Final Project*

PSTAT 231

Villaseñor-Derbez J.C. | 8749749

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

There are a number of factors that make it difficult to predict the outcome of voting. Even if we now people's political affiliation, the simple act of going out and actually voting (think about it as probability of voting) is governed by many determinustic processes. Moreover, census or random sample questionaires of support for a candidate may obscure a person's true voting intentions. In the scope of this project, the have high ressolution at the sub-county level, but still focus on mean trends, rather than on individual-level characteristics.

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

Unlike common polls, Nate combined predictions by many different polls, weighted by their historical accuracy, and demographic information about different electoral districts. This is, off course, in addition to standard ML procedures such as cross-validation and having testing and training dataset.

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

It is difficult to say what went wrong in 2016. My intuition is that not only were some polls skewed, but that the two strongest presidential candidates, Hillary and Trump, ma have caused a change in voting behavior -incentivizing more people to go out and vote. This, ultimately, might have changed the rules and conditions, as well as the composition, of total electoral turnout, compared to previous elections.

*Code available on GitHUb at: https://github.com/jcvdav/PSTAT231/tree/master/final_project

```r
election_raw <- read.csv(here("data", "election", "election.csv")) %>%
  as_tibble()

census_meta <- read.csv(here("data", "census", "metadata.csv"), sep = ";") %>%
  as_tibble()

census <- read.csv(here("data", "census","census.csv")) %>%
  as_tibble() %>%
  mutate(CensusTract = as.factor(CensusTract))
```

4. Remove summary rows from `election.raw` data:

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

```r
election_federal <- election_raw %>%
  filter(fips == "US")

election_state <- election_raw %>%
  filter(state != "US", is.na(county))

election <- election_raw %>%
  filter(!is.na(county))
```

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

There were 31 explicitly mentioned presidential candidates, plus a category of `None of these acandidates`. Figure 1 shows the votes (on a log_{10}-scale) that each candidate received.

```
election_federal %>%
  group_by(candidate) %>%
  summarize(votes = sum(votes, na.rm = T)) %>%
  ungroup() %>%
  mutate(candidate = fct_reorder(.f = candidate, .x = votes)) %>%
  ggplot(aes(x = candidate, y = votes)) +
  geom_col() +
  coord_flip() +
  scale_y_continuous(trans = "log10") +
  labs(x = "Candidate", y = "Votes (log-10 Scale)")
```

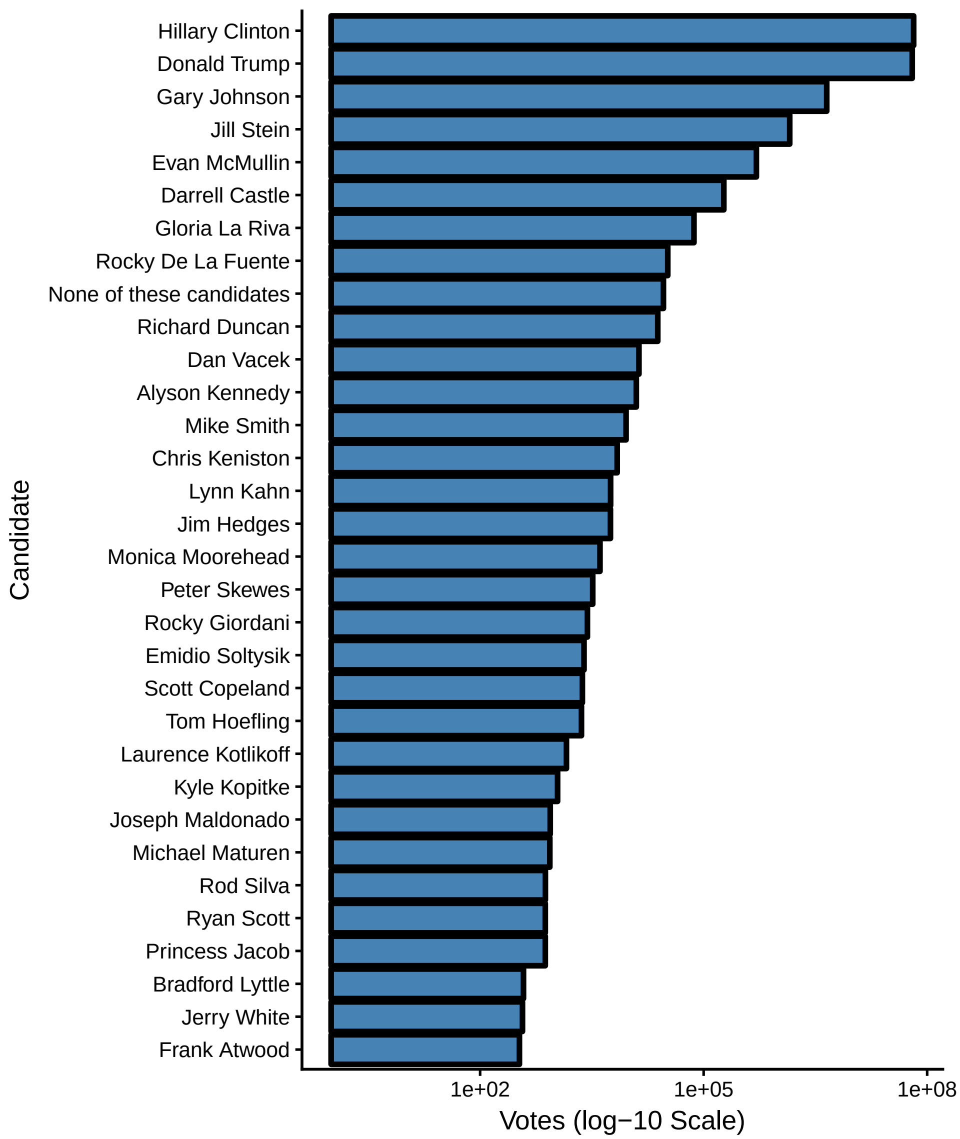


Figure 1: Number of votes that each presidential candidate received in the 2018 Presidential Elections.

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

```
county_winner <- election %>%
  group_by(fips) %>%
  mutate(total = sum(votes),
         pct = votes / total) %>%
  top_n(1) %>%
  ungroup() %>%
  mutate(fips = as.numeric(as.character(fips)))
```

```
## Selecting by pct
```

```
state_winner <- election_state %>%
  group_by(state) %>%
  mutate(total = sum(votes),
         pct = votes / total) %>%
  top_n(1) %>%
  ungroup() %>%
  mutate(state = as.character(state))
```

```
## Selecting by pct
```

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

```
my_fun <- function(long, lat) {
  data <- cbind(long, lat) %>%
    rbind(cbind(long[1], lat[1]))
  st_sfc(st_polygon(list(as.matrix(data))))
}
state_dictionary <- tibble(abb = state.abb, state = tolower(state.name))

states <- map_data("state") %>%
  group_by(group, region) %>%
  summarize(geometry = my_fun(long, lat)) %>%
  ungroup() %>%
  st_sf(crs = 4326) %>%
  left_join(state_dictionary, by = c("region" = "state"))

counties <- map_data("county") %>%
  group_by(group, region, subregion) %>%
  summarize(geometry = my_fun(long, lat)) %>%
  ungroup() %>%
  st_sf(crs = 4326)
```

```
ggplot(data = counties,
       mapping = aes(fill = subregion)) +
  geom_sf(data = counties, color = "white") +
  ggtheme_map() +
  theme(legend.position = "None")
```

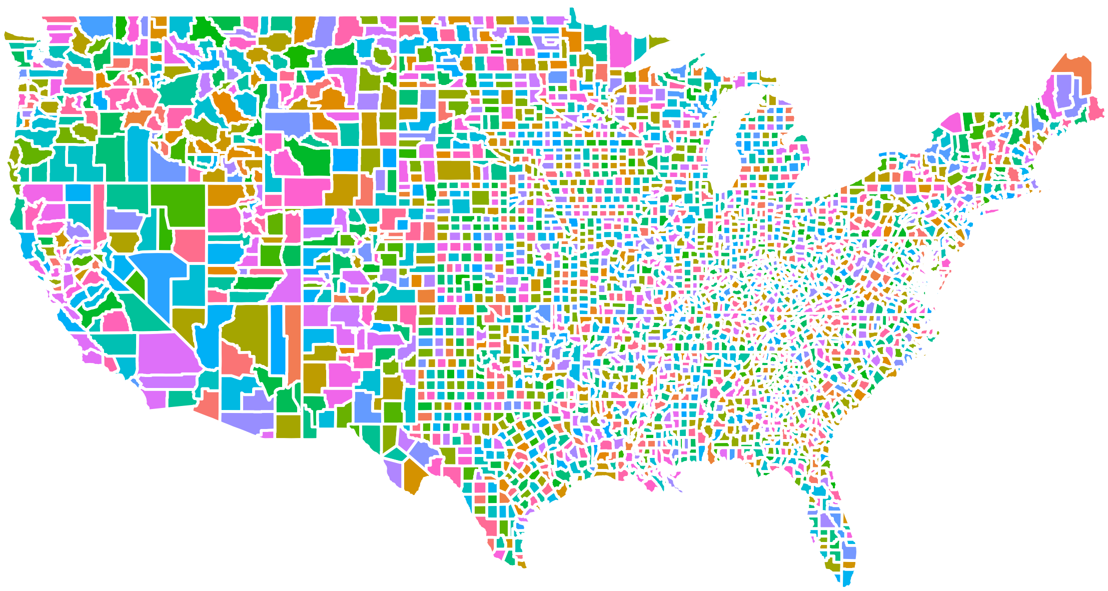


Figure 2: Map of all US counties.

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 %>%
  left_join(state_winner, by = c("abb" = "state")) %>%
  ggplot(mapping = aes(fill = candidate)) +
  geom_sf(color = "white") +
  ggtheme_map() +
  scale_fill_brewer(palette = "Set1") +
  guides(fill = guide_legend(title = "Winner"))
```

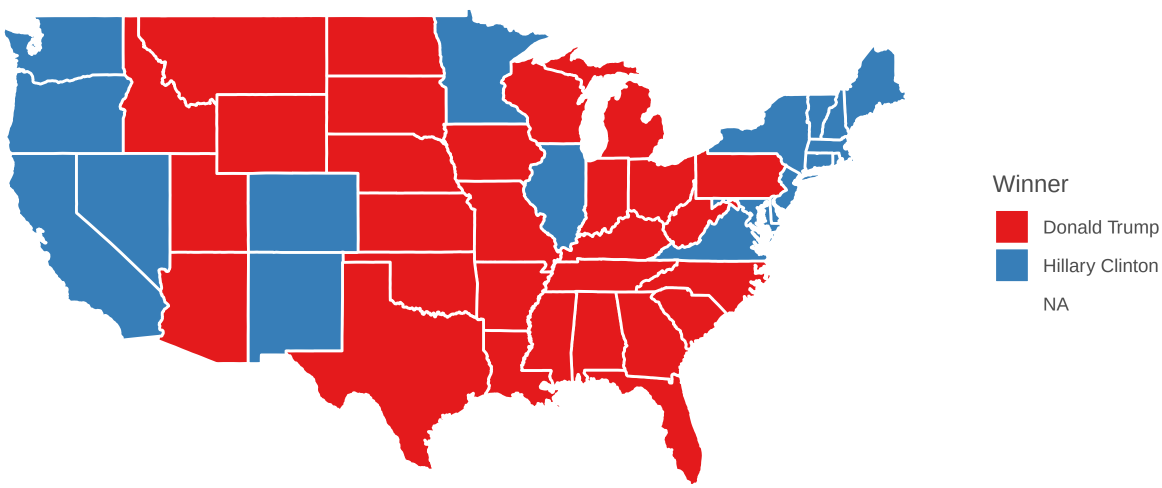


Figure 3: Map of US states colored by winning candidate.

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()` to 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.

```
county_fips <- maps::county.fips %>%
  mutate(region = str_extract(polyname, ".+,"),
         subregion = str_extract(polyname, ",.+"),
         region = str_remove(region, ","),
         subregion = str_remove(subregion, ",")) %>%
  select(-polyname)

counties %>%
  left_join(county_fips, by = c("region", "subregion")) %>%
  left_join(county_winner, by = "fips") %>%
  ggplot(mapping = aes(fill = candidate)) +
  geom_sf(color = "white") +
  ggtheme_map() +
  scale_fill_brewer(palette = "Set1") +
  guides(fill = guide_legend(title = "Winner"))
```

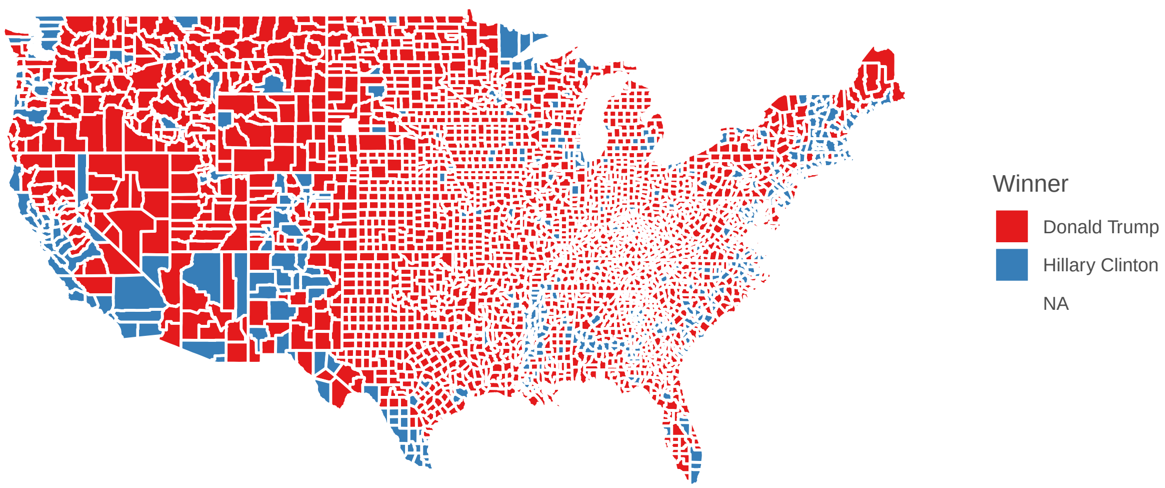


Figure 4: Map of US counties colored by winning candidate.

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.

I am going to use the census data and electoral outcomes to build a logistic regression model. In this case, I will use only demographics and population size as predictors for winning candidate at the county level. I begin by creating a dataset where I calculate the average measure across all demographics at the county level.

```
model_data <- census %>%
  select(-CensusTract) %>%
  group_by(State, County) %>%
  summarise_all(mean, na.rm = T) %>%
  ungroup() %>%
  mutate(State = tolower(State),
         County = tolower(County)) %>%
  rename(region = State,
         subregion = County) %>%
  left_join(county_fips, by = c("region", "subregion")) %>%
  left_join(county_winner, by = "fips") %>%
  drop_na(candidate) %>%
  mutate(candidate = ifelse(candidate == "Hillary Clinton", 0, 1)) %>%
  select(-fips)
```

I then fit a binary logistic regression, which predicts Hillary / Trump as options.

```
model <- glm(candidate ~  TotalPop + Men + Hispanic + White + Black + Native + Asian + Pacific + Citizen
```

I then take the predictions and plot them on a map. The map showing the predicted outcomes suggests that demographics are good predictors of election results. A regression table is also included.

```
model_pred <- model_data %>%
  mutate(prediction = model$fitted.values,
         prediction = ifelse(prediction <= 0.5, "Hillary Clinton", "Donald Trump")) %>%
  select(region, subregion, prediction, candidate)

counties %>%
  left_join(model_pred, by = c("region", "subregion")) %>%
  ggplot(mapping = aes(fill = prediction)) +
  geom_sf(color = "white") +
  ggtheme_map() +
  scale_fill_brewer(palette = "Set1") +
  guides(fill = guide_legend(title = "Predicted winner"))
```

```
stargazer::stargazer(model,
                     single.row = T,
                     header = F)
```

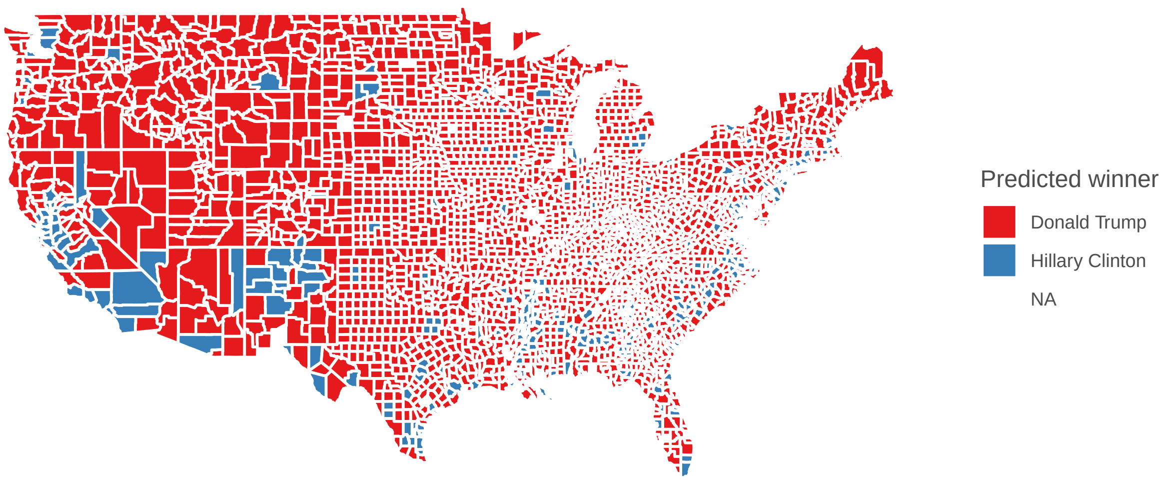


Figure 5: Map of US counties colored by PREDICTED winning candidate as a function of some demographic characteristics.

Table 1:

	Dependent variable:
	candidate
TotalPop	0.001** (0.0004)
Men	0.003*** (0.001)
Hispanic	−0.041 (0.047)
White	0.042 (0.048)
Black	−0.042 (0.047)
Native	−0.037 (0.050)
Asian	−0.807*** (0.076)
Pacific	−0.428 (0.428)
Citizen	−0.003*** (0.0004)
Constant	1.057 (4.679)
Observations	3,017
Log Likelihood	−730.654
Akaike Inf. Crit.	1,481.309
Note:	*p<0.1; **p<0.05; ***p<0.01

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.

I removed Women (complementary variable to Men) and all the variables that make up `Minority`.

```
census_del <- census %>%
  drop_na() %>%
  mutate(Men = Men / TotalPop,
         Employed = Employed / TotalPop,
         Citizen = Citizen / TotalPop,
         minority = Hispanic + Black + Native + Asian + Pacific) %>%
  select(-c(Walk, PublicWork, Construction, Hispanic, Black, Native, Asian, Pacific, Women, White))
```

- 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) %>%
  mutate(CountyTotal = sum(TotalPop)) %>%
  mutate(weight = TotalPop / CountyTotal) %>%
  ungroup()
```

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

```
census_ct <- census_subct %>%
  group_by(State, County) %>%
  summarize_at(.vars = vars(Men, Citizen, Income, IncomeErr,
                            IncomePerCap, IncomePerCapErr,
                            Poverty, ChildPoverty, Professional,
                            Service, Office, Production, Drive,
                            Carpool, Transit, OtherTransp, WorkAtHome,
                            MeanCommute, Employed, PrivateWork,
                            SelfEmployed, FamilyWork, Unemployment,
                            minority), list(~ weighted.mean(., w = weight))) %>%
  ungroup()
```

- Print few rows of `census.ct`:

```
head(census_ct)
```

```
## # A tibble: 6 x 26
##   State County   Men Citizen Income IncomeErr IncomePerCap IncomePerCapErr
##   <fct> <fct>  <dbl>   <dbl>  <dbl>     <dbl>        <dbl>           <dbl>
## 1 Alab~ Autau~ 0.484   0.737 51696.     7771.       24974.           3434.
## 2 Alab~ Baldw~ 0.488   0.757 51074.     8745.       27317.           3804.
## 3 Alab~ Barbo~ 0.538   0.769 32959.     6031.       16824.           2430.
## 4 Alab~ Bibb   0.534   0.774 38887.     5662.       18431.           3074.
## 5 Alab~ Blount 0.494   0.734 46238.     8696.       20532.           2052.
```

```
## 6 Alab~ Bullo~ 0.530   0.755 33293.     9000.       17580.           3111.
## # ... with 18 more variables: 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>
```

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.

```
census_subct_pca <- census_subct %>%
  select(-c(CensusTract, State, County, CountyTotal, weight)) %>%
  prcomp(scale = T, center = T)

census_ct_pca <- census_ct %>%
  select(-c(State, County)) %>%
  prcomp(scale = T, center = T)

subct_pc <- tibble(PC1 = census_subct_pca$x[, 1],
                   PC2 = census_subct_pca$x[, 2])

ct_pc <- tibble(PC1 = census_ct_pca$x[, 1],
                PC2 = census_ct_pca$x[, 2])
```

What are the most prominent loadings?

I calculate the total magnitude between the first two PCs, and then order them in descending order. For the sub-county data, the top 10 loadings are for the variables shown below. These same variables (arrows) can be seen in the biplot below.

```
tibble(var = rownames(census_subct_pca$rotation),
       l1 = census_subct_pca$rotation[, 1],
       l2 = census_subct_pca$rotation[, 2]) %>%
  mutate(l = sqrt(l1 ^ 2 + l2 ^ 2)) %>%
  arrange(desc(l)) %>%
  head(10)
```

```
## # A tibble: 10 x 4
##    var                  l1      l2     l
##    <chr>             <dbl>   <dbl> <dbl>
##  1 Drive            0.0634 -0.499  0.503
##  2 Transit         -0.0396  0.484  0.486
##  3 minority        -0.219   0.294  0.366
##  4 IncomePerCap     0.335   0.146  0.366
##  5 Professional     0.323   0.119  0.344
##  6 Poverty         -0.310   0.135  0.339
##  7 Income           0.321   0.0988 0.336
##  8 ChildPoverty    -0.305   0.101  0.321
##  9 IncomePerCapErr  0.229   0.209  0.310
## 10 IncomeErr        0.218   0.210  0.302
```

```r
biplot(census_subct_pca, xlabs = rep(".", nrow(census_subct)))
```

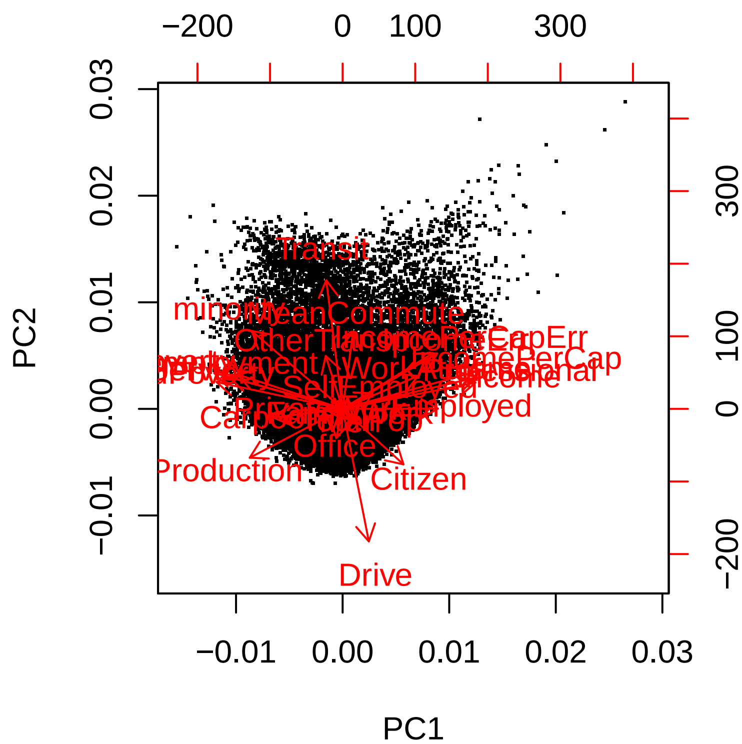


Figure 6: Biplot for subcounty PCA

For the county-level data, the top 10 variables are shown in the table, and then again in the biplot. Variables `Drive` and `Income` appear again, suggesting that they are important at different scales. The biplot below shows this information again.

```r
tibble(var = rownames(census_ct_pca$rotation),
       l1 = census_ct_pca$rotation[, 1],
       l2 = census_ct_pca$rotation[, 2]) %>%
  mutate(l = sqrt(l1 ^ 2 + l2 ^ 2)) %>%
  arrange(desc(l)) %>%
  head(10)
```

```
## # A tibble: 10 x 4
##    var              l1      l2     l
##    <chr>         <dbl>   <dbl> <dbl>
##  1 PrivateWork  0.0480  0.438  0.440
##  2 SelfEmployed 0.0946 -0.410  0.420
##  3 WorkAtHome   0.179  -0.375  0.416
##  4 IncomePerCap 0.367   0.0838 0.377
##  5 Income       0.339   0.155  0.373
##  6 Drive       -0.111   0.341  0.359
##  7 ChildPoverty -0.344  -0.0455 0.347
```

```
##   8 Poverty      -0.340  -0.0687 0.347
##   9 Employed      0.334   0.0438 0.337
## 10 Unemployment -0.286   0.0452 0.289
```

```r
biplot(census_ct_pca, xlabs = rep(".", nrow(census_ct)))
```

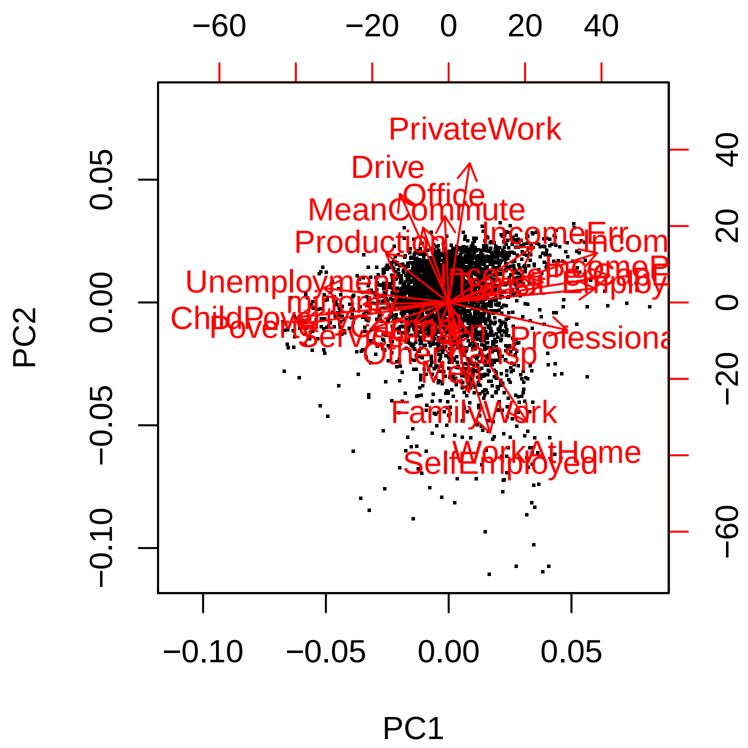
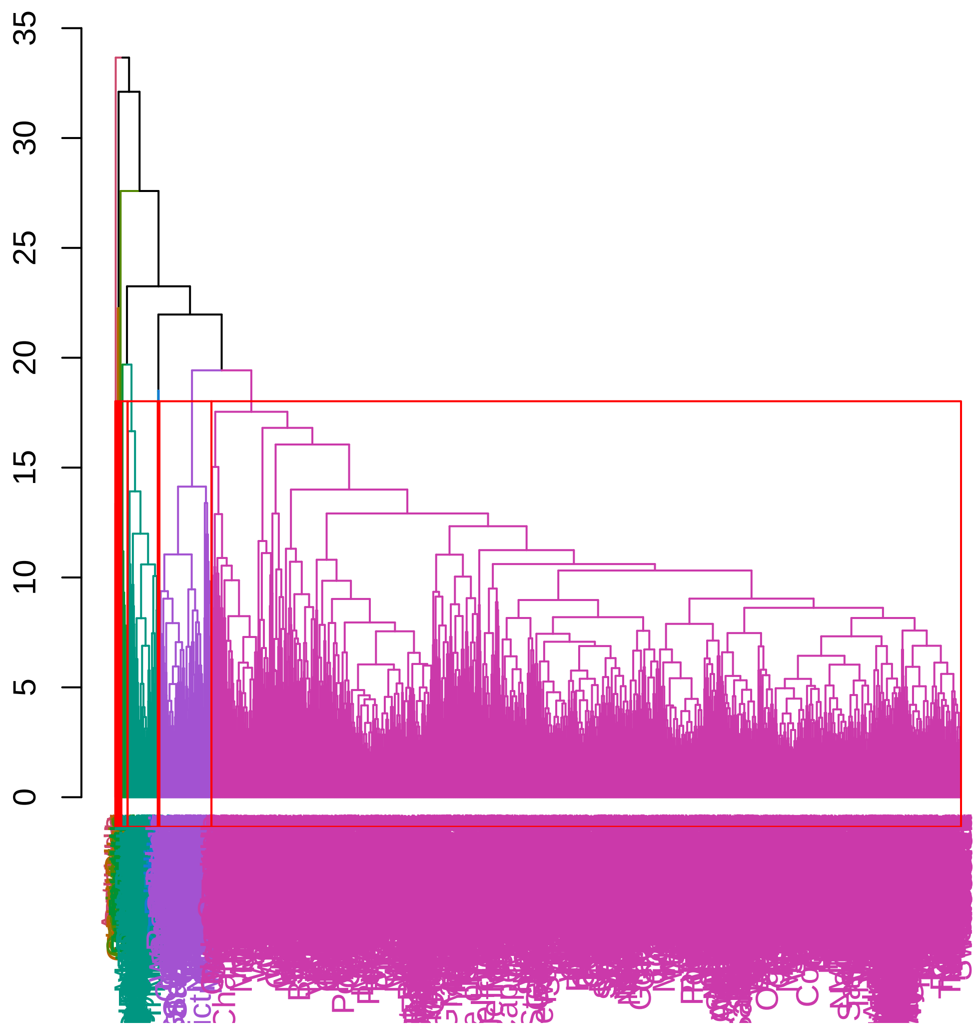


Figure 7: Biplot for county-level Principal Component Analyses.

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?

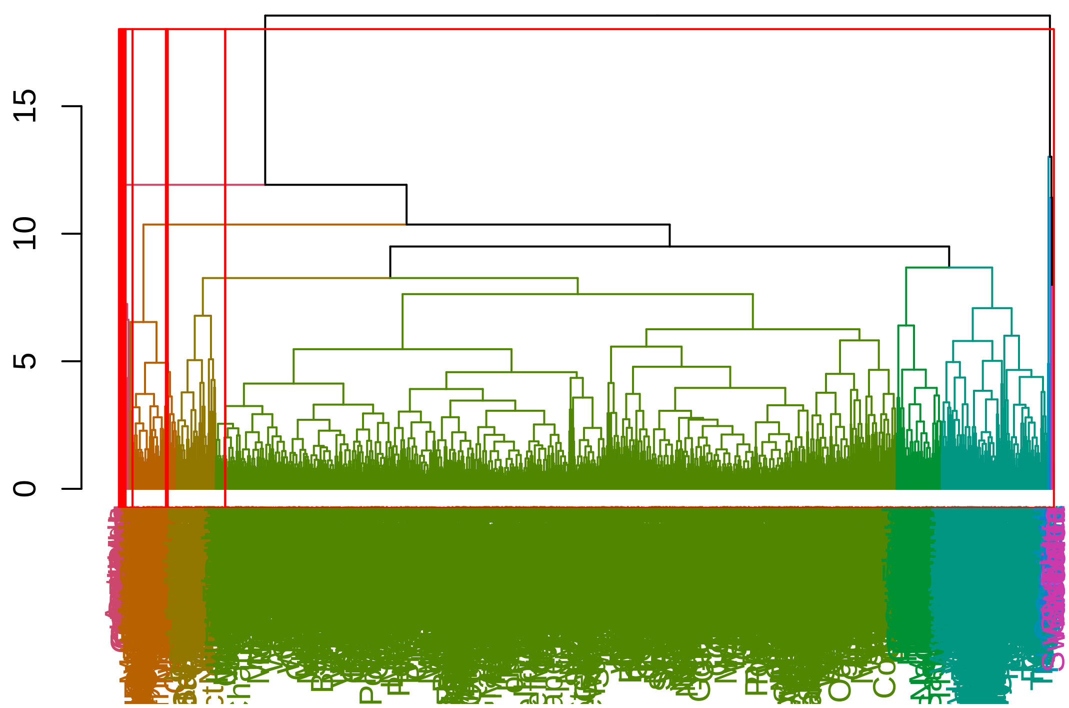
```
census_ct_clust <- census_ct %>%
  ungroup() %>%
  select(-c(State, County)) %>%
  scale() %>%
  dist() %>%
  hclust(method = "complete")
```

```
census_ct_clust %>%
  as.dendrogram() %>%
  color_labels(k = 10) %>%
  color_branches(k = 10) %>%
  set_labels(labels = census_ct$County) %>%
  plot()
rect.hclust(census_ct_clust, k = 10)
```



```
census_ct_pca_clust <- census_ct_pca$x[, 1:5] %>%
  scale() %>%
  dist() %>%
  hclust(method = "complete")

census_ct_pca_clust %>%
  as.dendrogram() %>%
  color_labels(k = 10) %>%
  color_branches(k = 10) %>%
  set_labels(labels = census_ct$County) %>%
  plot()
rect.hclust(census_ct_clust, k = 10)
```



I took a slightly different, more tidy approach to prepping the data for classification. But they produce the same datasets.

```
tmp_ct_winner <- county_winner %>%
  select(-c(county, state)) %>%
  left_join(county_fips, by = "fips") %>%
  rename(county = subregion, state = region) %>%
  mutate(county = str_remove(county, pattern = "county | columbia | city | parish"))

tmp_census_ct <- census_ct %>%
  mutate_at(vars(State, County), tolower) %>%
  rename(state = State, county = County)

election_cl <- tmp_ct_winner %>%
  left_join(tmp_census_ct, by = c("state", "county")) %>%
  drop_na()

election_cl_attr <- election_cl %>%
  select(county, fips, state, votes, pct)

attr(election_cl, "location") <- election_cl_attr

election_cl <- election_cl %>%
  select(-c(county, fips, state, votes, pct)) %>%
  mutate_at(vars(-candidate), scale)
```

Split into training and testing datasets

```
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, ]
```

Create 10 cross-validation folds

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

Create functions and objects to be used throughout

```
calc_error_rate <- function(predicted.value, true.value) {
  mean(true.value!=predicted.value)
}

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

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.

```
ctrl <- tree.control(nobs = length(trn_cl$candidate), minsize = 5, mindev = 1e-5)
main_tree <- tree(candidate ~ ., data = trn_cl, control = ctrl)
```

```
draw.tree(main_tree, nodeinfo = T, cex = 0.6)
```

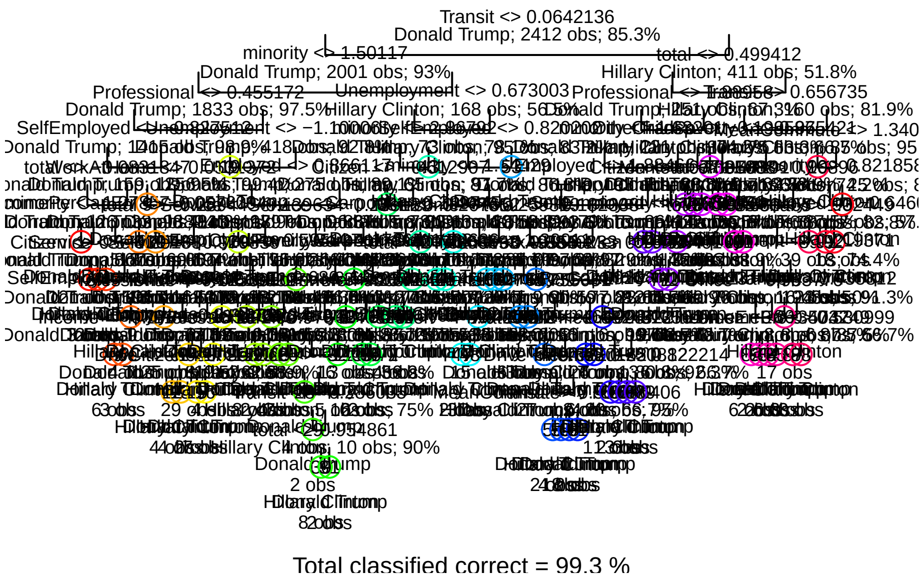


Figure 8: Original tree.

```
# Perform cross validation
cv_tree <- cv.tree(object = main_tree,
                   rand = folds,
                   method = "misclass")
```

Inspecting misclassification error visually and with a table, we see that a tree of size 8 produces the lowest misclassification error:

```
plot(cv_tree)
```

```
# create a tidy version of the diagnostics
cv_tree_tidy <- tibble(size = cv_tree$size,
                       misclass = cv_tree$dev)
```

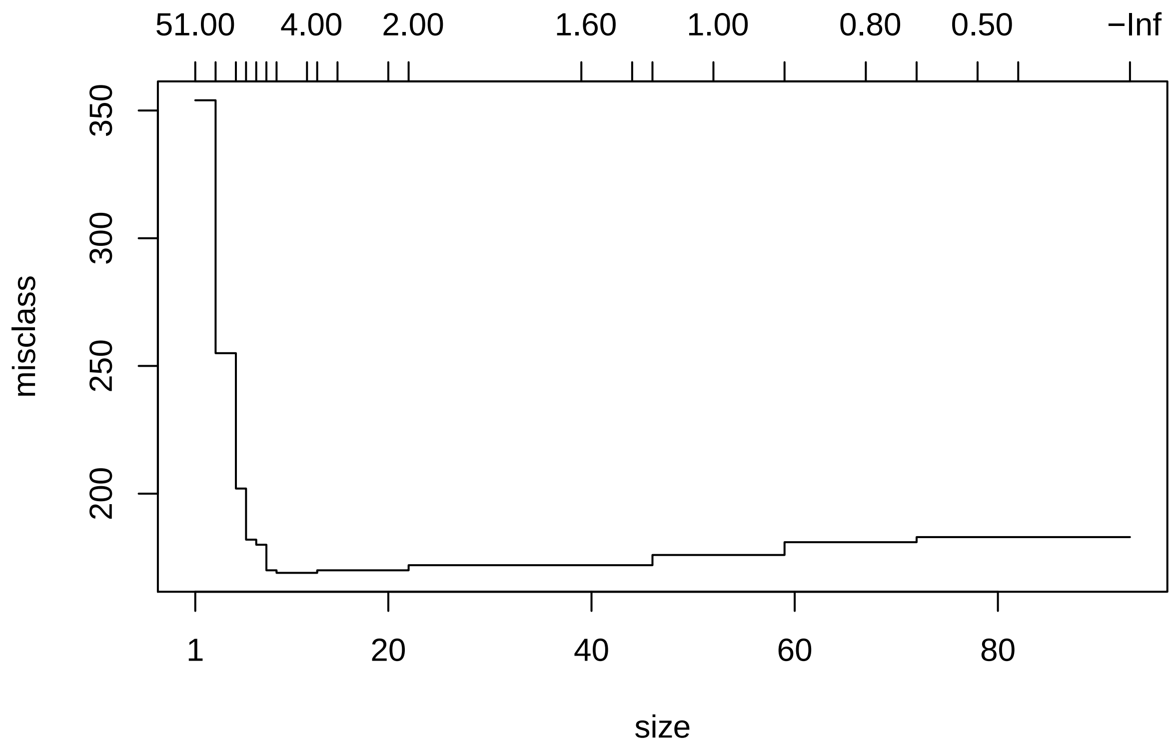


Figure 9: Misclassification error rates as a function of tree size.

```
# Find the smallest tree with the smallest misclassification error
cv_tree_tidy %>%
  filter(misclass <= min(misclass)) %>%
  filter(size == min(size))
```

```
## # A tibble: 1 x 2
##    size misclass
##   <int>    <dbl>
## 1     9      169
```

```
pruned_tree <- prune(main_tree, best = 9)
```

```
draw.tree(pruned_tree, nodeinfo = T, cex = 0.6)
```

```
predYtr <- predict(pruned_tree, type = "class")
predYts <- predict(pruned_tree, type = "class", newdata = tst_cl)

records[1, 1] <- calc_error_rate(predYtr, trn_cl$candidate)
records[1, 2] <- calc_error_rate(predYts, tst_cl$candidate)
```

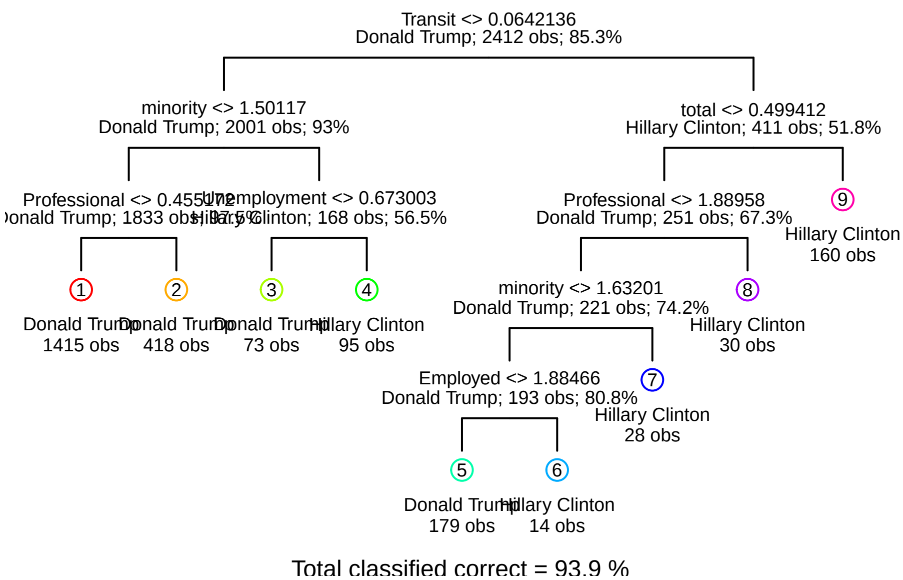


Figure 10: Pruned tree

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.

```
knn_cv <- function(chunkid, folddef, Xdat, Ydat, k){
  # Training
  # identify rows to be used in this fold
  train <- (folddef!=chunkid)
  # Extract predictors for this fold
  Xtr <- Xdat[train, ]
  # Extract outcome variable for this fold
  Ytr <- Ydat[train]

  # Testing
  # Get predictors for testing fold
  Xvl <- Xdat[!train, ]
  # Get outcomes for testing fold
  Yvl <- Ydat[!train]
  ## get classifications for current training chunks
  predYtr <- knn(train = Xtr, test = Xtr, cl = Ytr, k = k)

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

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

```
# Set up tunning parameter options
kvec <- c(1, 5, 8, 10, 15, 20, 25, 30)

# Set up covariates
Xdat <- trn_cl %>%
  select(-candidate) %>%
  as.matrix()
# Outcome variable
Ydat <- trn_cl$candidate

# Plan multiprocess for parallel fitting
plan(multiprocess)
# Fit all
set.seed(1)
out <- expand.grid(k = kvec, chunkid = 1:nfold) %>%
  as_tibble() %>%
  mutate(fit = future_map2(chunkid, k, knn_cv, folddef = folds, Xdat = Xdat, Ydat = Ydat)) %>%
  unnest()
```

```
ggplot(data = out, mapping = aes(x = k, y = val.error)) +
  stat_summary(geom = "ribbon", fun.data = "mean_se") +
  stat_summary(geom = "line", fun.y = "mean") +
  geom_point() +
  labs(x = "k", y = "Testing error")
```

```
best.kfold <- 5
# Matrix of training data covariates
```

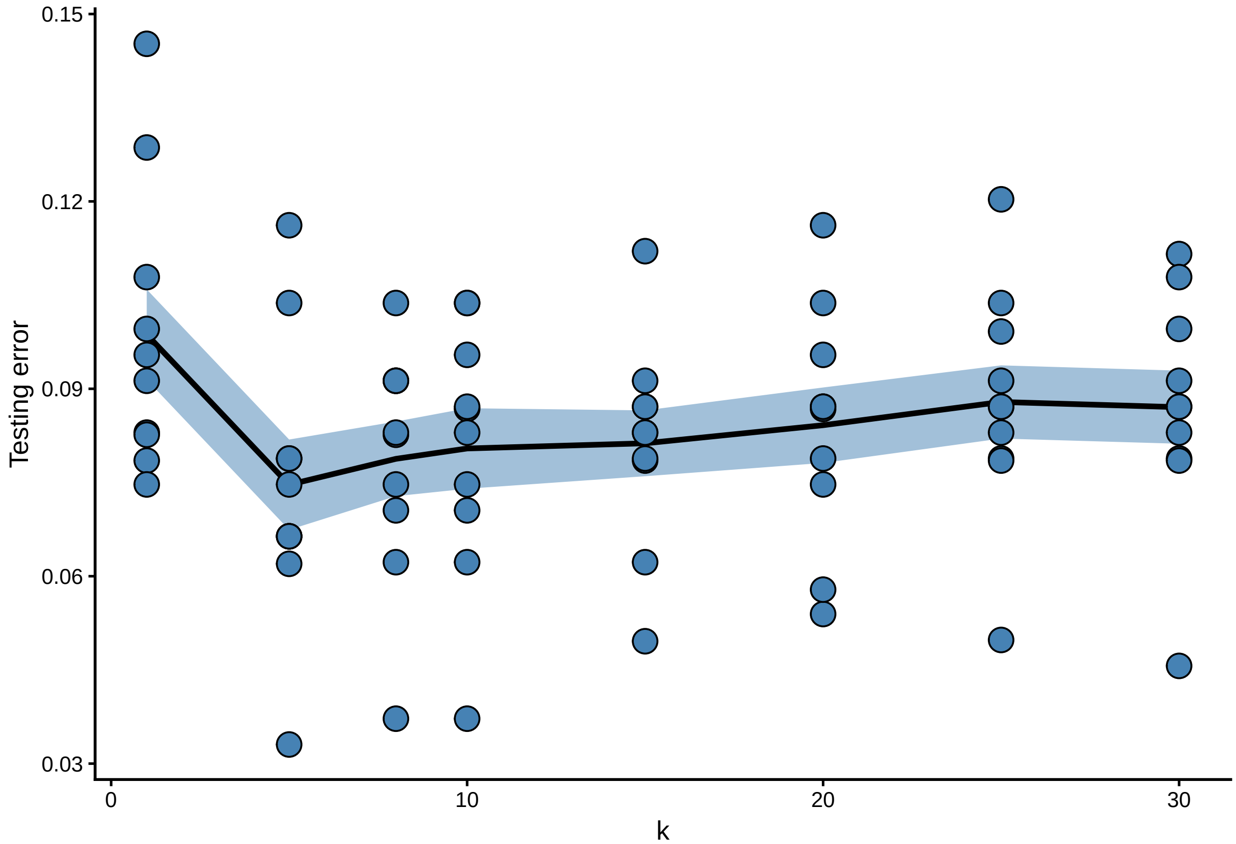


Figure 11: Misclassification errors for KNN CV.

```
Xtest <- tst_cl %>%
  select(-candidate) %>%
  as.matrix()

# Outcome variable
Ytest <- tst_cl$candidate

set.seed(1)
## get classifications for training set
predYtr <- knn(train = Xdat, test = Xdat, cl = Ydat, k = best.kfold)

## get classifications for testing set
predYvl <- knn(train = Xdat, test = Xtest, cl = Ydat, k = best.kfold)

records[2, 1] <- calc_error_rate(predYtr, Ydat)
records[2, 2] <- calc_error_rate(predYvl, Ytest)
```

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

Table 2: Testing and training errors for two different methods.

	train.error	test.error
tree	NA	NA
knn	NA	NA

```r
knitr::kable(pca.records, caption = "Testing and training errors for two different methods.")
```

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.

```
trn_pca <- trn_cl %>%
  select(-candidate) %>%
  as.matrix() %>%
  prcomp(scale = T, center = T)
```

```
pca_var <- tibble(component = 1:length(trn_pca$sdev),
       variance = trn_pca$sdev) %>%
  mutate(relative = variance / sum(variance),
         cumulative = cumsum(relative))

ggplot(data = pca_var ,
       aes(x = component, y = relative)) +
  geom_line(size = 0.5) +
  geom_point() +
  labs(x = "Principal Component", y = "Proportion of Variance") +
  geom_vline(xintercept = 19, linetype = "dashed")
```

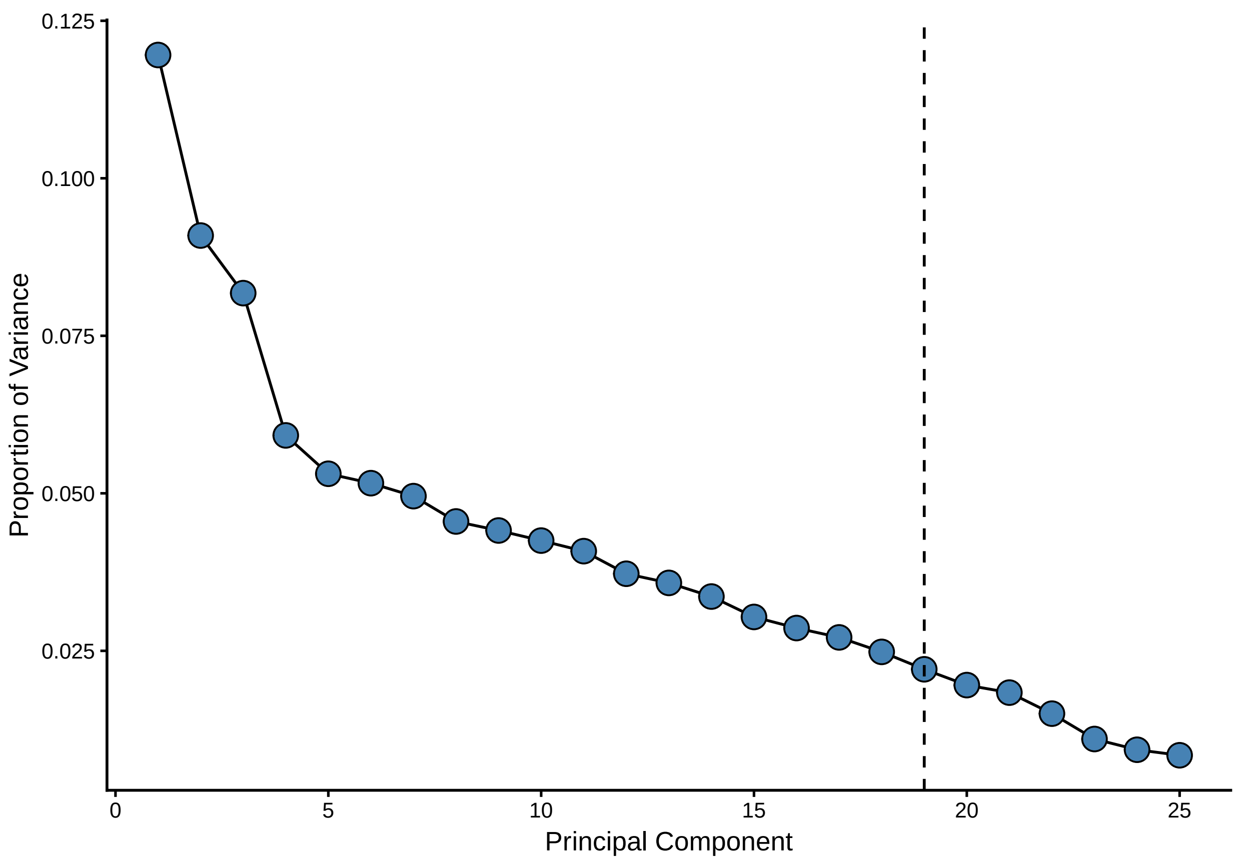


Figure 12: Proportion of variance explained by the 25 components

```
ggplot(data = pca_var ,
       aes(x = component, y = cumulative)) +
```

```r
  geom_point() +
  labs(x = "Principal Component", y = "Cumulative Variance") +
  geom_hline(yintercept = 0.9, linetype = "dashed") +
  geom_vline(xintercept = 19, linetype = "dashed")
```

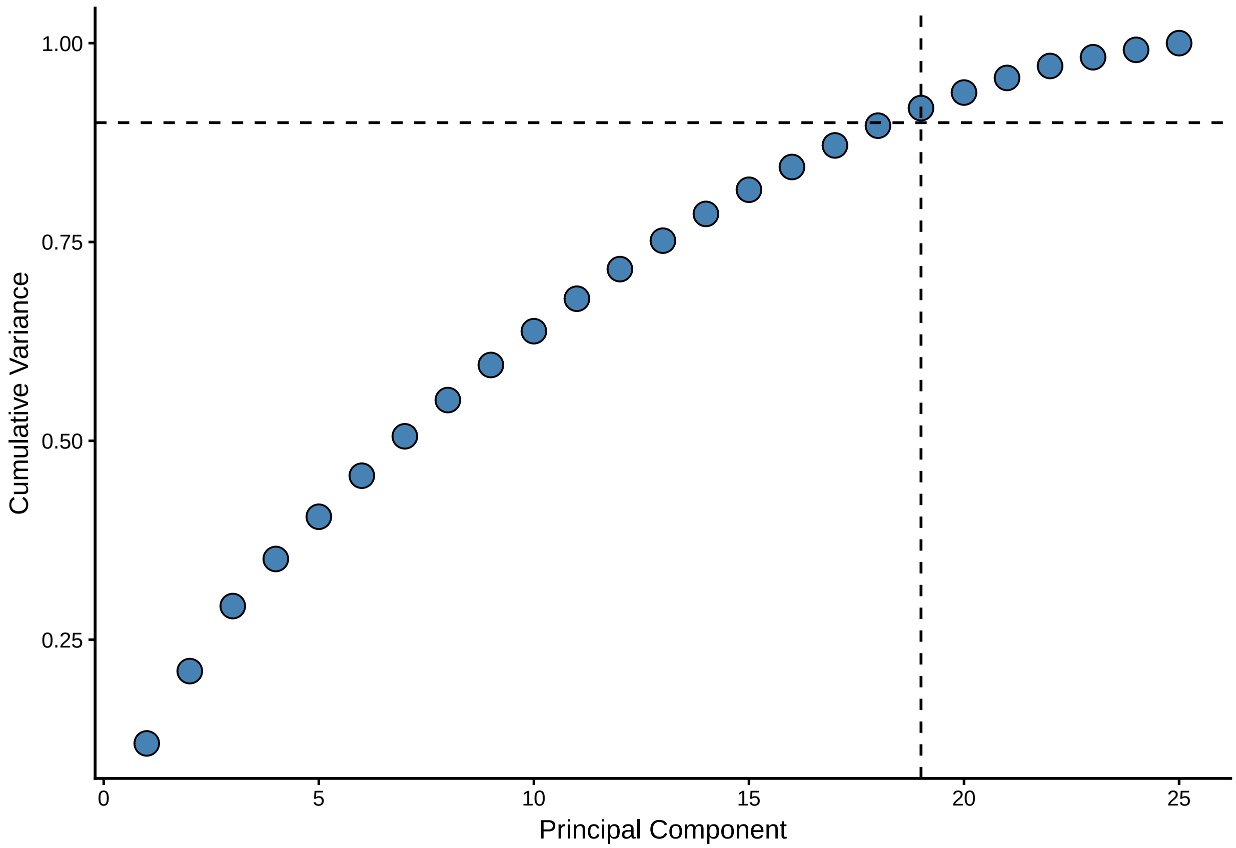


Figure 13: Cumulative proportional variance explained

The following table shows that the first 19 PC are needed to explain 91% of the variance.

```r
pca_var %>%
  filter(cumulative >= 0.89)
```

```
## # A tibble: 8 x 4
##   component variance relative cumulative
##       <int>    <dbl>    <dbl>      <dbl>
## 1        18    0.521  0.0248       0.896
## 2        19    0.463  0.0221       0.918
## 3        20    0.410  0.0196       0.938
## 4        21    0.385  0.0184       0.956
## 5        22    0.315  0.0150       0.971
## 6        23    0.231  0.0110       0.982
## 7        24    0.195  0.00930      0.992
## 8        25    0.177  0.00842      1
```

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 <- as_tibble(trn_pca$x[, 1:19]) %>%
  mutate(candidate = trn_cl$candidate) %>%
  select(candidate, everything())

test_dat <- tst_cl %>%
  select(-candidate) %>%
  as.matrix()

project_loadings <- function(data, pca_obj, n_pc) {
  # Define empty matrix
  projected <- matrix(data = NA, nrow = nrow(data), ncol = n_pc)

  # Iterate across principal components
  for(i in 1:n_pc) {
    projected[, i] <- data %*% pca_obj$rotation[, i]
  }

  colnames(projected) <- paste0("PC", 1:n_pc)

  return(as_tibble(projected))
}

test_pca <- project_loadings(data = test_dat, pca_obj = trn_pca, n_pc = 19) %>%
  mutate(candidate = tst_cl$candidate) %>%
  select(candidate, everything())
```

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

```r
pca_main_tree <- tree(candidate ~ ., data = tr_pca, control = ctrl)
```

```r
draw.tree(pca_main_tree, nodeinfo = T, cex = 0.5)
```

```r
pca_cv_tree <- cv.tree(object = pca_main_tree, rand = folds)
```

```r
plot(pca_cv_tree)
```

```r
# create a tidy version of the diagnostics
pca_cv_tree_tidy <- tibble(size = pca_cv_tree$size,
                        misclass = pca_cv_tree$dev)

# Find the smallest tree with the smallest misclassification error
pca_cv_tree_tidy %>%
  filter(misclass <= min(misclass)) %>%
  filter(size == min(size))
```

```
## # A tibble: 1 x 2
##    size misclass
##   <int>    <dbl>
## 1     7    1475.
```

```r
pca_pruned_tree <- prune(pca_main_tree, best = 7)
```

```r
draw.tree(pca_pruned_tree, nodeinfo = T, cex = 0.6)
```

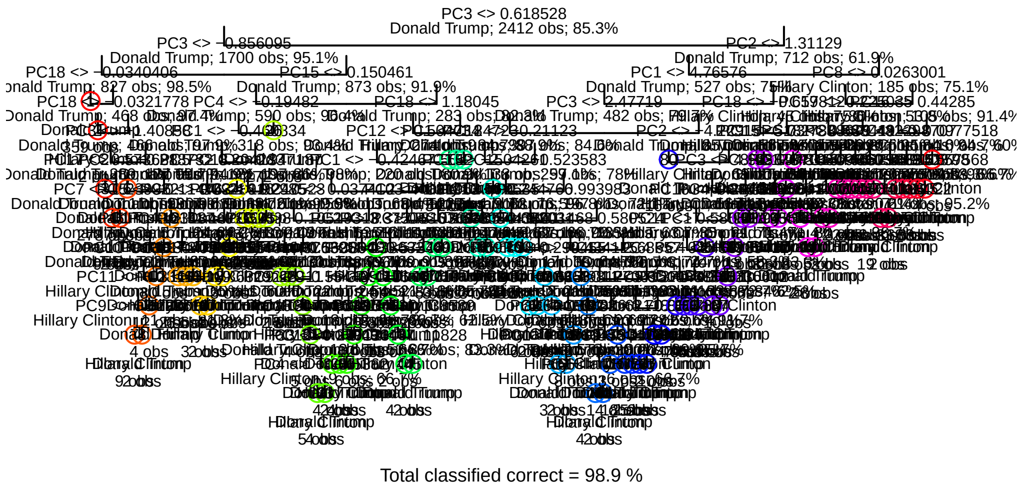


Figure 14: Decission tree with PCA data

```r
predYtr <- predict(pca_pruned_tree, type = "class")
predYts <- predict(pca_pruned_tree, type = "class", newdata = test_pca)

pca.records[1, 1] <- calc_error_rate(predYtr, tr_pca$candidate)
pca.records[1, 2] <- calc_error_rate(predYts, test_pca$candidate)
```

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

```r
# Set up covariates
Xdat <- tr_pca %>%
  select(-candidate) %>%
  as.matrix()
# Outcome variable
Ydat <- tr_pca$candidate

# Plan multiprocess for parallel fitting
plan(multiprocess)
# Fit all
set.seed(1)
out <- expand.grid(k = kvec, chunkid = 1:nfold) %>%
  as_tibble() %>%
```

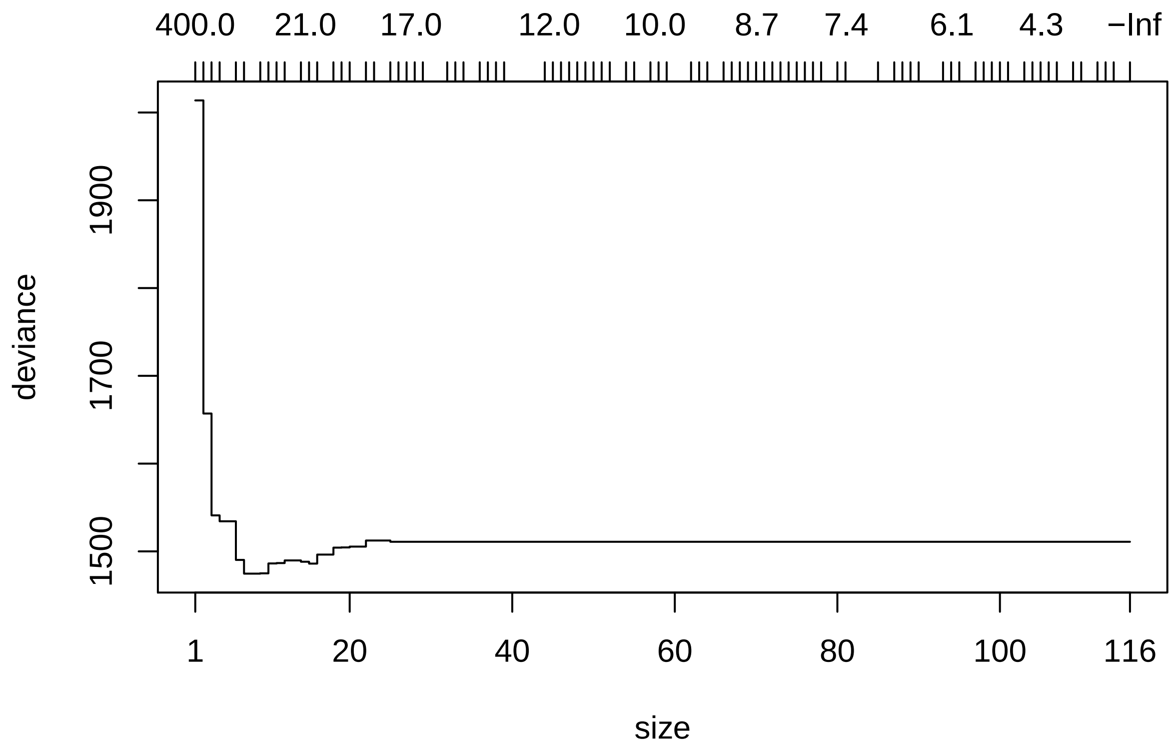


Figure 15: Misclassification error rates as a function of tree size for PCA tree.

```
  mutate(fit = future_map2(chunkid, k, knn_cv, folddef = folds, Xdat = Xdat, Ydat = Ydat)) %>%
  unnest()
```

```
ggplot(data = out, mapping = aes(x = k, y = val.error)) +
  stat_summary(geom = "ribbon", fun.data = "mean_se") +
  stat_summary(geom = "line", fun.y = "mean") +
  geom_point() +
  labs(x = "k", y = "Testing error")
```

```
best.kfold <- 5
# Matrix of training data covariates
Xtest <- test_pca %>%
  select(-candidate) %>%
  as.matrix()

# Outcome variable
Ytest <- test_pca$candidate

set.seed(1)
## get classifications for training set
predYtr <- knn(train = Xdat, test = Xdat, cl = Ydat, k = best.kfold)
```

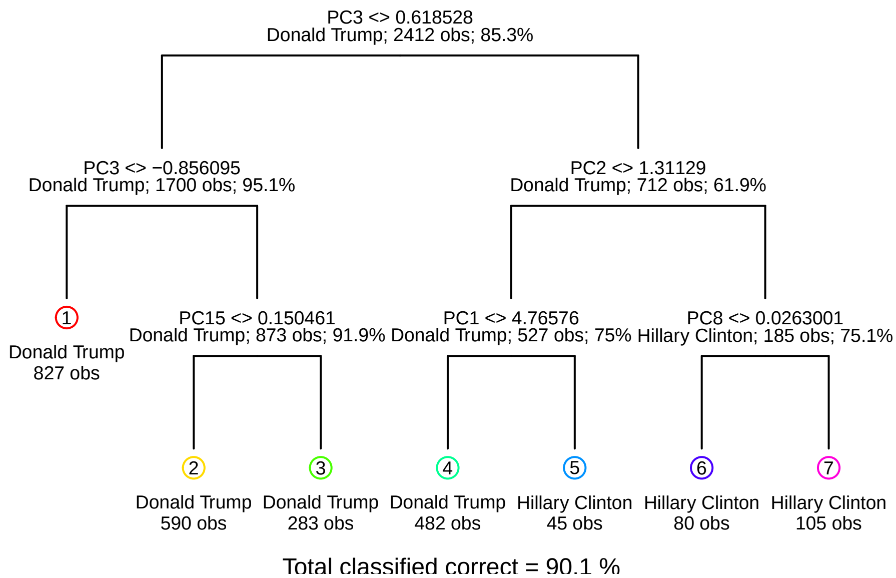


Figure 16: Pruned decision tree using PCA data.

```
## get classifications for testing set
predYvl <- knn(train = Xdat, test = Xtest, cl = Ydat, k = best.kfold)

pca.records[2, 1] <- calc_error_rate(predYtr, Ydat)
pca.records[2, 2] <- calc_error_rate(predYvl, Ytest)
```

```
knitr::kable(pca.records, caption = "Table showing training and testing errors for both methods when PC
```

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 understand

20. Propose and tackle at least one interesting question

I will visualize the predictions of 3 models: decision tree, KNN, binary logistic regression. I will produce county-level maps of the predictions, and will use the PCA training dataset, and predict across the entire data.

```
pca_logistic <- glm(candidate ~ ., data = tr_pca, family = binomial)
```

```
train_reconstruct <- cbind(election_cl_attr[in.trn, ], tr_pca)
test_reconstruct <- cbind(election_cl_attr[-in.trn, ], test_pca)
```

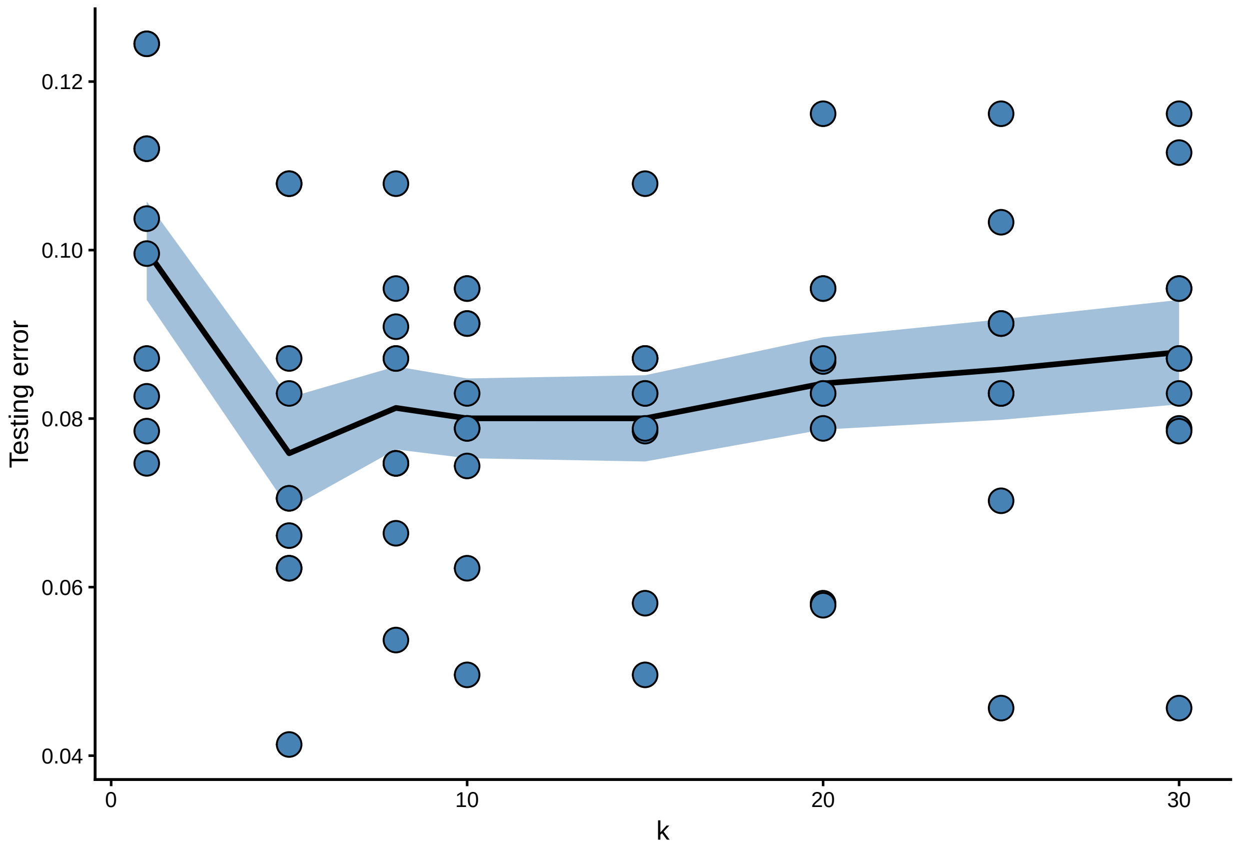


Figure 17: Misclassification errors for KNN CV using PCA covariates.

Table 3: Table showing training and testing errors for both methods when PCs are used to train the models.

	train.error	test.error
tree	0.0986733	0.1158940
knn	0.0514096	0.0778146

```
recons <- rbind(train_reconstruct, test_reconstruct) %>%
  mutate(tree_pred = predict(pca_pruned_tree, newdata = ., type = "class"),
         logistic_pred = predict(pca_logistic, newdata = ., type = "response"),
         logistic_pred = ifelse(logistic_pred >= 0.5, "Hillary Clinton", "Donald Trump"))

test_knn <- recons %>%
  select(contains("PC")) %>%
  select(-pct)

knn_pred <- knn(train = Xdat, test = test_knn , cl = Ydat, k = best.kfold)

recons <- recons %>%
  mutate(knn_pred = knn_pred) %>%
  select(state, county, truth = candidate, tree_pred, knn_pred, logistic_pred) %>%
  mutate_at(vars(truth, tree_pred, knn_pred, logistic_pred), as.character)
```

```
predicted_winners <- counties %>%
  mutate(subregion = str_remove(subregion, pattern = "county | columbia | city | parish")) %>%
  left_join(recons, by = c("region" = "state", "subregion" = "county"))
```

```
ggplot(data = predicted_winners,
                        mapping = aes(fill = tree_pred == truth)) +
  geom_sf(color = "white") +
  ggtheme_map() +
  scale_fill_brewer(palette = "Set1") +
  guides(fill = guide_legend(title = "Prediction correct?"))
```

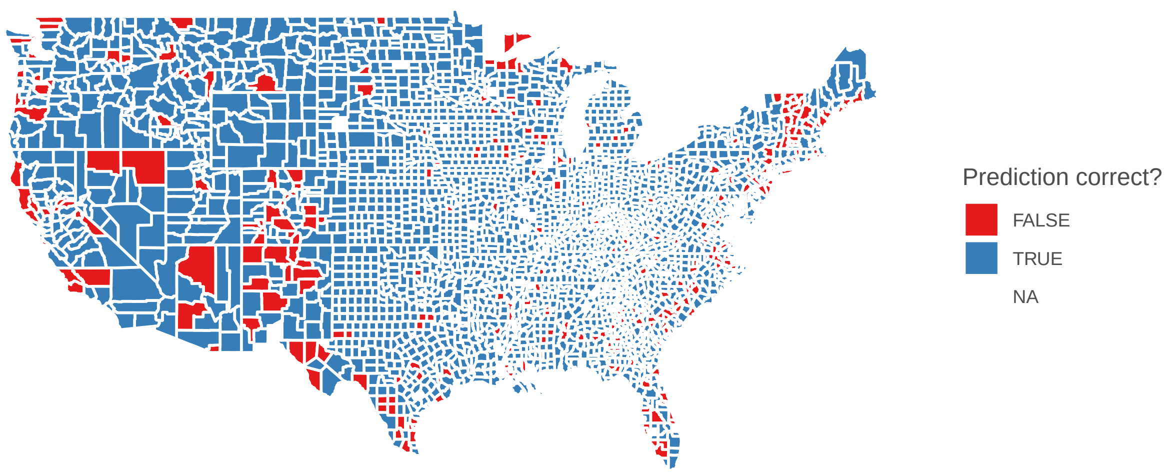


Figure 18: Map of predictions by decision tree

```r
ggplot(data = predicted_winners,
                       mapping = aes(fill = knn_pred == truth)) +
  geom_sf(color = "white") +
  ggtheme_map() +
  scale_fill_brewer(palette = "Set1") +
  guides(fill = guide_legend(title = "Prediction correct?"))
```

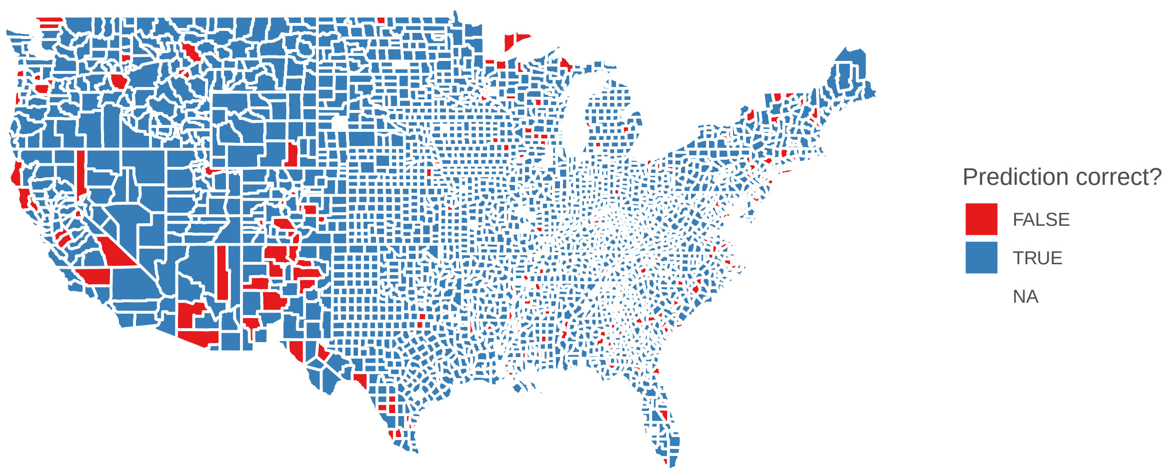


Figure 19: Map of predictions by KNN

```r
ggplot(data = predicted_winners,
                           mapping = aes(fill = logistic_pred == truth)) +
  geom_sf(color = "white") +
  ggtheme_map() +
  scale_fill_brewer(palette = "Set1") +
  guides(fill = guide_legend(title = "Prediction correct?"))
```

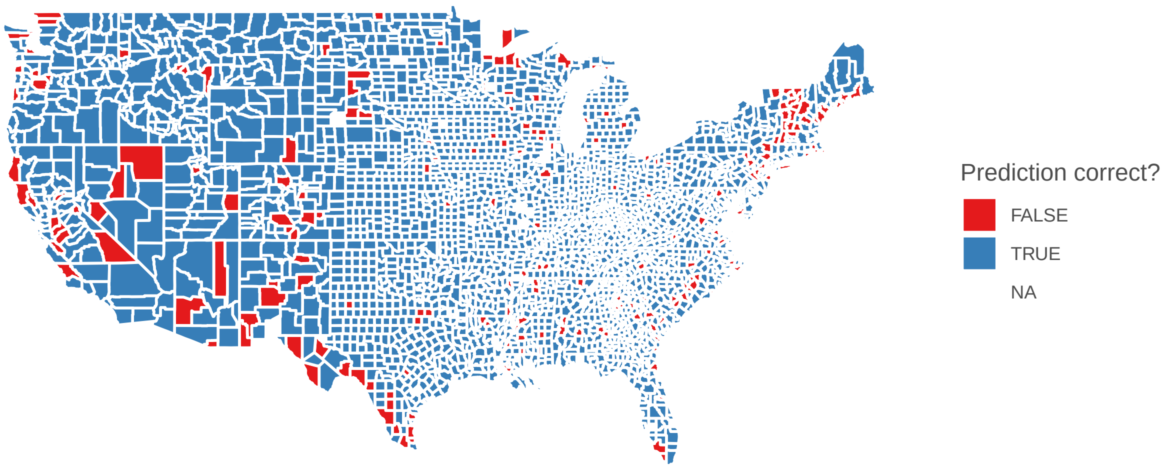


Figure 20: Map of predictions by binary logistic regression