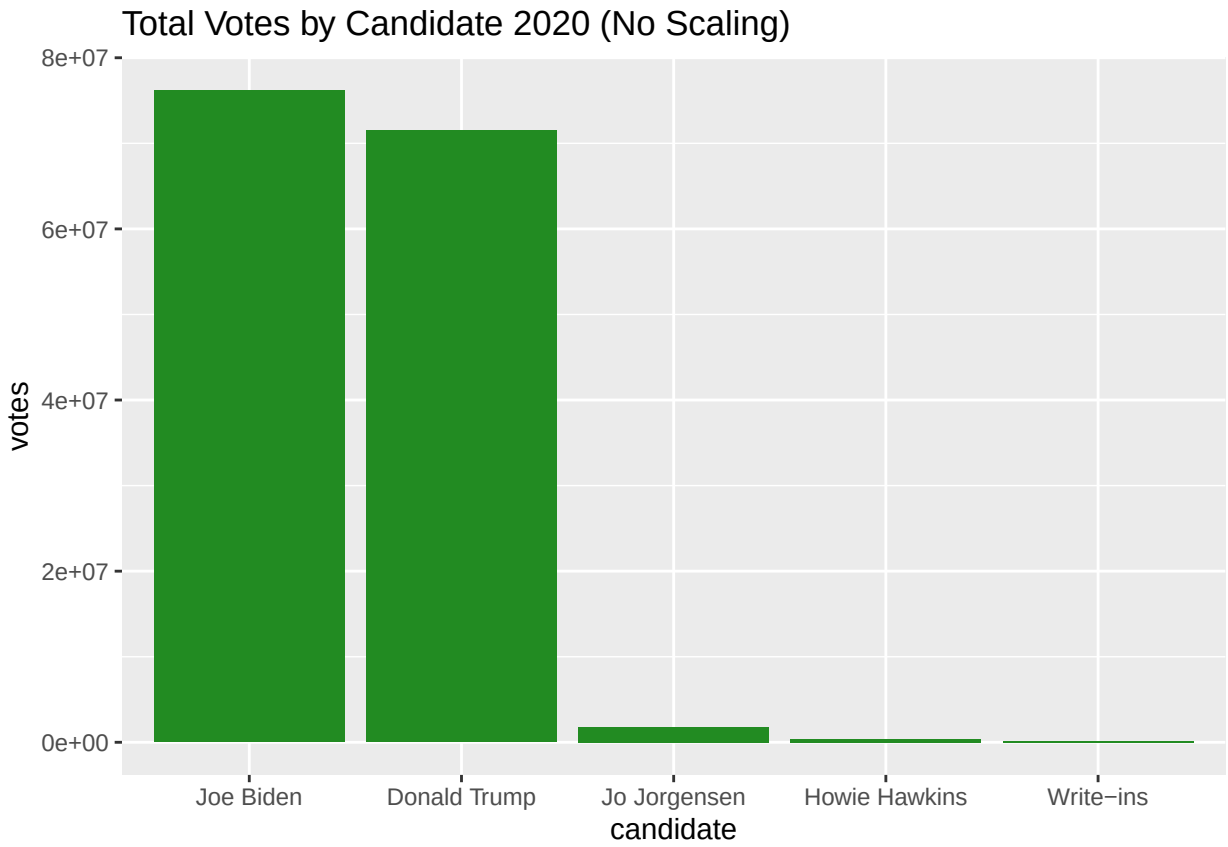
```r
# Number of named candidates
length(unique(election.total$candidate)) # Note there are no duplicates in election.total
```

```
## [1] 38
```

```r
# 38 candidates
# 26 parties

# bar chart of votes by candidate
ggplot(data = election.total, aes(y = log(votes), x = candidate)) +
  geom_bar(stat = 'identity', fill = 'forest green') + coord_flip() +
  scale_x_discrete(limits = election.total$candidate) +
  ggtitle('Total Votes by Candidate 2020 (Log Scale)')
```




```r
# bar chart of top 5 by votes
top5 <- election.total[1:5,]
ggplot(data = top5, aes(y = votes, x = candidate)) +
  geom_bar(stat = 'identity', fill = 'forest green') +
  scale_x_discrete(limits = top5$candidate) +
  ggtitle('Total Votes by Candidate 2020 (No Scaling)')
```



Thus we can see there were 38 candidates and 26 parties represented on the ballot in 2020. The race was largely between Joe Biden and Donald Trump as seen by the disparity in votes between every other candidate highlighted in the barchart with no scaling.

5 data sets for county and state winners

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

# takes raw data set,  groups each column by the county values, creates 2 new columns with mutate where


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

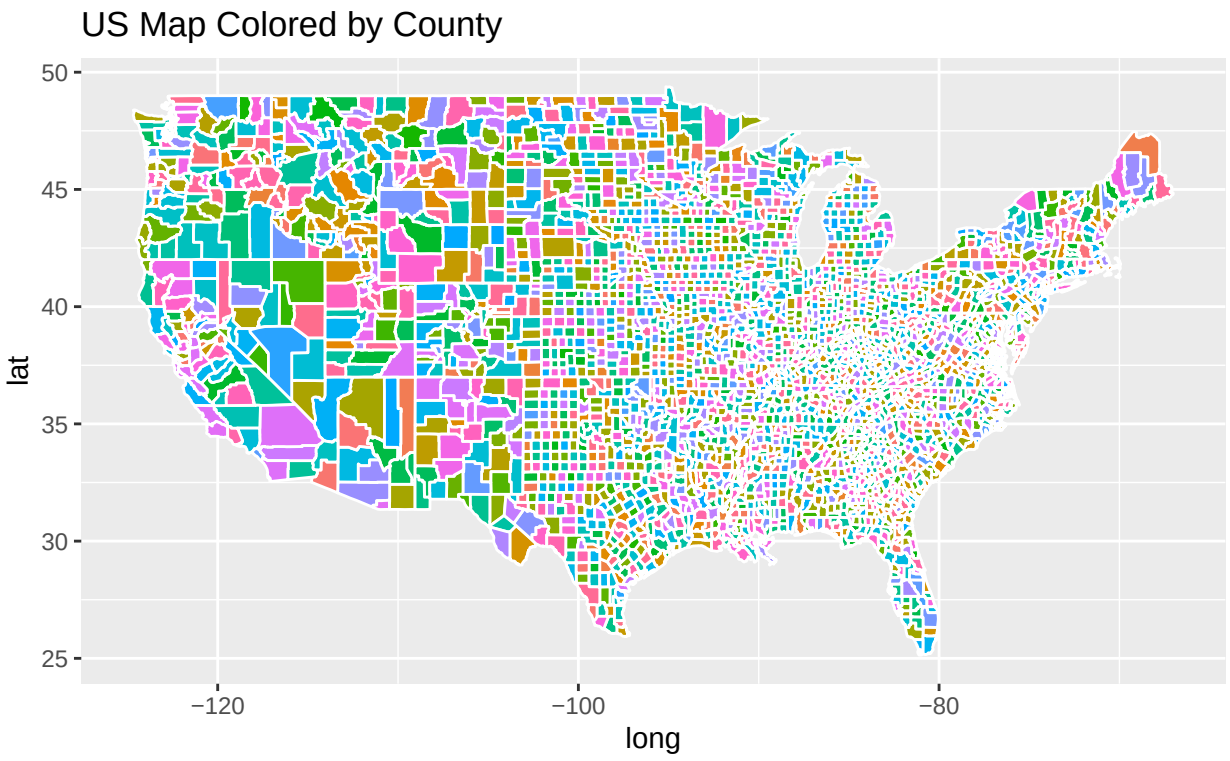
6 - Visualization

```
# Library for US map
library('maps')
```

```
## Warning: package 'maps' was built under R version 3.6.3
```

```
##
## Attaching package: 'maps'

## The following object is masked from 'package:cluster':
##
##     votes.repub

## The following object is masked from 'package:purrr':
##
##     map
```

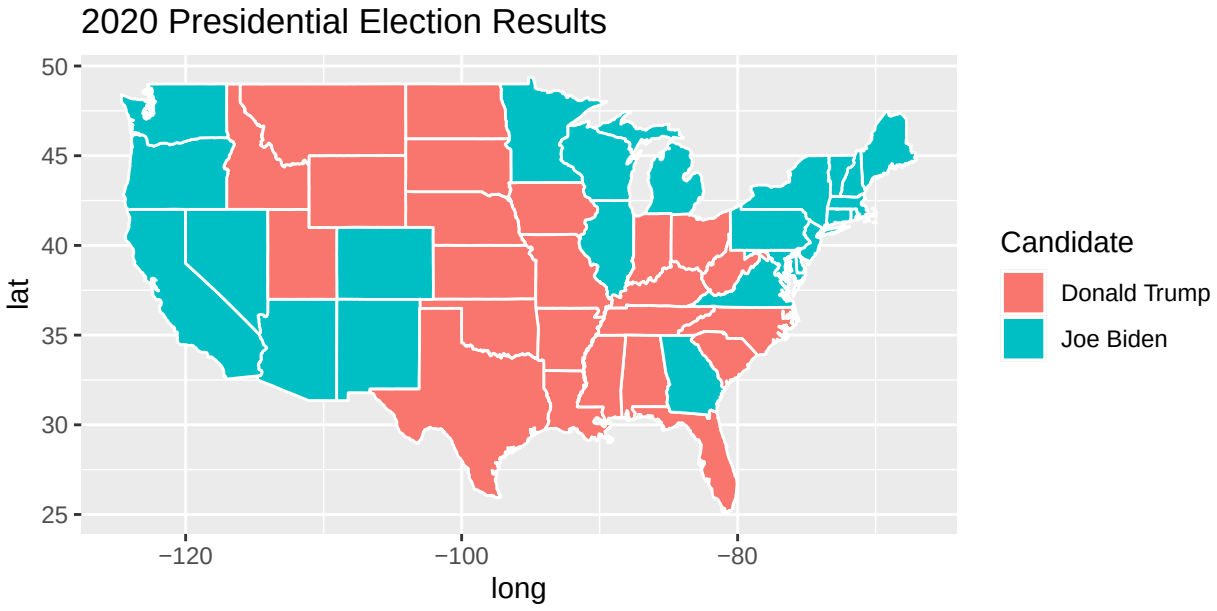
```r
# Visualization
states <- map_data("state")
counties <- map_data("county")

# Plot: US colored by count
ggplot(data = counties) +
  geom_polygon(aes(x = long, y = lat, fill = subregion, group = group),
               color = "white") +
               coord_fixed(1.3) +
               guides(fill=FALSE) +
              ggtitle('US Map Colored by County')
```



Here we can see a map of the US with county lines in white.

7 - Color State by winner

```
## Join state.winner with states data to encode winner of each state
# rename columns to be the same for left_join
# (Cap first letter to match)
states$state <- str_to_title(states$region)

states <- left_join(state.winner, states, by = 'state')

# US map colored by Winning candidate in each state: state.winner
ggplot(data = states) +
  geom_polygon(aes(x = long, y = lat, fill = candidate, group = group),
               color = "white") +
  coord_fixed(1.3) +
  guides(fill=guide_legend(title = 'Candidate',title.position = 'top'))+
  ggtitle('2020 Presidential Election Results')
```



8 - California Counties colored by presidential candidate

```
# Color the map of the state of California by the winning candidate for each county.

# Get map data for California
cali <- counties[counties$region == 'california',]

# Get county data for California
cali.raw <- county.winner[county.winner$state == 'California',]
```

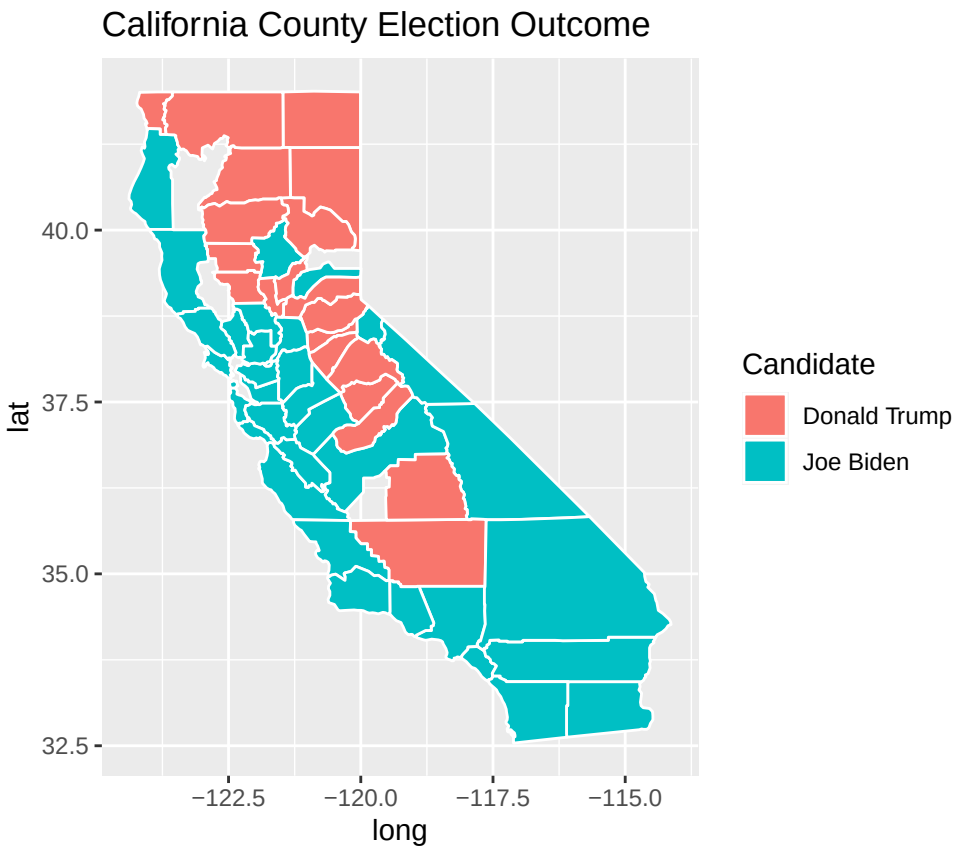
```r
# Check number of counties in each
length(unique(cali.raw$county)) # 54 - not all counties have a winner
```

```
## [1] 54
```

```r
length(unique(cali$subregion)) # 58
```

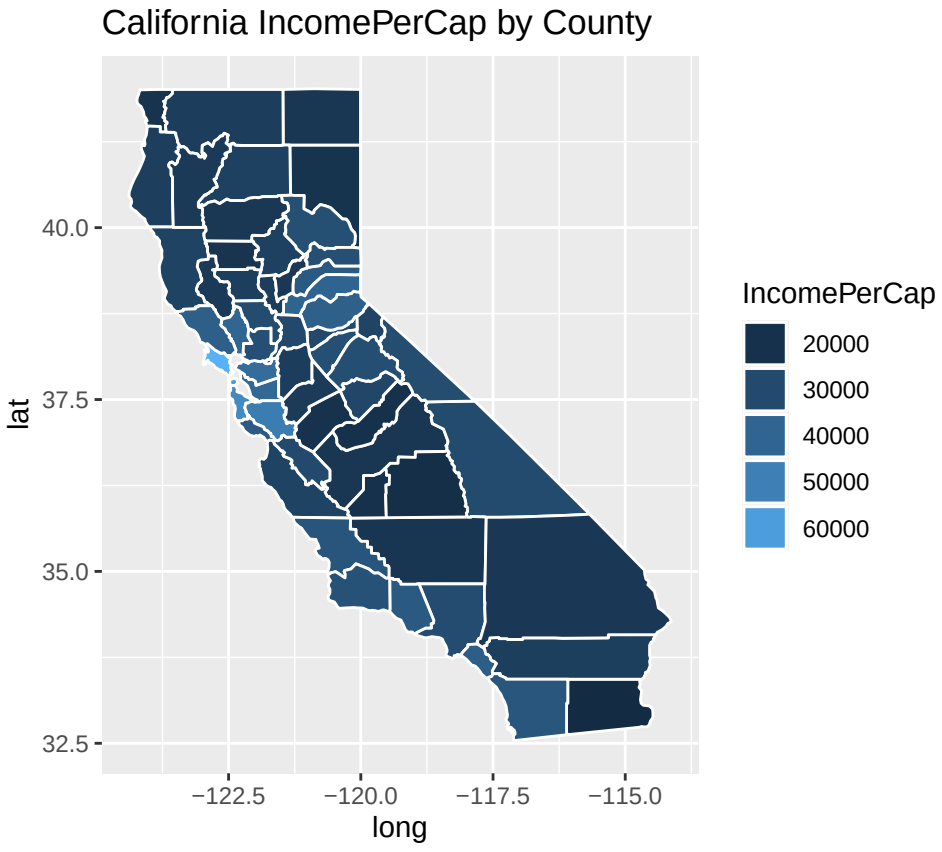
```
## [1] 58
```

```r
# left_join cali.raw data to get winner by county
  # create similar columns to join on
  cali$county <- str_to_title(cali$subregion)
california <- left_join(cali.raw, cali, by = 'county')



# plot
ggplot(data = california) +
  geom_polygon(aes(x = long, y = lat, fill = candidate, group = group),
               color = "white") +
               coord_fixed(1.3) +
               guides(fill=guide_legend(title = 'Candidate',title.position = 'top')) +
              ggtitle('California County Election Outcome')
```

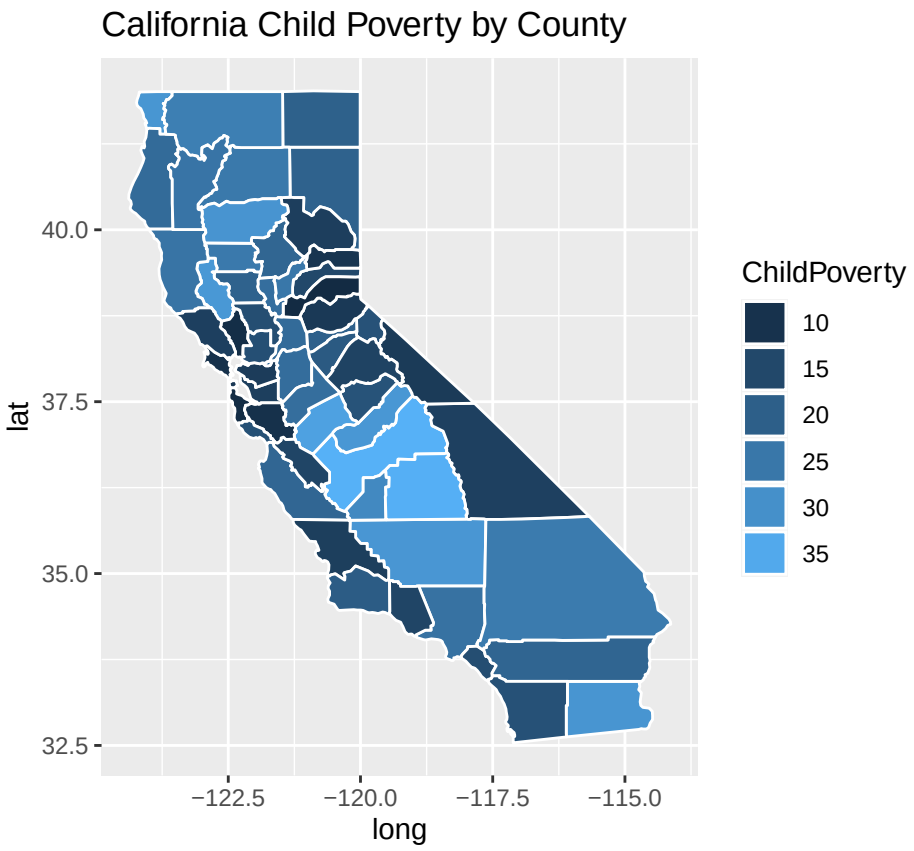


9

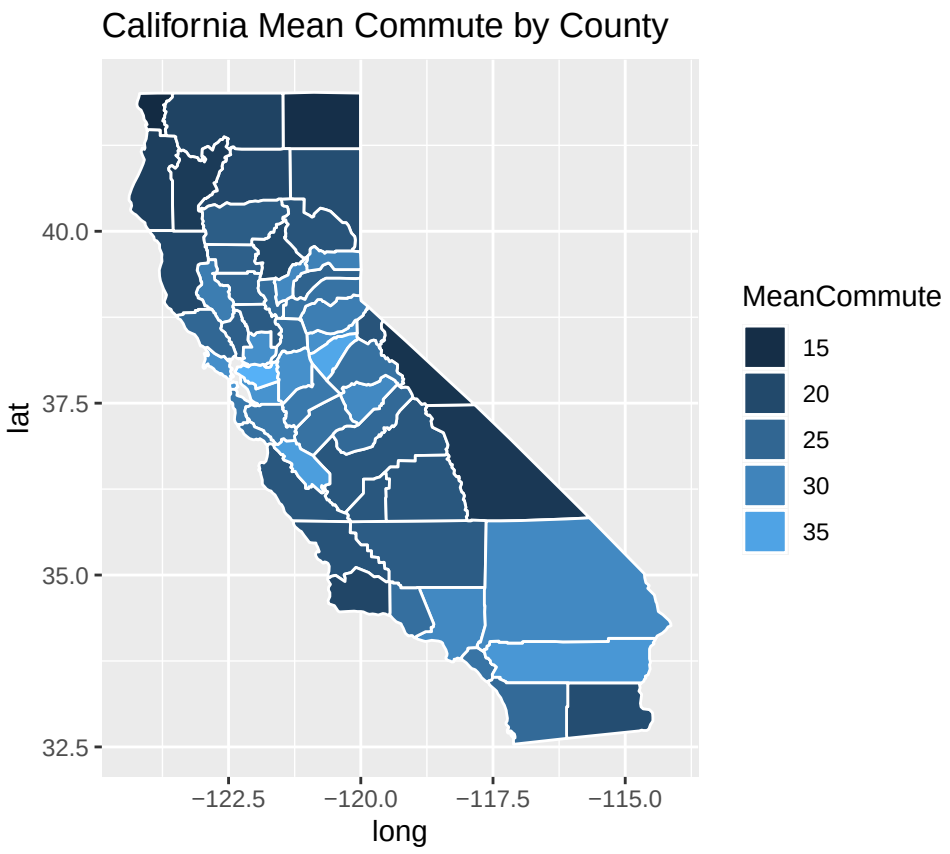
```r
# Use census data in California to show IncomePerCap by county
cali.census <- census[census$State == 'California',]

# remove the word county from County: using remove.words
cali.census$County <- remove.words(cali.census$County, 'County')
# subset cali.census to have only relevant data points
cali.census.wanted <- subset(cali.census, select = c('IncomePerCap', 'ChildPoverty','MeanCommute','Coun



# Map data: cali
# left join cali.census IncomePerCap & ChildPoverty & MeanCommute

  # Change subregion to County for left join
cali$County <- str_to_title(cali$subregion)
cali.9 <- left_join( cali, cali.census.wanted, by = 'County')


# Make the plot (heat map colors)
ggplot(data = cali.9) +
  geom_polygon(aes(x = long, y = lat, fill = IncomePerCap, group = group),
               color = "white") +
               coord_fixed(1.3) +
               guides(fill=guide_legend(title = 'IncomePerCap',title.position = 'top')) +
              ggtitle('California IncomePerCap by County')
```



```
# Child Poverty
ggplot(data = cali.9) +
  geom_polygon(aes(x = long, y = lat, fill = ChildPoverty, group = group),
               color = "white") +
               coord_fixed(1.3) +
               guides(fill=guide_legend(title = 'ChildPoverty',title.position = 'top')) +
              ggtitle('California Child Poverty by County')
```




```
# MeanCommute
ggplot(data = cali.9) +
  geom_polygon(aes(x = long, y = lat, fill = MeanCommute, group = group),
               color = "white") +
               coord_fixed(1.3) +
               guides(fill=guide_legend(title = 'MeanCommute',title.position = 'top')) +
              ggtitle('California Mean Commute by County')
```



10 data cleaning

```
# census.clean
# Remove missing values
census.clean <- census[complete.cases(census),] # removed 1 row

# length(unique(census.clean$CountyId))
# length(unique(census.clean$County)) # More unique CountyId's than Counties
# Need to use CountyId for mutation to avoid error

# convert Men, Employed, VotingAgeCitizens to percentages
census.clean <- census.clean %>% group_by(CountyId) %>%
  mutate(Employed = (Employed/TotalPop) * 100,
         VotingAgeCitizen = (VotingAgeCitizen/TotalPop) * 100, Men = (Men/TotalPop) * 100)

# combine Hispanic, Black, Native, Asian, Pacific to Minority
census.clean <- census.clean %>% group_by(CountyId) %>%
  mutate(Minority = sum(Hispanic + Native + Asian + Pacific+ Black))
# Note: could easily have included in above code, kept seperate for legiblity

# Remove Columns no longer desired
census.clean <- subset(census.clean, select = -c(IncomeErr,IncomePerCap,
                                                 IncomePerCapErr, Walk, PublicWork, Construction,Hispani

# Identify and Remove Columns which are linearly correlated
library(caret) # can find values in corr matrix which are greater than a cutoff
```

```
ColToDrop <- findCorrelation(cor(census.clean[,-c(1,2,3)]), cutoff = 0.7) # 70% correlated cutoff
 #census.clean[,-c(1,2,3)][1,ColToDrop] # White, ChildPoverty, Poverty,  and Women

# drop values from cleaned data:
  # Note: Not all need to be dropped (correlated with themselves)
  # Will drop Women, and both poverty metrics (assume they are highly correlated with income)
census.clean <- subset(census.clean, select = -c(Women, ChildPoverty, Poverty))

# Remove the word county from the counties
census.clean$County <- remove.words(census.clean$County, "County")

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

```
## # A tibble: 5 x 24
## # Groups:   CountyId [5]
##   CountyId State County TotalPop   Men White VotingAgeCitizen Income
##      <dbl> <chr> <chr>     <dbl> <dbl> <dbl>            <dbl>  <dbl>
## 1     1001 Alab~ Autau~    55036  48.9  75.4             74.5  55317
## 2     1003 Alab~ Baldw~   203360  48.9  83.1             76.4  52562
## 3     1005 Alab~ Barbo~    26201  53.3  45.7             77.4  33368
## 4     1007 Alab~ Bibb      22580  54.3  74.6             78.2  43404
## 5     1009 Alab~ Blount    57667  49.4  87.4             73.7  47412
## # ... with 16 more variables: 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>
```

Thus here during our data cleaning we chose to remove Women, ChildPoverty, Poverty as they were highly correlated with other predictors in our model and likely would not add extra detail.

Dimensionality Reduction

11

```
# Perform PCA on census.clean (State and County excluded: [,-c(2,3)])
  # will scale and center our data for best results
    # The range of data
census.PC <- prcomp(census.clean[,-c(2,3)], scale = TRUE, center = TRUE)


# Save PC1, PC2
pc.county <- as.data.frame(census.PC$x[,c(1,2)])

pc.feature <- census.PC$rotation[,c(1,2)]

# features with largest absolute values of the first component
sort(pc.feature[,1], decreasing = TRUE)
```

```
##     Unemployment         Minority          Service            Drive
##      0.372309600      0.327152273      0.223659084      0.184104710
```

```
##       Production      MeanCommute           Office          Carpool
##      0.129017498      0.128459935      0.116726752      0.075995045
##         CountyId      PrivateWork      OtherTransp         TotalPop
##      0.068900936      0.004443398      0.002862430     -0.021683040
##              Men VotingAgeCitizen          Transit       FamilyWork
##     -0.027329299     -0.038327832     -0.055322208     -0.130277393
##     SelfEmployed     Professional       WorkAtHome            White
##     -0.219831646     -0.296413546     -0.308281994     -0.323258800
##           Income         Employed
##     -0.327905911     -0.383606324
```

```
  # Employed, Unemployment, and Income are the top 3 (in that order)
```

I chose to Scale and center my values as they were in seperate ranges to one another, while many were in percentages and likely not to be an issue, I felt they would not combine well with metrics such as Total Population or income. Further, given the number of counties/states, I felt we had enough data points to effectively normalize the data.

The largest values given in PC1 were Employed, Unemployment, and Income implying that they had the greatest affect.

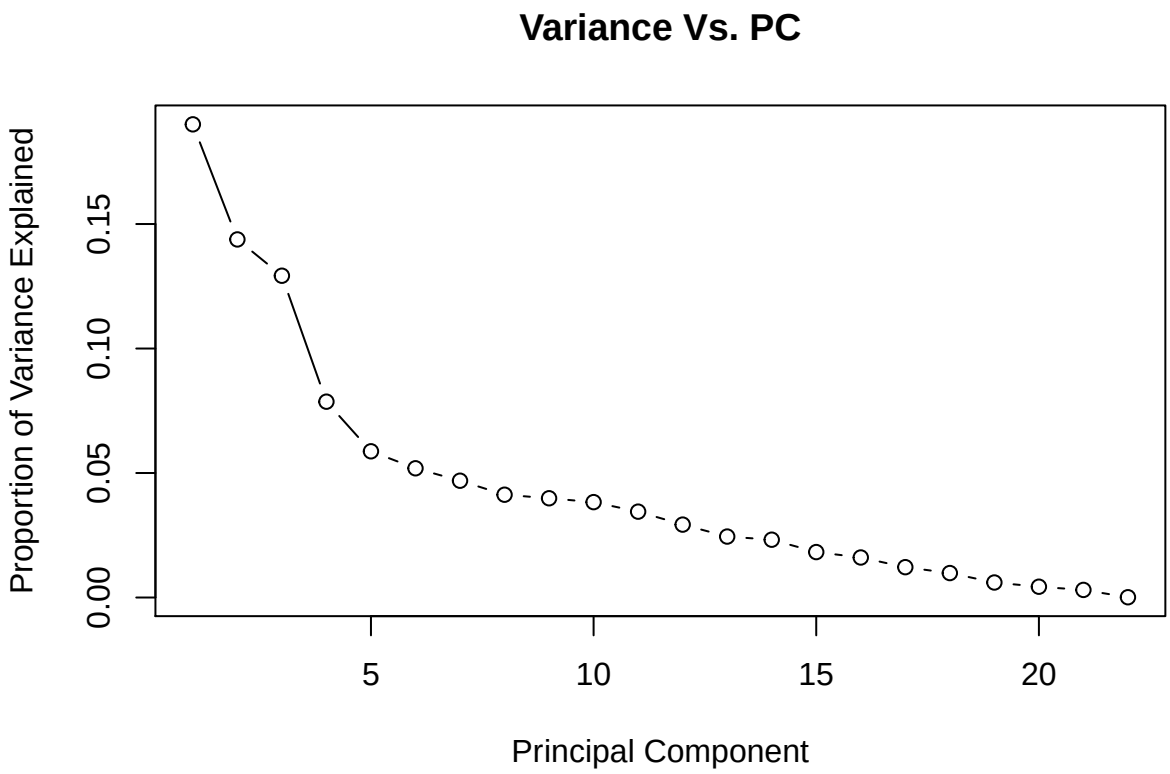
A positive value with a higher score is more important to that component, if it is negative, a larger absolute value indicates a lower score on the PC. This implies that of our top 3 largest values, 2 of them were to indicate lower scores in PC1, where Unemployment appeared to be the most positively significant.

12

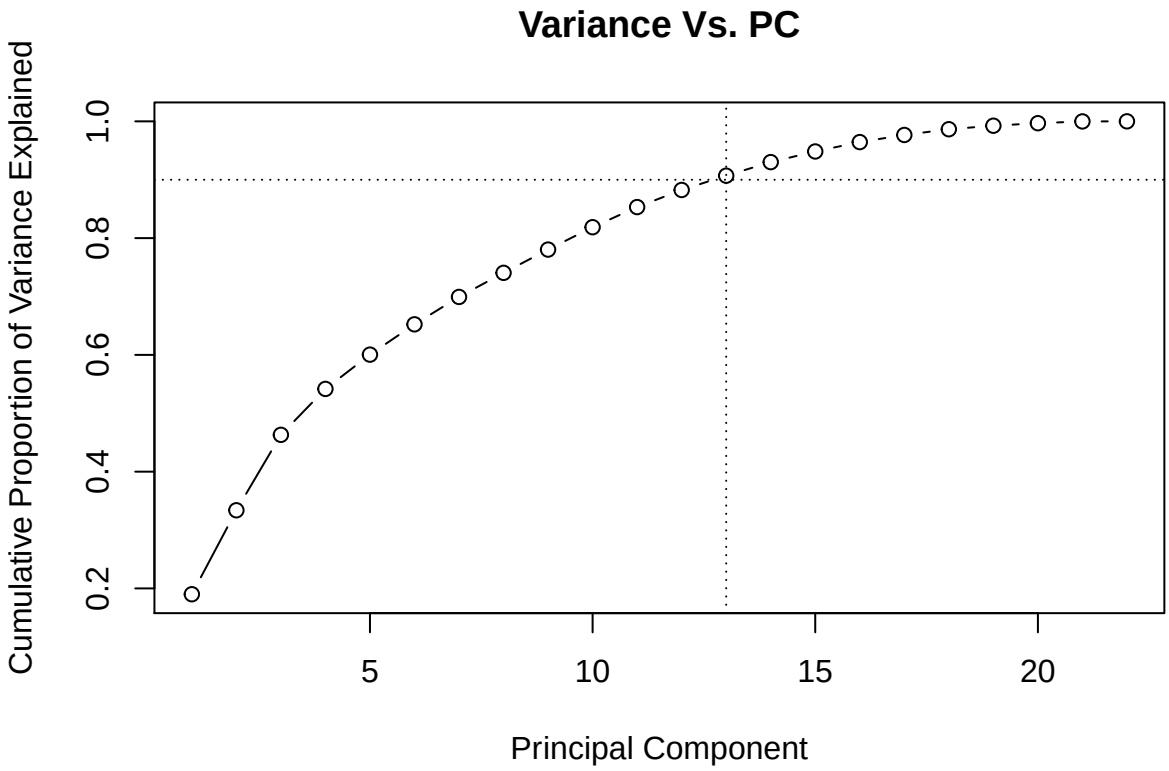
```
# Find number of PC's needed to capture 90% of variance
PC.var <- census.PC$sdev^2 # get variance of each
  # Proportion of Variance Explained
  pve = PC.var/sum(PC.var)

# Number needed for 90%: 13
length(pve[cumsum(pve) < .9]) # counts the False PC's until the Cumsum hits our 90% (# needed - 1)
```

```
## [1] 12
```

```
# Plot proportion of variance explained(PVE)
plot(pve, xlab = 'Principal Component', ylab = 'Proportion of Variance Explained',
     type = 'b', main = 'Variance Vs. PC')
```




```
# Plot cumulative PVE
plot(cumsum(pve), xlab = 'Principal Component', ylab = 'Cumulative Proportion of Variance Explained',
     type = 'b', main = 'Variance Vs. PC')
abline(v = 13, h = 0.9, lty = 'dotted') # 90% above position 12 below 13
```



Clustering

13

```
# hierarchical clustering with complete linkage
census.clean2 <- census.clean[,-c(2,3)] # Remove State and County (For scaling)
  # Scale Data
  scaled.census <- scale(census.clean2, center = TRUE, scale = TRUE)

# Create distance matrix
census.dist = dist(scaled.census)

# set seed
set.seed(1)

# Build tree
census.hclust <- hclust(census.dist) # default method is complete linkage

# cut Tree into 10 clusters
census.hclust.10 <- cutree(census.hclust, k = 10)

### Use PC loadings and run analysis again, which group does SB land in?

# Build tree 2: PC's are already scaled
census.dist2 <- dist(pc.county)
```

```
census.pc.hclust <- hclust(census.dist2) # form tree

# Cut Tree
census.pc.hclust.10 <- cutree(census.pc.hclust, k = 10)

# Compare results: Cluster assignments
table(census.hclust.10)
```

```
## census.hclust.10
##    1    2    3    4    5    6    7    8    9   10
## 3105    6    6    5   48    1   13    4   27    4
```

```
table(census.pc.hclust.10)
```

```
## census.pc.hclust.10
##    1    2    3    4    5    6    7    8    9   10
## 1486 1165  218  101  162   13   46    1   20    7
```

Thus by using our Principle loadings and the initial data, both scaled and centered, we were able to use hierarchical clustering to split our counties into 10 seperate groups. Santa Barbara County was in group 1 the initial runthrough and then using the PC's it was placed into group 5. Looking at the distribution from the table function calls we can see that the pc version has much more spread in the groupings. The census data has all groups besides group 1 with less than 50 observations, group 1 contains 3105 (whereas in the PC version group 1 had only 1486 observations).

```
### Eval on SB
# cluster.eval <- left_join(county.winner[c('state','county','candidate')], census.clean, by = 'county'
# Note I had too many issues with keeping census.clean index the same as hclust objects
# Had initially wanted to check who each county voted for
# and check the group SB belongs to compared to who SB voted for
# the join with county.winner was affecting the index positions


# Join the state names to the groups and assess groupings
census.hclust.10 <- cbind(census.hclust.10, census.clean['State'])
census.pc.hclust.10 <- cbind(census.pc.hclust.10, census.clean['State'])

# find SB county: Row 228 (based on row position in census.clean)
census.hclust.10[228,] # group 1
```

```
##     census.hclust.10      State
## 228                1 California
```

```
# find SB county: Row 228
census.pc.hclust.10[228,] # group 5
```

```
##     census.pc.hclust.10      State
## 228                   5 California
```

```
# Compare results: # of states for each
  # Base Results
y = table(census.hclust.10[census.hclust.10$census.hclust.10 == 1,])[1,]

sort(y, decreasing = TRUE)[1:5]
```

```
##    Texas  Georgia Virginia Kentucky Missouri
##      237      153      128      120      115
```

```
# [1,] at end to turn to vector
# [1:5] gets the top 5 counts

  # PC Results
x = table(census.pc.hclust.10[census.pc.hclust.10$census.pc.hclust.10 == 3,])[1,]
sort(x, decreasing = TRUE)[1:5]
```

```
##          Texas        Georgia    Mississippi North Carolina        Alabama
##             31             23             19             15             14
```

I would argue that the PC version has a worse clustering for Santa Barbara County as the states it is associated with the most, as witnessed by the counts for the respective groups, they did not vote in line with what we saw in Santa Barbara. The initial screens 3 of top 5 voted Red whereas with the PC version 4 of the top 5 had voted Red. Important to note is that this is a very arbitrary metric to judge the performance of our clustering. If I had more time I would ideally have been able to code in the candidate for each county and then table the candidate based on who won in that grouping. While that method would be more complete it would still be an arbitrary metric to use, however given this is a classification problem I think it would make the most sense.

Classification

can we use census information in a county to predict the winner in that county?

14

We removed the party from election.cl in order to prevent unwanted correlation when making a predictive model (wouldn't want one of the predictors to be a function of the response/ perfectly linearly correlated)

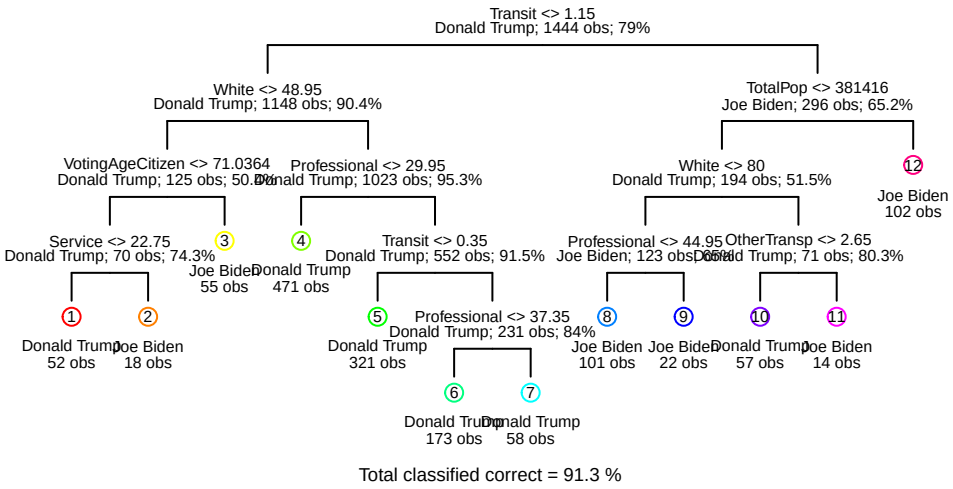
```
# split data into train and test
set.seed(10)
n <- nrow(election.cl)
idx.tr <- sample.int(n, 0.8*n) # creates index to use
election.tr <- election.cl[idx.tr, ]
election.te <- election.cl[-idx.tr, ]

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

# Function to test error
calc_error_rate = function(predicted.value, true.value){
  return(mean(true.value!=predicted.value))
```

```
}
records = matrix(NA, nrow=3, ncol=2)
colnames(records) = c("train.error","test.error")
rownames(records) = c("tree","logistic","lasso")
```

15

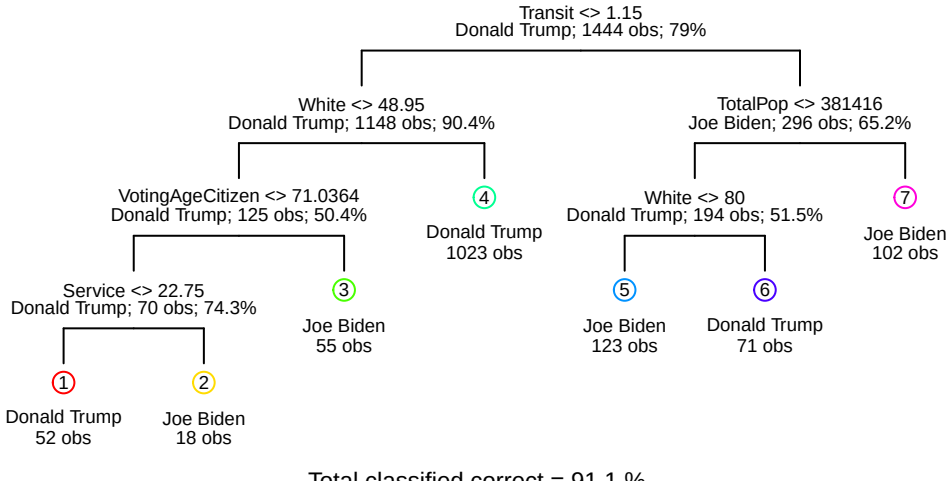
Decision tree: Model to predict candidate

```
# Data: election.cl
set.seed(2)

# Create Tree
tree.election <- tree(candidate~., data = election.tr, method = 'classification')

# Plot of unhedged tree
draw.tree(tree.election, nodeinfo=TRUE, cex = .5)
```




```
### Prune the tree

# Use Cross Validation to find best prune point
cv = cv.tree(tree.election, FUN = prune.misclass, rand = folds) # Note: rand is our fold assignment

# determine best prune point
best.cv = min(cv$size[cv$dev == min(cv$dev)]) # find min size for which the dev is least
```

```
# Prune the tree
election.pruned <- prune.misclass(tree.election, best = best.cv)

# Plot tree
draw.tree(election.pruned, nodeinfo = TRUE, cex = 0.6)
```



Thus from the above we can see our initial tree and the pruned tree based on 10 folds assigned previously. Our inital tree had 12 leaf nodes and the pruned tree reduced this number to 7 while reducing deviation.

Transit is consistently the first predictor used to make a split indicating it has the largest effect on the model. In many states with large scale public transport (that are realistic to use to get to work/ daily life), a unifying theme is typically dense populations which are often associated with democratic constituents. On the flip side of this, places with very sparse public transport typically in rural country settings are considered consistently conservitive. The next largest factors are the percent of the population that is white, and the total population (both of which could be looked at from the rural/urban perspective).

```
############ determine training/test error

##### Training

# training predictions
elect.training.err <- predict(election.pruned, election.tr, type = 'class')



# Training Error rate
```

```
calc_error_rate(elect.training.err, election.tr$candidate)
```

```
## [1] 0.08933518
```

```
# Update value in df
records[1,1] = round(calc_error_rate(elect.training.err, election.tr$candidate), digits = 4)
# records[tree, train.error]

##### Test

# test error prediction:
elect.test.err <- predict(election.pruned, election.te, type = 'class')

# Test Error rate
calc_error_rate(elect.test.err, election.te$candidate)
```

```
## [1] 0.1301939
```

```
# Update value in df
records[1,2] = round(calc_error_rate(elect.test.err, election.te$candidate), digits = 4)
# records[tree, test.error]

records
```

```
##          train.error test.error
## tree          0.0893     0.1302
## logistic          NA         NA
## lasso             NA         NA
```

Thus we can see we had a training error of roughly 8% and a test error of 13% with our pruned tree.

16 - Logistic Regression

```
# election.te (test) election.tr (training)
# Logistic regression
glm.election <- glm(candidate~., data = election.tr, family = binomial)
```

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

```
# View coefficients
summary(glm.election)
```

```
##
## Call:
## glm(formula = candidate ~ ., family = binomial, data = election.tr)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.8478  -0.2553  -0.0991  -0.0215   3.3482
##
## Coefficients:
```

```
##                     Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -1.944e+01  1.169e+01  -1.663  0.09638 .
## TotalPop          1.857e-06  7.584e-07   2.448  0.01436 *
## Men              -4.975e-03  6.000e-02  -0.083  0.93392
## White            -2.061e-01  8.503e-02  -2.424  0.01535 *
## VotingAgeCitizen  1.746e-01  3.063e-02   5.700 1.20e-08 ***
## Income           -1.854e-05  1.672e-05  -1.109  0.26746
## Professional      3.188e-01  4.923e-02   6.476 9.43e-11 ***
## Service           3.296e-01  6.157e-02   5.353 8.65e-08 ***
## Office            1.603e-01  6.253e-02   2.563  0.01037 *
## Production        2.334e-01  5.240e-02   4.455 8.40e-06 ***
## Drive            -2.198e-01  5.535e-02  -3.970 7.17e-05 ***
## Carpool          -2.279e-01  7.376e-02  -3.090  0.00200 **
## Transit           6.270e-02  1.133e-01   0.553  0.57994
## OtherTransp       1.251e-01  1.160e-01   1.078  0.28110
## WorkAtHome       -1.353e-01  8.578e-02  -1.577  0.11481
## MeanCommute       3.524e-02  3.180e-02   1.108  0.26780
## Employed          2.425e-01  4.046e-02   5.995 2.03e-09 ***
## PrivateWork       7.088e-02  2.752e-02   2.576  0.01000 **
## SelfEmployed      2.067e-02  5.827e-02   0.355  0.72284
## FamilyWork       -3.697e-01  3.351e-01  -1.103  0.26984
## Unemployment      1.515e-01  5.085e-02   2.980  0.00288 **
## Minority         -5.943e-02  8.218e-02  -0.723  0.46963
## ---
## 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:  523.65  on 1422  degrees of freedom
## AIC: 567.65
##
## Number of Fisher Scoring iterations: 7
```

Here we see for the logistic regression model that our most important predictors are Professional, Employed, service, production and VotingAgeCitizen. Our decision tree had indicated the most significant were "Transit", "White", "VotingAgeCitizen", "Service", "TotalPop". They shared Service and VotingAgeCitizen though neither were the most significant in either model.

In the logistic regression the most significant predictors on the model were professional and service as witnessed by their relatively large estimate values, large z values, and highly siginificant p values. For instance should we have 10 unit changes in each would increase our value by roughly 3 each. Noting that in logistic regression our estimates would be our β values in the following equation.

$$p = \frac{e^{\beta_0+\beta_1 x_1+\ldots+\beta_p x_p}}{1+e^{\beta_0+\beta_1 x_1+\ldots+\beta_p x_p}}$$

Where p is our probality that Joe Biden is the chosen candidate, thus by increasing the values in the exponential our p value would tend further towards 1, or push the probability of our logistic regression towards predicting president Biden.

We can think of larger relative professional workforces as being associated with urban areas and therefore more liberal politicaly as well as diverse in nature. Likewise service jobs are much more common in densely populated areas. This correlation between political idealogy and population density is known to be highly correlated with US election outcomes as witnessed in many previous elections. Looking at our data we can

show that on average counties with a higher percentage of Professional and service workers voted for Joe Biden as predicted by the skew of cities.

```
# Professional
aggregate(Professional~ candidate, election.cl, mean)
```

```
##      candidate Professional
## 1 Donald Trump     30.79405
## 2    Joe Biden     37.09523
```

```
# Service
aggregate(Service~ candidate, election.cl, mean)
```

```
##      candidate  Service
## 1 Donald Trump 17.94608
## 2    Joe Biden 19.42679
```

```
# Transit
aggregate(Transit~ candidate, election.cl, mean)
```

```
##      candidate   Transit
## 1 Donald Trump 0.5387955
## 2    Joe Biden 3.8729443
```

Though of interest is that the greatest relative difference in means between predictors is for transit which the tree deemend much more significant.

Training/Test error

```
# training error
glm.training <- predict(glm.election, type = 'response')

# Threshold at 0.5 for predictions
glm.training <- ifelse(glm.training >= 0.5, 'Joe Biden', 'Donald Trump')

# Confusion matrix
table(pred = glm.training, true = election.tr$candidate)
```

```
##               true
## pred           Donald Trump Joe Biden
##   Donald Trump         1107        66
##   Joe Biden              34       237
```

```
# Training error rate
mean(glm.training != election.tr$candidate)
```

```
## [1] 0.06925208
```

```
#####################

# test error
glm.test <- predict(glm.election, election.te, type = 'response')

# Threshold at 0.5 for predictions
  # Note this is an arbitary limit further analysis with a confusion matrix could lead to further optim
glm.test <- ifelse(glm.test >= 0.5, 'Joe Biden', 'Donald Trump')

# Confusion matrix
table(pred = glm.test, true = election.te$candidate)
```

```
##               true
## pred           Donald Trump Joe Biden
##   Donald Trump          279        22
##   Joe Biden               8        52
```

```
# Test error rate
mean(glm.test != election.te$candidate)
```

```
## [1] 0.08310249
```

```
# update records
records[2,1] = round(mean(glm.training != election.tr$candidate), digits = 4)
records[2,2] = round(mean(glm.test != election.te$candidate), digits = 4)

# display records
records
```

```
##          train.error test.error
## tree          0.0893     0.1302
## logistic      0.0693     0.0831
## lasso             NA         NA
```

Thus we have seen that our logistic model appears better in terms of test error. Looking at the confusion matrices we can note that Joe Biden is misclassified at a larger relative rate, this implies our model may be improved by moving our threshold of predicting Joe Biden from 0.5.

17 Overfitting - Regularization of logistic regression model

```
set.seed(2)
# lasso penalty
library(glmnet)
### Define x and y vectors for cv.glmnet
# x is election.tr, y is election.te$candidate
x.train <- droplevels(election.tr$candidate)


x.train <- as.matrix(election.tr[,-1]) # remove candidate to convert all to int
# x.train <- cbind(election.tr[,1], x.train) # turns back into list
```

```
y.train <- ifelse(election.tr$candidate == 'Joe Biden', 1, 0) # Biden = 1
y.test <- ifelse(election.te$candidate == 'Joe Biden', 1, 0) # Trump = 0


# Find lambda, using 10-fold CV (10 folds is default)
cv.lasso <- cv.glmnet(x.train, y.train, alpha = 1, lambda = seq(1, 50) * 1e-4, family = 'binomial')

# Optimal value of lambda
cv.lasso$lambda.min
```

```
## [1] 8e-04
```

```
#coefficients
coef(cv.lasso)
```

```
## 22 x 1 sparse Matrix of class "dgCMatrix"
##                               1
## (Intercept)      -1.544539e+01
## TotalPop          1.743259e-06
## Men              -2.737042e-02
## White            -1.063722e-01
## VotingAgeCitizen  1.365790e-01
## Income            .
## Professional      1.542994e-01
## Service           1.469733e-01
## Office            1.566258e-02
## Production        4.439140e-02
## Drive            -8.545096e-02
## Carpool          -7.906326e-02
## Transit           1.024783e-01
## OtherTransp       1.494640e-01
## WorkAtHome        .
## MeanCommute       .
## Employed          1.504688e-01
## PrivateWork       4.515097e-02
## SelfEmployed     -4.523344e-02
## FamilyWork       -1.604264e-01
## Unemployment      1.254228e-01
## Minority          .
```

Thus we can see that Minority, MeanCommute, WorkAtHome, and Income have all been removed by our lasso model. For completeness of the method I will update our logistic regression model to remove these predictors and then store the training and test error of the method as lasso. Also the optimal value of lamda is given by cv.lasso$lambda.min.

```
set.seed(3)
# Create reduced model
glm.election2 <- glm(candidate~. - Minority - MeanCommute - WorkAtHome - Income,
                     data = election.tr, family = binomial)
# fitted probabilities numerically warning persists

# training error
```

```r
glm.training.lasso <- predict(glm.election2, type = 'response')

# Threshold at 0.5 for predictions
glm.training.lasso <- ifelse(glm.training.lasso >= 0.5, 'Joe Biden', 'Donald Trump')

# Confusion matrix
table(pred = glm.training.lasso, true = election.tr$candidate)
```

```
##                true
## pred            Donald Trump Joe Biden
##   Donald Trump          1106        67
##   Joe Biden               35       236
```

```r
# Training error rate
mean(glm.training.lasso != election.tr$candidate)
```

```
## [1] 0.07063712
```

```r
######################

# test error
glm.test.lasso <- predict(glm.election2, election.te, type = 'response')

# Threshold at 0.5 for predictions
  # Note this is an arbitary limit further analysis with a confusion matrix
  # could lead to further optimization
glm.test.lasso <- ifelse(glm.test.lasso >= 0.5, 'Joe Biden', 'Donald Trump')

# Confusion matrix
table(pred = glm.test.lasso, true = election.te$candidate)
```

```
##                true
## pred            Donald Trump Joe Biden
##   Donald Trump           279        22
##   Joe Biden                8        52
```

```r
# Test error rate
mean(glm.test.lasso != election.te$candidate)
```

```
## [1] 0.08310249
```

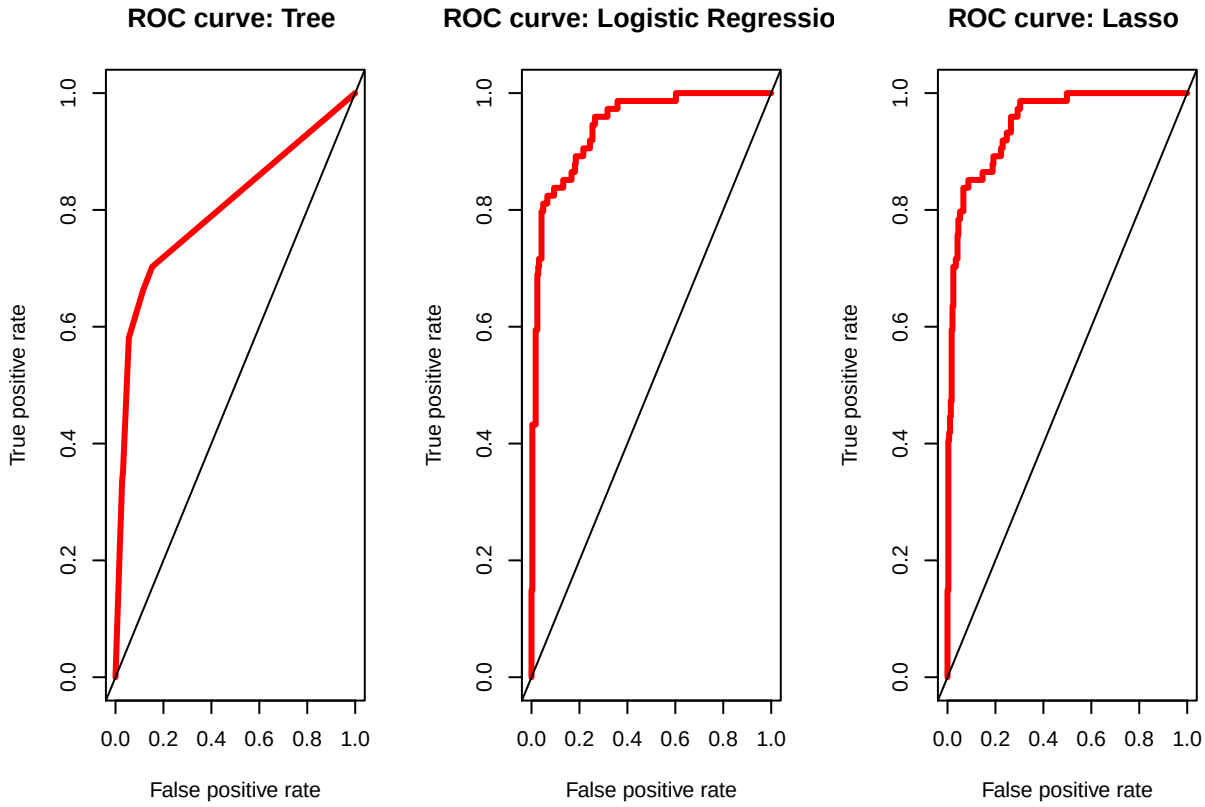
```r
# update records
records[3,1] = round(mean(glm.training.lasso != election.tr$candidate), digits = 4)
records[3,2] = round(mean(glm.test.lasso != election.te$candidate), digits = 4)

# display records
records
```

```
##          train.error test.error
## tree          0.0893     0.1302
## logistic      0.0693     0.0831
## lasso         0.0706     0.0831
```

18 ROC curves

```
library(ROCR)

#### Tree
# Create vector for use in prediction
tree.test.err <- predict(election.pruned, election.te)

## Create prediction object
pred.tree <- prediction(tree.test.err[,2], election.te$candidate)
# use second column for Biden

# create object to be plotted
perf.tree <- performance(pred.tree, measure = 'tpr', x.measure = 'fpr')

# Area under the Curve AUC
AUC.tree <- performance(pred.tree, 'auc')@y.values



#### Logistic Regression
# Probability vector for use in prediction
log.test.err <- predict(glm.election, election.te, type = 'response')

# Create prediction object
pred.log <- prediction(log.test.err, election.te$candidate)

# Create performance object
perf.log <- performance(pred.log, measure = 'tpr', x.measure = 'fpr')

# Area under the Curve AUC
AUC.log <- performance(pred.log, 'auc')@y.values

#### Lasso Model
# Probability vector for use in prediction
glm.test.lasso <- predict(glm.election2, election.te, type = 'response')

# Create prediction object
pred.lasso <- prediction(glm.test.lasso, election.te$candidate)

# Create performance object
perf.lasso <- performance(pred.lasso, measure = 'tpr', x.measure = 'fpr')

# Area under the Curve AUC
AUC.lasso <- performance(pred.lasso, 'auc')@y.values


# Display AUC of all 3
AUC <- cbind(AUC.tree, AUC.log, AUC.lasso)

### display all 3 in one graph
# 3 figures arranged in 3 rows and 1 column
par(mfcol=c(1,3)) # set up framework
```

```
# Tree plot
plot(perf.tree, col=2, lwd=3, main="ROC curve: Tree ")
abline(0,1)

# Logistic Regression Plot
plot(perf.log, col=2, lwd=3, main="ROC curve: Logistic Regression ")
abline(0,1)

# Lasso Plot
plot(perf.lasso, col=2, lwd=3, main="ROC curve: Lasso ")
abline(0,1)
```




```
# AUC of each
AUC
```

```
##       AUC.tree  AUC.log   AUC.lasso
## [1,] 0.8011583 0.9439213 0.9472643
```

19 Additional classification methods: random forest and SVM

```
set.seed(5)
# Random Forest Model: using training/test split, as opposed to OOB error rate
rf.election <- randomForest(candidate~., data = election.tr, n.trees = 1500)
```

```r
# Chose to keep importance = False (default), lowers OOB error estimate
importance(rf.election)
```

```
##                   MeanDecreaseGini
## TotalPop                 56.598279
## Men                      17.310273
## White                    50.568856
## VotingAgeCitizen         15.233095
## Income                   21.819058
## Professional             30.801550
## Service                  13.264049
## Office                    8.865174
## Production               20.250933
## Drive                    13.937845
## Carpool                   7.891115
## Transit                  74.819242
## OtherTransp              12.479107
## WorkAtHome                8.603333
## MeanCommute              11.146200
## Employed                 19.475356
## PrivateWork              10.318505
## SelfEmployed             15.378319
## FamilyWork                3.726080
## Unemployment             17.940901
## Minority                 47.748571
```

Thus we can see our model where we randomly sampled 4 variables from our predictors at each split point to build our tree, with 1500 trees grown, similiar to our previous tree method the most impactful predictors on our model are Transit, TotalPop, and White.

Test error estimate

```r
# Prediction
pred.rf <- predict(rf.election, newdata = election.te)

# Test error
mean(pred.rf != election.te$candidate)
```

```
## [1] 0.08864266
```

```r
# Using OOB error as training error create vector of scores
rf.perf <- c(.073, mean(pred.rf != election.te$candidate))

# insert into records
records <- rbind(records, rf.perf)

records
```

```
##          train.error test.error
## tree          0.0893 0.13020000
## logistic      0.0693 0.08310000
## lasso         0.0706 0.08310000
## rf.perf       0.0730 0.08864266
```

Thus from the above we can see that our random forest performed signficantly better in terms of test error as compared to our inital tree model, however the method is currently still worse than the logistic regressions.

SVM

```
library(e1071)
set.seed(8)
# Create model
svmfit.lin <- svm(candidate~., data = election.tr, kernel="linear", cost=10, scale=TRUE)
svmfit.rad <- svm(candidate~., data = election.tr, kernel="radial", cost=10, scale=TRUE)
svmfit.poly <- svm(candidate~., data = election.tr, kernel="polynomial", cost=10, scale=TRUE)

# Cross validate: Tune # Note: Cost is best between 1:10
# discovered best cost range by using C = .001, .01, .1, 1, 10, 100
tune.out.lin <- tune(svm, candidate~., data = election.tr, kernal = 'linear',
                     ranges = list(cost = seq(1, 10, .5)))
tune.out.rad <- tune(svm, candidate~., data = election.tr, kernal = 'radial',
                     ranges = list(cost = seq(1, 10, .5)))
tune.out.poly <- tune(svm, candidate~., data = election.tr, kernal = 'polynomial',
                     ranges = list(cost = seq(1, 10, .5)))

# Store the best models for testing
lin.best <- tune.out.lin$best.model
rad.best <- tune.out.rad$best.model
poly.best <- tune.out.poly$best.model

# Find training/test error rate
### Training error
# Create prediction vectors
lin.pred <- predict(lin.best, election.tr)
rad.pred <- predict(rad.best, election.tr)
poly.pred <- predict(poly.best, election.tr)

# Find training error
lin.tr.err <- mean(lin.pred != election.tr$candidate)
rad.tr.err <- mean(rad.pred != election.tr$candidate)
poly.tr.err <- mean(poly.pred != election.tr$candidate)

### Test error
# Create test vectors
lin.test <- predict(lin.best, election.te)
rad.test <- predict(rad.best, election.te)
poly.test <- predict(poly.best, election.te)

# Find test error
lin.te.err <- mean(lin.test != election.te$candidate)
rad.te.err <- mean(rad.test != election.te$candidate)
poly.te.err <- mean(poly.test != election.te$candidate)


# Create dataframe to compare
svm.result <- matrix(c(lin.tr.err, rad.tr.err, poly.tr.err, lin.te.err, rad.te.err, poly.te.err), nrow =
colnames(svm.result) <- c('Training error', 'Test error')
rownames(svm.result) <- c('Linear Kernal', 'Radial Kernal', 'Polynomial Kernal')
```

```r
# SVM Results
svm.result
```

```
##                   Training error Test error
## Linear Kernal          0.03531856 0.08310249
## Radial Kernal          0.03808864 0.08033241
## Polynomial Kernal      0.03047091 0.08310249
```

```r
# Compare with previous methods
records <- rbind(records, svm.result[2,])
rownames(records) <- c('tree', 'logistic','lasso','random forest', 'SVM: Radial')
records
```

```
##                train.error test.error
## tree            0.08930000 0.13020000
## logistic        0.06930000 0.08310000
## lasso           0.07060000 0.08310000
## random forest   0.07300000 0.08864266
## SVM: Radial     0.03808864 0.08033241
```

Here we can see a fairly significant improvement in our best model using a Radial Kernal with cost = 1.5, our test error rate achieved was 8%, where our previous best models all seemed to be capped at around 8.3% error. Of interest as well is that the model with the best Test error also had the worst training error among the support vector models.

20

Time has been a major limiting factor for this final and unfortunately I will not have time to tackle a further interesting question. In retrospect working with a partner would have greatly improved the quality of work I would have been able to present. Something I may choose to work on during the break and had hoped to include in this assignment is to create a fake news engagement index based on google trends/ searches. With this index created based off a handful of popular conspiracies, I would then attempt to normalize the index with respect to the other states and use it as a predictor in a handful of models to see if it is able to add context to the purple states. Fake news has impacted many people and their political opinions, so doing an analysis with the rise of QANON, pizzagate or various other larger conspiracies with regards to changes in voting patterns (purple states) over the past 2 elections could potentially add context.

21

Of all the methods we attempted the incredibly low training error and best test error with radial support vector machine could hold insight into the nature of the data. A radial kernal implies to me that a KNN model, or models which group based on locality in hyperspace, are likely one of the best ways to pursuit improving our model. This is likely true because of various counties behaving like microcosms/ associations between factors not being additive. Some predictors in the model when associated with other values that are higher or lower may indicate different things about which candidate is likely to be voted for. Of interest though is that all of the support vector models had training error less than 4%, so perhaps a SVM is the most effective for splitting apart the smaller/ more granular groupings.