Final Project - 2016 Election Prediction

PSTAT131: Catherine Miao, Joheun Kang

5/26/2019

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?

We suppose three main reasons why election forecasting can be a very difficult problem to tackle. First of all, a great number of American citizens are not motivated enough to vote, since they don't believe their individual vote could make a difference. According to the U.S. Census Bureau, overall there are only about 61% of the U.S. citizens voted in the 2016 election. This turnout might not represent the most prevailing opinion of the entire U.S. population, leaving great uncertainty for forecasting. Besides, many citizens would not disclose their honest opinions in the election poll.Some of the voters even change their mind at the election day. Thus, such popular situations may also lead to the erroneous election prediction. In addition, numerous political institutions, such as electoral college, political parties, and interest groups, have great influence on the election outcome. For instance, the effect that numerous political campaigns pose on voter behavior complicates the election result.

2.

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

The uniqueness about Nate Silver's approach is that he employs a time series method to predict voter behaviors. His logic is mainly based on the fact that voters' opinions change across time. His time series approach was to select a baseline of how people in the 50 states would vote if the election were held on any particular day. While forecasting the election outcome, he separates the prediction to national and state level and incorporates variations from polling results as well as unobserved shocks into consideration. Employing Bayes Theorem, he built the mathematical formula: actual percentage + the house effect + sampling variation. He also conducts the state-level prediction with the hierarchical modeling. In addition, Nate Silver ignored third party candidates in his prediction, considering only the race between Obama and Romney.

3.

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

The surprising victory of Donald Trump indicates that the election poll did not coincide with the true prevailing public opinion in the 2016 presidential election. As many of Trump's public speeches were resentful towards the minorities, immigrants, or females, they are greatly against cultural conventions. As a result,

social media exaggerate the negativity of Trump's public speech, imposing an over-pessimistic impression on the public that Trump will lose the election. The public receive the messages and generate a "confirmation bias" in their mind. As a result, republicans and Trump's supporters have to disclose their political opinion in order to avoid being criticized by the media and the general public. Another reason leading to the false prediction of the election result is that a significant portion of the population did not like either of the candidates. Consequently, they decide to vote for other political parties or randomly vote for one on the actual election day.

Data

```
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:

```
kable(election.raw %>% 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

Census data

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

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

CensusTract	State	County	TotalPop	Men	Women	Hispanic	White	Black	Native	Asian	Pacific	C
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

CensusTract	Census.tract.ID	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
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") # only keep fips is US

election.raw1 <- election.raw %>% filter(fips!="US") # remove fips not US from original dataset

election_state <- election.raw1 %>% filter(is.na(county) == T) # remove county-level data

election <- election.raw1 %>% filter(is.na(county)==F)
```

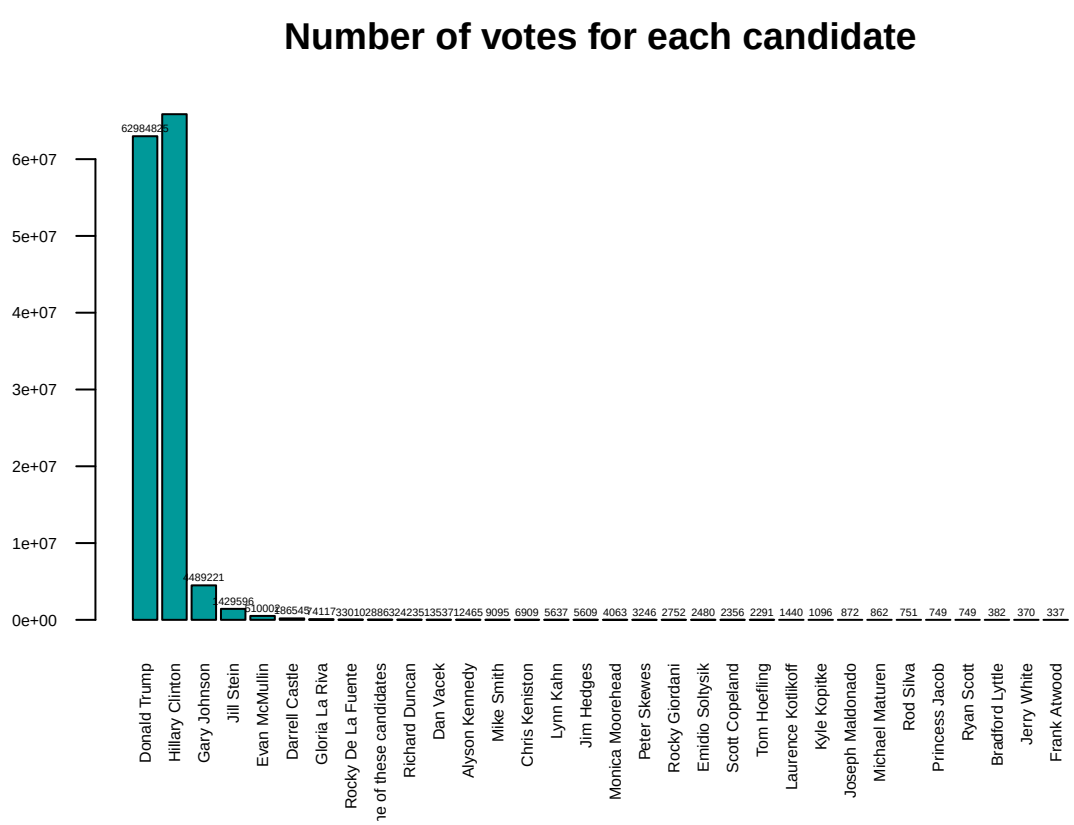
5.

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

```
levels(election.raw$candidate)
```

```
##  [1] " None of these candidates" "Alyson Kennedy"
##  [3] "Bradford Lyttle"           "Chris Keniston"
##  [5] "Dan Vacek"                 "Darrell Castle"
##  [7] "Donald Trump"              "Emidio Soltysik"
##  [9] "Evan McMullin"             "Frank Atwood"
## [11] "Gary Johnson"              "Gloria La Riva"
## [13] "Hillary Clinton"           "Jerry White"
## [15] "Jill Stein"                "Jim Hedges"
## [17] "Joseph Maldonado"          "Kyle Kopitke"
## [19] "Laurence Kotlikoff"        "Lynn Kahn"
## [21] "Michael Maturen"           "Mike Smith"
## [23] "Monica Moorehead"          "Peter Skewes"
## [25] "Princess Jacob"            "Richard Duncan"
## [27] "Rocky De La Fuente"        "Rocky Giordani"
## [29] "Rod Silva"                 "Ryan Scott"
## [31] "Scott Copeland"            "Tom Hoefling"
```

```
bar <- barplot(election_federal$votes,main = "Number of votes for each candidate",cex.axis = 0.5, names
#ylim = 6.1e+7
text(x=bar, y=election_federal$votes+1000000 , paste(election_federal$votes,sep="") ,cex=0.3,xpd=F)
```



There are 32 named presidential candidates in the 2016 election.

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).

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

```
## # A tibble: 3,111 x 7
## # Groups:   fips [3,111]
##    county             fips  candidate       state   votes   total   pct
##    <fct>              <fct> <fct>           <fct>   <int>   <int> <dbl>
##  1 Los Angeles County 6037  Hillary Clinton CA    2464364 3421533 0.720
##  2 Cook County        17031 Hillary Clinton IL    1611946 2156395 0.748
##  3 Maricopa County    4013  Donald Trump    AZ     747361 1536743 0.486
##  4 Harris County      48201 Hillary Clinton TX     707914 1305434 0.542
##  5 San Diego County   6073  Hillary Clinton CA     735476 1291078 0.570
##  6 Orange County      6059  Hillary Clinton CA     609961 1186203 0.514
##  7 King County        53033 Hillary Clinton WA     718322  998499 0.719
##  8 Miami-Dade County  12086 Hillary Clinton FL     624146  980204 0.637
##  9 Broward County     12011 Hillary Clinton FL     553320  831950 0.665
## 10 Kings County       36047 Hillary Clinton NY     640553  800393 0.800
## # ... with 3,101 more rows
```

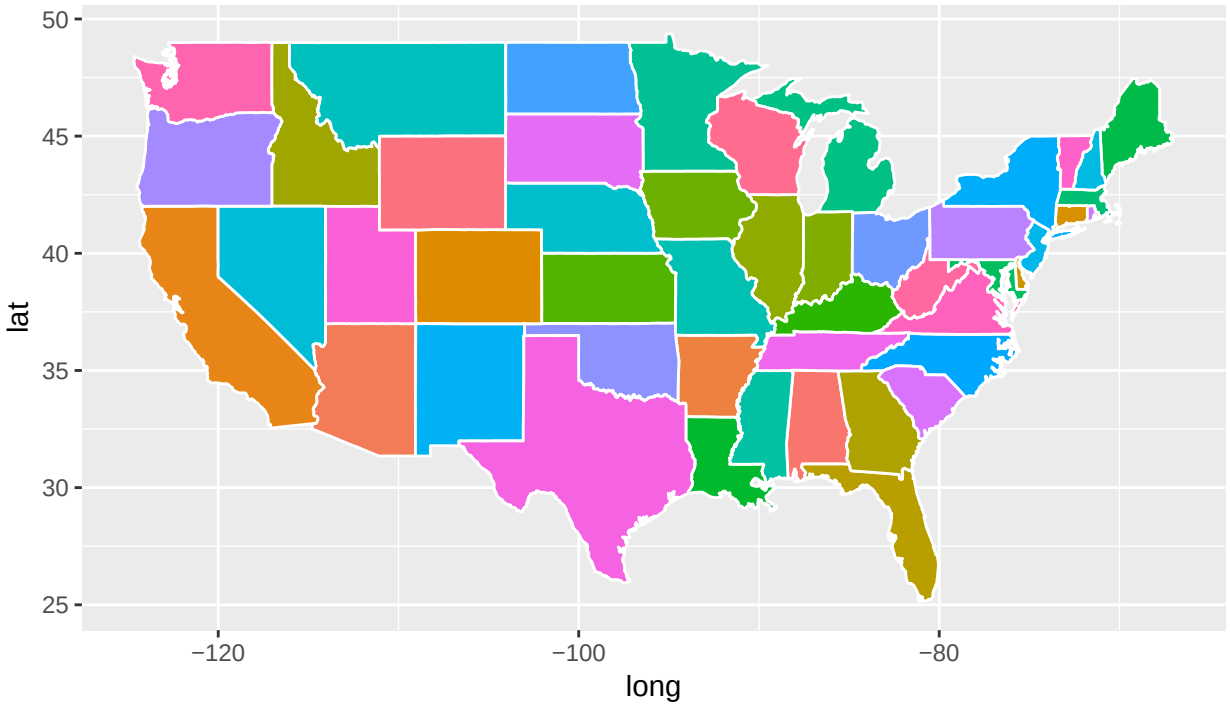
```
state_winner <- election_state %>% group_by(state) %>% mutate(total=sum(votes), pct=votes/total) %>% to
```

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
```

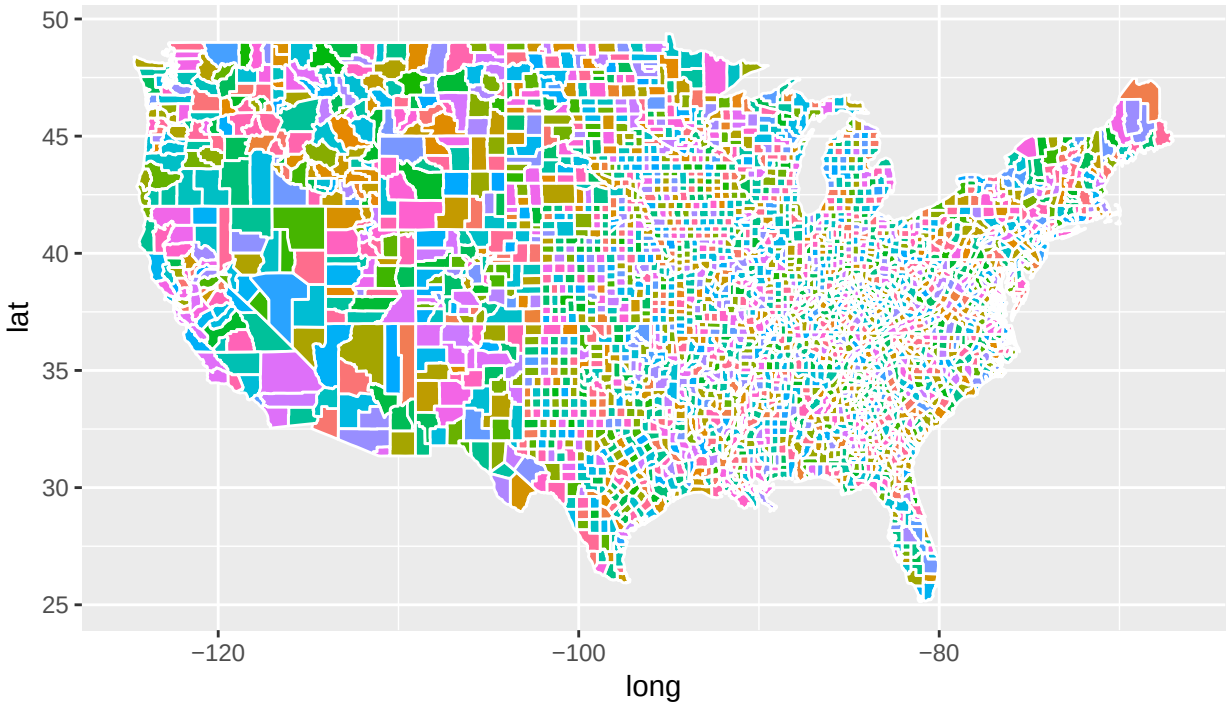


The variable `states` contain information to draw white polygons, and fill-colors are determined by `region`.

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 = subregion, group = group), color = "white") +
  coord_fixed(1.3) +
  guides(fill=FALSE)
```



8.

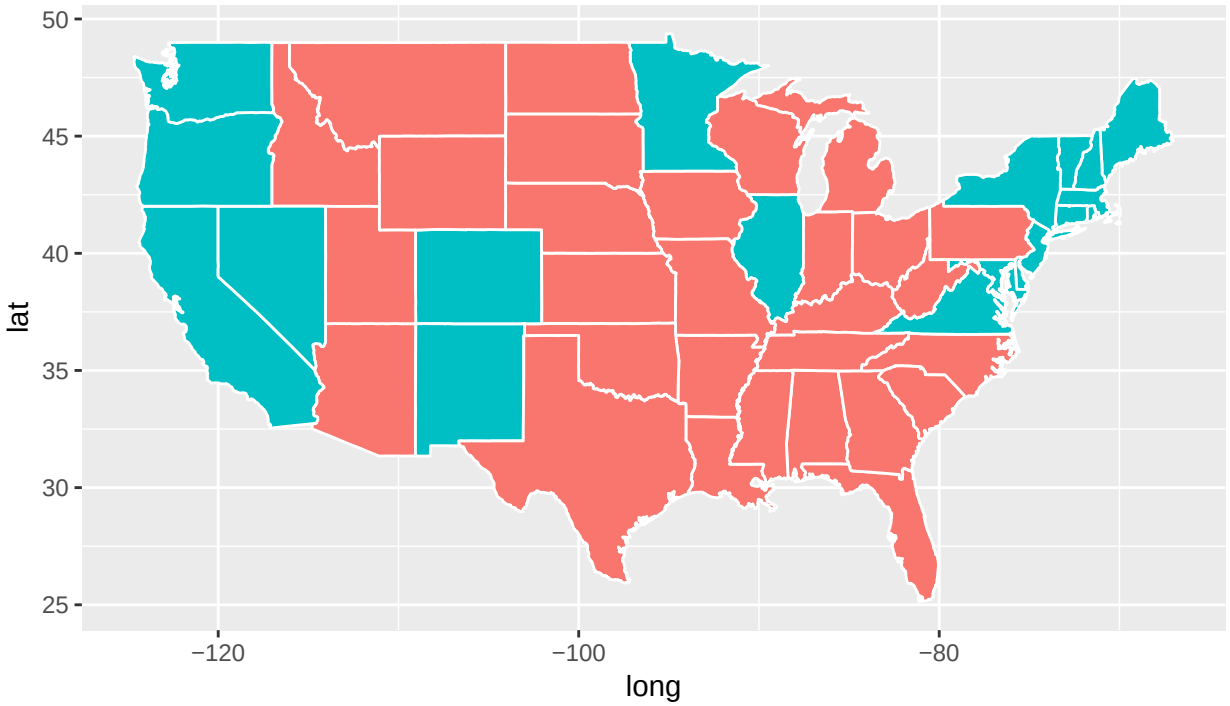
Now color the map by the winning candidate for each state. First, combine `states` variable and `state_winner` we created earlier using `left_join()`. Note that `left_join()` needs to match up values of states to join the tables; however, they are in different formats: e.g. `AZ` vs. `arizona`. Before 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()`. Your figure will look similar to state_level New York Times map.

```
states<-states %>% mutate(fips = state.abb[match(states$region, tolower(state.name))])

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

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

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

```
states <- map_data("state")
ggplot(data = lj_state) +
  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`. Your figure will look similar to county-level New York Times map.

```
library(stringr)

county.fips<-county.fips %>% mutate(region=str_split_fixed(county.fips$polyname, ",", 2)[,1]) %>%
  mutate(subregion=str_split_fixed(county.fips$polyname, ",", 2)[,2])


lj_county<-left_join(counties,county.fips)
```

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

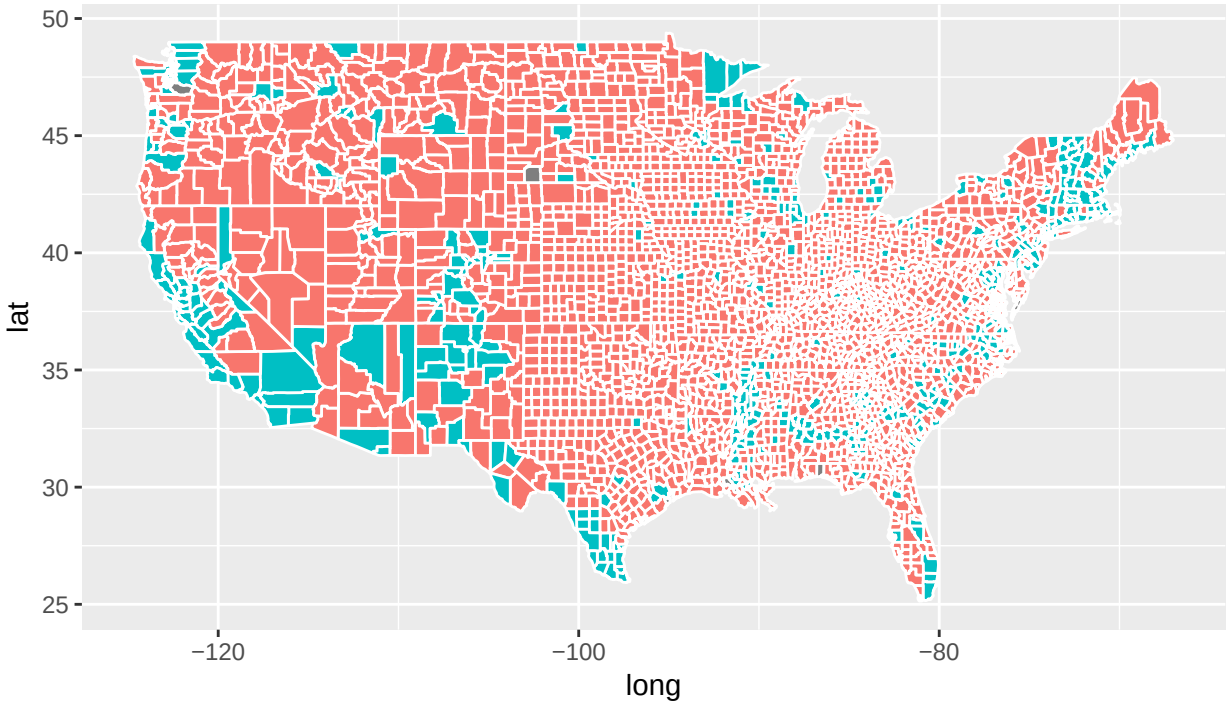
```
lj_county$fips<-as.character(lj_county$fips)
lj_county_winner<-left_join(lj_county,county_winner)
```

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

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

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

```
  guides(fill=FALSE)
```



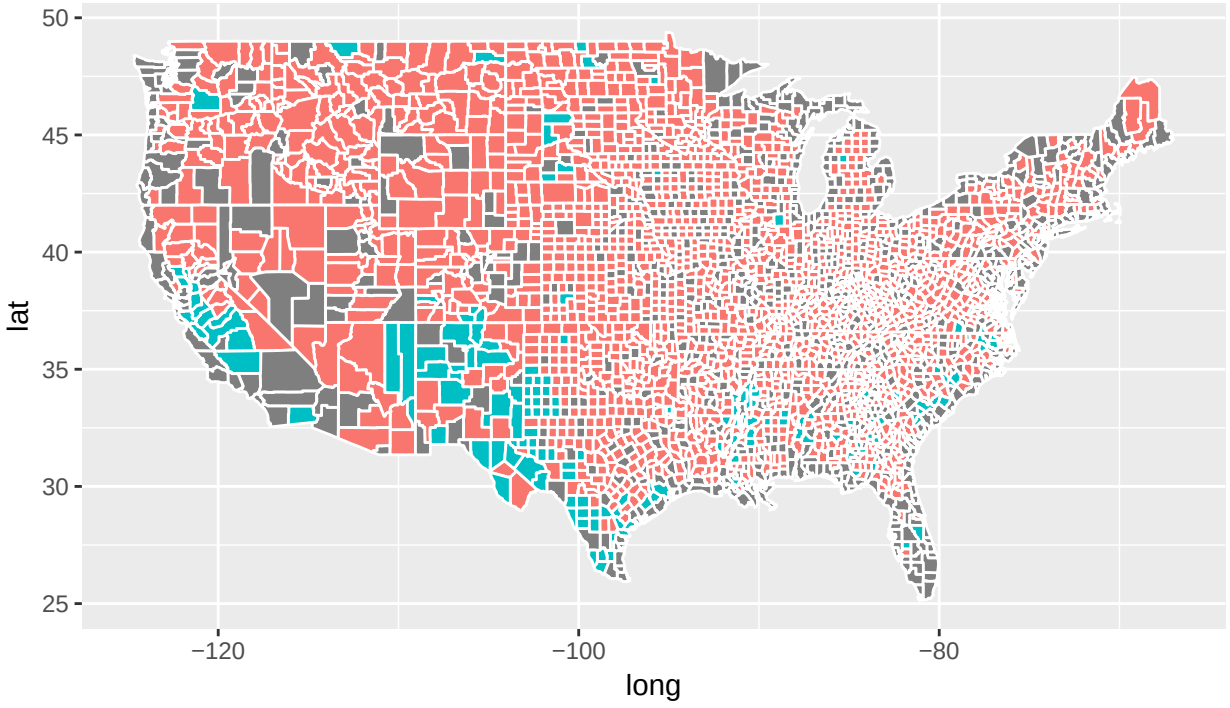
10.

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.

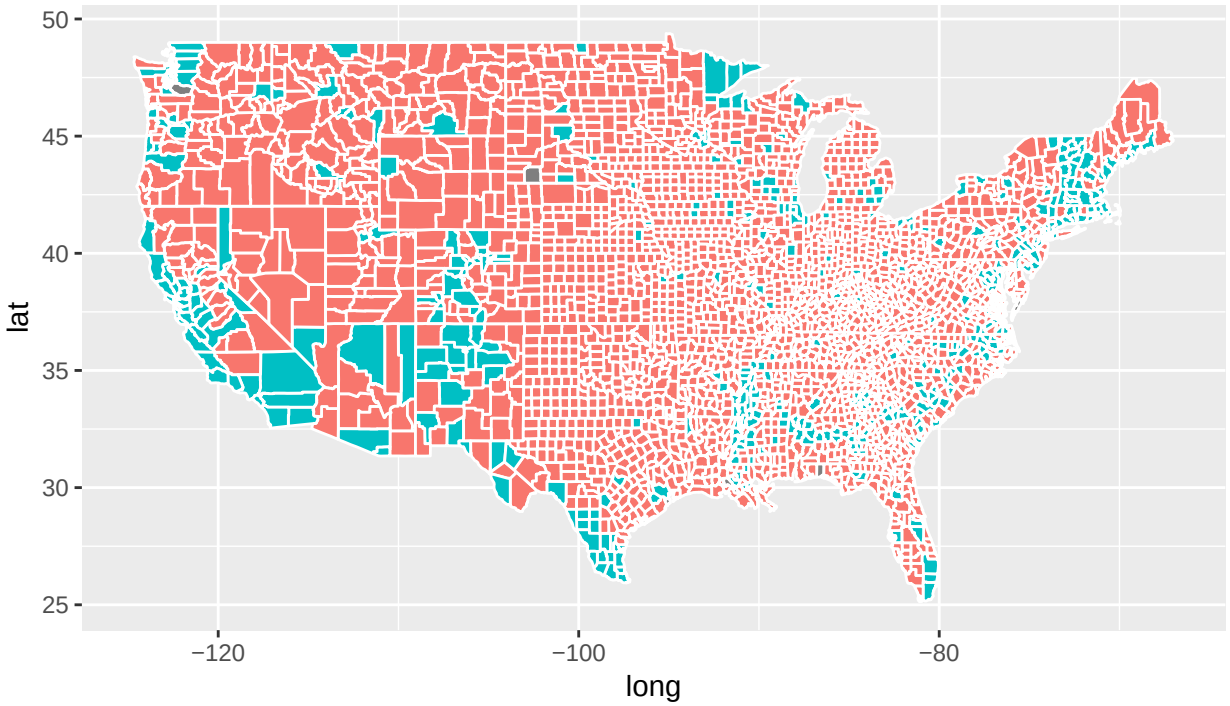
```
# Create total for white populations
wcensus<-census %>% mutate(wpopulation=White*TotalPop/100) %>% dplyr::select(-State)

# Aggregate white and total population by county
wccensus<-aggregate(wcensus$wpopulation,by=list(wcensus$County),sum)
wpcensus<-aggregate(wcensus$TotalPop,by=list(wcensus$County),sum)

# Bind white and total population
wcensus<-cbind(wccensus,wpcensus[,2])

# Create a variable that shows a true/false for white majority
wcensus<-wcensus%>% mutate(wmaj=ifelse(wcensus[,2]/wcensus[,3] >0.5, "false","true"))
wcensus$subregion<-tolower(wcensus[,1])

# Aggregate the census data with the geographical data
wcensus<-left_join(lj_county,wcensus)
```

```
## Joining, by = "subregion"
```

```
county <- map_data("county")

op = par(mfrow = c(1,2))
ggplot(data = wcensus) +
  geom_polygon(aes(x = long, y = lat, fill = wmaj, group = group), color = "white") +
  coord_fixed(1.3) +
  guides(fill=FALSE)
```




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




```
par(op)
```

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`_:
```

```
# Creating 'census.del'

census.del<-census[-1]%>%filter(complete.cases(census))

census.del = census[-1] %>%
  na.omit() %>%
  mutate( Men = 100*Men/TotalPop, Employed = 100*Employed/TotalPop, Citizen = 100*Citizen/TotalPop) %>%
  mutate(Minority = Hispanic + Black + Native + Asian + Pacific) %>%
  dplyr::select(-c(Hispanic, Black, Native, Asian, Pacific, Walk, PublicWork, Construction, Women))
```

```
#Creating 'census.subct'

census.subct = census.del %>%
  group_by(State, County) %>%
  add_tally(wt = TotalPop)
   # creates a variable to sum up TotalPop within a county

names(census.subct)[29] = 'CountyTotal' # renames 'n' to 'CountyTotal'

census.subct = census.subct %>% mutate(CountyWeight = TotalPop/CountyTotal)
#head(census.subct)
```

```
#Creating 'census.ct'

census.ct = census.subct %>% summarize_at(.vars = vars(Men:CountyTotal), .funs = funs(weighted.mean(., C
head(census.ct)
```

```
## # A tibble: 6 x 28
## # Groups:   State [1]
##   State County   Men White Citizen Income IncomeErr IncomePerCap
##   <fct> <fct>  <dbl> <dbl>   <dbl>  <dbl>     <dbl>        <dbl>
## 1 Alab~ Autau~  48.4  75.8    73.7 51696.     7771.       24974.
## 2 Alab~ Baldw~  48.8  83.1    75.7 51074.     8745.       27317.
## 3 Alab~ Barbo~  53.8  46.2    76.9 32959.     6031.       16824.
## 4 Alab~ Bibb    53.4  74.5    77.4 38887.     5662.       18431.
## 5 Alab~ Blount  49.4  87.9    73.4 46238.     8696.       20532.
## 6 Alab~ Bullo~  53.0  22.2    75.5 33293.     9000.       17580.
## # ... with 20 more variables: IncomePerCapErr <dbl>, Poverty <dbl>,
## #   ChildPoverty <dbl>, Professional <dbl>, Service <dbl>, Office <dbl>,
## #   Production <dbl>, Drive <dbl>, Carpool <dbl>, Transit <dbl>,
## #   OtherTransp <dbl>, WorkAtHome <dbl>, MeanCommute <dbl>,
## #   Employed <dbl>, PrivateWork <dbl>, SelfEmployed <dbl>,
## #   FamilyWork <dbl>, Unemployment <dbl>, Minority <dbl>,
## #   CountyTotal <dbl>
```

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?

```
# PCA for county level data
```

```r
pr_ct = prcomp(census.ct[-c(1,2)], scale=T, center =T) #delete the non-numerical cols

# data frame for PC1 and PC2
ct.pc = pr_ct$x[,1:2]

# rotation matrix of the PC1
PC1.ct = pr_ct$rotation[,1]
```

```r
## Prominent loadings for subct.pc

# Order the absolute values of PC1 in decreasing order
PC1.ct.ordered = abs(PC1.ct[order(abs(PC1.ct), decreasing = T)])
PC1.ct.ordered[1:3] #The Top 3 most prominent loadings for county level data are IncomePerCap, ChildPov
```

```
## IncomePerCap ChildPoverty      Poverty
##    0.3530767    0.3421530    0.3405832
```

```r
#  PCA for sub-county level data
pr_subct = prcomp(census.subct[-c(1,2)], scale=T, center =T)
subct.pc = pr_subct$x[,1:2]
```

```r
# Prominent loadings for subct.pc
PC1.subct = pr_subct$rotation[,1]
PC1.subct.ordered = abs(PC1.subct[order(abs(PC1.subct), decreasing = T)])
PC1.subct.ordered[1:3]
```

```
## IncomePerCap Professional      Poverty
##    0.3181199    0.3064366    0.3046886
```

```r
#The top 3 most pominent loadings for subcounty level data are IncomePerCap, Professional, and Poverty
```

The Top 3 most prominent loadings in PC1 for the **county** level data are: **IncomePerCap, ChildPoverty, and Poverty**.
The Top 3 most prominent loadings in PC1 for the**subcounty** level data are: **IncomePerCap, Professional, and Poverty**.

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?

```r
library(dendextend)
```

```
##
## ---------------------
## Welcome to dendextend version 1.9.0
## Type citation('dendextend') for how to cite the package.
##
## Type browseVignettes(package = 'dendextend') for the package vignette.
## The github page is: https://github.com/talgalili/dendextend/
##
## Suggestions and bug-reports can be submitted at: https://github.com/talgalili/dendextend/issues
```

```
## Or contact: <tal.galili@gmail.com>
##
##  To suppress this message use:  suppressPackageStartupMessages(library(dendextend))
## ----------------------
```

```
##
## Attaching package: 'dendextend'
```

```
## The following object is masked from 'package:rpart':
##
##     prune
```

```
## The following object is masked from 'package:stats':
##
##     cutree
```

```
dist.ct = dist(census.ct[-c(1,2)], method = "euclidean")

ct.hclust = hclust(dist.ct, method = "complete")
clust = cutree(ct.hclust, k = 10)
dfclust = data.frame(census.ct, clust)
SMclust = dfclust %>% filter(dfclust[2] == "San Mateo") # county is "SM"
SMgroup = dfclust %>% filter(clust == as.integer(SMclust$clus))
#SMgroup
n.SMgroup = nrow(SMgroup)
n.SMgroup.Cal = SMgroup %>% filter(SMgroup[1]== "California") %>% count()
n.SMgroup.Cal
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1     6
```

```
n.SMgroup
```

```
## [1] 69
```

```
n.SMgroup.Cal/n.SMgroup
```

```
##            n
## 1 0.08695652
```

Using the complete dataset, of the 69 observations in the San Mateo group, 8.696% of the counties are in California.

```
census.ct.5PC = pr_ct$x[,1:5]

dist.ct.5PC = dist(census.ct.5PC, method = "euclidean")
census.ct.5PC.hclust = hclust(dist.ct.5PC, method = "complete")
clus.5PC = cutree(census.ct.5PC.hclust, k = 10)

dfclus.5PC = data.frame(census.ct, clus.5PC)

# filter the cluster which contains San Mateo
SMclust.5PC = dfclus.5PC %>% filter(dfclus.5PC[2] == "San Mateo")

# all observations in the cluster that includes San Mateo's
SMgroup.5PC = dfclus.5PC %>% filter(clus.5PC == as.integer(SMclust.5PC$clus.5PC))
```

```
n.SMgroup.5PC = nrow(SMgroup.5PC) # size of San Mateo's cluster
n.SMgroup.5PC
```

```
## [1] 47
```

```
n.SMgroup.Cal.5PC = SMgroup.5PC %>% filter(SMgroup.5PC[1]== "California") %>% count()
n.SMgroup.Cal.5PC
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1    12
```

```
n.SMgroup.Cal.5PC/n.SMgroup.5PC
```

```
##           n
## 1 0.2553191
```

Using the first 5 principle component, of the 47 observations in the San Mateo group, 25.53% of the counties are in California.

For both hierarchical clustering methods with the entire dataset and only the first 5 principal components, there are only a small number of observations in the clusters that contain San Mateo. This is possibly because the bay area is a place with such unique features that not many counties are similar to. Since using only the first 5 principal components leads to a better overall prediction and simplifies the problem at the same time, we think clustering with prinicple components is more favorable over clustering with the whole dataset. However, as the sample sizes for both methods are relatively small, it is difficult to make a definite decision on which method is strictly better than the other.

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 %>% ungroup %>% mutate_at(vars(State, County), tolower)

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

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

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

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

```
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:

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

Using the following error rate function:

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

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.

```
#remove the constant row (cleaning the data)
trn.cl<-trn.cl %>% select(-total)
tst.cl<-tst.cl %>% select(-total)

trn.cl.tree = tree(candidate ~., data = trn.cl)
summary(trn.cl.tree)
```

```
##
## Classification tree:
## tree(formula = candidate ~ ., data = trn.cl)
## Variables actually used in tree construction:
## [1] "Transit"      "White"        "Income"       "Unemployment"
## [5] "Production"   "CountyTotal"  "Professional" "Service"
## Number of terminal nodes:  12
## Residual mean deviance:  0.3598 = 879.3 / 2444
## Misclassification error rate: 0.06433 = 158 / 2456
```

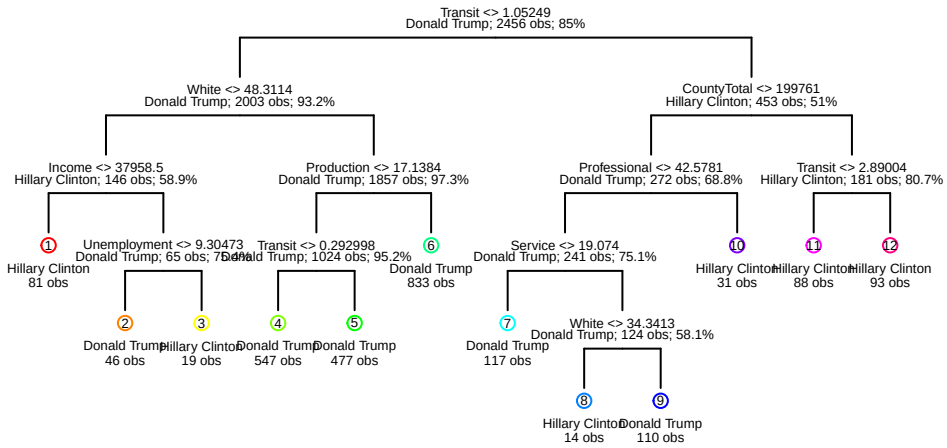
```
# Use cross validation on a tree with 10 folds
cv = cv.tree(trn.cl.tree, rand = folds, FUN=prune.misclass)

# Find which tree size minimizes deviance
best.cv <- min(cv$size[which(cv$dev==min(cv$dev))])
best.cv
```

```
## [1] 9
```

```
paste0('The best tree size which minimizes deviance is ',best.cv)
```

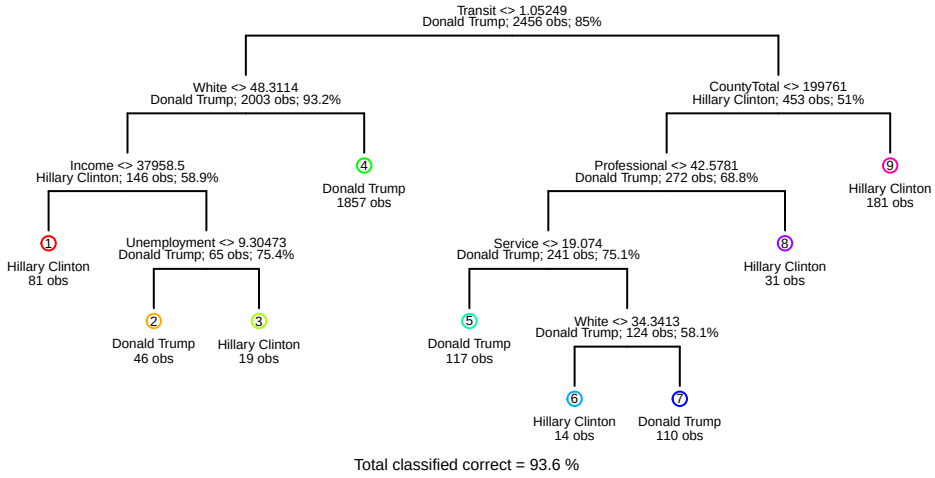
```
## [1] "The best tree size which minimizes deviance is 9"
```

```
# Draw non-pruned tree
draw.tree(trn.cl.tree, nodeinfo = TRUE, cex = 0.4)
```



Total classified correct = 93.6 %

```
# Prune the tree to minimize misclassification error
trn.cl.tree.pruned = prune.tree(trn.cl.tree, best=best.cv, method="misclass")

# Draw pruned tree (same as non-pruned)
draw.tree(trn.cl.tree.pruned, nodeinfo = TRUE, cex = 0.4)
```




```
# Calculate training and testing error
tree.train.pred=predict(trn.cl.tree.pruned, trn.cl,type = "class")
tree.test.pred=predict(trn.cl.tree.pruned, tst.cl,type = "class")

# Record the error
records[1,1] = sprintf("%.5f",calc_error_rate(tree.train.pred, trn.cl$candidate))
records[1,2] = sprintf("%.5f",calc_error_rate(tree.test.pred, tst.cl$candidate))
records
```

```
##      train.error test.error
## tree "0.06433"   "0.08143"
## knn  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`.

```
set.seed(1)
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(fold=chunkid,
               train.error = calc_error_rate(predYtr, Ytr),
               val.error = calc_error_rate(predYvl, Yvl))
}
```

```
library(plyr)
```

```
## ------------------------------------------------------------------------------
## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first, then dplyr:
## library(plyr); library(dplyr)
## ------------------------------------------------------------------------------
##
## Attaching package: 'plyr'
## The following object is masked from 'package:maps':
##
##     ozone
## The following objects are masked from 'package:dplyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize
## The following object is masked from 'package:purrr':
##
##     compact
```

```
X.trn <- trn.cl %>% select(-candidate)
Y.trn <- trn.cl$candidate

X.test <- tst.cl %>% select(-candidate)
Y.test <- tst.cl$candidate

set.seed(1)
error.folds = NULL
kvec = c(1, seq(10, 50, length.out=9))


for (j in kvec){
tmp = ldply(1:nfold, do.chunk, folddef=folds, Xdat=X.trn, Ydat=Y.trn, k=j)
tmp$neighbors = j
error.folds = rbind(error.folds, tmp)
}

val.error.means = error.folds %>%
  select(neighbors, val.error) %>%
  group_by(neighbors) %>%
  summarise_each(funs(mean), val.error) %>%
  ungroup()%>%
  filter(val.error == min(val.error))
```

```
## `summarise_each()` is deprecated.
## Use `summarise_all()`, `summarise_at()` or `summarise_if()` instead.
## To map `funs` over a selection of variables, use `summarise_at()`
```

```
numneighbor = max(val.error.means$neighbors)
numneighbor
```

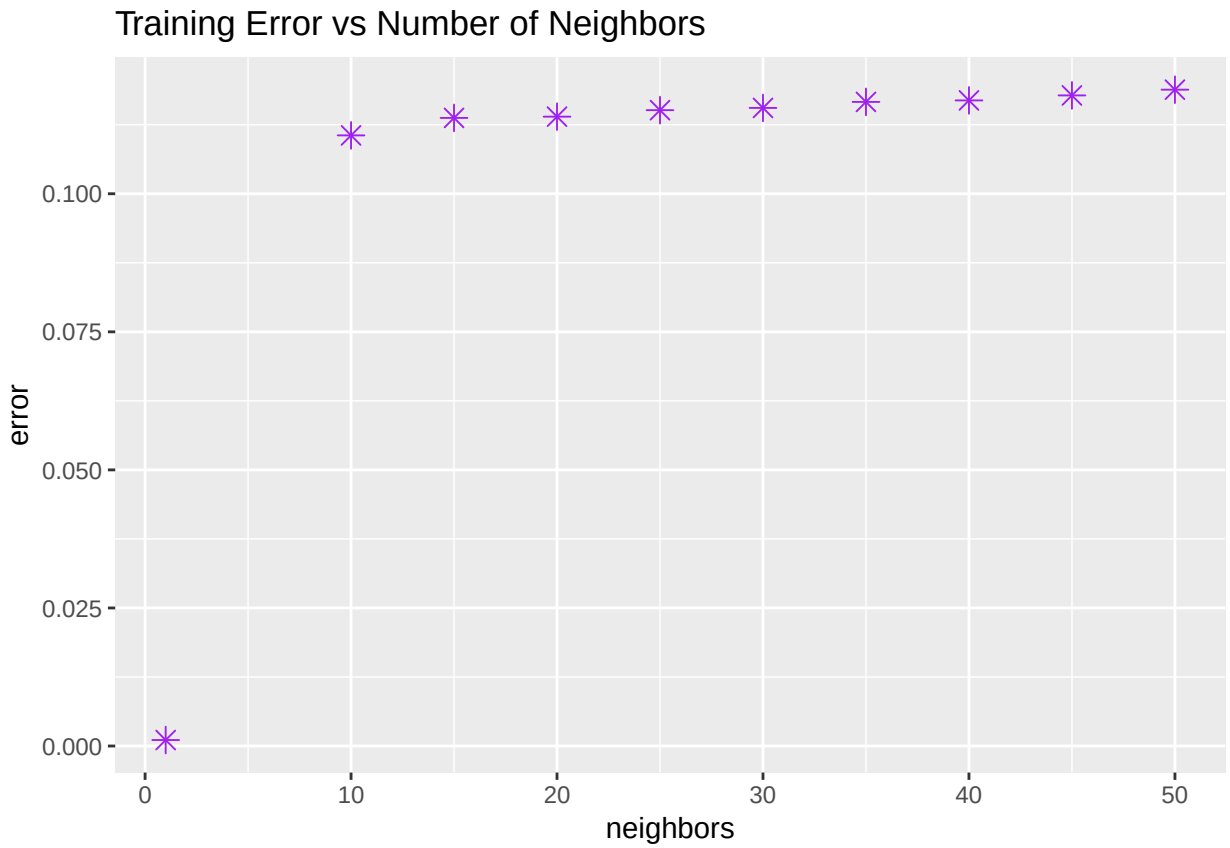
```
## [1] 15
```

```
set.seed(1)
pred.Ytr =knn(train = X.trn, test = X.trn, cl = Y.trn, k = numneighbor,l = 0, prob = TRUE, use.all = FA
pred.Yts =knn(train = X.trn, test = X.test, cl = Y.trn, k= numneighbor,l = 0, prob = TRUE, use.all = FA

records[2,1] = sprintf("%.5f",calc_error_rate(pred.Ytr, trn.cl$candidate))
records[2,2] = sprintf("%.5f",calc_error_rate(pred.Yts, tst.cl$candidate))
records
```

```
##      train.error test.error
## tree "0.06433"   "0.08143"
## knn  "0.11319"   "0.12378"
```

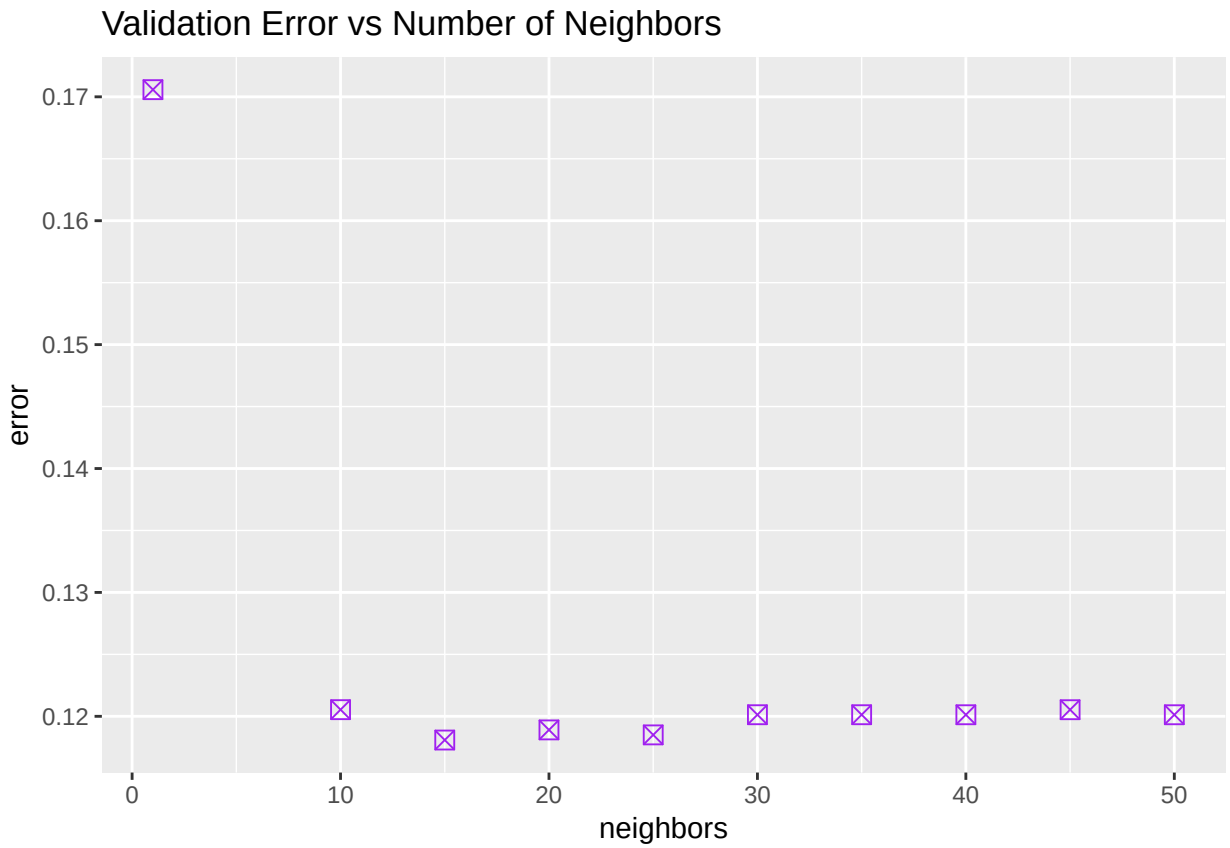
plot number of neighbors vs. resulting training

```
errors <- melt(error.folds, id.vars = c("fold","neighbors"),value.name = "error")
train.error.means <- errors %>%
  filter(variable=="train.error") %>%
  group_by(neighbors) %>%
  summarise_at(vars(error),funs(mean))

sp2 <- ggplot(train.error.means) +
  geom_point(aes(neighbors,error),color="purple",size=3,shape=8) +
  ggtitle("Training Error vs Number of Neighbors")
sp2
```



plot number of neighbors vs. validation errors

```
val.error.means <- errors %>%
  filter(variable=="val.error") %>%
  group_by(neighbors) %>%
  summarise_at(vars(error),funs(mean))

sp3 <- ggplot(val.error.means) + geom_point(aes(neighbors,error),color="purple",size=3,shape=7) +
  ggtitle("Validation Error vs Number of Neighbors")

sp3
```



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.

```
VarThresh = 0.9
##county level
pr.var.ct <- pr_ct$sdev^2
sum.pr.var.ct <- sum(pr.var.ct)
```

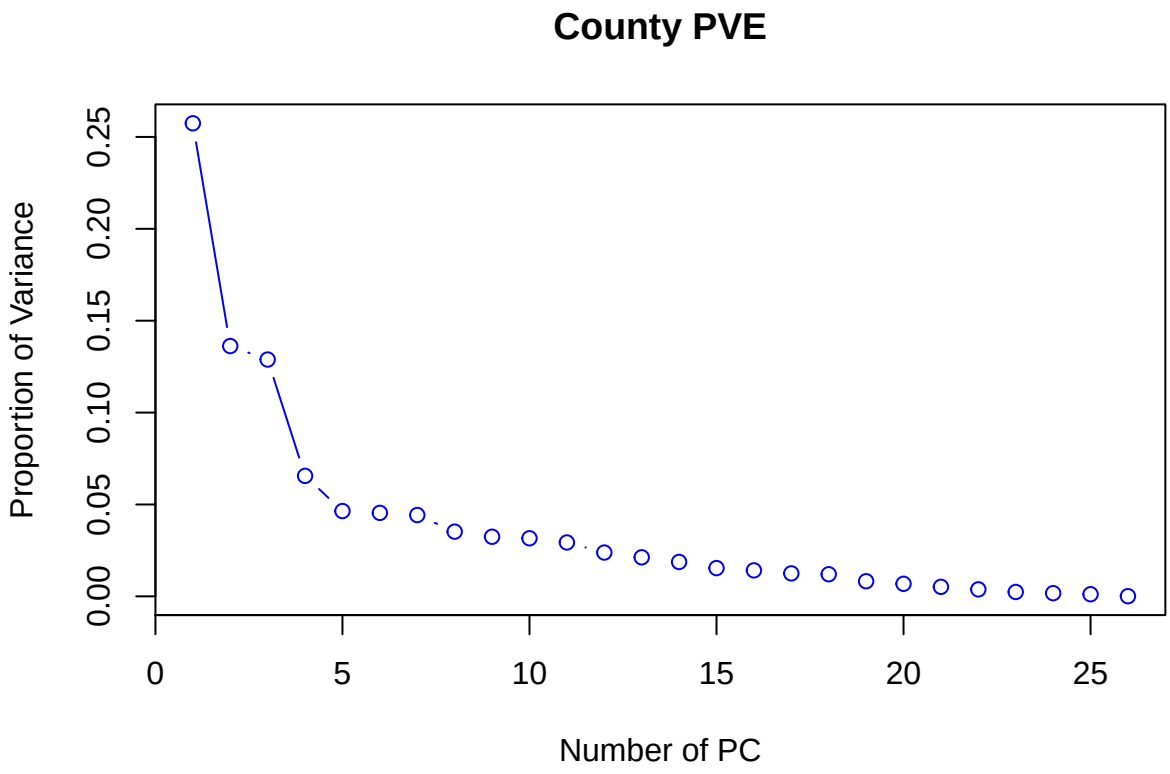
```
perct.var.ct <- pr.var.ct/sum.pr.var.ct
cum.perct.var.ct <- cumsum(perct.var.ct)
```

```
#minimum number of PC to exceed threshold
which(cum.perct.var.ct >= 0.9)[1]
```

```
## [1] 14
```

Minimum number of PCs needed to capture 90% of the variacne (county): 14

```
n.ct = length(cum.perct.var.ct)
size.ct = as.data.frame(cbind(cum.perct.var.ct, 1:n.ct))
abv.thres.ct = size.ct %>% filter(cum.perct.var.ct > VarThresh)

plot(perct.var.ct, type="b", col = "blue",main = "County PVE", xlab = "Number of PC", ylab = "Proportio
```




```
#subcounty level
pr.var.subct <- pr_subct$sdev^2
perct.var.subct <- pr.var.subct/sum(pr.var.subct)
cum.perct.var.subct <- cumsum(perct.var.subct)
```

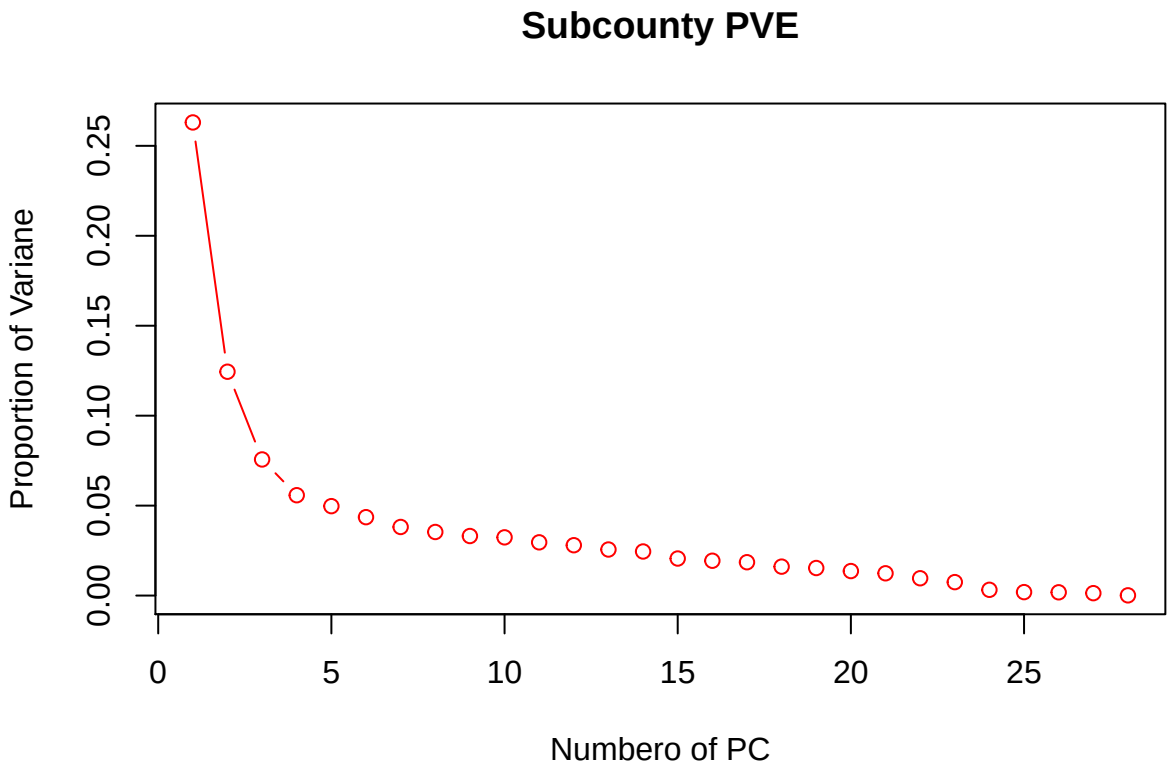
```
#minimum number of PC to exceed threshold
which(cum.perct.var.subct >= 0.9)[1]
```

```
## [1] 17
```

Minimum number of PCs needed to capure 90% of the variacne (subcounty): 17

```
n.subct = length(cum.perct.var.subct)
#n.subct #28
size.subct = as.data.frame(cbind(cum.perct.var.subct, 1:n.subct))
#size.subct
abv.thres.subct = size.subct %>% filter(cum.perct.var.subct > VarThresh)
#abv.thres.subct
```

```
plot(perct.var.subct, type="b",col = "red", main = "Subcounty PVE", xlab = "Numbero of PC", ylab = "Prop
```



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`.

```
# Training part
X.trn <- trn.cl %>% select(-candidate)
trn.pca <- prcomp(X.trn, scale=TRUE)
trn.pc <- data.frame(trn.pca$x)
```

```
# Test part
X.tst <- X.test
tst.pca <- prcomp(X.tst, scale = TRUE)
tst.pc <- data.frame(tst.pca$x)
```

```
#tr.pca & test.pca
tr.pca <- trn.pc %>% mutate(candidate=trn.cl$candidate)
test.pca <- tst.pc %>% mutate(candidate=tst.cl$candidate)
```

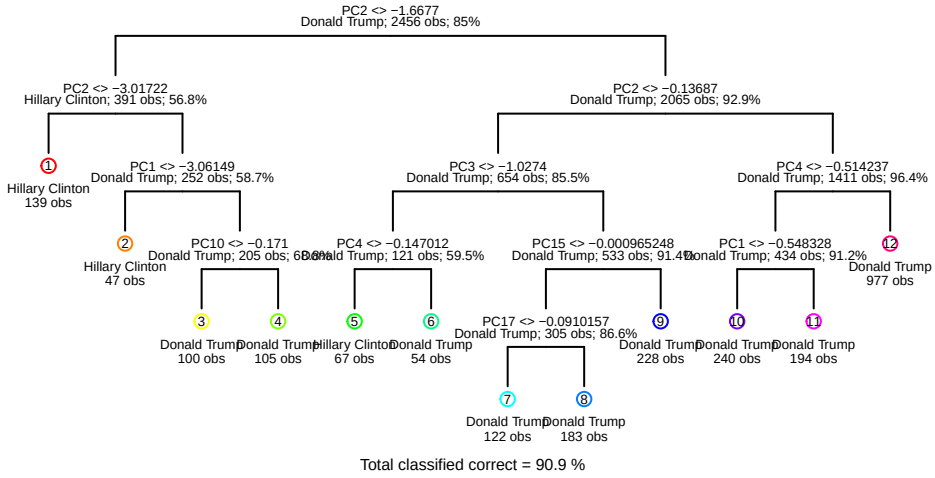
17.

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

```
#get original tree
tree.pc <- tree(candidate ~., tr.pca) #pc.tree -> tree.pc
```

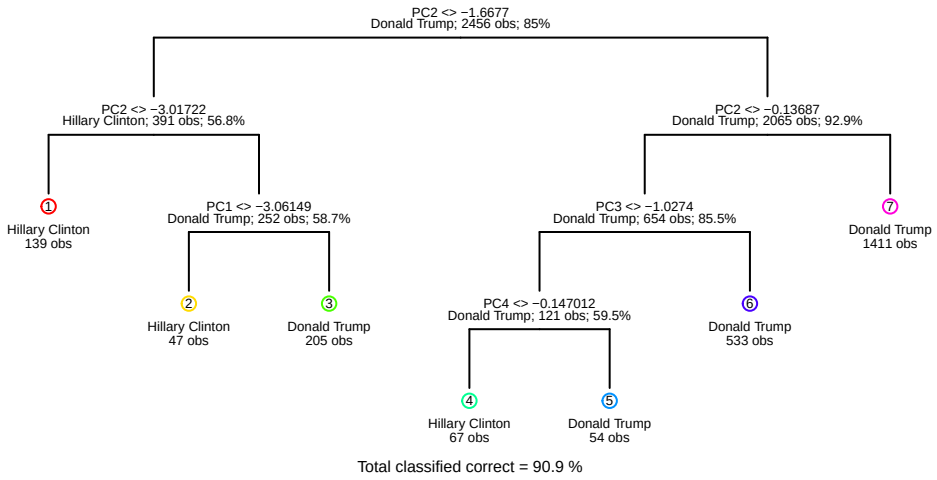
```
# Unpruned tree
draw.tree(tree.pc, nodeinfo = TRUE, cex = 0.4)
title("17. Unpruned Decision Tree")
```

17. Unpruned Decision Tree




```
#get best cv to draw tree to draw pruned tree
tree.pc.cv = cv.tree(tree.pc, rand = folds, FUN=prune.misclass)
best.pc.cv <- min(cv$size[which(tree.pc.cv$dev==min(tree.pc.cv$dev))])

# Pruned tree
tree.pc.pruned = prune.tree(tree.pc, best=best.pc.cv, method="misclass")
draw.tree(tree.pc.pruned, nodeinfo = TRUE, cex = 0.4)
title("17. Pruned Decision Tree")
```

17. Pruned Decision Tree




```
# Calculate training and testing error
tree.pc.train.pred=predict(tree.pc.pruned,trn.pc,type = "class")
tree.pc.test.pred=predict(tree.pc.pruned,test.pca,type = "class")

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

```
#record decision tree

pca.records[1,1] = sprintf("%.5f",calc_error_rate(tree.pc.train.pred, tr.pca$candidate))
pca.records[1,2] = sprintf("%.5f",calc_error_rate(tree.pc.test.pred, test.pca$candidate))
pca.records
```

```
##      train.error test.error
## tree "0.09080"   "0.11238"
## knn  NA          NA
## lda  NA          NA
```

18

K-nearest neighbor: repeat training of KNN classifier using principal components as independent variables. Record resulting errors.

```
library(plyr)

set.seed(1)
```

```
error.folds = NULL
kvec = c(1, seq(10, 50, length.out=9))


for (j in kvec){
tmp = ldply(1:nfold, do.chunk, folddef=folds, Xdat=trn.pc, Ydat=tr.pca$candidate, k=j)
tmp$neighbors = j
error.folds = rbind(error.folds, tmp)
}


val.error.means = error.folds %>%
  select(neighbors, val.error) %>%
  group_by(neighbors) %>%
  summarise_each(funs(mean), val.error) %>%
  ungroup()%>%
  filter(val.error == min(val.error))
```

```
## `summarise_each()` is deprecated.
## Use `summarise_all()`, `summarise_at()` or `summarise_if()` instead.
## To map `funs` over a selection of variables, use `summarise_at()`
```

```
numneighbor = max(val.error.means$neighbors)
numneighbor
```

```
## [1] 10
```

```
set.seed(1)
pred.Ytr =knn(train = trn.pc, test = trn.pc, cl = tr.pca$candidate, k = numneighbor,l = 0, prob = TRUE,
pred.Yts =knn(train = trn.pc, test = tst.pc, cl = tr.pca$candidate, k= numneighbor,l = 0, prob = TRUE, t

pca.records[2,1] = sprintf("%.5f",calc_error_rate(pred.Ytr, tr.pca$candidate))
pca.records[2,2] = sprintf("%.5f",calc_error_rate(pred.Yts, test.pca$candidate))
pca.records
```

```
##      train.error test.error
## tree "0.09080"   "0.11238"
## knn  "0.06393"   "0.10912"
## 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)

From the above analysis, we determine that PCA has a significant influence on the classification methods. We notice that decision tree method outperforms the knn method with the original training dataset and vice versa with the principle component training dataset. With principal component analysis, the pruned tree size was largely shrunk and the corresponding error rate was inflated. Therefore, it is our belief that comparing the learning capability of different classification methods is of great importance when dataset is transformed.

We also determine that the effect of race on the election prediction can be confounding. As is seen from the decision tree, the white citizens has a major impact on the election outcome, which seems plausible to us because white is the most dominant race in the United States. However, it is difficult to identify the individual impact of citizens from other races with this particular dataset. We also notice that variables concerning economic status, such as income per capita, are leading factors of prediction outcome. This result can be easily viewed from the prominent loadings of principal components at both the county and the sub-county level. Additionally, even though how election regions should be divided cannot be further explored with the given dataset, we believe it can potentially be a variable of great interest regarding the election outcome.

While determining the optimal machining learning method as well as interest variables for future presidential election prediction, we should consistently weigh the bias-variance tradeoff for model identification and selection. Applying bootstrap methods such as random forest might result in ideal forecasting as it replicates the original data and refine the data distribution. However, the downside turns out to be the fact that interpretability might be low. On the other hand, shrinking methods such as Lasso regression also worth considering for predicting the presidential election, since it removes irrelevant predictors and preserve the most desirable information.

Taking it further

20. The question that we propose: Examining additional methods: LDA and logistic regression, how do these methods compare to the previous two methods and which is the best classification method in specific to this dataset?

```
library(MASS)
```

```
##
## Attaching package: 'MASS'
```

```
## The following object is masked from 'package:dplyr':
##
##     select
```

```
#LDA

linDA.trn = lda(candidate~., data = trn.cl)
```

```
## Warning in lda.default(x, grouping, ...): groups None of these candidates
## Alyson Kennedy Bradford Lyttle Chris Keniston Dan Vacek Darrell Castle
## Emidio Soltysik Evan McMullin Frank Atwood Gary Johnson Gloria La Riva
## Jerry White Jill Stein Jim Hedges Joseph Maldonado Kyle Kopitke Laurence
## Kotlikoff Lynn Kahn Michael Maturen Mike Smith Monica Moorehead Peter
## Skewes Princess Jacob Richard Duncan Rocky De La Fuente Rocky Giordani Rod
## Silva Ryan Scott Scott Copeland Tom Hoefling are empty
```

```
linDA.trn.pred = predict(linDA.trn)$class
linDA.tst.pred = predict(linDA.trn, tst.cl)$class
```

```
error.train<- calc_error_rate(linDA.trn.pred, trn.cl$candidate)
error.test <- calc_error_rate(linDA.tst.pred, tst.cl$candidate)
```

```
#logistics

#model
lgst<- glm(candidate~., data = trn.cl, family = binomial)
```

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

```
#prediction on training  & testing
lgst.prob.training<- predict(lgst, X.trn, type = "response")
lgst.prob.testing<- predict(lgst,X.test, type = "response")

#set candidate for training and testing
lgst.pred.training<-rep("Donald Trump", length(Y.trn))
lgst.pred.testing<- rep("Donald Trump", length(Y.test))

#compare

lgst.pred.training[lgst.prob.training > 0.5 ] = "Hillary Clinton wins"
lgst.pred.testing[lgst.prob.testing >0.5] = "Hillary Clinton wins "

# Training Error & Test Error
error.training <- calc_error_rate(lgst.pred.training, Y.trn)
error.testing <- calc_error_rate(lgst.pred.testing, Y.test)
```

```
# error comparison
records.comp= matrix(NA, nrow=2, ncol=2)
colnames(records.comp) = c("train.error","test.error")
rownames(records.comp) = c("lda","lgst")

records.comp[1,1] = sprintf("%.5f", error.train)
records.comp[1,2] = sprintf("%.5f",error.test)

records.comp[2,1] = sprintf("%.5f", error.training)
records.comp[2,2] =  sprintf("%.5f", error.testing)

records
```

```
##      train.error test.error
## tree "0.06433"   "0.08143"
## knn  "0.11319"   "0.12378"
```

```
records.comp
```

```
##      train.error test.error
## lda  "0.06800"   "0.07980"
## lgst "0.17060"   "0.18567"
```

Observing the LDA and the logistic regression models in addition to the previous two models (decision tree, knn), we determine that decision tree and LDA preform comparably well with the original dataset. Decision tree and LDA are also the top methods in terms of simplicity and interpretability. We suspect that knn does not perform as ideally due to the fact that we did not set the algorithm detailed enough to run through all possible number of neighbors. However, if we did, the algorithm would be much less efficient. Additionally, as mentioned in Question 19, it might be the less relevant variables in the dataset that raise

the error rates for knn. For logistic regression, we believe that the default link function in R may not be the most appropriate for the election dataset, which results in a higher error rates. Overall, it is our belief that LDA is the best model for this specific dataset. To summarize, when determining the most perferable model, we should always strike the balance between bias and variance while at the same time take into account the model simplicity and interpretability.