Final Project

PSTAT 131/231, Winter 2018

Evan Azevedo

3/23/2018

Questions

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

Voter intention is given by polls, but they are often done months in advance of the election and do not count as actual votes. This gives respondents the option to 1. Be disingenuous or not fully consider their intentions on which candidate to vote for, and 2. Change who they will vote for before the actual election. Thus, voter polls are merely a forecast of voter intention. Another complication which makes voter prediction a hard problem is the inherent hierarcy in presidential elections, where the outcome is based on the number of electoral votes a candidate recieves when he or she wins a state. Thus, a candidate can win the majority vote for predsidency but still loose due to not having enough electoral votes.

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

He calculated the probabilities of each level of support for a candidate (51%, 52%, …) on a given day, and used conditional probabilities when the poll data for that day was released to get a better idea of where the true voter count was for each candidate. For example, if support for Obama is 53% on a day, he calculates the conditional probabilities of the true percentage of Obama voters, N, given the polled support for the candidate, S, $P(N = i|S = 53)$ for $i = 50, 51, 52, \ldots$. Now, when the polls for the next day come out with 56% support for Obama, N can more accurately be predicted given the computed probabilities from the previous day.

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

There were many systematic errors unaccounted for due to the unique circumstances of the 2016 Presidential Election. For example, a Los Angeles poll noted that women were significantly less likely to tell a pollster that they were supporting Trump in the election. A blog post on fivethirtyeight.com noted that the polls failed in several ways to account for the polarizing nature of the Republican presidental candidate, which is made evident by the amount that Donald Trump beat the polls in states with a high population of whites without college degrees. One point to this that the blog post mentions is that polls are expected to be inaccurate, but they are also assumed that their errors are in many areas and will effectively cancel out over several polling agencies and their districts, but we see that in the above demographic and in some important swing regions, these errors were huge and unaccounted for even up until election day. I think improvement of future predictions in cases such as these is a matter of more highly educated Political Scientists and researchers to keep track of the bigger picture of the polling: what demographics are we underrepresenting? What kind of systematic errors are we ignoring/inccuring with our current method? From there, the polling methods can be adjusted to continue to give accurate standings of the race.

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

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`.

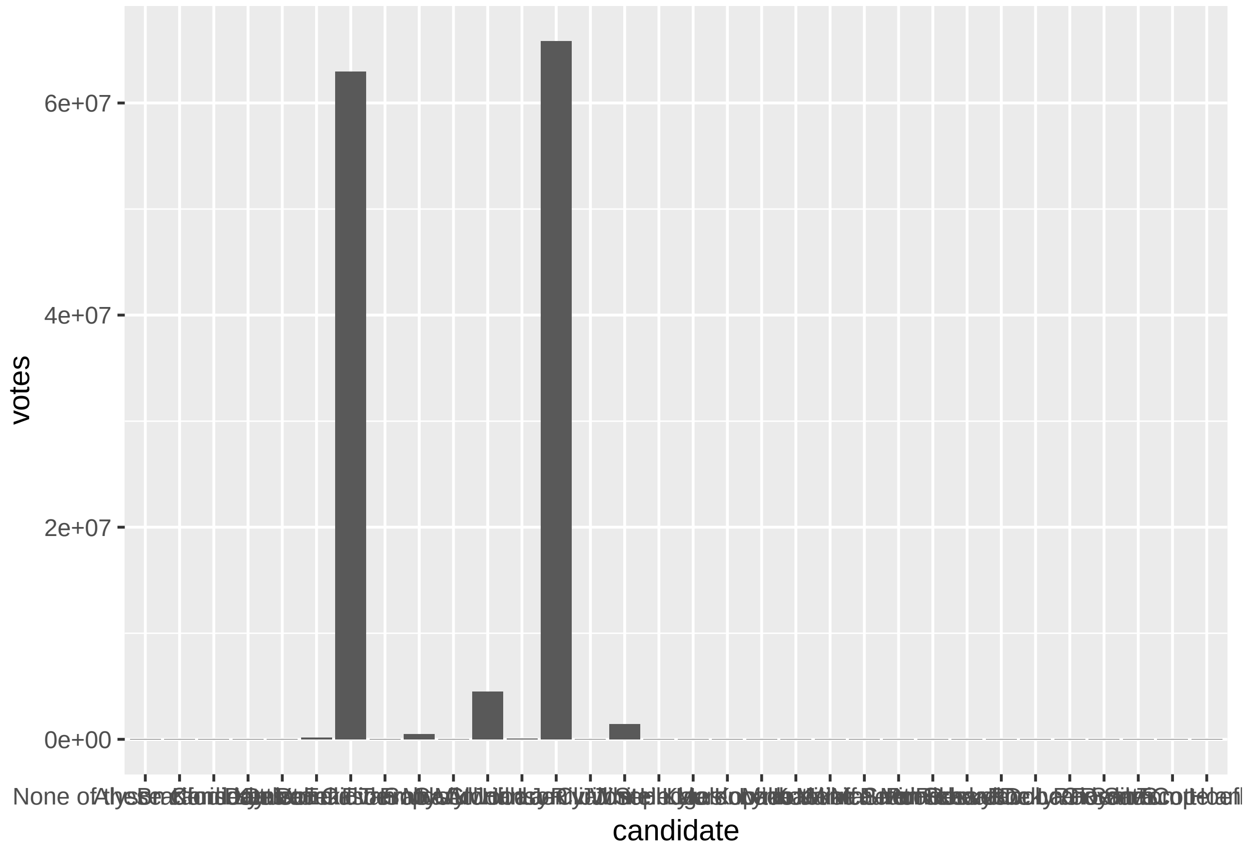
```r
# get the names of states
states = unique(election.raw$state)

# separate the federal summaries by NA county values
election_federal = filter(election.raw, is.na(county), fips=="US")

# separate state summaries as ones with fips = state name
election_state = filter(election.raw, is.na(county), fips %in% states) %>%
  filter(fips != "US")

# get the election data without NA counties
election = filter(election.raw, is.na(county) == FALSE)
```

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

```r
# get the number of named pres. candidates from election_federal
length(unique(election_federal$candidate))
```

```
## [1] 32
```

```r
# bar chart of votes recieved
ggplot(election_federal, aes(x = candidate, y = votes)) + geom_col()
```



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).

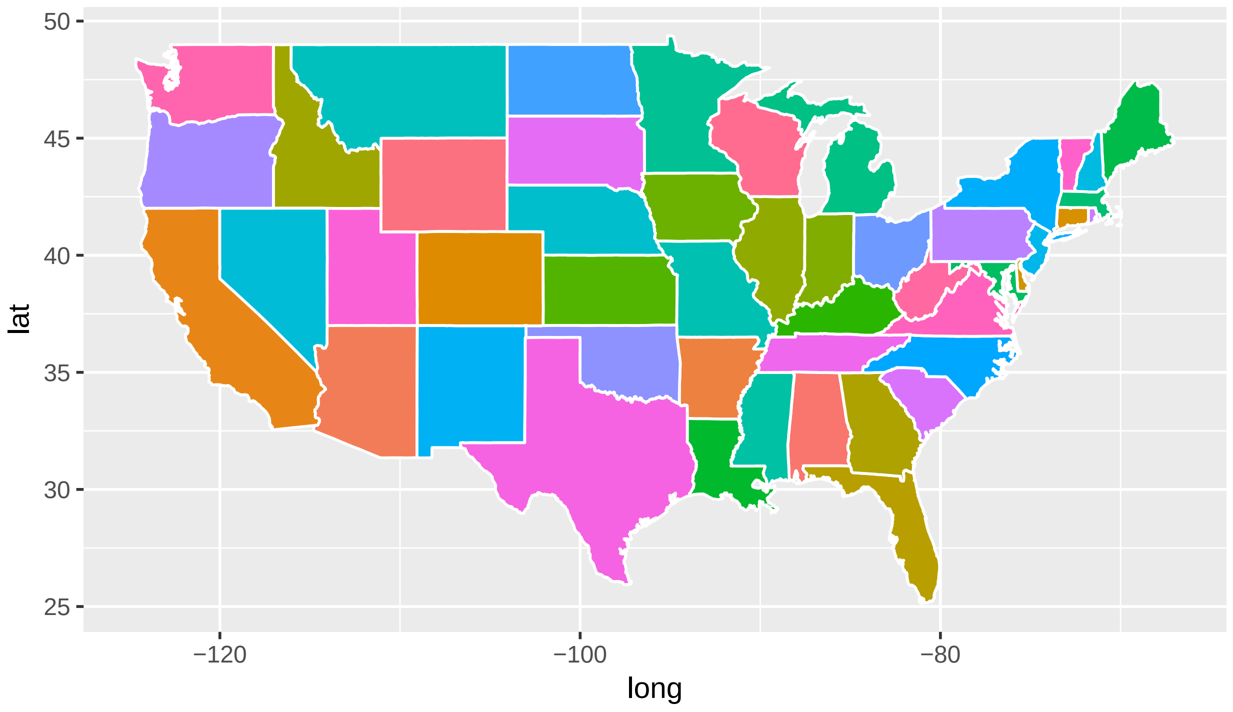
```
# calculate the total votes
total = sum(election_federal$votes)

county_winner = election %>%
  group_by(fips) %>% # group by county
  mutate(pct = votes/total) %>%
  top_n(1, pct) # get the candidate with the most votes

state_winner = election %>% # do the same for the state winners
  group_by(state) %>% # group by state
  mutate(pct = votes/total) %>%
  top_n(1, 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.

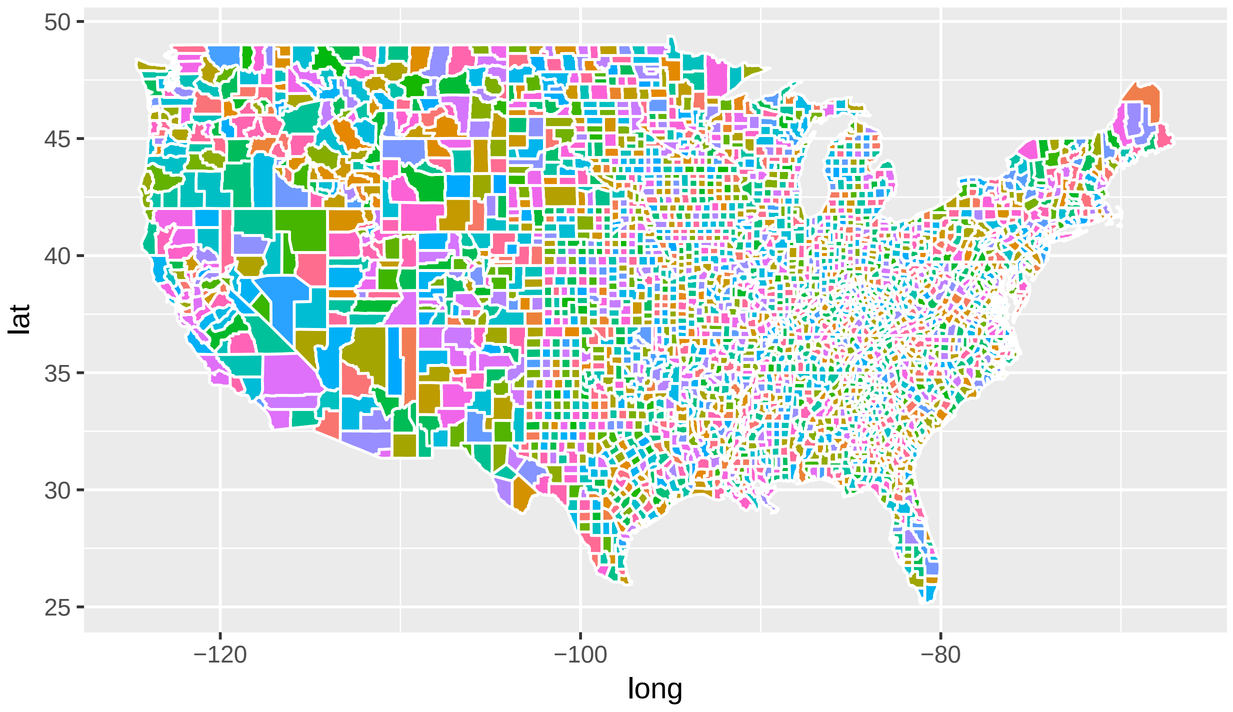
```
states = map_data("state")

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

```r
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)  # color legend is unnecessary and takes too long
```

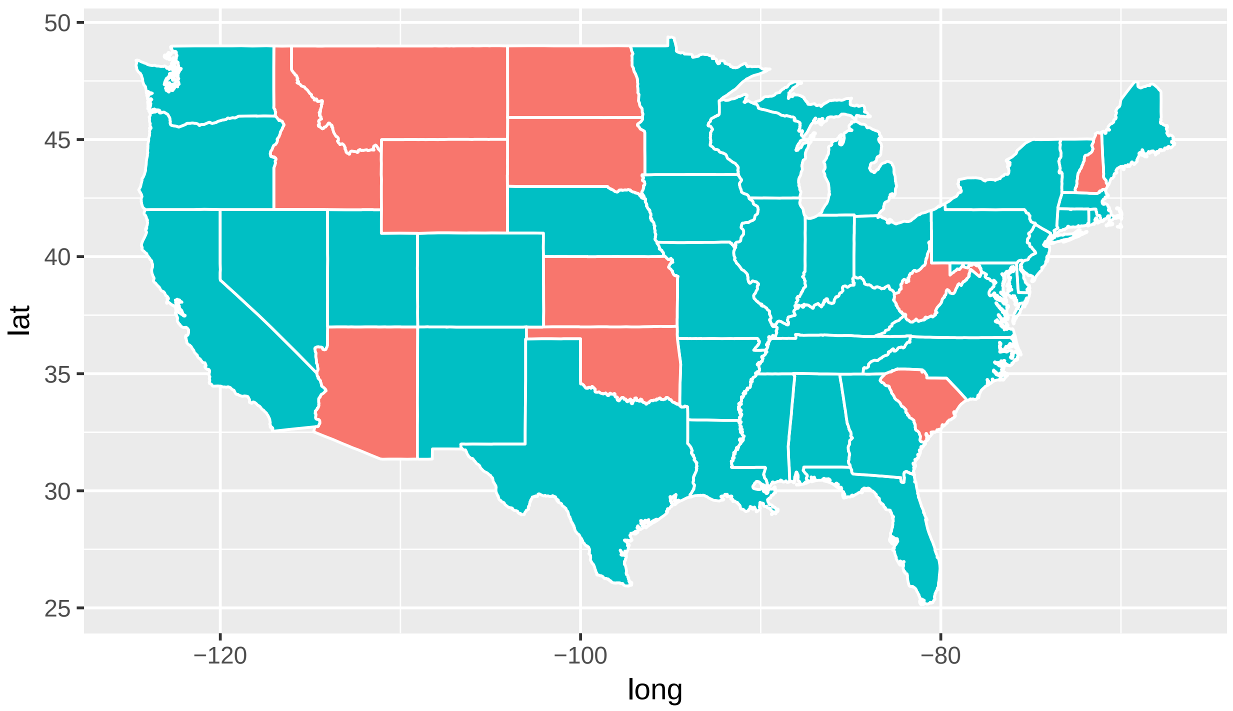


8. Now color the map by the winning candidate for each state. First, combine `states` variable and `state_winner` we created earlier using `left_join()`. Note that `left_join()` needs to match up values of states to join the tables; however, they are in different formats: e.g. `AZ` vs. `arizona`. Before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

```r
# make the replacement for the fips column
fips = state.abb[match(states$region, tolower(state.name))]

# append fips to states
states = states %>%
  mutate(region = as.factor(fips))

# add state_winner to states
states = left_join(states, state_winner, by = c("region" = "state"))
```

```
## Warning: Column `region`/`state` joining factors with different levels,
## coercing to character vector
```

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



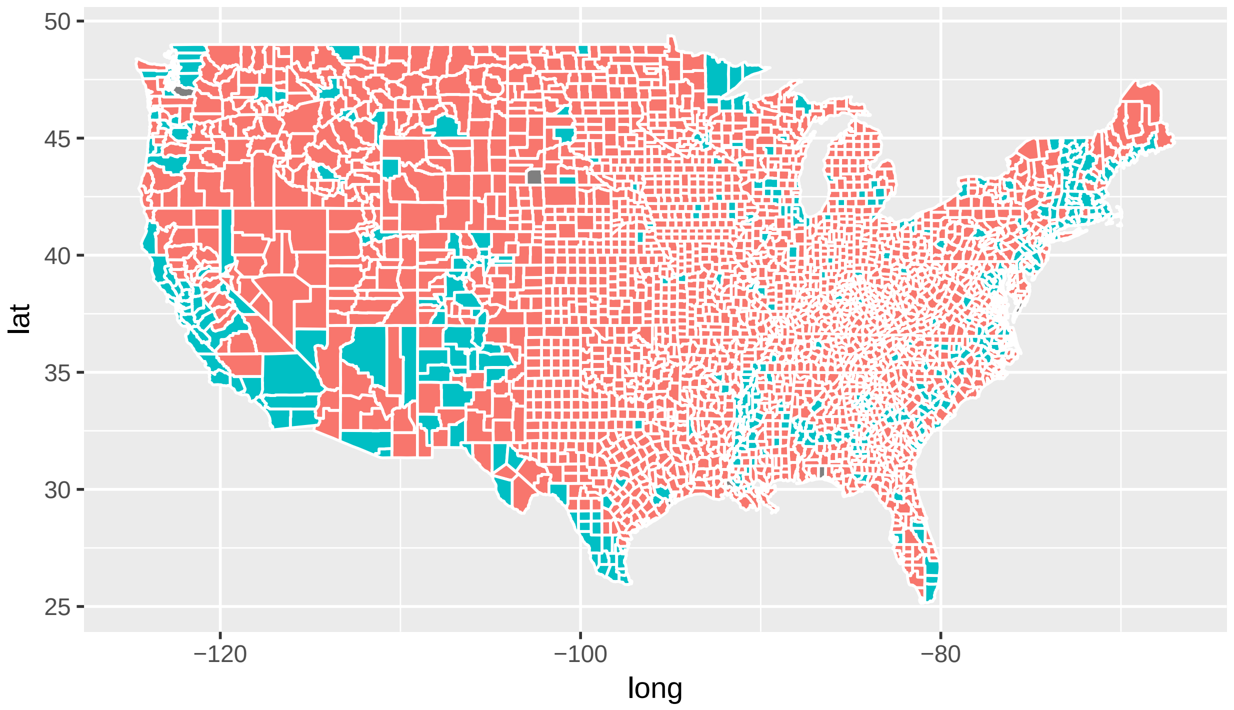
9. The variable `county` does not have `fips` column. So we will create one by pooling information from `maps::county.fips`. Split the `polyname` column to `region` and `subregion`. Use `left_join()` combine `county.fips` into `county`. Also, `left_join()` previously created variable `county_winner`.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

```r
# save the county fips assignments
county.fips = maps::county.fips

# separate region from subregion in $polyname
county.region = unlist(strsplit(county.fips$polyname, ","))

# remove $polyname
county.fips$polyname <- NULL

# make a blank vector of the form we want in our table, county.fips for relist()
skeleton <- vector(length = length(county.region[c(TRUE, FALSE)]), mode = "character")

# make the columns we want to add to county.fips
state.region = relist(county.region[c(TRUE, FALSE)], skeleton)
county.subregion = relist(county.region[c(FALSE, TRUE)], skeleton)

# add the columns to county.fips
county.fips$region <- state.region
county.fips$subregion <- county.subregion
# convert fips to factor to aid with left_join
county.fips$fips <- as.factor(county.fips$fips)
```

```r
# use leftjoin to combine county.fips into county
county = left_join(counties, county.fips, by = c("subregion", "region"))

# left_join the county_winners
county = left_join(county, county_winner)
```

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

```r
# plot the county winners
ggplot(data = county) +
  geom_polygon(aes(x = long, y = lat, fill = candidate, group = group), color = "white") +
  coord_fixed(1.3) +
  guides(fill=FALSE)  # color legend is unnecessary and takes too long
```



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.

```r
head(census)
```

```
## # A tibble: 6 x 37
##   CensusTract   State  County TotalPop   Men Women Hispanic White Black
##        <fctr>  <fctr>  <fctr>    <int> <int> <int>    <dbl> <dbl> <dbl>
## 1  1001020100 Alabama Autauga     1948   940  1008      0.9  87.4   7.7
## 2  1001020200 Alabama Autauga     2156  1059  1097      0.8  40.4  53.3
## 3  1001020300 Alabama Autauga     2968  1364  1604      0.0  74.5  18.6
## 4  1001020400 Alabama Autauga     4423  2172  2251     10.5  82.8   3.7
## 5  1001020500 Alabama Autauga    10763  4922  5841      0.7  68.5  24.8
## 6  1001020600 Alabama Autauga     3851  1787  2064     13.1  72.9  11.9
## # ... with 28 more variables: Native <dbl>, Asian <dbl>, Pacific <dbl>,
## #   Citizen <int>, Income <dbl>, IncomeErr <int>, IncomePerCap <int>,
## #   IncomePerCapErr <int>, Poverty <dbl>, ChildPoverty <dbl>,
## #   Professional <dbl>, Service <dbl>, Office <dbl>, Construction <dbl>,
## #   Production <dbl>, Drive <dbl>, Carpool <dbl>, Transit <dbl>,
## #   Walk <dbl>, OtherTransp <dbl>, WorkAtHome <dbl>, MeanCommute <dbl>,
## #   Employed <int>, PrivateWork <dbl>, PublicWork <dbl>,
## #   SelfEmployed <dbl>, FamilyWork <dbl>, Unemployment <dbl>
```

```
#unfortunately, didn't have time to get to this part
```

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.

```
census.del = census %>%
  filter(complete.cases(census)) %>% # remove rows with missing values
  mutate(Men = Men/TotalPop, Employed = Employed/TotalPop, Citizen = Citizen/TotalPop)  %>% # ‘Tot
  mutate(Minority = Hispanic + Black + Native + Asian + Pacific) %>% # combine into minority featu
  mutate(Hispanic=NULL, Black=NULL, Native=NULL, Asian=NULL, Pacific=NULL) %>% # remove the combin
  mutate(Walk = NULL, PublicWork = NULL, Construction = NULL) # remove ‘Walk‘, ‘PublicWork‘, ‘Cons
```

   - *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`.

```
census.subct = census.del %>%
  group_by(State, County) %>%
  add_tally() %>%
  #rename(., CountyTotal = n) %>%
  mutate(weight = TotalPop/n)
```

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

```
census.ct = census.subct %>%
  # summarize the numeric columns
  summarize_at(vars(TotalPop:Minority), funs(weighted.mean(., weight))) %>%
  ungroup()
```

   - *Print few rows of* `census.ct`:

```
head(census.ct)
```

```
## # A tibble: 6 x 29
##     State  County TotalPop       Men    Women    White   Citizen   Income
##    <fctr>  <fctr>    <dbl>     <dbl>    <dbl>    <dbl>     <dbl>    <dbl>
## 1 Alabama Autauga 6486.404 0.4843266 3348.805 75.78823 0.7374912 51696.29
## 2 Alabama Baldwin 7698.957 0.4884866 3934.167 83.10262 0.7569406 51074.36
## 3 Alabama Barbour 3325.195 0.5382816 1491.941 46.23159 0.7691222 32959.30
## 4 Alabama    Bibb 6380.718 0.5341090 2930.106 74.49989 0.7739781 38886.63
## 5 Alabama  Blount 7018.573 0.4940565 3562.081 87.85385 0.7337550 46237.97
## 6 Alabama Bullock 4263.211 0.5300618 1968.034 22.19918 0.7545420 33292.69
## # ... with 21 more variables: IncomeErr <dbl>, IncomePerCap <dbl>,
## #   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>
```

Dimensionality reduction

12. Run PCA for both county & sub-county level data. Save the principal components data frames, call them `ct.pc` and `subct.pc`, respectively. What are the most prominent loadings of the first two principal components PC1 and PC2?

```
# perfom pca
ct.pc = prcomp(census.ct[, 3:ncol(census.ct)], scale = TRUE)
subct.pc = prcomp(census.subct[, 4:ncol(census.subct)], scale = TRUE)

# get the first two loadings
ct.pc$rotation[,1:2]
```

```
##                          PC1         PC2
## TotalPop        -0.031162028  0.37386008
## Men              0.009450326 -0.13661721
## Women           -0.028112996  0.38630141
## White            0.225752691 -0.12273890
## Citizen          0.008867693 -0.16555896
## Income           0.315240742  0.20280923
## IncomeErr        0.164950947  0.23544269
## IncomePerCap     0.349077653  0.11780826
## IncomePerCapErr  0.192471942  0.11744885
## Poverty         -0.341718733 -0.02585608
## ChildPoverty    -0.342740984 -0.02898598
## Professional     0.248820729  0.03668661
## Service         -0.181364276 -0.02230406
## Office          -0.021065129  0.26353170
## Production      -0.117716950 -0.01955066
## Drive           -0.097748454  0.17680223
## Carpool         -0.076227800 -0.03810379
## Transit          0.068308102  0.13019164
## OtherTransp     -0.009612914 -0.02212970
## WorkAtHome       0.180197059 -0.26590269
## MeanCommute     -0.063573094  0.21488307
## Employed         0.327195096  0.03349149
## PrivateWork      0.050102052  0.29885655
## SelfEmployed     0.105343099 -0.34808659
## FamilyWork       0.053330049 -0.22807766
## Unemployment    -0.292568879  0.08454037
## Minority        -0.229194895  0.12100691
```

```
subct.pc$rotation[,1:2]
```

```
##                         PC1           PC2
## TotalPop        -0.03799670  0.0006207675
## Men             -0.01641401 -0.0507997584
## Women           -0.03663314  0.0087662351
## White           -0.23959677 -0.3058424607
## Citizen         -0.15916427 -0.2259363566
## Income          -0.30292370  0.1526980669
## IncomeErr       -0.19843494  0.2205612774
## IncomePerCap    -0.31760985  0.1672712559
## IncomePerCapErr -0.21124105  0.1967913952
## Poverty          0.30500855  0.0524833818
## ChildPoverty     0.29814504  0.0282760768
## Professional    -0.30621740  0.1415925623
## Service          0.26890100  0.0591383629
## Office           0.01309728 -0.0421516909
## Production       0.20689560 -0.1927033392
## Drive           -0.07963736 -0.3805592160
## Carpool          0.16212164 -0.0420656440
## Transit          0.05774953  0.3964700866
## OtherTransp      0.04558216  0.1333186937
## WorkAtHome      -0.17231202  0.0912108594
## MeanCommute     -0.01056043  0.2758307984
## Employed        -0.22132444  0.0606639690
## PrivateWork      0.04146074  0.0487410274
## SelfEmployed    -0.06884070  0.0143647386
## FamilyWork      -0.01487263 -0.0409877889
## Unemployment     0.25304861  0.0775269161
## Minority         0.24119599  0.3027304026
## CountyTotal      0.02544136  0.2866127377
## weight           0.01519161 -0.2331720362
```

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
# calculate the euclidean distance matrix
ct.dist = dist(census.ct[, 3:ncol(census.ct)], method="euclidean")

# perform hierarchial clustering with complete linkage
set.seed(1)
ct.h = hclust(ct.dist)
ct.clust = cutree(ct.h, 10) # cut to 10 clusters

# cluster again with ‘ct.pc‘
ct.pc.dist = dist(ct.pc$x[,1:5], method="euclidean")
ct.pc.h = hclust(ct.pc.dist)
ct.pc.clust = cutree(ct.pc.h, 10) # cut to 10 clusters
```

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))
```

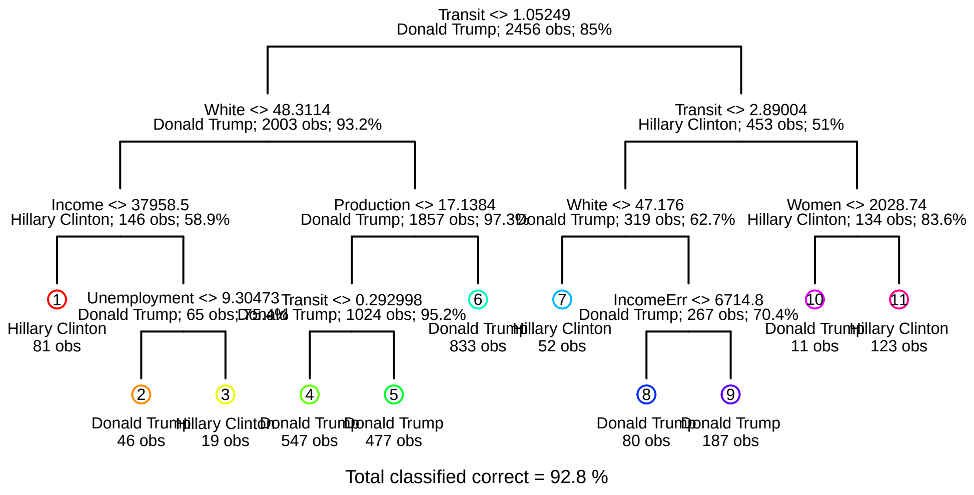
Using the following error rate function:

```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")
```

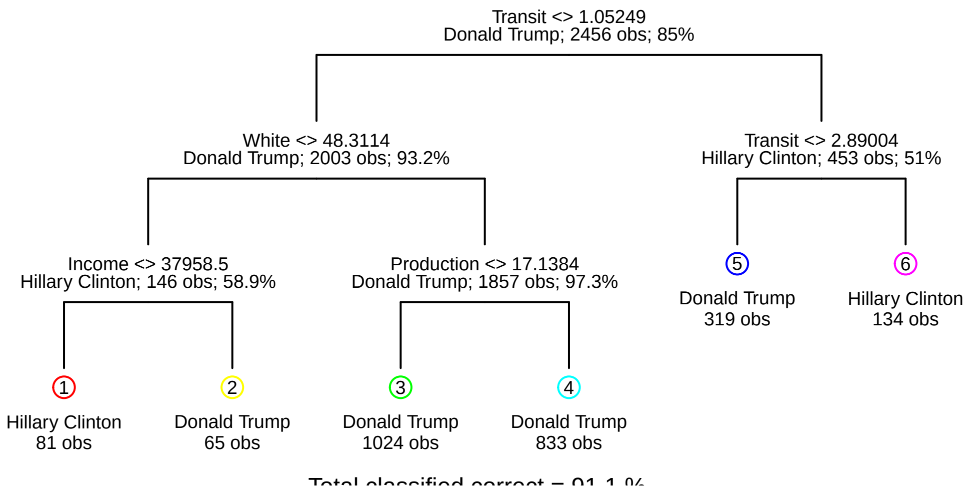
Classification: native attributes

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
    # make the decision tree to pick candidates
    elec.tree = tree(candidate~., data = trn.cl)
    # show the tree before pruning
    draw.tree(elec.tree, cex = 0.5, nodeinfo = TRUE)
    ```



```r
# train the tree with cross validation
cv.model = cv.tree(elec.tree, rand = folds, K = nfold, method="misclass")

# find the best size, and choose
best.size <- min(cv.model$size[which(cv.model$dev==min(cv.model$dev))])
# prune to the best size
elec.tree.cv = prune.tree(elec.tree, best = best.size)

# visualize the tree after pruning
draw.tree(elec.tree.cv, cex = 0.6, nodeinfo = TRUE)
```



```
get.error.tree <- function(tree, Xtr, Xvl, Ytr, Yvl){
  ## get classifications for current training chunks
  predYtr = predict(object = tree, type = "class", newdata = Xtr)
  ## get classifications for current test chunk
  predYvl = predict(object = tree, type = "class", newdata = Xvl)
  data.frame(train.error = calc_error_rate(predYtr, Ytr),
             test.error = calc_error_rate(predYvl, Yvl))
}

records$tree <- get.error.tree(elec.tree.cv, trn.cl[, -1], tst.cl[, -1], trn.cl$candidate, tst.cl$cand
```

```
## Warning in records$tree <- get.error.tree(elec.tree.cv, trn.cl[, -1],
## tst.cl[, : Coercing LHS to a list
```

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`.

```
do.chunk <- function(chunkid, folddef, Xdat, Ydat, k){# the do.chunk code

  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)
```

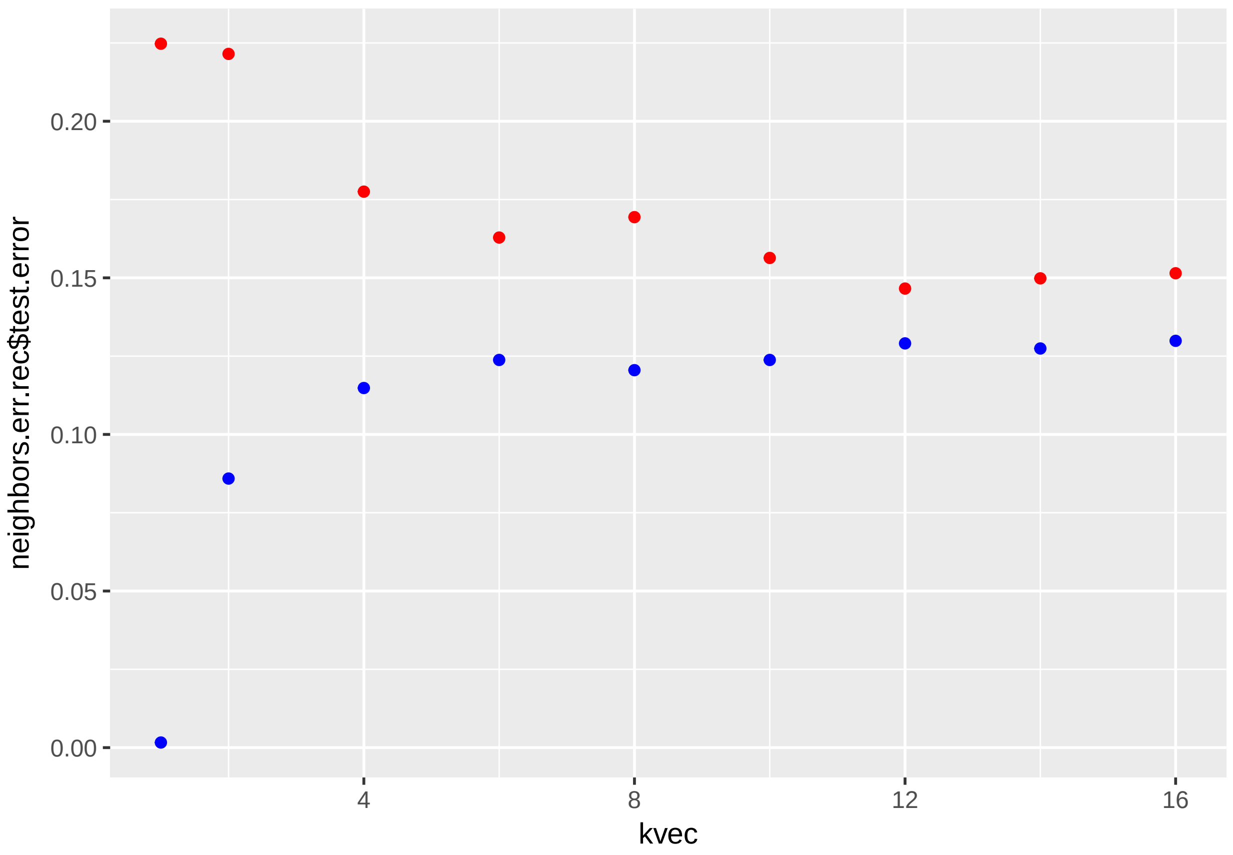
```r
  ## get classifications for current test chunk
  predYvl = knn(train = Xtr, test = Xvl, cl = Ytr, k = k)

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

set.seed(10) #set the desired seed

# create the possible number of neighbors, k
kvec = c(1, seq(2, 16, length.out=8))
error.folds.rec <- list()

# perform cross validation
for (j in 1:length(kvec)){
  error.folds=NULL
  error.folds = ldply(1:nfold, do.chunk, folds, trn.cl[, -1], trn.cl$candidate, k = kvec[j])
  error.folds.rec[[j]] <- error.folds
}

#concatinate the cross validation
error.folds.rec = do.call(rbind, error.folds.rec)

# determine the best number of neighbors given all of the chunks
k_best = kvec[which.min(error.folds.rec[,2]) %% 10]
```

```r
# Now we will get the error rates for each neighor
get.error.knn <- function(Xtr, Xvl, Ytr, Yvl, k){
  ## 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),
             test.error = calc_error_rate(predYvl, Yvl))
}

# save the errors to records
records$knn <- get.error.knn(trn.cl[, -1], tst.cl[, -1], trn.cl$candidate, tst.cl$candidate, k = k_best)
# print the validation and training errors for the knn
records$knn
```

```
##   train.error test.error
## 1   0.1123779  0.1775244
```

```r
# Now, to plot neighbors vs. tr and vl errors
## first, get the train and test err for each neighbor
### initiate variables
neighbors.err = NULL
neighbors.err.rec = list()

### loop through each neighbor and get the train & test errors
for (j in 1:length(kvec)){
  neighbors.err = get.error.knn(trn.cl[, -1], tst.cl[, -1], trn.cl$candidate, tst.cl$candidate, k = kvec[j]
  neighbors.err.rec[[j]] <- neighbors.err
}

### concatinate the resulting list
```

```
neighbors.err.rec = do.call(rbind, neighbors.err.rec)

## plot the train and test errors on top of each other
ggplot(neighbors.err.rec) + geom_point(aes(x = kvec, y = neighbors.err.rec$test.error), color = "red") + ge
```



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.

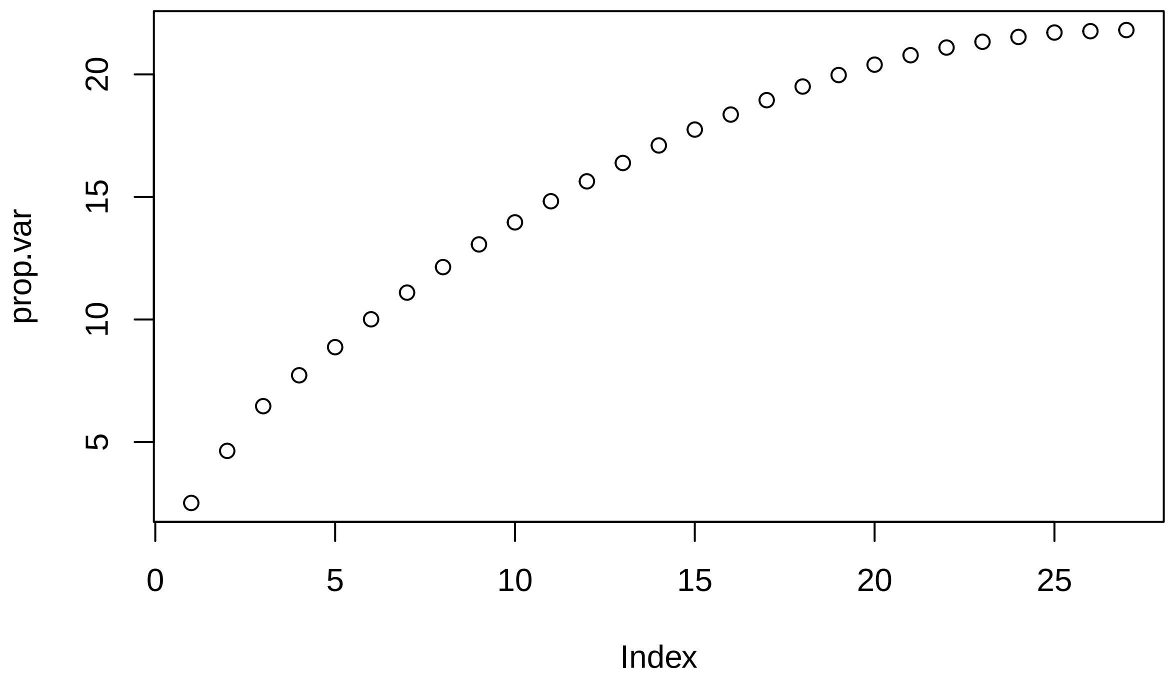
```
‘‘‘r
pca.records = matrix(NA, nrow=3, ncol=2)
colnames(pca.records) = c("train.error","test.error")
rownames(pca.records) = c("tree","knn","lda")
‘‘‘
```

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.

```
# compute the principal components
election.pc = prcomp(election.cl[, -1], scale = TRUE)
# use ‘summary‘ to get the cumulative variance explained
summary(election.pc)
```

```
## Importance of components:
##                           PC1    PC2    PC3     PC4     PC5     PC6
## Standard deviation     2.5154 2.1243 1.8246 1.25968 1.14802 1.13628
## Proportion of Variance 0.2344 0.1671 0.1233 0.05877 0.04881 0.04782
## Cumulative Proportion  0.2344 0.4015 0.5248 0.58356 0.63237 0.68019
##                            PC7     PC8     PC9    PC10    PC11   PC12
## Standard deviation     1.08819 1.04166 0.92524 0.90011 0.86100 0.8117
## Proportion of Variance 0.04386 0.04019 0.03171 0.03001 0.02746 0.0244
## Cumulative Proportion  0.72405 0.76424 0.79594 0.82595 0.85341 0.8778
##                          PC13    PC14    PC15    PC16    PC17    PC18
## Standard deviation     0.7493 0.71521 0.64798 0.61240 0.58752 0.55607
## Proportion of Variance 0.0208 0.01895 0.01555 0.01389 0.01278 0.01145
## Cumulative Proportion  0.8986 0.91755 0.93310 0.94699 0.95978 0.97123
##                           PC19    PC20    PC21    PC22    PC23    PC24
## Standard deviation     0.46798 0.42405 0.38557 0.31237 0.23535 0.19679
## Proportion of Variance 0.00811 0.00666 0.00551 0.00361 0.00205 0.00143
## Cumulative Proportion  0.97934 0.98600 0.99151 0.99512 0.99717 0.99861
##                          PC25    PC26    PC27
## Standard deviation     0.1801 0.05621 0.04500
## Proportion of Variance 0.0012 0.00012 0.00007
## Cumulative Proportion  0.9998 0.99993 1.00000
```

```r
# plot the proportion of variance explained
prop.var = cumsum(election.pc$sdev)
plot(prop.var)
```



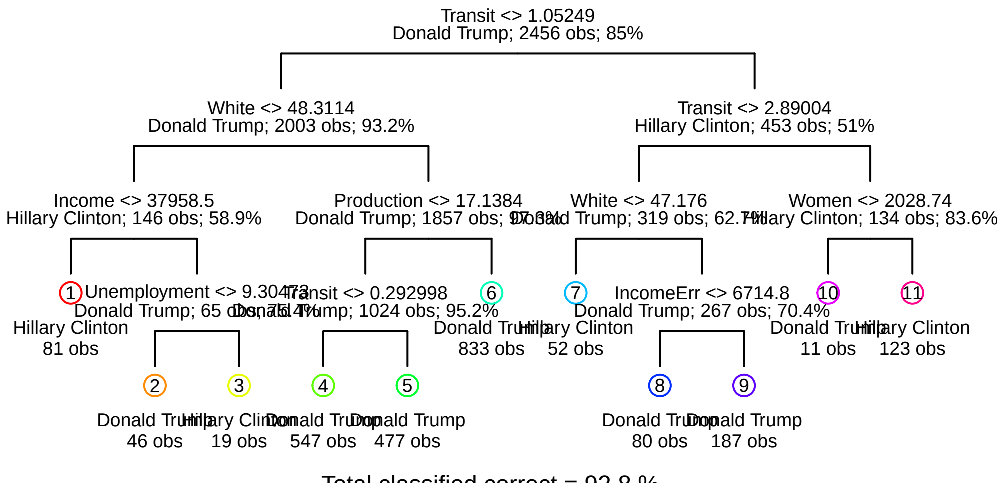
From the `summary`, we find that 14 is the minimimum number of PCA components needed to capture 90% of the variance.

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`.

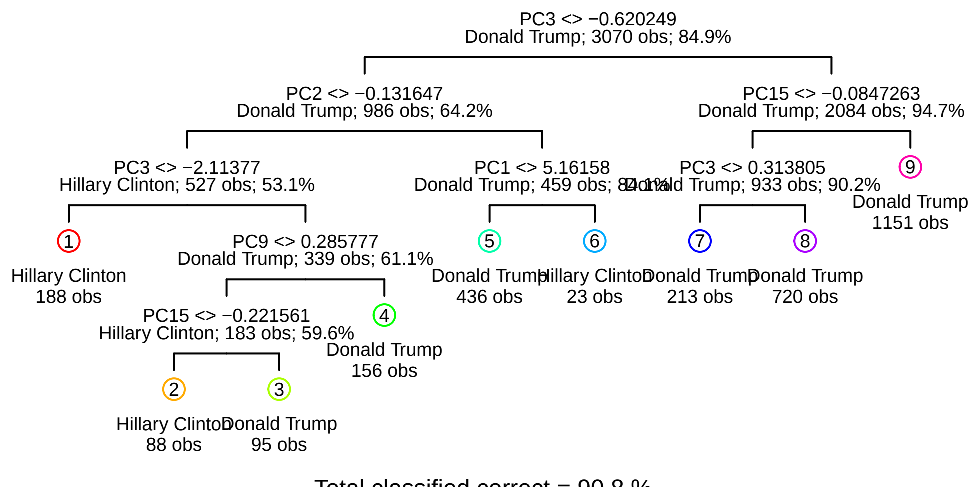
```r
tr.pca = cbind(election.cl, election.pc$x) %>%
  select(candidate, PC1:PC27) %>%
  as.data.frame()

# make the test data
theta = (as.matrix(election.cl[, -1]) %*% (election.pc$rotation/election.pc$scale))

test.pca = cbind(election.cl, theta) %>%
  select(candidate, PC1:PC27) %>%
  as.data.frame()
```

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

```
# make the decision tree to pick candidates
pca.tree = tree(candidate~., data = tr.pca)
# show the tree before pruning
draw.tree(elec.tree, cex = 0.6, nodeinfo = TRUE)
```



```
# train the tree with cross validation
cv.model = cv.tree(pca.tree, rand = folds, K = nfold, method="misclass")

# find the best size, and choose
best.size <- min(cv.model$size[which(cv.model$dev==min(cv.model$dev))])
# prune to the best size
pca.tree.cv = prune.tree(pca.tree, best = best.size)

# visualize the tree after pruning
draw.tree(pca.tree.cv, cex = 0.6, nodeinfo = TRUE)
```



```
get.error.tree <- function(tree, Xtr, Xvl, Ytr, Yvl){
  ## get classifications for current training chunks
  predYtr = predict(object = tree, type = "class", newdata = Xtr)
  ## get classifications for current test chunk
  predYvl = predict(object = tree, type = "class", newdata = Xvl)
  data.frame(train.error = calc_error_rate(predYtr, Ytr),
             test.error = calc_error_rate(predYvl, Yvl))
}

pca.records$tree <- get.error.tree(pca.tree.cv, tr.pca[, -1], test.pca[, -1], tr.pca$candidate, test.p
```

```
## Warning in pca.records$tree <- get.error.tree(pca.tree.cv, tr.pca[, -1], :
## Coercing LHS to a list
```

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

```
set.seed(10) #set the desired seed

# create the possible number of neighbors, k
error.folds.rec <- list()

# perform cross validation
for (j in 1:length(kvec)){
  error.folds=NULL
  error.folds = ldply(1:nfold, do.chunk, folds, tr.pca[, -1], tr.pca$candidate, k = kvec[j])
  error.folds.rec[[j]] <- error.folds
}

#concatinate the cross validation
```

```
error.folds.rec = do.call(rbind, error.folds.rec)

# determine the best number of neighbors given all of the chunks
k_best = kvec[which.min(error.folds.rec[,2]) %% 10]

# save the errors to records
pca.records$knn <- get.error.knn(tr.pca[, -1], test.pca[, -1], test.pca$candidate, test.pca$candidate,
# print the validation and training errors for the knn on pca
pca.records$knn
```

```
##   train.error test.error
## 1  0.05960912  0.4912052
```

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 tree plots in question 13, I see two important splits occuring on the `transit` response favoring Donald Trump for lower transit numbers. This gives a possible insight that supports the blog post that the people who voted for Trump had low poll turn out, but still came out and voted in the general election.

Unfortunately, the test data from the PCA loadings is not scaled properly, and I didn't have time to fix the error. This is because the loadings are scaled, but the responses in the `election.cl` data are not. I think using some form of `predict` might make more useable data.

Also, the state level results map is felatively innacurate when compared to the Washington Post results map. This may partially be due to the fact that the WP map is colored by electoral college votes.

A possible direction is to purse the transportation as a factor of voter turnout for Donald Trump by taking more finely grained data on county to county and state level transport.

Taking it further

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

    I was not able to make it to this question either.