2016 Election Prediction

Jorge Sifuentes & Pedro Aristizabal (PSTAT 131)

5/21/2019

Predicting voter behavior is complicated for many reasons despite the tremendous effort in collecting, analyzing, and understanding many available datasets. For our final project, we will analyze the 2016 presidential election dataset, but, first, some background.

Background

The presidential election in 2012 did not come as a surprise. Some correctly predicted the outcome of the election correctly including Nate Silver, and many speculated his approach.

Despite the success in 2012, the 2016 presidential election came as a big surprise to many, and it was a clear example that even the current state-of-the-art technology can surprise us.

Answer the following questions in one paragraph for each.

1. What makes voter behavior prediction (and thus election forecasting) a hard problem?

   For one there are multiple dimmensions to consider (location, age, education level, etc . . .). There is also the number of canidates to consider and also how canidate popularity is sensitive to unforseen real time change (A canidates scandal gets uncovered for instance). It's not obvious how some factors may corrilate. The statitistics may be compromised as politicians in that position tend to be nefarious in nature. Electoral College is complicated.

2. What was unique to Nate Silver's approach in 2012 that allowed him to achieve good predictions?

Nate Silver prediction model was unique in the following ways: FiveThirtyEight's probabilities are based on the accuracy of polling averages. In addition to a systematic national polling error, they also simulate potential errors across regional or demographic lines. They assume that polling errors are correlated.Their historical models and the recent election show that making inferences from early voting is genreally not good.

3. What went wrong in 2016? What do you think should be done to make future predictions better?

From the artical the errors in predictions were coming from wrong assumptions and over generalizations, such as state vs national polls. It seems that more information can be accurately derived based on historical trends of polls and election outcomes. The accuracy seems to not come directly from the modelbut how well we can predict the error. Some challanges for future elections is predicting trends with late undediced voters, and the changing of party preferences of some states such as Arizona and Pennsylvania. Journalist.

Data

```r
election.raw = read.csv("election.csv") %>% as.tbl
census_meta = read.csv("metadata.csv", sep = ";") %>% as.tbl
census = read.csv("census.csv") %>% as.tbl
census$CensusTract = as.factor(census$CensusTract)
```

Election data

Following is the first few rows of the `election.raw` data:

county	fips	candidate	state	votes
NA	US	Donald Trump	US	62984825
NA	US	Hillary Clinton	US	65853516
NA	US	Gary Johnson	US	4489221
NA	US	Jill Stein	US	1429596
NA	US	Evan McMullin	US	510002
NA	US	Darrell Castle	US	186545

The meaning of each column in `election.raw` is clear except `fips`. The accronym is short for Federal Information Processing Standard.

In our dataset, `fips` values denote the area (US, state, or county) that each row of data represent: i.e., some rows in `election.raw` are summary rows. These rows have `county` value of `NA`. There are two kinds of summary rows:

- Federal-level summary rows have `fips` value of `US`.
- State-level summary rows have names of each states as `fips` value.

Census data

Following is the first few rows of the `census` data:

CensusTract	State	County	TotalPop	Men	Women	Hispanic	White	Black	Native	Asian	Pacific
1001020100	Alabama	Autauga	1948	940	1008	0.9	87.4	7.7	0.3	0.6	0.0
1001020200	Alabama	Autauga	2156	1059	1097	0.8	40.4	53.3	0.0	2.3	0.0
1001020300	Alabama	Autauga	2968	1364	1604	0.0	74.5	18.6	0.5	1.4	0.3
1001020400	Alabama	Autauga	4423	2172	2251	10.5	82.8	3.7	1.6	0.0	0.0
1001020500	Alabama	Autauga	10763	4922	5841	0.7	68.5	24.8	0.0	3.8	0.0
1001020600	Alabama	Autauga	3851	1787	2064	13.1	72.9	11.9	0.0	0.0	0.0

Census data: column metadata

Column information is given in `metadata`.

CensusTract	Census.tract.ID	numeric
State	State, DC, or Puerto Rico	string
County	County or county equivalent	string
TotalPop	Total population	numeric
Men	Number of men	numeric
Women	Number of women	numeric
Hispanic	% of population that is Hispanic/Latino	numeric
White	% of population that is white	numeric
Black	% of population that is black	numeric
Native	% of population that is Native American or Native Alaskan	numeric
Asian	% of population that is Asian	numeric
Pacific	% of population that is Native Hawaiian or Pacific Islander	numeric
Citizen	Number of citizens	numeric
Income	Median household income ($)	numeric
IncomeErr	Median household income error ($)	numeric

CensusTract	Census.tract.ID	numeric
IncomePerCap	Income per capita ($)	numeric
IncomePerCapErr	Income per capita error ($)	numeric
Poverty	% under poverty level	numeric
ChildPoverty	% of children under poverty level	numeric
Professional	% employed in management, business, science, and arts	numeric
Service	% employed in service jobs	numeric
Office	% employed in sales and office jobs	numeric
Construction	% employed in natural resources, construction, and maintenance	numeric
Production	% employed in production, transportation, and material movement	numeric
Drive	% commuting alone in a car, van, or truck	numeric
Carpool	% carpooling in a car, van, or truck	numeric
Transit	% commuting on public transportation	numeric
Walk	% walking to work	numeric
OtherTransp	% commuting via other means	numeric
WorkAtHome	% working at home	numeric
MeanCommute	Mean commute time (minutes)	numeric
Employed	% employed (16+)	numeric
PrivateWork	% employed in private industry	numeric
PublicWork	% employed in public jobs	numeric
SelfEmployed	% self-employed	numeric
FamilyWork	% in unpaid family work	numeric
Unemployment	% unemployed	numeric

Data wrangling

4. Remove summary rows from `election.raw` data: i.e.,

    - Federal-level summary into a `election_federal`.
    - State-level summary into a `election_state`.
    - Only county-level data is to be in `election`.

```
election_federal <- election.raw %>% filter(fips == "US")
election_state <- election.raw %>% filter(fips != "US" & is.na(county) & fips != "2000" & fips != "4610
election <- election.raw %>% filter(!is.na(county))

##Federal level summary
kable(election_federal %>% head)
```

county	fips	candidate	state	votes
NA	US	Donald Trump	US	62984825
NA	US	Hillary Clinton	US	65853516
NA	US	Gary Johnson	US	4489221
NA	US	Jill Stein	US	1429596
NA	US	Evan McMullin	US	510002
NA	US	Darrell Castle	US	186545

```
##State level summary
kable(election_state%>% head)
```

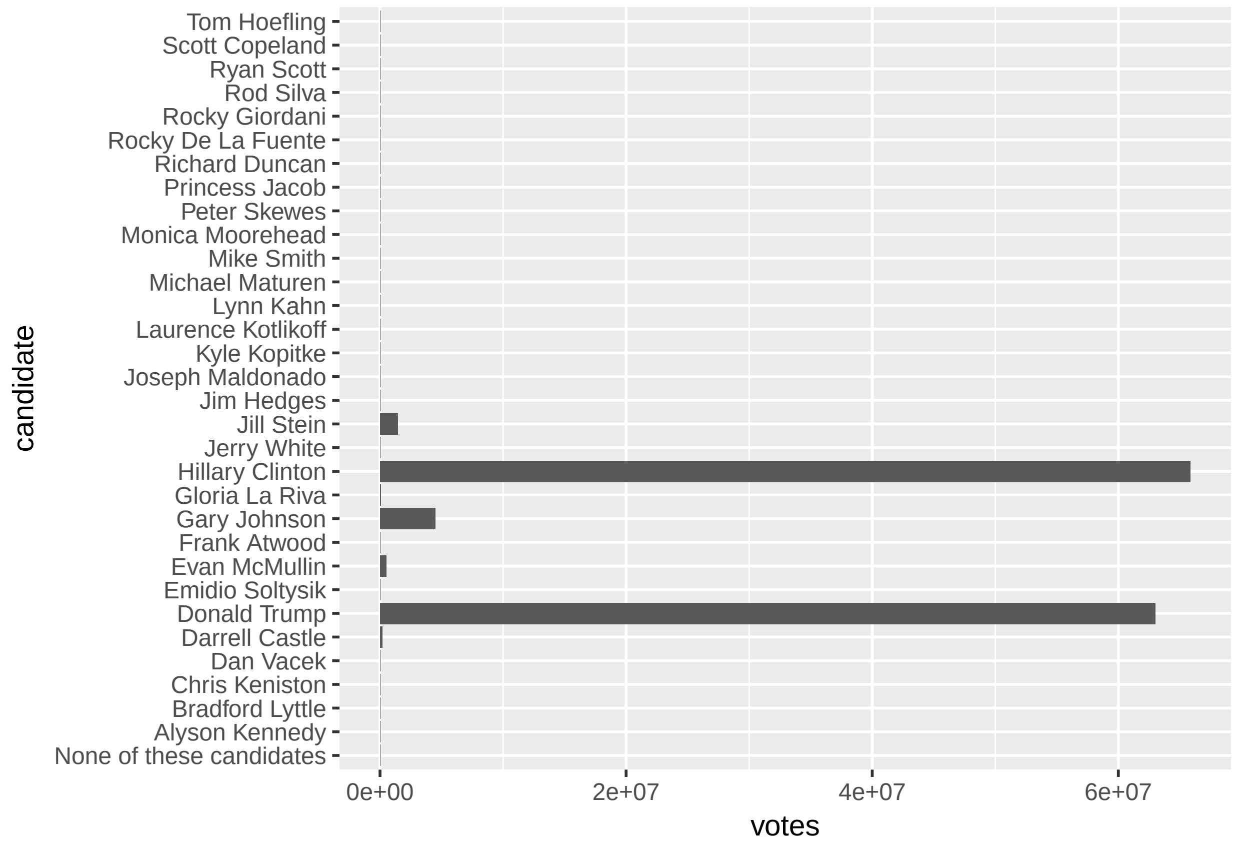
county	fips	candidate	state	votes
NA	CA	Hillary Clinton	CA	8753788
NA	CA	Donald Trump	CA	4483810
NA	CA	Gary Johnson	CA	478500
NA	CA	Jill Stein	CA	278657
NA	CA	Gloria La Riva	CA	66101
NA	FL	Donald Trump	FL	4617886

```
#county level data only
kable(election %>% head)
```

county	fips	candidate	state	votes
Los Angeles County	6037	Hillary Clinton	CA	2464364
Los Angeles County	6037	Donald Trump	CA	769743
Los Angeles County	6037	Gary Johnson	CA	88968
Los Angeles County	6037	Jill Stein	CA	76465
Los Angeles County	6037	Gloria La Riva	CA	21993
Cook County	17031	Hillary Clinton	IL	1611946

5. How many named presidential candidates were there in the 2016 election? Draw a bar chart of all votes received by each candidate

```
ggplot(election_federal, aes(x = candidate, y = votes,)) + geom_bar(stat="identity")+coord_flip()
```



As We can see, there were 32 candidates for the 2016 presidential elections.

6. Create variables `county_winner` and `state_winner` by taking the candidate with the highest proportion of votes. Hint: to create `county_winner`, start with `election`, group by `fips`, compute `total` votes, and `pct = votes/total`. Then choose the highest row using `top_n` (variable `state_winner` is similar).

```
election <- election %>% group_by(fips) %>% mutate(total = sum(votes))
election <- election %>% group_by(fips) %>% mutate(pct = votes/total)

## County winner variable
county_winner <- election %>% group_by(fips) %>% top_n(1)
```

```
## Selecting by pct
```

```
election_state <- election_state %>% group_by(fips) %>% mutate(total = sum(votes))
election_state <- election_state %>% group_by(fips) %>% mutate(pct = votes/total)

#state winner variable
state_winner <- election_state %>% group_by(fips) %>% top_n(1)
```

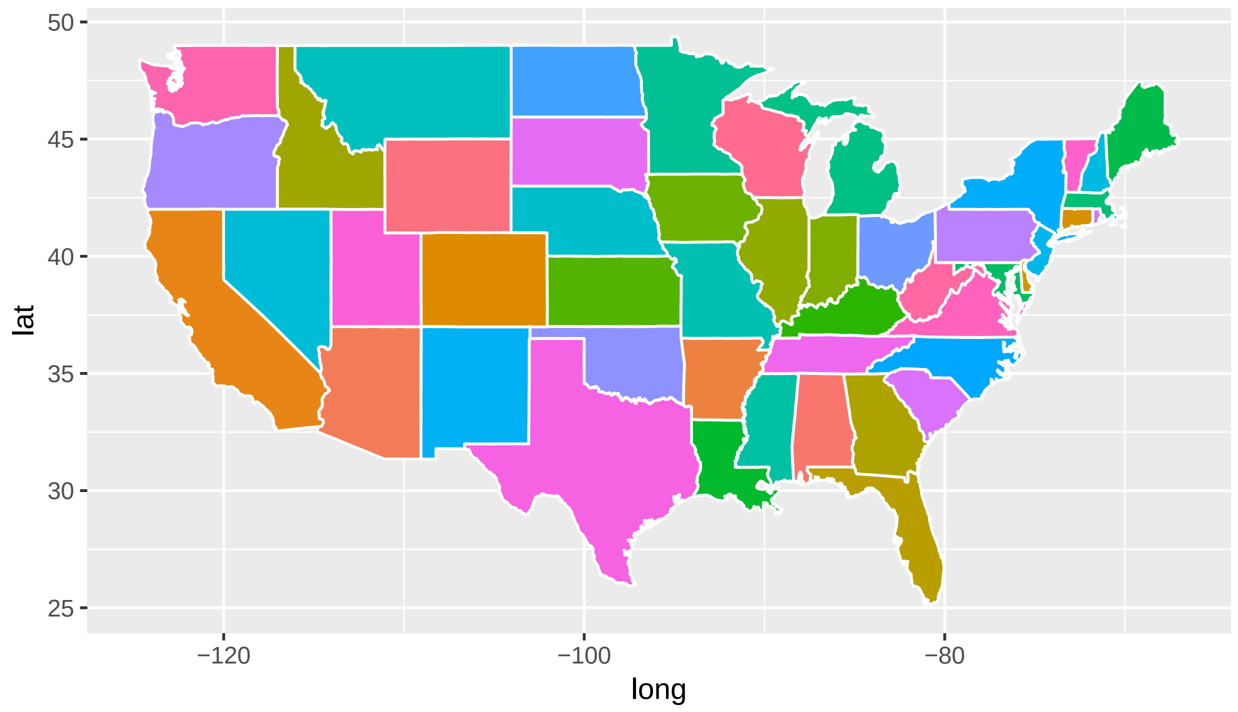
```
## Selecting by pct
```

Visualization

Visualization is crucial for gaining insight and intuition during data mining. We will map our data onto maps.

The R package `ggplot2` can be used to draw maps. Consider the following code.

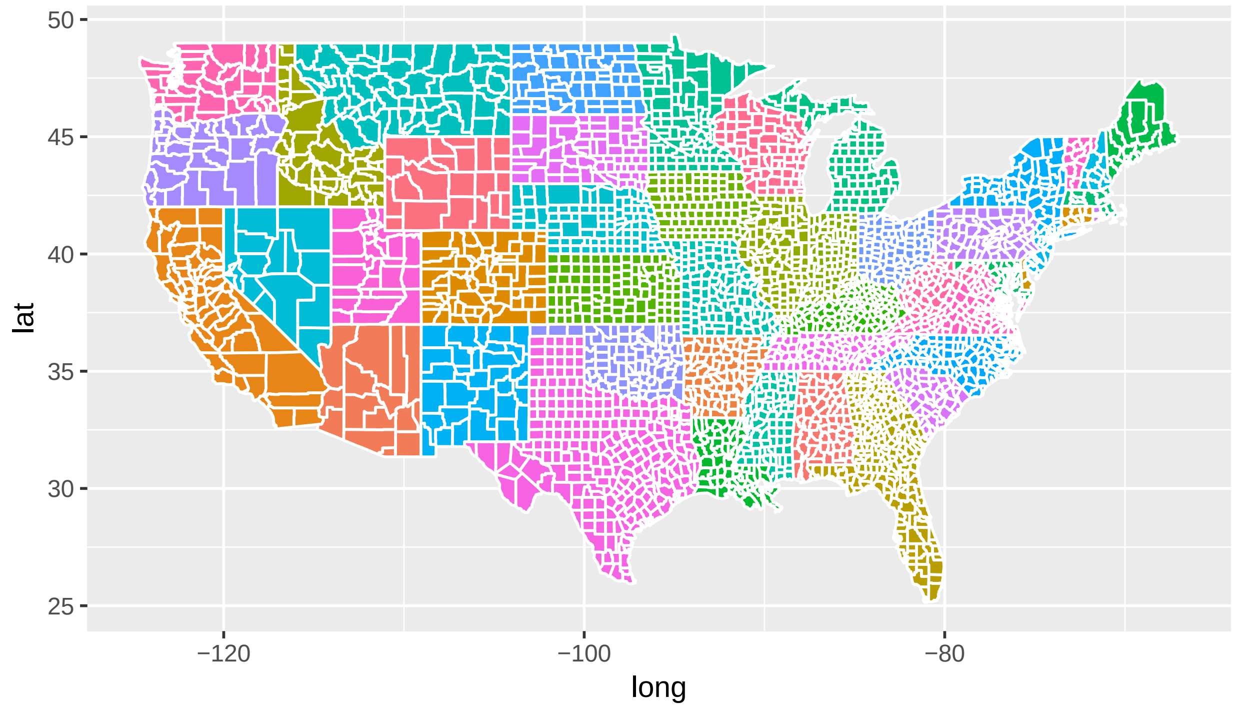
```
states = map_data("state")

ggplot(data = states) +
  geom_polygon(aes(x = long, y = lat, fill = region, group = group), color = "white") +
  coord_fixed(1.3) +
  guides(fill=FALSE)  # color legend is unnecessary and takes too long
```



The variable `states` contain information to draw white polygons, and fill-colors are determined by `region`.

7. Draw county-level map by creating `counties = map_data("county")`. Color by county

```
counties = map_data("county")

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



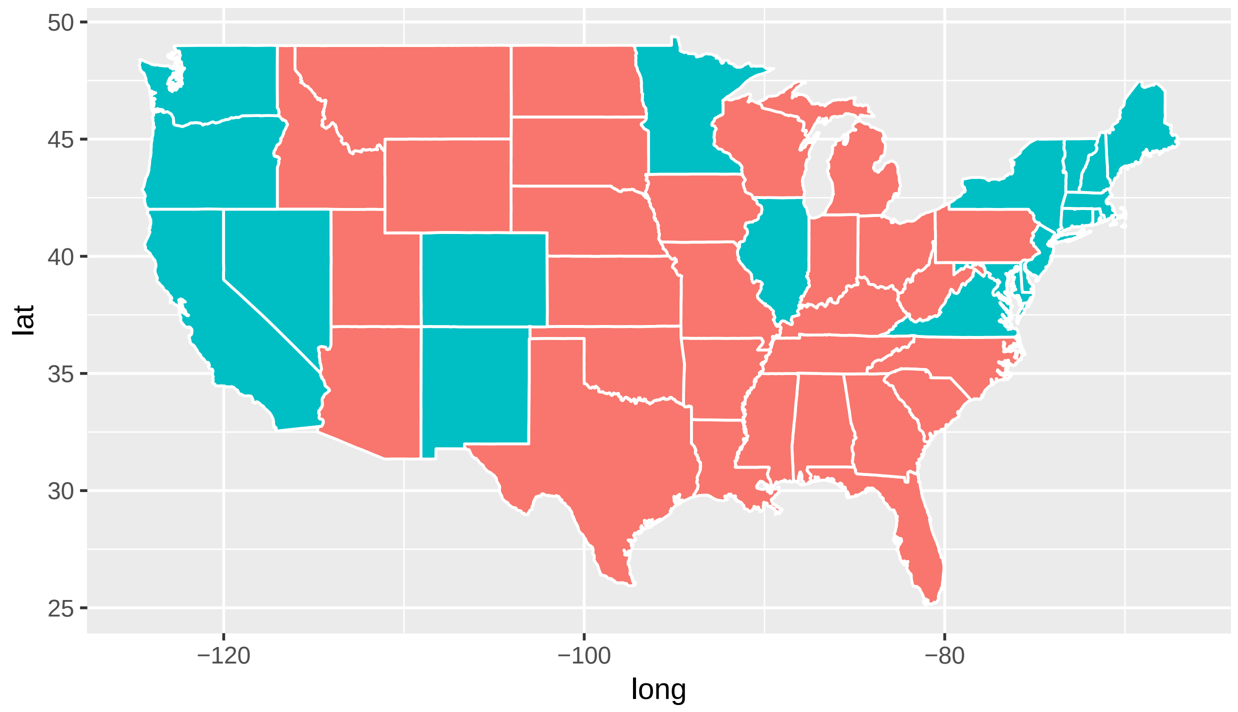
8. Now color the map by the winning candidate for each state. First, combine `states` variable and `state_winner` we created earlier using `left_join()`. Note that `left_join()` needs to match up values of states to join the tables; however, they are in different formats: e.g. `AZ` vs. `arizona`. Before using `left_join()`, create a common column by creating a new column for `states` named `fips = state.abb[match(some_column, some_function(state.name))]`. Replace `some_column` and `some_function` to complete creation of this new column. Then `left_join()`.

```
fips = state.abb[match(states$region, tolower(state.name))]

 states <- cbind(states, fips)

 states <- left_join(state_winner,states)
```

```
## Joining, by = "fips"
```

```
## Warning: Column `fips` joining factors with different levels, coercing to
## character vector
```

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



9. The variable `county` does not have `fips` column. So we will create one by pooling information from `maps::county.fips`. Split the `polyname` column to `region` and `subregion`. Use `left_join()` combine `county.fips` into `county`. Also, `left_join()` previously created variable `county_winner`.

```
counties <- left_join( maps::county.fips %>% separate(polyname, c("region","subregion"), sep = ",", ext
```

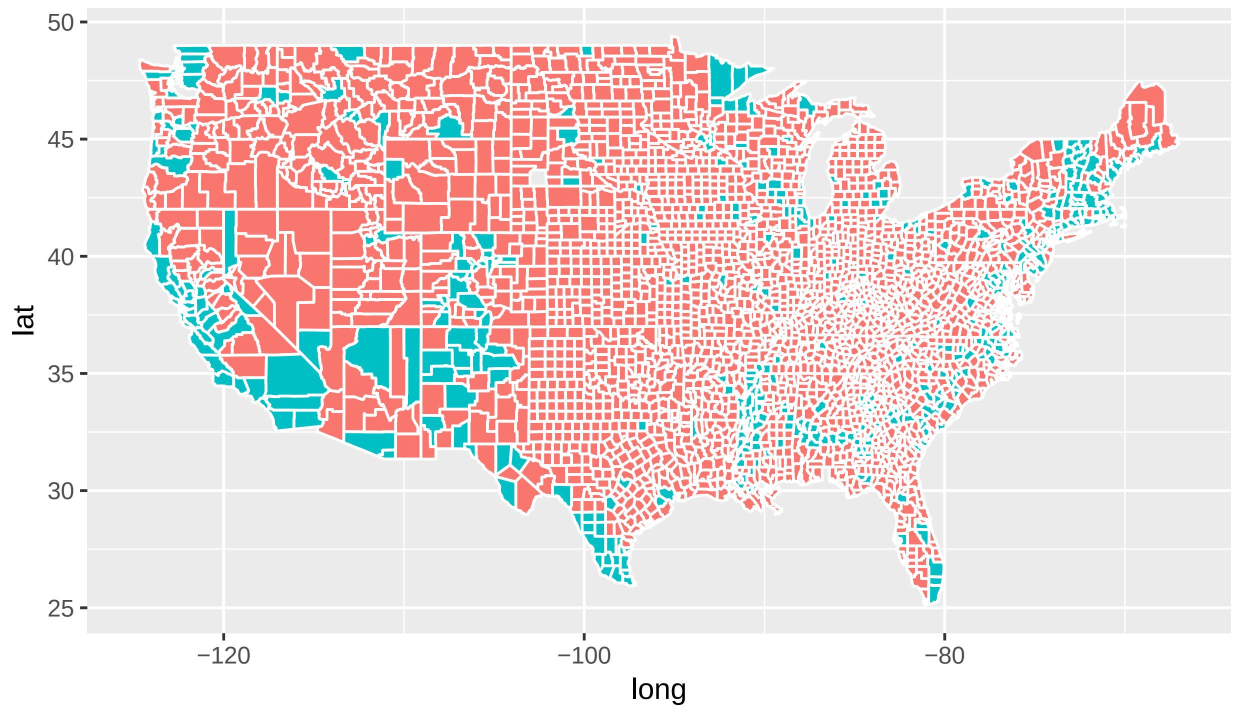
```
## Joining, by = c("region", "subregion")
```

```
counties$fips <- as.factor(counties$fips)

counties <- left_join(county_winner,counties)
```

```
## Joining, by = "fips"
```

```
## Warning: Column `fips` joining factors with different levels, coercing to
## character vector
```

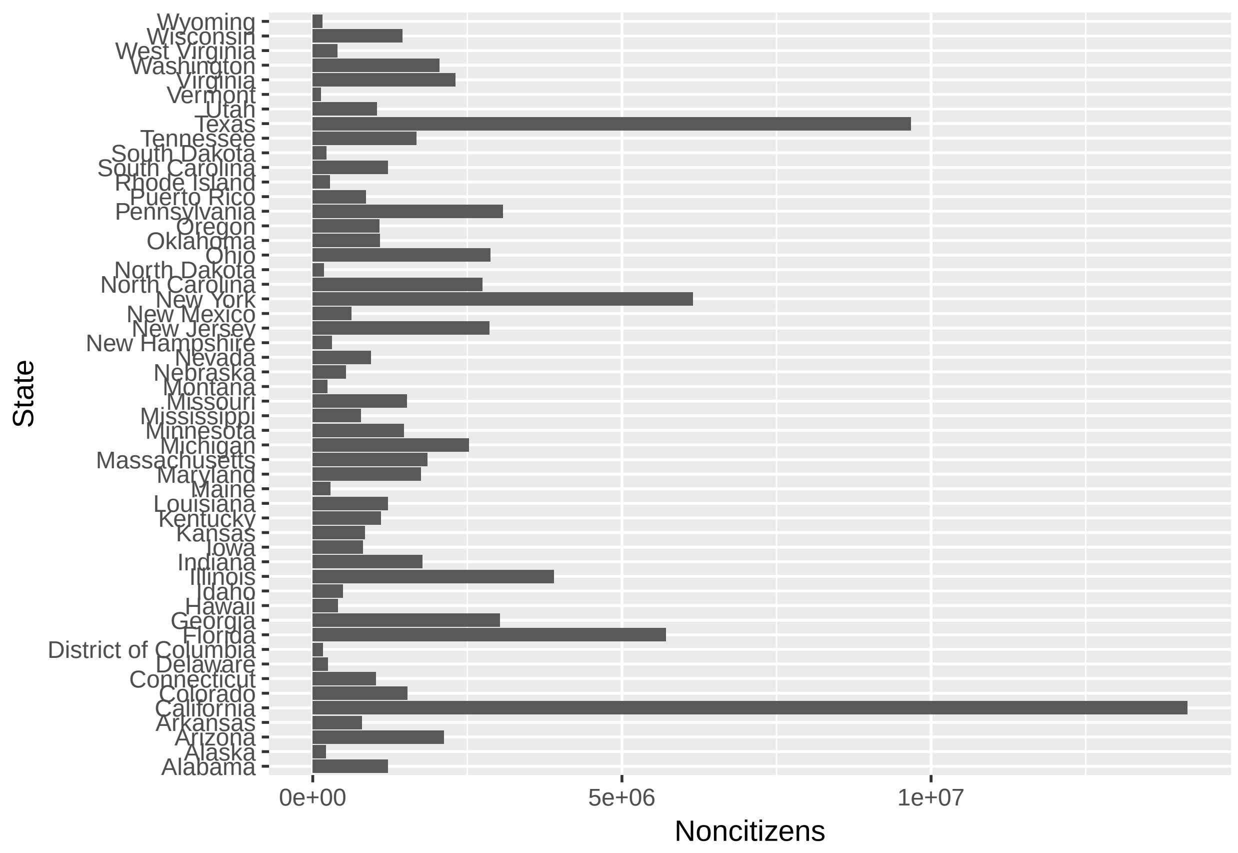
```
 ggplot(data = counties) +
   geom_polygon(aes(x = long, y = lat, fill = candidate, group = group), color = "white") +
   coord_fixed(1.3) +
   guides(fill=FALSE)
```



10. Create a visualization of your choice using `census` data. Many exit polls noted that demographics played a big role in the election. Use this Washington Post article and this R graph gallery for ideas and inspiration.

For this question, we decided to create a barplot showing the amount of non-citizens found in each state, looking to find which states have the highest population. This info might be helpful to notice voting patters for future elections.

```
census <- census %>% group_by(County) %>% mutate(Noncitizens = (TotalPop - Citizen))

 census <- census %>% group_by(County) %>% mutate(Noncitizen_percentage = Noncitizens/TotalPop)

 ggplot(census, aes(x = State, y = Noncitizens)) + geom_bar(stat="identity") + coord_flip()
```



11. The `census` data contains high resolution information (more fine-grained than county-level). In this problem, we aggregate the information into county-level data by computing `TotalPop`-weighted average of each attributes for each county. Create the following variables:

```
* _Clean census data `census.del`_:
  start with `census`, filter out any rows with missing values,
  convert {`Men`, `Employed`, `Citizen`} attributes to a percentages (meta data seems to be inaccurate)
  compute `Minority` attribute by combining {Hispanic, Black, Native, Asian, Pacific}, remove {`Walk`, `
  _Many columns seem to be related, and, if a set that adds up to 100%, one column will be deleted._


* _Sub-county census data, `census.subct`_:
  start with `census.del` from above, `group_by()` two attributes {`State`, `County`},
  use `add_tally()` to compute `CountyTotal`. Also, compute the weight by `TotalPop/CountyTotal`.


* _County census data, `census.ct`_:
  start with `census.subct`, use `summarize_at()` to compute weighted sum


* _Print few rows of `census.ct`_:
```

```r
census = read.csv("census.csv") %>% as.tbl
census$CensusTract = as.factor(census$CensusTract)

census.del <- census %>% filter(complete.cases(census))

census.del <- census.del %>% mutate(Men = Men/TotalPop)
census.del <- census.del %>% mutate(Employed = Employed/TotalPop)
census.del <- census.del %>% mutate(Citizen = Citizen/TotalPop)

census.del <- census.del %>% mutate(Minority = Hispanic + Black + Native + Asian + Pacific)

census.del <- census.del %>% select(-c(Walk, PublicWork, Construction))

census.del<- census.del %>% select(-Hispanic,-Black,-Native,-Asian,-Pacific,-Women)

census.subct <- census.del %>% group_by(State, County) %>% add_tally(TotalPop)

census.subct <- census.subct %>% mutate(CountyTotal = n)

census.subct<- census.subct %>% select(-n)

census.subct <- census.subct %>% mutate(weight = TotalPop/CountyTotal)

census.ct<-census.subct %>% summarise_at(vars(Men:CountyTotal), funs(weighted.mean(., weight)))

census.ct<-as.data.frame(census.ct)

census.subct<-as.data.frame(census.subct)

kable(census.ct) %>% head
```

```
## [1] "State                  County                                   Men        White     Citizen
## [2] "---------------------  -----------------------------------  ----------  -----------  ----------
## [3] "Alabama                Autauga                               0.4843266   75.7882273   0.7374912
## [4] "Alabama                Baldwin                               0.4884866   83.1026163   0.7569406
## [5] "Alabama                Barbour                               0.5382816   46.2315944   0.7691222
## [6] "Alabama                Bibb                                  0.5341090   74.4998894   0.7739781
```

Dimensionality reduction

12. Run PCA for both county & sub-county level data. Save the first two principle components PC1 and PC2 into a two-column data frame, call it `ct.pc` and `subct.pc`, respectively. What are the most prominent loadings?

```
#for Subcounty
pr.out_sub=prcomp(census.subct[,4:31],scale=TRUE)
subct.pc<-pr.out_sub$rotation[,1:2] ##gives out PC1 and PC2

max(abs(subct.pc)[1:28]) ##Value corresponds to IncomePerCap (PC1)
```

```
## [1] 0.3181199
```

```
max(abs(subct.pc)[29:56]) ## Value corresponds to Transit (PC2)
```

```
## [1] 0.3937091
```

```
#County level
pr.out_ct=prcomp(census.ct[,3:28],scale=TRUE)
ct.pc<-pr.out_ct$rotation[,1:2] ##gives out PC1 and PC2

max(abs(ct.pc)[1:26]) ## Value corresponds to IncomePerCar
```

```
## [1] 0.3530767
```

```
max(abs(ct.pc)[27:52]) ## Value corresponds to IncomeErr
```

```
## [1] 0.3145022
```

As we see, the most prominent loadings at the sub county level are IncomePerCap (PC1) and Transit (PC2). For the county level, we get IncomePerCar(PC1) and IncomeErr(PC2). # Clustering

13. With `census.ct`, perform hierarchical clustering using Euclidean distance metric complete linkage to find 10 clusters. Repeat clustering process with the first 5 principal components of `ct.pc`. Compare and contrast clusters containing San Mateo County. Can you hypothesize why this would be the case?

```
#standardizing census.ct
scaled_ct = scale(census.ct[, -c(1,2)], center=TRUE, scale=TRUE)
#evaluating distance w/ eucledian method
census.dist = dist(scaled_ct,method="euclidean")


set.seed(1)
census.hclust = hclust(census.dist) # complete linkage
census.hclust=cutree(census.hclust, 10) ## k=10
table(census.hclust)
```

```
## census.hclust
##    1    2    3    4    5    6    7    8    9   10
## 2632  501    6    7    5    1   11   13   38    4
```

```
#now for top 5 PC

scaled_ct_PCA<-scale(pr.out_ct$x[,1:5])
PCA_dist<-dist(scaled_ct_PCA)
census.hclust.PCA<- cutree(hclust(PCA_dist),k=10)
 table(census.hclust.PCA)
```

```
## census.hclust.PCA
##    1    2    3    4    5    6    7    8    9   10
## 2441  525   97    6    8   31    5   18    7   80
```

Classification

In order to train classification models, we need to combine `county_winner` and `census.ct` data. This seemingly straightforward task is harder than it sounds. Following code makes necessary changes to merge them into `election.cl` for classification.

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

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

## saves meta information to attributes
```

```r
attr(election.cl, "location") = election.cl %>% select(c(county, fips, state, votes, pct))
election.cl = election.cl %>% select(-c(county, fips, state, votes, pct))
```

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

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

Using the following code, define 10 cross-validation folds:

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

Using the following error rate function:

```r
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","knn","lda")
```

Classification: native attributes

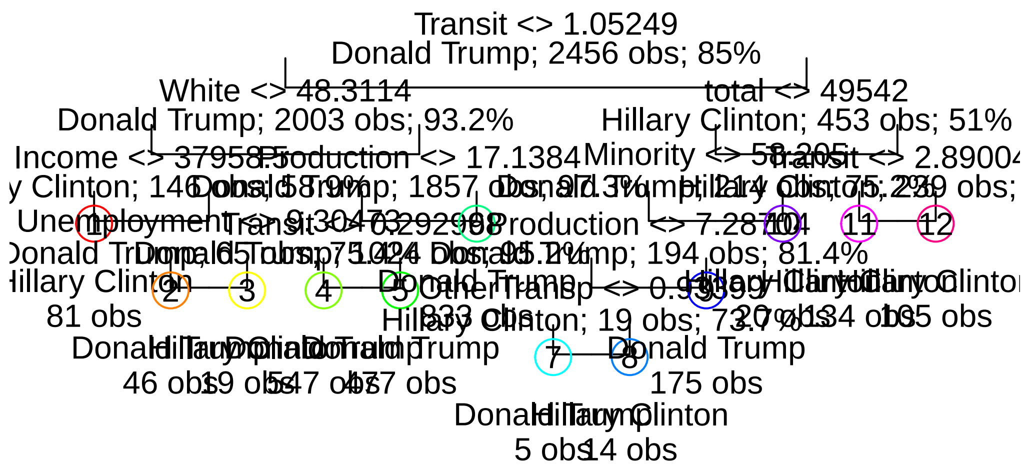
13. Decision tree: train a decision tree by `cv.tree()`. Prune tree to minimize misclassification. Be sure to use the `folds` from above for cross-validation. Visualize the trees before and after pruning. Save training and test errors to `records` variable.

```r
tmpwinner = county_winner %>% ungroup %>%
   mutate(state = state.name[match(state, state.abb)]) %>%
   mutate_at(vars(state, county), tolower) %>%
   mutate(county = gsub(" county| columbia| city| parish", "", county))
tmpcensus = census.ct %>% mutate_at(vars(State, County), tolower)

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

attr(election.cl, "location") = election.cl %>% select(c(county, fips, state, votes, pct))

election.cl = election.cl %>% select(-c(county, fips, state, votes, pct))


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


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

```r
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","knn","lda")

 #actual stuff

cv.tree <- tree(trn.cl$candidate~.,trn.cl)
summary(cv.tree)
```

```
##
## Classification tree:
## tree(formula = trn.cl$candidate ~ ., data = trn.cl)
## Variables actually used in tree construction:
## [1] "Transit"      "White"        "Income"       "Unemployment"
## [5] "Production"   "total"        "Minority"     "OtherTransp"
## Number of terminal nodes:  12
## Residual mean deviance:  0.3612 = 882.8 / 2444
## Misclassification error rate: 0.06393 = 157 / 2456
```

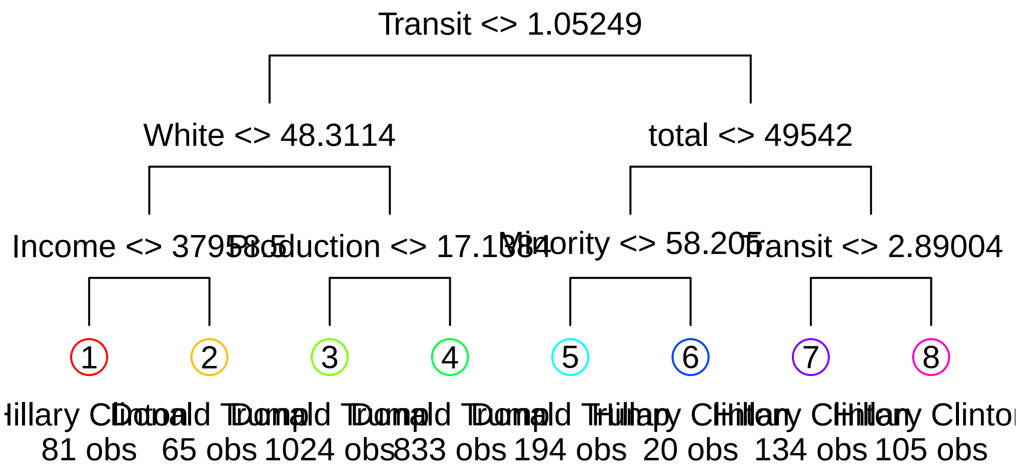
```r
 ##Un-pruned
library (rpart)
library("maptree")
draw.tree(cv.tree,nodeinfo=TRUE)
```



```r
#pruned w K-cross means we have to use cv.tree()
cv = cv.tree(cv.tree, folds, FUN=prune.misclass, K=10)
best.cv = cv$size[which.min(cv$dev)]
best.cv
```

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

```r
pruned.tree<-prune.tree(cv.tree,best=best.cv)
draw.tree(pruned.tree)
```



```
prediction<- predict(pruned.tree,tst.cl,type='class')
test.error<-calc_error_rate(prediction, tst.cl$candidate)

train.prediction<-predict(pruned.tree,trn.cl, type='class')
train.error<-calc_error_rate(train.prediction,trn.cl$candidate)

records[1,]<-c(train.error,test.error)
records
```

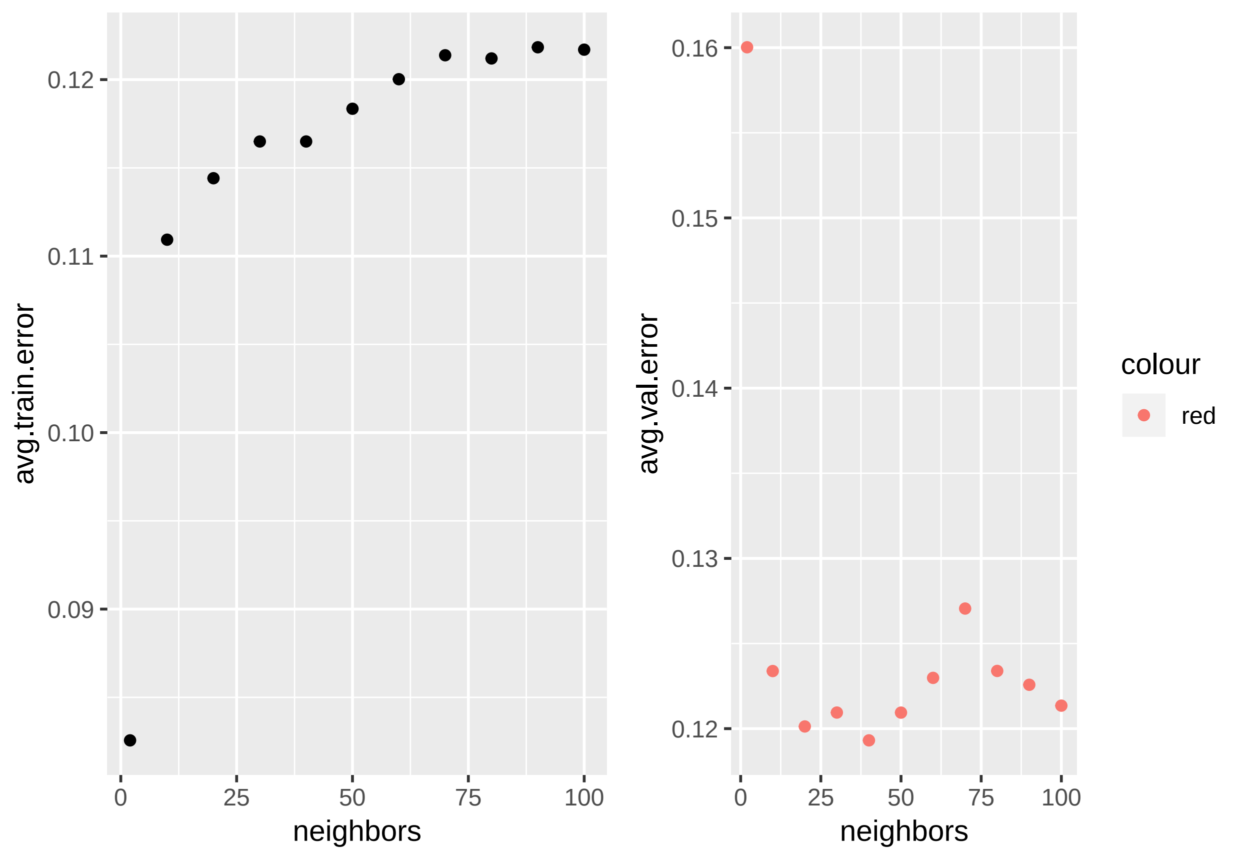
```
##      train.error test.error
## tree  0.07247557 0.08469055
## knn           NA         NA
## lda           NA         NA
```

14. K-nearest neighbor: train a KNN model for classification. Use cross-validation to determine the best number of neighbors, and plot number of neighbors vs. resulting training and validation errors. Compute test error and save to `records`.

```
kvec = c(2, seq(10,100, length.out=10))

do.chunk <- function(chunkid, folddef, Xdat, Ydat, k){
    train = (folddef!=chunkid)
    Xtr = Xdat[train,]
    Ytr = Ydat[train]

    Xvl = Xdat[!train,]
    Yvl = Ydat[!train]
    ## get classifications for current training chunks
    predYtr = knn(train = Xtr, test = Xtr, cl = Ytr, k = k)
    ## get classifications for current test chunk
    predYvl = knn(train = Xtr, test = Xvl, cl = Ytr, k = k)

    data.frame(train.error = calc_error_rate(predYtr, Ytr),
        val.error = calc_error_rate(predYvl, Yvl))
}

tmpDF <- data.frame(neighbors = 0,avg.train.error = 0,
        avg.val.error = 0)

for (i in kvec){

    tmp = ldply(1:nfold, do.chunk, # Apply do.chunk() function to each fold
      folddef=folds, Xdat=trn.cl[3:28], Ydat=trn.cl$candidate , k=i)


    avg.val.error = mean(tmp$val.error)
    avg.train.error = mean(tmp$train.error)

    tmpDF <- rbind(tmpDF,data.frame(neighbors = i,avg.train.error = avg.train.error,
        avg.val.error = avg.val.error))

    cat("Average test error: ", avg.val.error, "With: ", i, " neighbors.\n")
}
```

```
## Average test error:  0.1600232 With:  2  neighbors.
## Average test error:  0.1233831 With:  10  neighbors.
## Average test error:  0.1201278 With:  20  neighbors.
## Average test error:  0.1209441 With:  30  neighbors.
## Average test error:  0.1193114 With:  40  neighbors.
## Average test error:  0.1209408 With:  50  neighbors.
## Average test error:  0.1229816 With:  60  neighbors.
## Average test error:  0.1270533 With:  70  neighbors.
## Average test error:  0.1233897 With:  80  neighbors.
## Average test error:  0.1225751 With:  90  neighbors.
## Average test error:  0.1213506 With:  100  neighbors.
```

```
tmpDF = tmpDF[-1,]

p1 <- qplot(neighbors, avg.train.error, data = tmpDF)
p2 <- qplot(neighbors, avg.val.error, data = tmpDF, colour= 'red')

grid.arrange(p1, p2, nrow = 1)
```



```
#Pick lowest val error for k in knn. #THIS IS DYNAMIC

Xtr = trn.cl[3:28]
Ytr = trn.cl$candidate

Xvl = tst.cl[3:28]
Yvl = tst.cl$candidate



predYtr = knn(train = Xtr, test = Xtr, cl = Ytr, k = 40)#might have to change k

predYvl = knn(train = Xtr, test = Xvl, cl = Ytr, k = 40)#might have to change k

records[2,]<-c(calc_error_rate(predYtr, Ytr),calc_error_rate(predYvl, Yvl))
records
```

```
##      train.error test.error
## tree  0.07247557 0.08469055
## knn   0.11482085 0.13355049
## lda           NA         NA
```

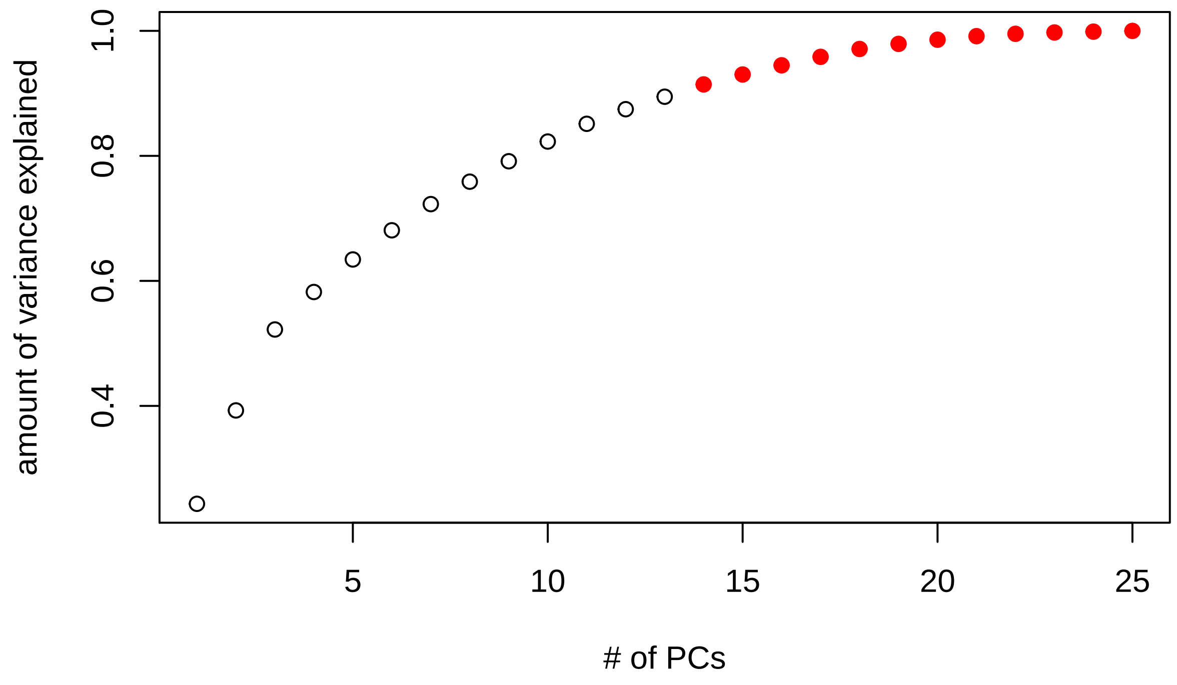
Classification: principal components

Instead of using the native attributes, we can use principal components in order to train our classification models. After this section, a comparison will be made between classification model performance between using native attributes and principal components.

```
pca.records = matrix(NA, nrow=3, ncol=2)
colnames(pca.records) = c("train.error","test.error")
rownames(pca.records) = c("tree","knn","lda")
```

15. Compute principal components from the independent variables in training data. Then, determine the number of minimum number of PCs needed to capture 90% of the variance. Plot proportion of variance explained.

```
library(dplyr)
pca.records = matrix(NA, nrow=3, ncol=2)
 colnames(pca.records) = c("train.error","test.error")
 rownames(pca.records) = c("tree","knn","lda")


 trn.cl <- trn.cl %>%dplyr::select(-total)
 PC.trn<-prcomp(trn.cl[2:27],scale=TRUE)

 explained=0
 values<-PC.trn$sdev^2
 sum=sum(values)
 y=NULL
 for (i in 1:25){
   explained=explained + (values[i]/sum)
   y<-rbind(y,explained)}

 plot(y, col=ifelse(y>=.90, "red", "black"),pch=ifelse(y>=.90,19,1),xlab='# of PCs',ylab='amount of var
```



16. Create a new training data by taking class labels and principal components. Call this variable `tr.pca`. Create the test data based on principal component loadings: i.e., transforming independent variables in test data to principal components space. Call this variable `test.pca`.

```
pca.records = matrix(NA, nrow=3, ncol=2)
colnames(pca.records) = c("train.error","test.error")
rownames(pca.records) = c("tree","knn","lda")

tr.pca<-(PC.trn$x)
tr.pca<-as.data.frame(tr.pca)
tr.pca<-mutate(tr.pca,candidate = trn.cl$candidate)

test.pca <-(prcomp(tst.cl[,2:27],scale = TRUE)$x)
test.pca<-as.data.frame(test.pca)
test.pca<-mutate(test.pca,candidate=tst.cl$candidate)

length(test.pca)
```

```
## [1] 27
```

```
length(tr.pca)
```

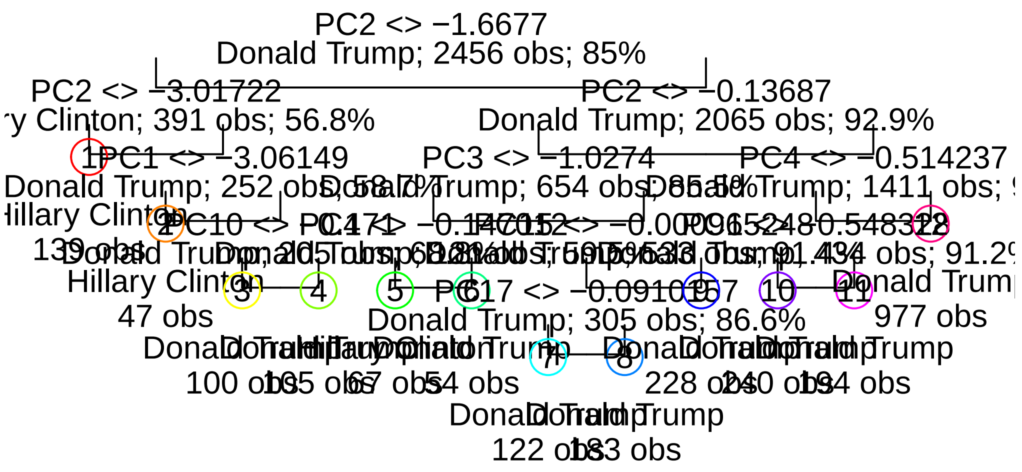
```
## [1] 27
```

17. Decision tree: repeat training of decision tree models using principal components as independent variables. Record resulting errors.

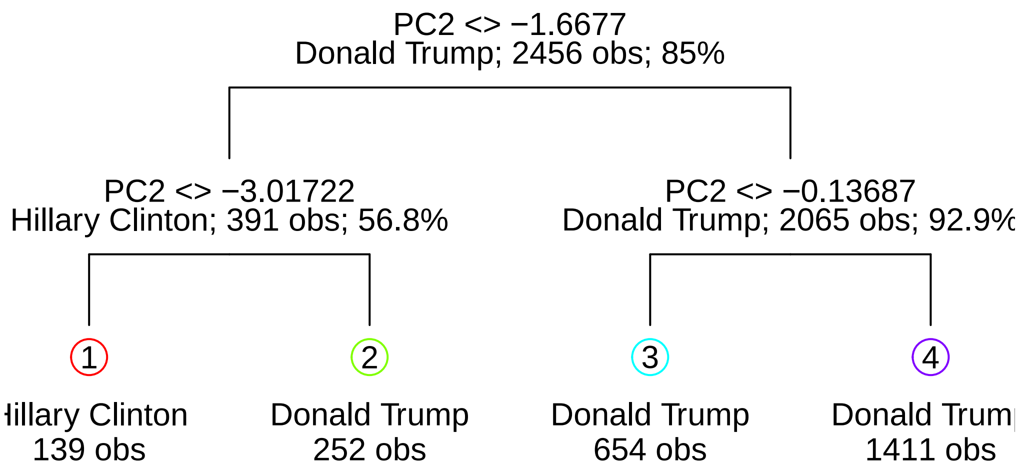
```
tree.2<-tree(candidate~.,data = tr.pca)
summary(tree.2)
```

```
##
## Classification tree:
## tree(formula = candidate ~ ., data = tr.pca)
## Variables actually used in tree construction:
## [1] "PC2"  "PC1"  "PC10" "PC3"  "PC4"  "PC15" "PC17"
## Number of terminal nodes:  12
## Residual mean deviance:  0.4685 = 1145 / 2444
## Misclassification error rate: 0.0908 = 223 / 2456
```

```
draw.tree(tree.2,nodeinfo=TRUE)
```



```r
cv.2 = cv.tree(tree.2, folds, FUN=prune.misclass, K=10)
best.2 = cv.2$size[which.min(cv.2$dev)]


pruned.tree.2<-prune.tree(tree.2,best=best.2)
draw.tree(pruned.tree.2,nodeinfo=TRUE)
```



```
train.prediction.2<-predict(pruned.tree.2,tr.pca[-length(tr.pca)],type='class')
train.error.2<-calc_error_rate(train.prediction.2,tr.pca$candidate)

test.prediction.2<- predict(pruned.tree.2,test.pca[-length(test.pca)],type='class')
test.error.2<-calc_error_rate(test.prediction.2, test.pca$candidate)

#keeping track of errors
pca.records[1,]<-c(train.error.2,test.error.2)
pca.records
```

```
##      train.error test.error
## tree    0.110342  0.2149837
## knn           NA         NA
## lda           NA         NA
```

Interpretation & Discussion

19. This is an open question. Interpret and discuss any insights gained and possible explanations. Use any tools at your disposal to make your case: visualize errors on the map, discuss what does/doesn't seems reasonable based on your understanding of these methods, propose possible directions (collecting additional data, domain knowledge, etc)

To summarise we preformed ten fold cross validation on the county cencus data set using the native attributes and the principal components as independent variables. We trained decision trees and KNN (K nearest neighbors) classifiers with the two variations of the datasets; the results shown:

```
records
```

```
##      train.error test.error
## tree  0.07247557 0.08469055
## knn   0.11482085 0.13355049
## lda           NA         NA
```

```
pca.records
```

```
##      train.error test.error
## tree    0.110342  0.2149837
## knn           NA         NA
## lda           NA         NA
```

Taking it further

20. Propose and tackle at least one interesting question. Be creative! Some possibilities are:

    - Data preprocessing: we aggregated sub-county level data before performing classification. Would classification at the sub-county level before determining the winner perform better? What implicit assumptions are we making?
    - Feature engineering: would a non-linear classification method perform better? Would you use native features or principal components?
    - Additional classification methods: logistic regression, LDA, QDA, SVM, random forest, etc. (You may use methods beyond this course). How do these compare to KNN and tree method?
    - Bootstrap: Perform boostrap to generate plots similar to Figure 4.10/4.11. Discuss the results.

We use subcounty data to train a decision tree with native attributes and then compare it the tree trained for the aggregated data.

```r
records2 = matrix(NA, nrow=1, ncol=2)
colnames(records2) = c("train.error","test.error")
rownames(records2) = c("tree")

pca.records2 = matrix(NA, nrow=1, ncol=2)
colnames(pca.records2) = c("train.error","test.error")
rownames(pca.records2) = c("tree")

#-------------------------------------------
tmpwinner = county_winner %>% ungroup %>%
  mutate(state = state.name[match(state, state.abb)]) %>%
  mutate_at(vars(state, county), tolower) %>%
  mutate(county = gsub(" county| columbia| city| parish", "", county))
tmpcensus = census.subct %>% mutate_at(vars(State, County), tolower)

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

attr(election.cl, "location") = election.cl %>% dplyr::select(c(county, fips, state, votes, pct))

election.cl = election.cl %>% dplyr::select(-c(county, fips, state, votes, pct))


set.seed(69)
```

```r
n = nrow(election.cl)
in.trn= sample.int(n, 0.8*n)
trn.cl = election.cl[ in.trn,]
tst.cl = election.cl[-in.trn,]
```