2016 Election Prediction

Anton Sunico & Danny Baerman (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?

There are a variety of factors that make voter behavior prediction difficult, but one of the most prominent of these factors is the change of voting intention over time. For example, a voter that claims he would vote for a certain candidate at a certain time can change his response in the future as a result of a specific event, such as the voter becoming unemployed. This possibility of change over time requires time series models for more accurate predictions. Another factor that adds to the difficulty of predicting voter behavior is in the polls. When people are asked on their voting intentions, various errors can take place. One such error is sampling error. Since only a sample of people are asked about their voting intentions, there becomes a possibility that a majority of the sample support a specific candidate, but this majority is not representative of the population.

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

Silver's approach was unique in that, instead of looking at the maximum probability, he looked at a range of probabilities. Silver would calculate the probability of a candidate's support for each date, and then for the following day, he would calculate the probability that the candidate's support shifts from one percentage to another. This approach is based off of Bayes' Theorem

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

The approximate 4-point national miss on the polls is likely attributed to various polling errors underestimating Trump's support. One such error is that Trump supporters were less likely to reveal their support without anonymity. Another is that Trump supporters were more distrusting of pollers, and thus misrepresented in polls. Voter turnout was also predicted to be higher than it actually was, with the turnout models being inaccurate in numerous states. Future predictions could be improved by taking into consideration the demographics of each area. Including these additional variables when making predicitions could reduce the inaccuracy that is a result from some of the polling errors mentioned above.

Data

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

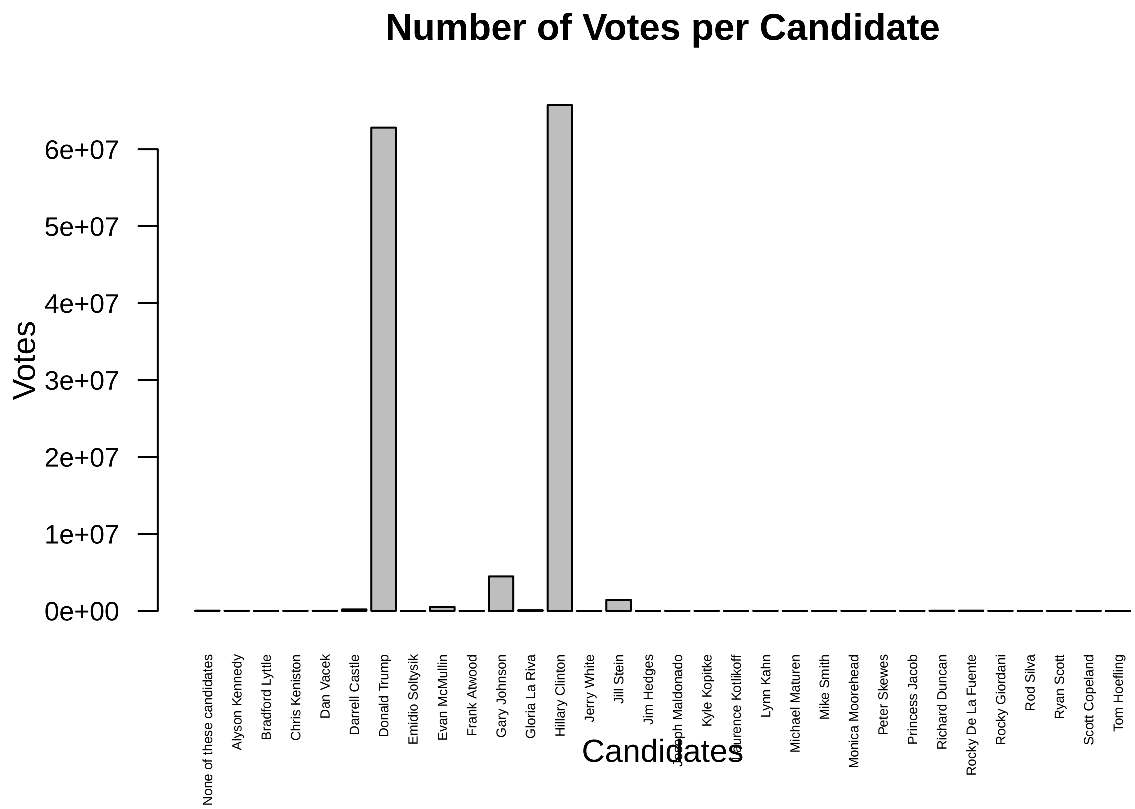
4. Remove summary rows from `election.raw` data: i.e.,

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

```r
# County-level data in election
election <- filter(election.raw, county != "NA")

# Federal-level summary in election_federal
election_federal <- filter(election.raw, fips == "US")

# State-level summary in election_state
election_state <- election.raw %>%
  filter(is.na(county), fips != "US")
```

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

```r
candidates <- election %>%
  select(candidate, votes) %>%
  group_by(candidate) %>%
  summarise(total_votes = sum(votes))

par(las = 2)
barplot(candidates$total_votes, names.arg = candidates$candidate, main = "Number of Votes per Candidate
```




```
# From the graph, there are a total 32 categories for candidates. However, one category is counted as n
```

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)) %>%
  mutate(pct = votes/total) %>%
  top_n(1, wt = pct)

state_winner <- election_state %>%
  group_by(fips) %>%
  mutate(total = sum(votes)) %>%
  mutate(pct = votes/total) %>%
  top_n(1, wt = pct)
```

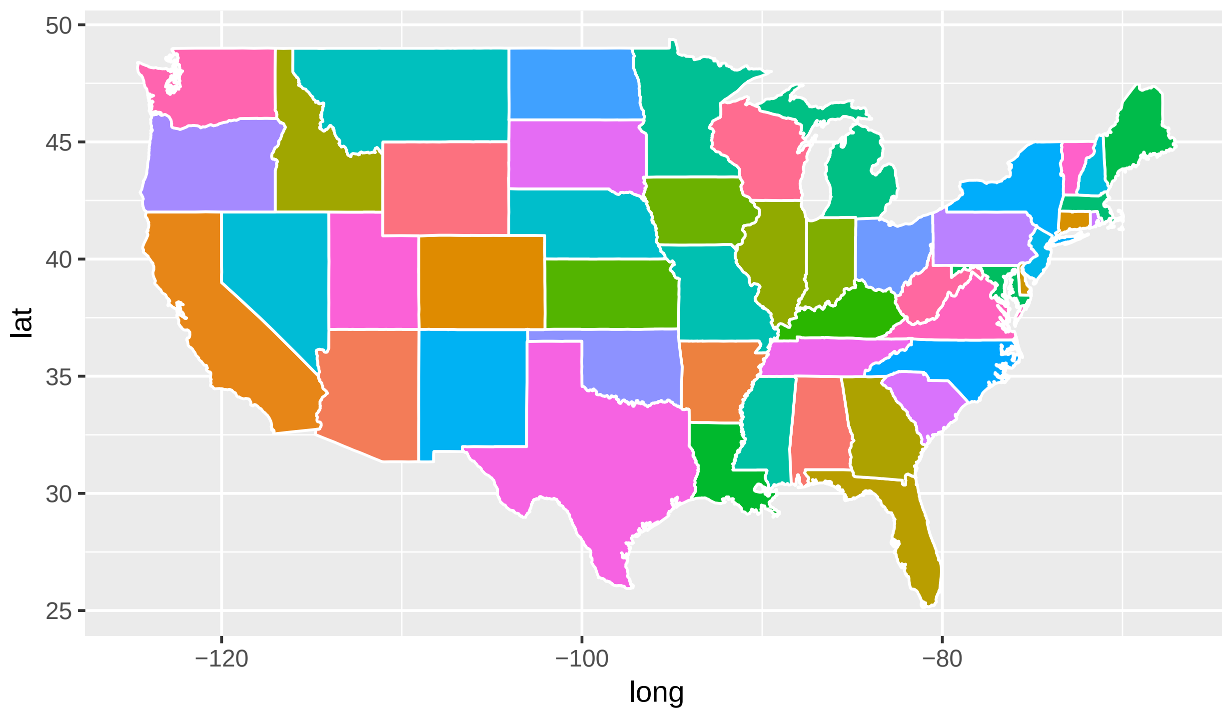
Visualization

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) +
```

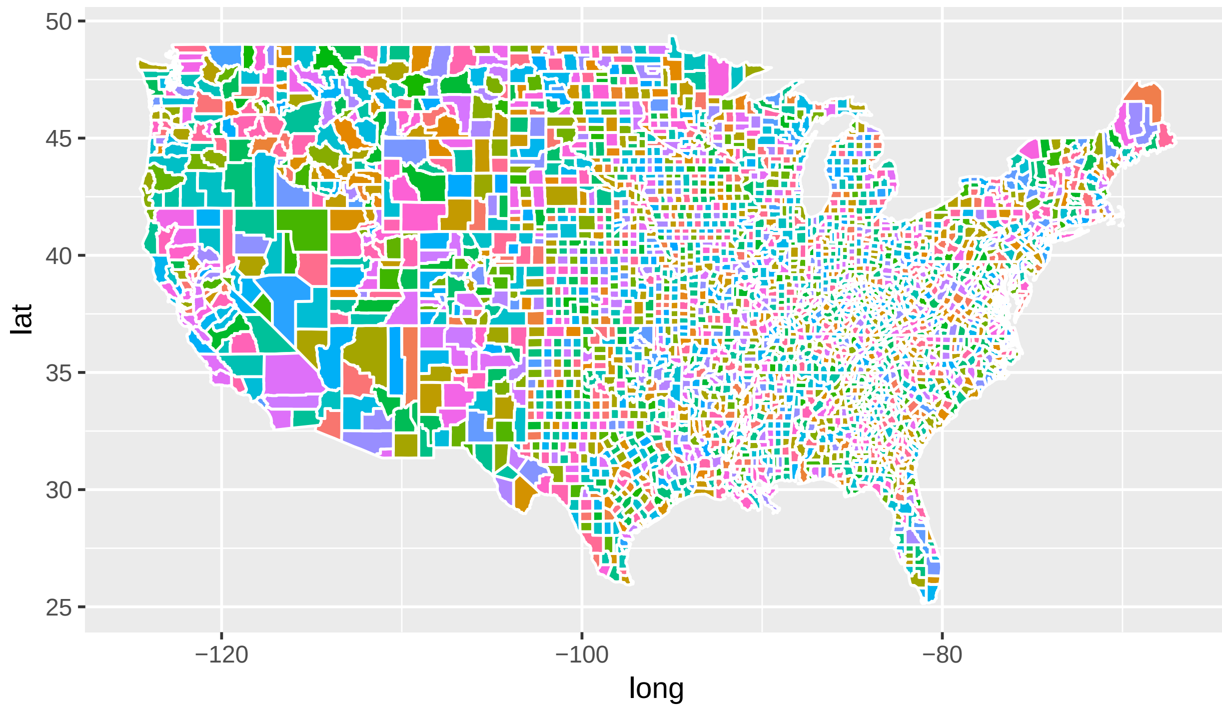
```
  geom_polygon(aes(x = long, y = lat, fill = region, group = group), color = "white") +
  coord_fixed(1.3) +
  guides(fill=FALSE)  # color legend is unnecessary and takes too long
```



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

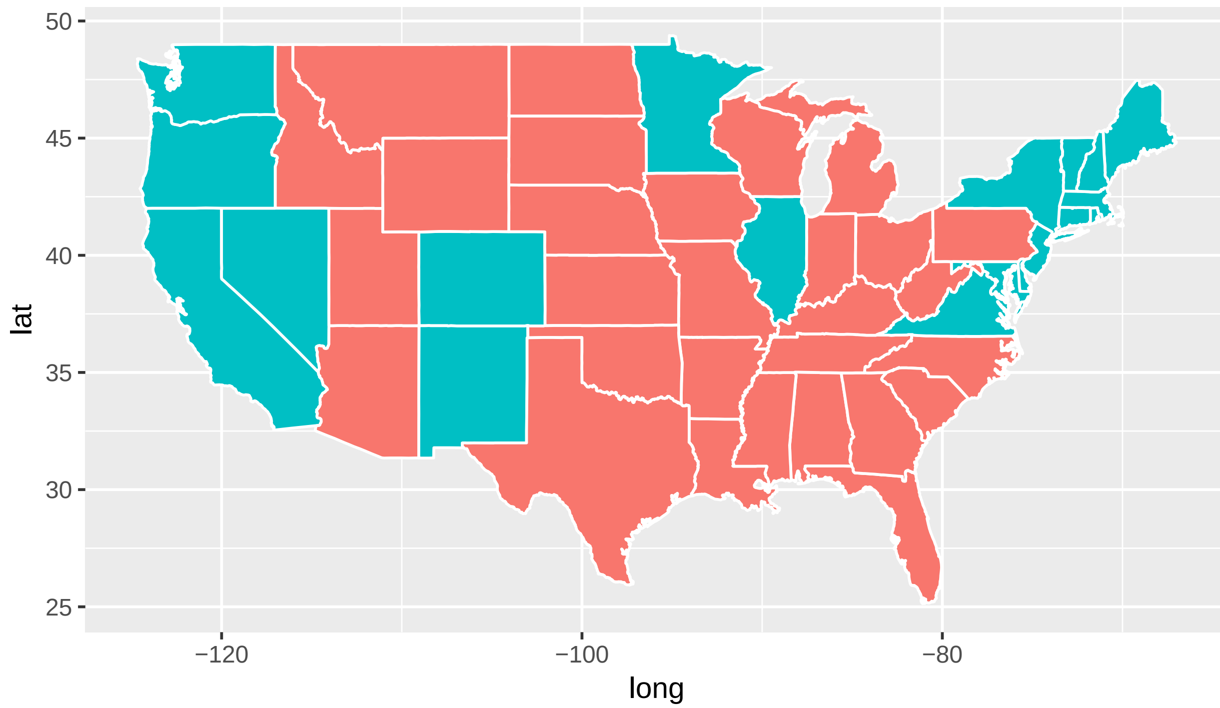


8. Now color the map by the winning candidate for each state. First, combine `states` variable and `state_winner` we created earlier using `left_join()`. Note that `left_join()` needs to match up values of states to join the tables; however, they are in different formats: e.g. `AZ` vs. `arizona`. Before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

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

state_winner_map <- left_join(new_states, state_winner, by = "fips")
```

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

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

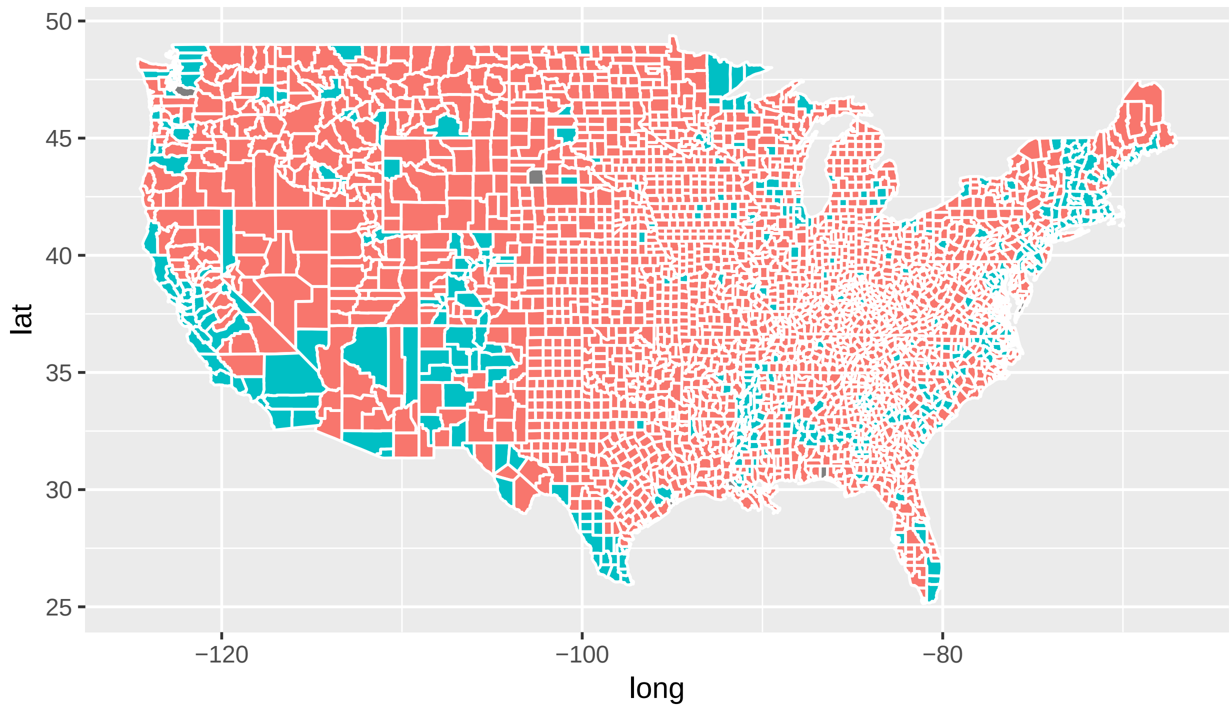


9. The variable `county` does not have `fips` column. So we will create one by pooling information from `maps::county.fips`. Split the `polyname` column to `region` and `subregion`. Use `left_join()` combine `county.fips` into `county`. Also, `left_join()` previously created variable `county_winner`.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

```
new_county <- separate(county.fips, polyname, c("region", "subregion"), sep = ",", remove = TRUE)

county2 <- left_join(counties, new_county, by = c("region", "subregion"))

county3 <- transform(county2, fips = as.factor(fips))
county_winner_map <- left_join(county3, county_winner, by = "fips")
```

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

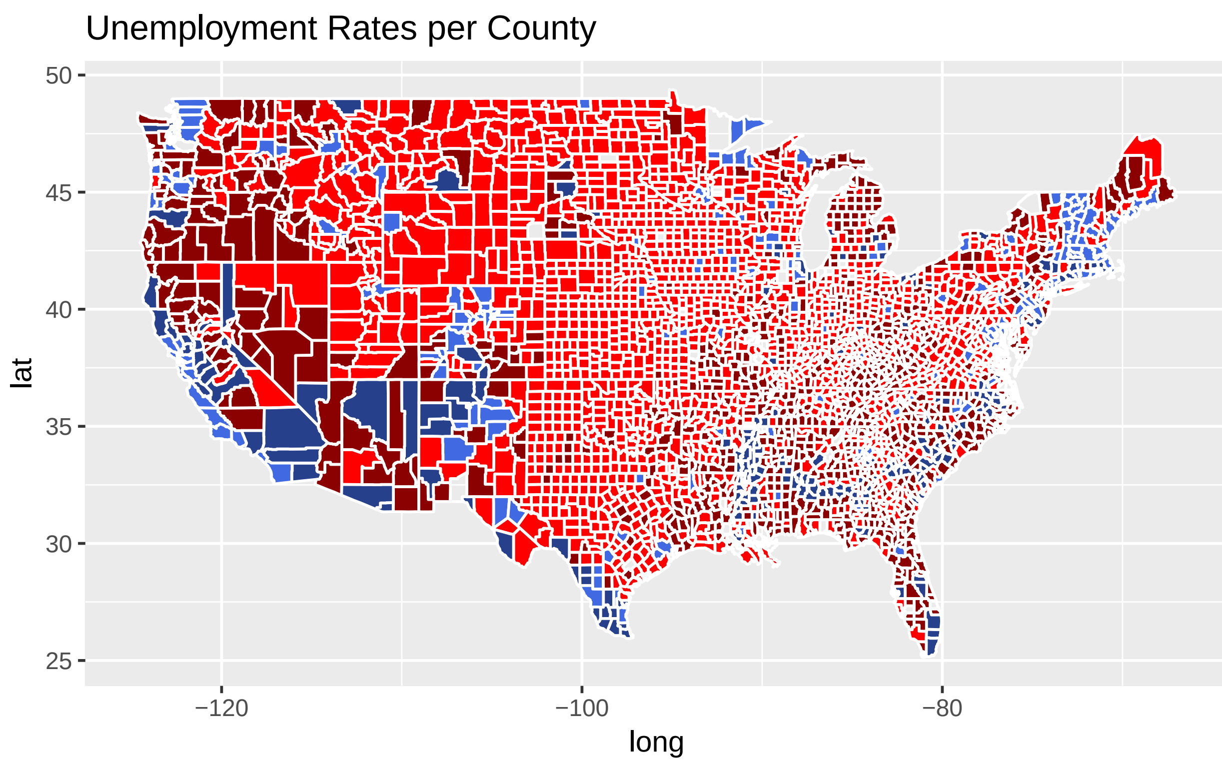
```
ggplot(data = county_winner_map) +
  geom_polygon(aes(x = long, y = lat, fill = candidate, group = group), color = "white") +
  coord_fixed(1.3) +
  guides(fill=FALSE)
```



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.

```
census_unemp <- census %>%
  select(State, County, Unemployment) %>%
  group_by(State, County) %>%
  mutate(avg_unemp = mean(Unemployment, na.rm = TRUE)) %>%
  mutate(region = tolower(State)) %>%
  mutate(subregion = tolower(County)) %>%
  distinct(subregion, .keep_all = TRUE) %>%
  ungroup() %>%
  select(region, subregion, avg_unemp)

counties_unemp <- left_join(new_county, census_unemp, by = c("region", "subregion"))

counties_unemp2 <- transform(counties_unemp, fips = as.character(fips))
county_winner_unemp <- left_join(county_winner_map, counties_unemp2, by = c("fips", "region", "subregio

unemp_map <- county_winner_unemp %>%
  mutate(unemp_factor = as.factor(ifelse(avg_unemp < 9 & candidate == "Donald Trump", "0", ifelse(candid

ggplot(data = unemp_map) +
  geom_polygon(aes(x = long, y = lat, fill = unemp_factor, group = group), color = "white") +
  scale_fill_manual("", labels = c("Trump, Below Avg", "Trump, Above Avg", "Clinton, Below Avg", "Clint
  ggtitle("Unemployment Rates per County") +
  coord_fixed(1.3) +
```

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



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`, `PublicWork`, `Construction`}.
     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.del <- na.omit(census) %>%
  mutate(Men = (Men/TotalPop)*100, Employed = (Employed/TotalPop)*100, Citizen = (Citizen/TotalPop)*100
  select(-Women, -Walk, -PublicWork, -Construction, -Hispanic, -Black, -Native, -Asian, -Pacific)

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

```r
  mutate(CountyTotal = n) %>%
  mutate(Weight = TotalPop/CountyTotal) %>%
  select(-n)

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

census.ct <- as.data.frame(census.ct)
print(head(census.ct))
```

```
##     State  County      Men    White  Citizen   Income IncomeErr
## 1 Alabama Autauga 48.43266 75.78823 73.74912 51696.29  7771.009
## 2 Alabama Baldwin 48.84866 83.10262 75.69406 51074.36  8745.050
## 3 Alabama Barbour 53.82816 46.23159 76.91222 32959.30  6031.065
## 4 Alabama    Bibb 53.41090 74.49989 77.39781 38886.63  5662.358
## 5 Alabama  Blount 49.40565 87.85385 73.37550 46237.97  8695.786
## 6 Alabama Bullock 53.00618 22.19918 75.45420 33292.69  9000.345
##   IncomePerCap IncomePerCapErr  Poverty ChildPoverty Professional  Service
## 1     24974.50        3433.674 12.91231     18.70758     32.79097 17.17044
## 2     27316.84        3803.718 13.42423     19.48431     32.72994 17.95092
## 3     16824.22        2430.189 26.50563     43.55962     26.12404 16.46343
## 4     18430.99        3073.599 16.60375     27.19708     21.59010 17.95545
## 5     20532.27        2052.055 16.72152     26.85738     28.52930 13.94252
## 6     17579.57        3110.645 24.50260     37.29116     19.55253 14.92420
##     Office Production    Drive   Carpool    Transit OtherTransp WorkAtHome
## 1 24.28243   17.15713 87.50624  8.781235 0.09525905   1.3059687  1.8356531
## 2 27.10439   11.32186 84.59861  8.959078 0.12662092   1.4438000  3.8504774
## 3 23.27878   23.31741 83.33021 11.056609 0.49540324   1.6217251  1.5019456
## 4 17.46731   23.74415 83.43488 13.153641 0.50313661   1.5620952  0.7314679
## 5 23.83692   20.10413 84.85031 11.279222 0.36263213   0.4199411  2.2654133
## 6 20.17051   25.73547 74.77277 14.839127 0.77321596   1.8238247  3.0998783
##   MeanCommute Employed PrivateWork SelfEmployed FamilyWork Unemployment
## 1    26.50016 43.43637    73.73649     5.433254 0.00000000     7.733726
## 2    26.32218 44.05113    81.28266     5.909353 0.36332686     7.589820
## 3    24.51828 31.92113    71.59426     7.149837 0.08977425    17.525557
## 4    28.71439 36.69262    76.74385     6.637936 0.39415148     8.163104
## 5    34.84489 38.44914    81.82671     4.228716 0.35649281     7.699640
## 6    28.63106 36.19592    79.09065     5.273684 0.00000000    17.890026
##   Minority CountyTotal
## 1 22.53687       55221
## 2 15.21426      195121
## 3 51.94382       26932
## 4 24.16597       22604
## 5 10.59474       57710
## 6 76.53587       10678
```

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?

```r
ct.pca <- prcomp(census.ct[3:28], scale = TRUE)
subct.pca <- prcomp(census.subct[4:28], scale = TRUE)

ct.pc <- data.frame(ct.pca$rotation)
subct.pc <- data.frame(subct.pca$rotation)

rownames(ct.pc)[which(abs(ct.pc$PC1) == max(abs(ct.pc$PC1)))]
```

```
## [1] "IncomePerCap"
```

```r
rownames(ct.pc)[which(abs(ct.pc$PC2) == max(abs(ct.pc$PC2)))]
```

```
## [1] "IncomeErr"
```

```r
rownames(subct.pc)[which(abs(subct.pc$PC1) == max(abs(subct.pc$PC1)))]
```

```
## [1] "IncomePerCap"
```

```r
rownames(subct.pc)[which(abs(subct.pc$PC2) == max(abs(subct.pc$PC2)))]
```

```
## [1] "Drive"
```

```r
# The most prominent loadings of PC1 is Income per Capital for both the county level and subcounty leve
```

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
scale.census.ct <- scale(census.ct[3:28])
distance <- dist(scale.census.ct, method = "euclidian")
hc.census.ct <- hclust(distance, method = "complete")
clusters <- cutree(hc.census.ct, k = 10)
table(clusters)
```

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

```r
ct.pc.five <- data.frame(ct.pca$x[,1:5])
scale.ct.pc <- scale(ct.pc.five)
distance2 <- dist(scale.ct.pc, method = "euclidian")
hc.ct.pc <- hclust(distance2, method = "complete")
clusters2 <- cutree(hc.ct.pc, k = 10)
table (clusters2)
```

```
## clusters2
##    1    2    3    4    5    6    7    8    9   10
## 2441  525   97    6    8   31    5   18    7   80
```

```r
clusters[which(census.ct$County == "San Mateo")]
```

```
## [1] 2
```

```r
clusters2[which(census.ct$County == "San Mateo")]
```

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

```r
check <- census.ct[which(clusters == 2),]
check2 <- census.ct[which(clusters2 == 1),]

# Based on the components of each cluster, it appears that the cluster that uses census.ct is more desi
```

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.

```r
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
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
x.trn.cl<-trn.cl %>%
  select(-candidate)
```

```r
y.trn.cl<-trn.cl$candidate
x.tst.cl<-tst.cl %>%
  select(-candidate)
y.tst.cl<-tst.cl$candidate

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

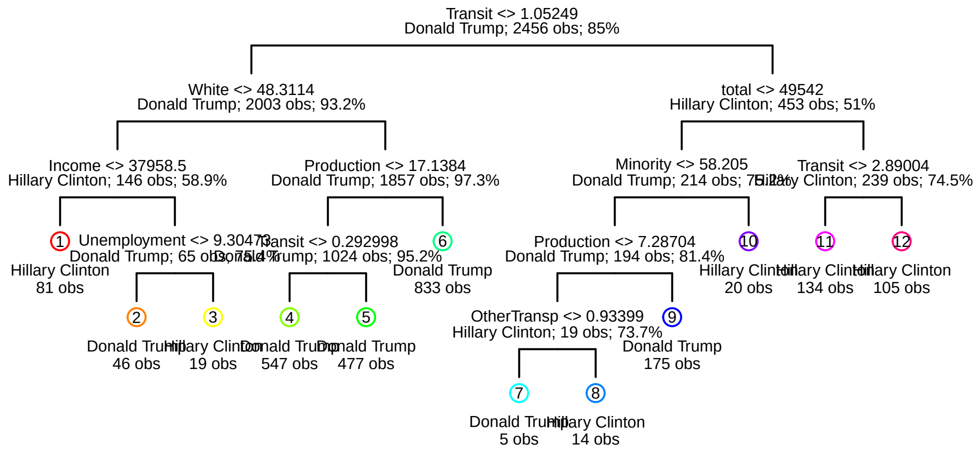
```
##
## Classification tree:
## tree(formula = candidate ~ ., data = trn.cl)
## Variables actually used in tree construction:
## [1] "Transit"      "White"        "Income"       "Unemployment"
## [5] "Production"   "total"        "Minority"     "OtherTransp"
## Number of terminal nodes:  12
## Residual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

```r
tree.cv<-cv.tree(tree, rand = folds, FUN = prune.misclass)

tree.cv.2<-min(tree.cv$size[which(tree.cv$dev==min(tree.cv$dev))])

tree.cv.2
```

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

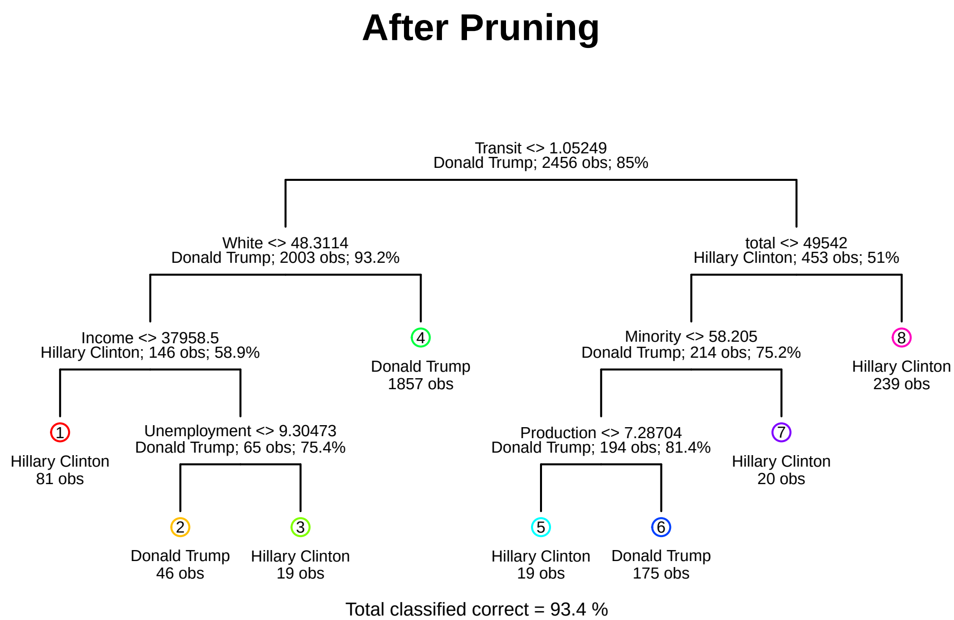
```r
#Pruning

set.seed(58)

tree.prune<-prune.tree(tree, best = tree.cv.2, method = "misclass")
draw.tree(tree, nodeinfo = TRUE, cex = 0.5)
title("Before Pruning")
```

Before Pruning



Total classified correct = 93.6 %

```r
draw.tree(tree.prune, nodeinfo = TRUE, cex = 0.5)
title("After Pruning")
```




```r
set.seed(58)

#Errors

tree.pred.trn<-predict(tree.prune, x.trn.cl, type = "class")
error.trn<-calc_error_rate(tree.pred.trn,y.trn.cl)

tree.pred.tst<-predict(tree.prune, x.tst.cl, type = "class")
error.tst<-calc_error_rate(tree.pred.tst,y.tst.cl)

records[1,1]<-error.trn
records[1,2]<-error.tst
records
```

```
##      train.error test.error
## tree  0.06596091 0.07980456
## 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`.

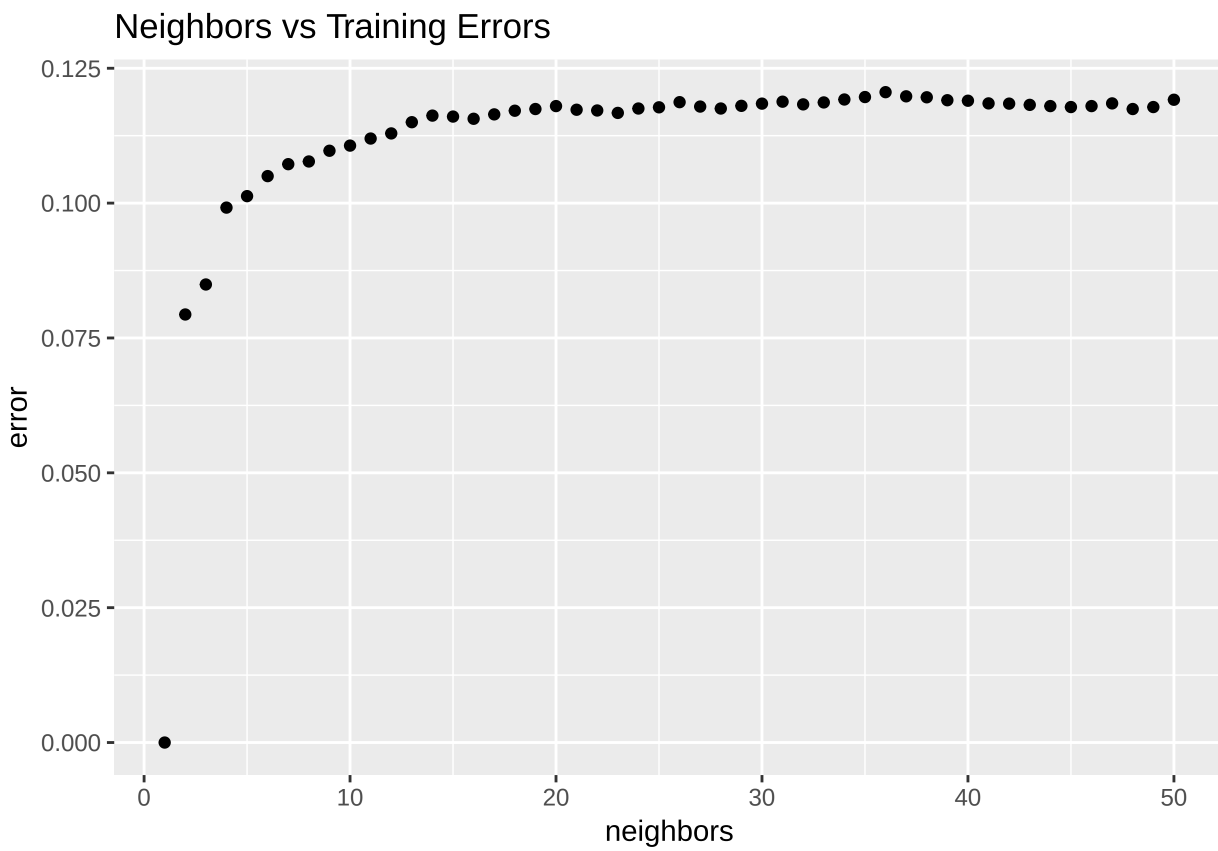
```r
do.chunk <- function(chunkid, folddef, Xdat, Ydat, ...){

    train = (folddef!=chunkid)
```

```r
    Xtr = Xdat[train,]
    Ytr = Ydat[train]

    Xvl = Xdat[!train,]
    Yvl = Ydat[!train]

    predYtr = knn(train=Xtr, test=Xtr, cl=Ytr, ...)
    predYvl = knn(train=Xtr, test=Xvl, cl=Ytr, ...)

    data.frame(fold = chunkid, # k folds
               train.error = mean(predYtr != Ytr),
               val.error = mean(predYvl != Yvl))

}

allK <- 1:50
error.folds <- NULL
set.seed(784)

for (j in allK){

    tmp = plyr::ldply(1:nfold, do.chunk,
                folddef=folds, Xdat=x.trn.cl, Ydat=y.trn.cl, k=j)


    tmp$neighbors = j

    error.folds = rbind(error.folds, tmp)

}

errors <- melt(error.folds, id.vars=c('fold', 'neighbors'), value.name='error')

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

minimumerror<-val.error.means %>%
  filter(error==min(error))

kk<-max(minimumerror$neighbors)

kk
```

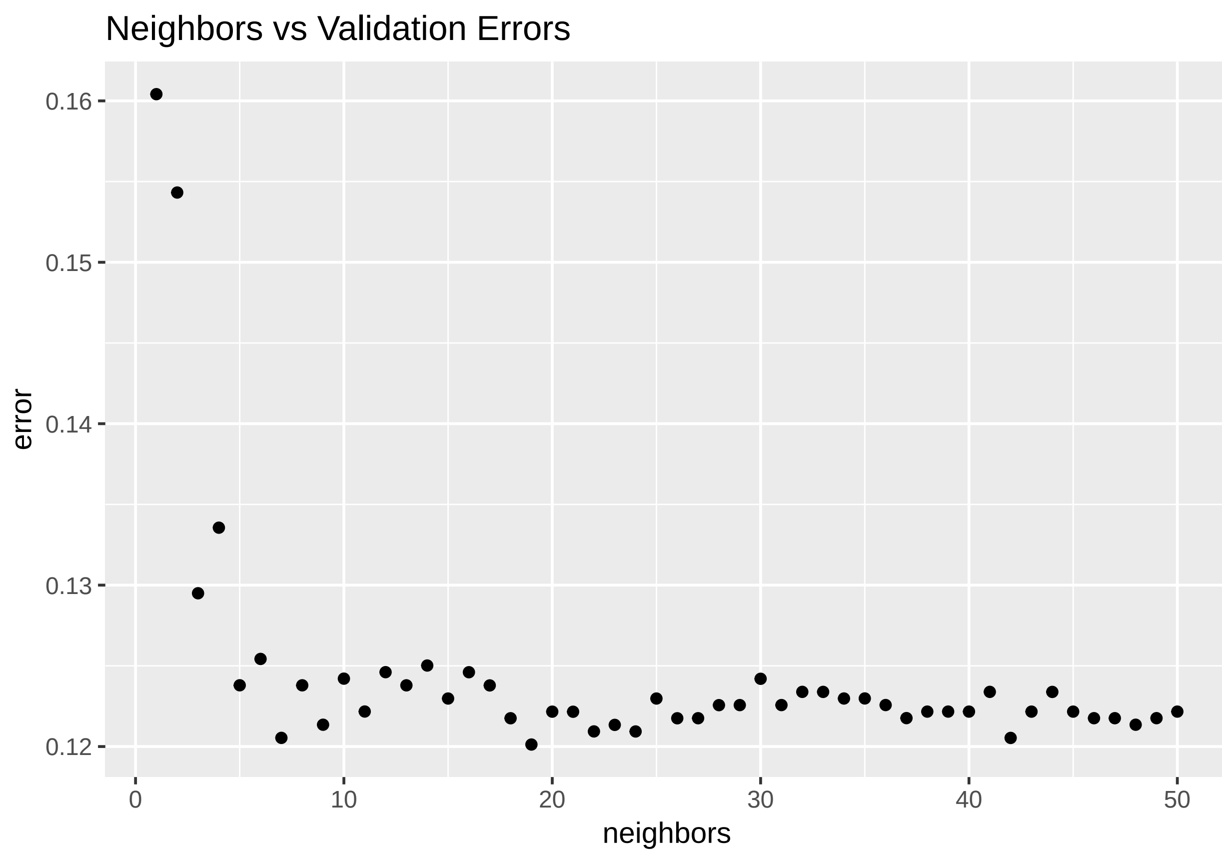
```
## [1] 19
```

```r
# k = 19 neighbors

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

```r
# Plotting

ggplot(trainingerrors) +
  geom_point(aes(neighbors,error)) +
  ggtitle("Neighbors vs Training Errors")
```




```r
ggplot(val.error.means) +
  geom_point(aes(neighbors, error)) +
  ggtitle("Neighbors vs Validation Errors")
```




```
#Records

knntraining<-knn(train = x.trn.cl, test = x.trn.cl, cl = y.trn.cl, k = kk)
knntrainingerror<-calc_error_rate(knntraining, y.trn.cl)

knntesting<-knn(train = x.trn.cl, test = x.tst.cl, cl = y.trn.cl, k = kk)
knntestingerror<-calc_error_rate(knntesting, y.tst.cl)

records[2,1] = knntrainingerror
records[2,2] = knntestingerror

records
```

```
##      train.error test.error
## tree  0.06596091 0.07980456
## knn   0.11604235 0.12540717
## 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.

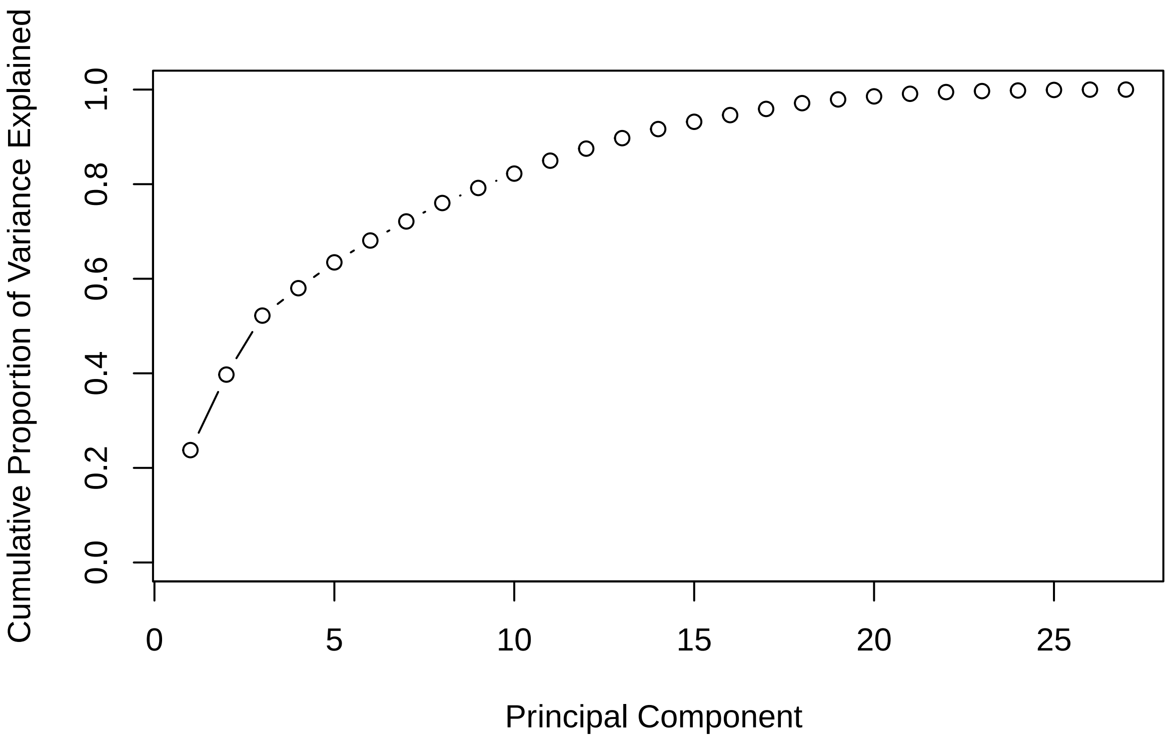
```
pca.records = matrix(NA, nrow=3, ncol=2)
colnames(pca.records) = c("train.error","test.error")
rownames(pca.records) = c("tree","knn","lda")
```

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.

```
pr.out<-prcomp(x.trn.cl,scale = TRUE)
pr.var<-pr.out$sdev^2
pve<-pr.var/sum(pr.var)

which(cumsum(pve)>=.9)[1]
```

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

```
plot(cumsum(pve), xlab="Principal Component ", ylab=" Cumulative Proportion of Variance Explained ", yli
```



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`.

```
proutdf<-data.frame(pr.out$x)
tr.pca<-proutdf %>%
  mutate(candidate=trn.cl$candidate)

pr.out.test<-prcomp(x.tst.cl,scale=TRUE)
prouttestdf<-data.frame(pr.out.test$x)
test.pca<-prouttestdf %>%
  mutate(candidate=tst.cl$candidate)
```

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

```r
x.trn.pc<-proutdf
y.trn.pc<-tr.pca$candidate
x.tst.pc<-prouttestdf
y.tst.pc<-test.pca$candidate

tree.pc<-tree(candidate~.,tr.pca)

tree.pr.cv<-cv.tree(tree.pc, rand = folds, FUN = prune.misclass)
tree.pr.cv.2<-min(tree.pr.cv$size[which(tree.pr.cv$dev==min(tree.pr.cv$dev))])

tree.pc.prune<-prune.tree(tree.pc, best = tree.pr.cv.2, method = "misclass")

#Errors

tree.pc.trn.pred<-predict(tree.pc.prune, x.trn.pc, type = "class")
pc.trn.error<-calc_error_rate(tree.pc.trn.pred, y.trn.pc)

tree.pc.tst.pred<-predict(tree.pc.prune, x.tst.pc, type = "class")
pc.tst.error<-calc_error_rate(tree.pc.tst.pred, y.tst.pc)

pca.records[1,1]<-pc.trn.error
pca.records[1,2]<-pc.tst.error

pca.records
```

```
##      train.error test.error
## tree  0.08713355   0.267101
## knn           NA         NA
## lda           NA         NA
```

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

```r
allKpca <- c(1, seq(10, 50, length.out = 3))
error.folds.pca <- NULL

for (j in allKpca) {
  tve <- plyr::ldply(1:nfold, do.chunk, folddef = folds, Xdat = x.trn.pc, Ydat = y.trn.pc, k = j)
  tve$neighbors <- j
  error.folds.pca <- rbind(error.folds.pca, tve)
}

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

minimumerror.pca <- val.means.error %>%
  filter(error==min(error))

kkpca<-max(minimumerror.pca$neighbors)
kkpca
```

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

```r
trainingerrors.pca <- pca.errors %>%
  filter(variable=="train.error") %>%
  group_by(neighbors) %>%
  summarise_at(vars(error), funs(mean))

pred.train <- knn(train = x.trn.pc, test = x.trn.pc, cl = y.trn.pc, k = kkpca)
error.train <- calc_error_rate(pred.train, y.trn.pc)

pred.test <- knn(train = x.trn.pc, test = x.tst.pc, cl = y.trn.pc, k = kkpca)
error.test <- calc_error_rate(pred.test, y.tst.pc)

# Records

pca.records[2,1] <- error.train
pca.records[2,2] <- error.test
pca.records
```

```
##      train.error test.error
## tree  0.08713355  0.2671010
## knn   0.06596091  0.1840391
## 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)

There are some missing data points, and therefore it may not be very representative of the population. Because there is such a large sample (US voters), it is hard to collect such a massive data set without making at least some errors. With these complications and difficulties in collecting large amounts of data, it's understandable to see why predicting elections can be challenging

Like any kind of data analysis, it can only be improved by addressing more variables. For instance, when analyzing the unemployment rates per county, we found that most of the counties that voted for Trump have a lower unemployment rate. There were several red states with high unemployment rates, too however. Several of the counties that voted for Clinton had a higher unemployment rate, overall. From our cluster data, Income per capita was one of the most influential factors in voting. A higher unemployment rate translates to a lower income per capita. Thus our analysis of the data lines up with the results.

By introducing other variables for analysis, we can address other potential reasons for why states voted the way they did, as well as by finding correlations between unemployment rates, income per capita, and other miscellaneous factors, such as industries that employees work in.

Taking it further

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

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

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

```r
# Errors
```

```r
fit.pred.train <- predict(fit, x.trn.cl, type = "response")
fit.pred.train2 <- rep("Donald Trump", length(y.trn.cl))
fit.pred.train2[fit.pred.train > .5] = "Hillary Clinton"
fit.trainingerror <- calc_error_rate(fit.pred.train2, y.trn.cl)

fit.pred.test <- predict(fit, x.tst.cl, type = "response")
fit.pred.test2 <- rep("Donald Trump", length(y.tst.cl))
fit.pred.test2[fit.pred.test > .5] = "Hillary Clinton"
fit.testingerror <- calc_error_rate(fit.pred.test2, y.tst.cl)

records[3,1] <- fit.trainingerror
records[3,2] <- fit.testingerror
records
```

```
##      train.error test.error
## tree  0.06596091 0.07980456
## knn   0.11604235 0.12540717
## lda   0.06555375 0.07491857
```

```r
# This yields some of the lowest errors.  It is similar to the classification tree errors, however.  We
```