2016 Election Prediction

Brandon Chester, Devin Thai, and Omar Jones (PSTAT 131/231)

3/12/2018

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? Predictions about voter behavior are based on data collected at a particular time. One problem with this method of analysis is that the presidential campaign lasts for many months, and the general purpose of the campaign is to persuade voters to choose a particular candidate. Thus a voter's choice for a candidate can be influenced by a variety of factors at any given time.
2. What was unique to Nate Silver's approach in 2012 that allowed him to achieve good predictions? Nate Silver's approach in 2012 was apparently standard for the most part. Normally, analysts only consider the proportion with the maximum likelihood. However, Nate Silver differentiates himself by considering the full range of possible support proportion for Obama. He combines Bayes' theorem and graph theory in order to weight each forward progression of his hierarchical model. With the abundance of available poll data, Silver was able to use his model confidently predict Obama's probability of winning the election.
3. What went wrong in 2016? What do you think should be done to make future predictions better? There can be many issues with the 2016 polls that could have led to such an incorrect result. Most polls overstated Hillary Clinton's advantage on Trump. Especially in areas of the country like the Midwest, the Democrats had a hard time getting its followers to show up to the polls voting day. Most polls underestimated the turnout for certain Trump voting groups, leading to Trump's advantage come election day. Future predictions can be more accurate if increased studies are done in all of the swing states. Taking into account things like voter motivation and census studies could help poll makers determine the winner with less error. The less error in their studies, the less likely a 2016 election happens again.

Data

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

Election data

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

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	C
1001020100	Alabama	Autauga	1948	940	1008	0.9	87.4	7.7	0.3	0.6	0.0	
1001020200	Alabama	Autauga	2156	1059	1097	0.8	40.4	53.3	0.0	2.3	0.0	
1001020300	Alabama	Autauga	2968	1364	1604	0.0	74.5	18.6	0.5	1.4	0.3	
1001020400	Alabama	Autauga	4423	2172	2251	10.5	82.8	3.7	1.6	0.0	0.0	
1001020500	Alabama	Autauga	10763	4922	5841	0.7	68.5	24.8	0.0	3.8	0.0	
1001020600	Alabama	Autauga	3851	1787	2064	13.1	72.9	11.9	0.0	0.0	0.0	

Census data: column metadata

Column information is given in `metadata`.

CensusTract	Census.tract.ID	numeric
State	State, DC, or Puerto Rico	string
County	County or county equivalent	string
TotalPop	Total population	numeric
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

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

Data wrangling

4. Remove summary rows from `election.raw` data: i.e.,
    - Federal-level summary into a `election_federal`.

```
election_federal = election.raw %>% filter(election.raw$fips=="US")
#election_federal = election_federal %>% select( fips, candidate, state, votes)
```

```
* State-level summary into a `election_state`.
```

```
election_state = election.raw %>% filter(fips != "US", fips!= 2000, fips != 46102, is.na(county))
#election_state = election_state %>% select( fips, candidate, state, votes)
```

```
* Only county-level data is to be in `election`.
```

```
election = election.raw %>% filter(election.raw$county!="NA")
```

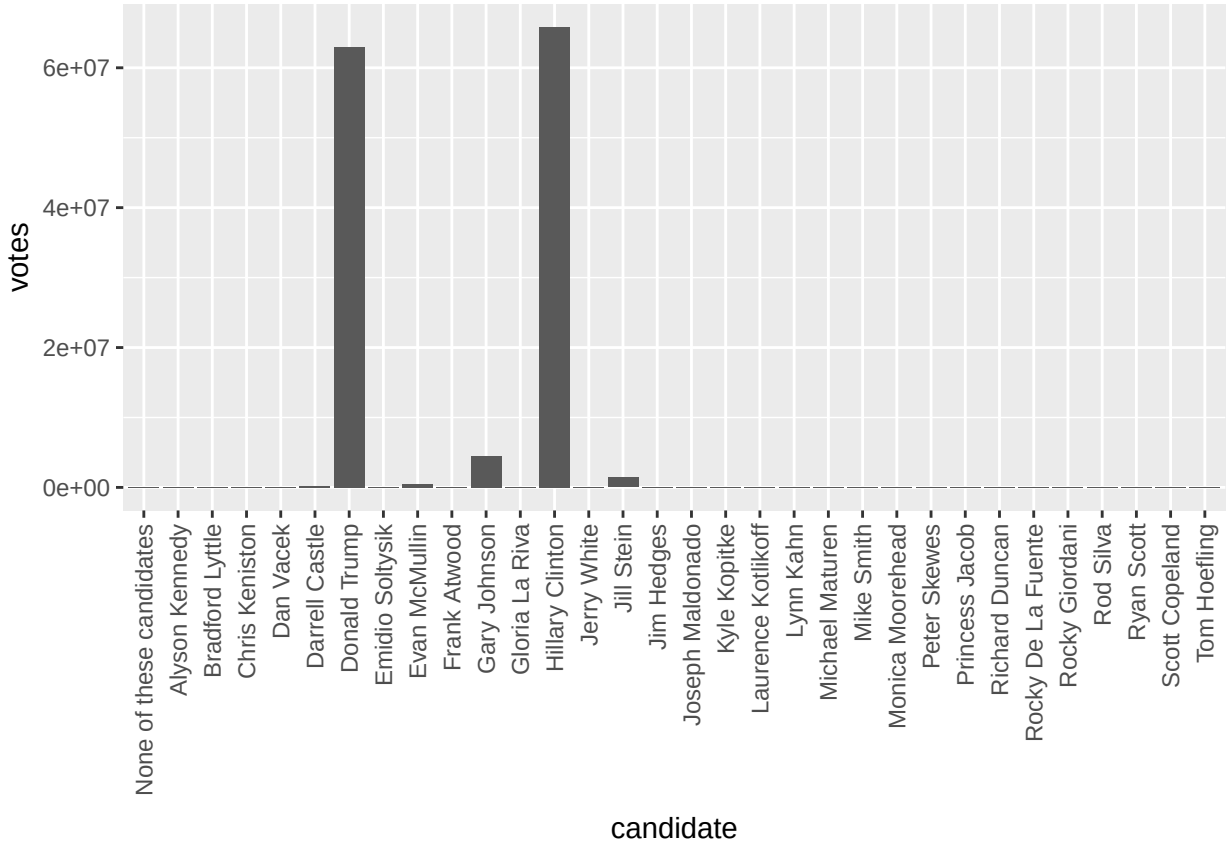
5. How many named presidential candidates were there in the 2016 election? Draw a bar chart of all votes received by each candidate

```
unique_cand = election_federal %>% select(candidate)
# unique_cand[-c(9),] if you want to get rid of the "none of the others"
unique(unique_cand)
```

```
## # A tibble: 32 x 1
##                    candidate
##                       <fctr>
```

```
##  1              Donald Trump
##  2           Hillary Clinton
##  3              Gary Johnson
##  4                Jill Stein
##  5              Evan McMullin
##  6             Darrell Castle
##  7             Gloria La Riva
##  8         Rocky De La Fuente
##  9  None of these candidates
## 10             Richard Duncan
## # ... with 22 more rows
```

```
ggplot(election_federal, aes(candidate, votes)) + geom_bar(stat = "identity") + theme(axis.text.x=eleme
```



From our tibble seen above, we can see that there were 31 unique candidates found in our data. We also have a category "none of the others" implying that there are more than just those candidates voted for.

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).

```
total = aggregate(election$votes, by=list(fips=election$fips), FUN=sum)
test1 = merge(election, total, by.x = 'fips', by.y = 'fips')
test2 = test1 %>% mutate(pct = votes/x)
county_winner = test2 %>% group_by(fips) %>% top_n(1)
```

```
## Selecting by pct
```

```
total2 = aggregate(election_state$votes, by=list(fips=election_state$fips), FUN=sum)
test3 = merge(election_state, total2, by.x = 'fips', by.y = 'fips')
test4 = test3 %>% mutate(pct = votes/x)
state_winner = test4 %>% group_by(fips) %>% top_n(1)
```

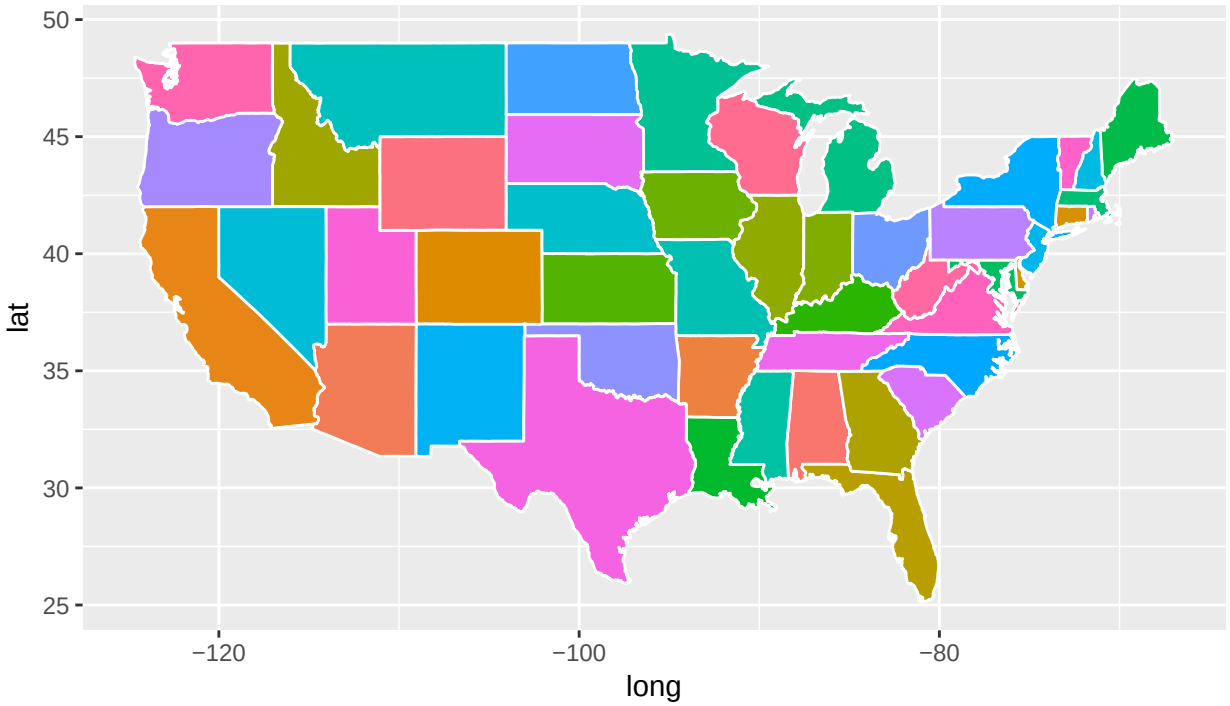
```
## Selecting by pct
```

Visualization

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

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

```
states = map_data("state")

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

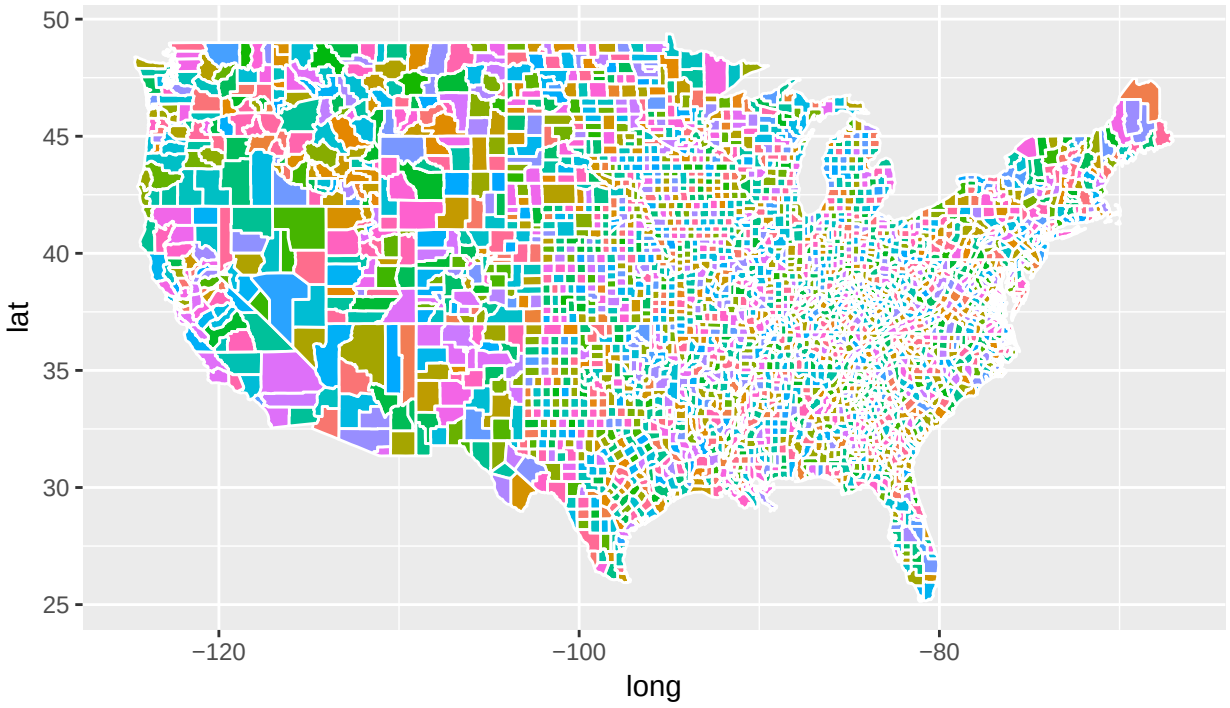


The variable `states` contain information to draw white polygons, and fill-colors are determined by `region`.

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

```
counties = map_data("county")

ggplot(data = counties) +
```

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

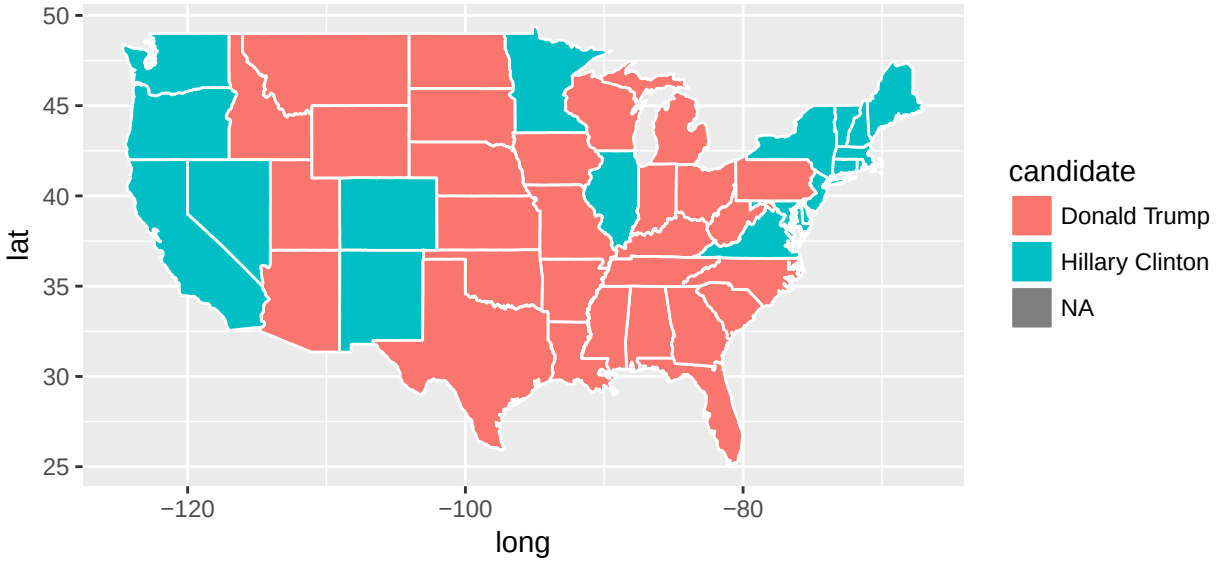


8. Now color the map by the winning candidate for each state. First, combine `states` variable and `state_winner` we created earlier using `left_join()`. Note that `left_join()` needs to match up values of states to join the tables; however, they are in different formats: e.g. `AZ` vs. `arizona`. Before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

```
states = states %>% mutate(fips = state.abb[match(region, sapply(state.name,tolower))])
states1 = left_join(states, state_winner, by = "fips")
```

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

```
ggplot(data = states1) +
  geom_polygon(aes(x = long, y = lat, color = candidate, fill = candidate, group = group), color = "whi
```

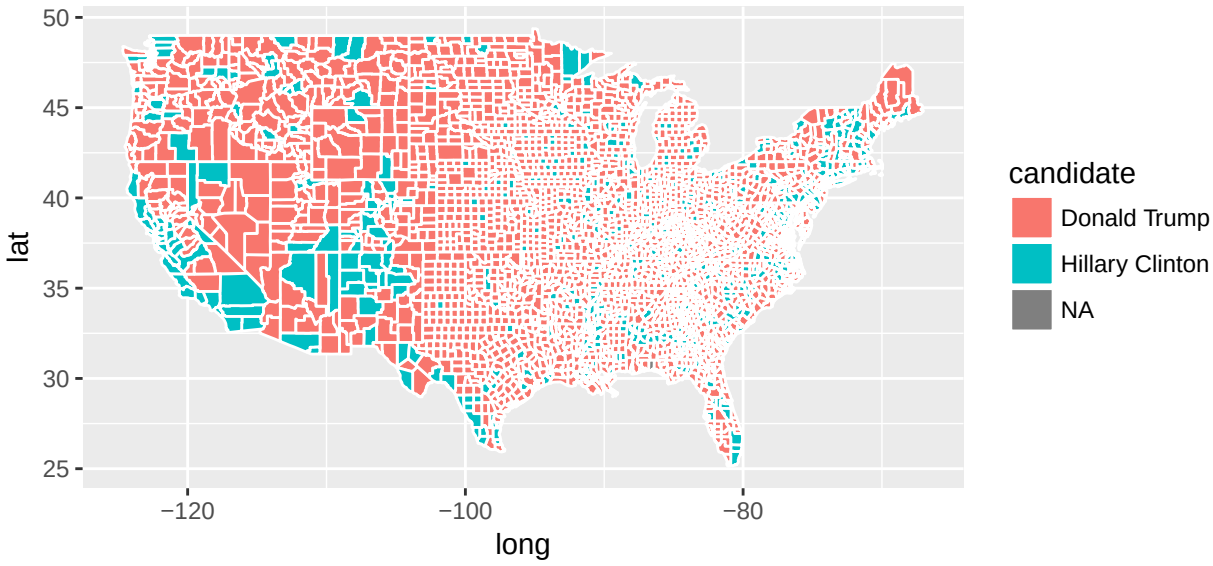


9. The variable `county` does not have `fips` column. So we will create one by pooling information from `maps::county.fips`. Split the `polyname` column to `region` and `subregion`. Use `left_join()` combine `county.fips` into `county`. Also, `left_join()` previously created variable `county_winner`.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

```
county.fips = maps::county.fips
county.fips = county.fips %>% mutate( region = sapply(strsplit(polyname, ","), "[", 1))
county.fips = county.fips %>% mutate( subregion = sapply(strsplit(polyname, ","), "[", 2))
county.fips = county.fips %>% select( fips, region, subregion)

counties = counties %>% mutate(fips = county.fips$fips[match(subregion,county.fips$subregion)])

counties$fips <- factor(counties$fips)

counties1 = left_join(counties, county_winner, by = c("fips" = "fips"))
```

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

```
ggplot(data = counties1) +
  geom_polygon(aes(x = long, y = lat, color = candidate, fill = candidate, group = group), color = "whi
```

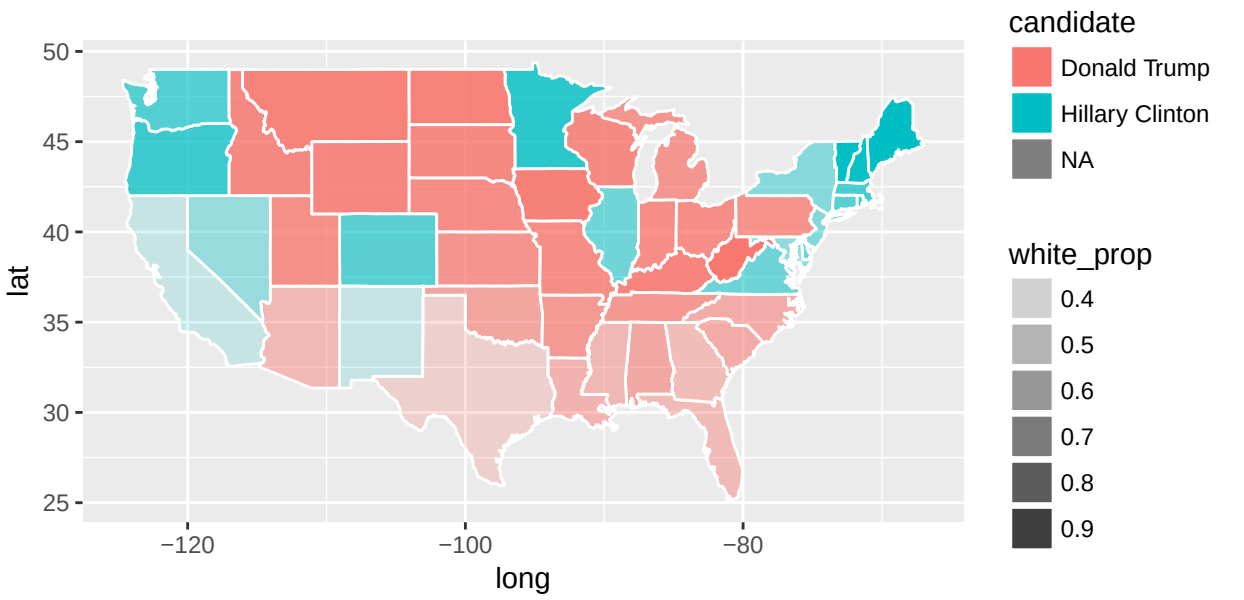


10. Create a visualization of your choice using `census` data. Many exit polls noted that demographics played a big role in the election. Use this Washington Post article and this R graph gallery for ideas and inspiration.

```
census.del = census %>% filter(complete.cases(census))
res = census.del %>% mutate_at(vars(White,Black),funs(pop = . * 0.01 * TotalPop))
res = res %>% group_by(State) %>% summarise_at(vars(TotalPop, White_pop), sum) %>% mutate(white_prop = W
res = res %>% mutate(fips = state.abb[match(State, state.name)]) %>% filter(!(State == "Puerto Rico"))
res[4] = rapply(res[4][1],function(x) ifelse(is.na(x),"DC",x), how = "replace")

res1 = left_join(res,state_winner, by = "fips")
```

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

```
states2 = plyr::join(states, res1, by = "fips")

ggplot(data = states2) +
  geom_polygon(aes(x = long, y = lat, color = candidate, fill = candidate, group = group, alpha = white_
```



We can see that in more than half of Donald Trump's winning states, the proportion of white votors compose at least half of the population. In comparison, of the states won by Hillary Clinton, it seems less than half of the states won by Clinton have the same proportions. It does seem like the proportion of white population does have a higher liklihood to be a red state. However, this does not seem to work in the southern states, in which minority proportions are higher and mostly went for Trump, excluding California and New Mexico.

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.del %>% mutate_at(vars(Men, Employed, Citizen), funs(./TotalPop))
census.del = select(census.del, -c(Walk, PublicWork, Construction))
census.del = census.del %>% mutate(Minority = Hispanic + Black + Native + Asian + Pacific)
```

    - *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(TotalPop)
census.subct = census.subct %>% mutate( weight = TotalPop / n)
census.subct = census.subct %>% select(-c(CensusTract, TotalPop, Men, Women, Hispanic, White, Black
```

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

weighted sum

```
census.ct = census.subct %>% group_by(State,County) %>% summarize_all(funs(weighted.mean(.,weight))
```

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

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

```
## # A tibble: 6 x 27
## # Groups:   State [1]
##     State  County   Citizen   Income IncomeErr IncomePerCap
##    <fctr>  <fctr>     <dbl>    <dbl>     <dbl>        <dbl>
## 1 Alabama Autauga 0.7374912 51696.29  7771.009     24974.50
## 2 Alabama Baldwin 0.7569406 51074.36  8745.050     27316.84
## 3 Alabama Barbour 0.7691222 32959.30  6031.065     16824.22
## 4 Alabama    Bibb 0.7739781 38886.63  5662.358     18430.99
## 5 Alabama  Blount 0.7337550 46237.97  8695.786     20532.27
## 6 Alabama Bullock 0.7545420 33292.69  9000.345     17579.57
## # ... with 21 more variables: IncomePerCapErr <dbl>, Poverty <dbl>,
## #   ChildPoverty <dbl>, Professional <dbl>, Service <dbl>, Office <dbl>,
## #   Production <dbl>, Drive <dbl>, Carpool <dbl>, Transit <dbl>,
## #   OtherTransp <dbl>, WorkAtHome <dbl>, MeanCommute <dbl>,
## #   Employed <dbl>, PrivateWork <dbl>, SelfEmployed <dbl>,
## #   FamilyWork <dbl>, Unemployment <dbl>, Minority <dbl>, n <dbl>,
## #   weight <dbl>
```

Dimensionality reduction

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

```
ct.pc = prcomp(census.ct[3:25], center = TRUE, scale = TRUE)
subct.pc = prcomp(census.subct[3:25], center = TRUE, scale = TRUE)

ct.pc2 = as.data.frame(ct.pc$x[,1:2])
subct.pc2 = as.data.frame(subct.pc$x[,1:2])
```

The most promiment loading would be the column PC1 found above, then followed by PC2, PC3, etc.

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?

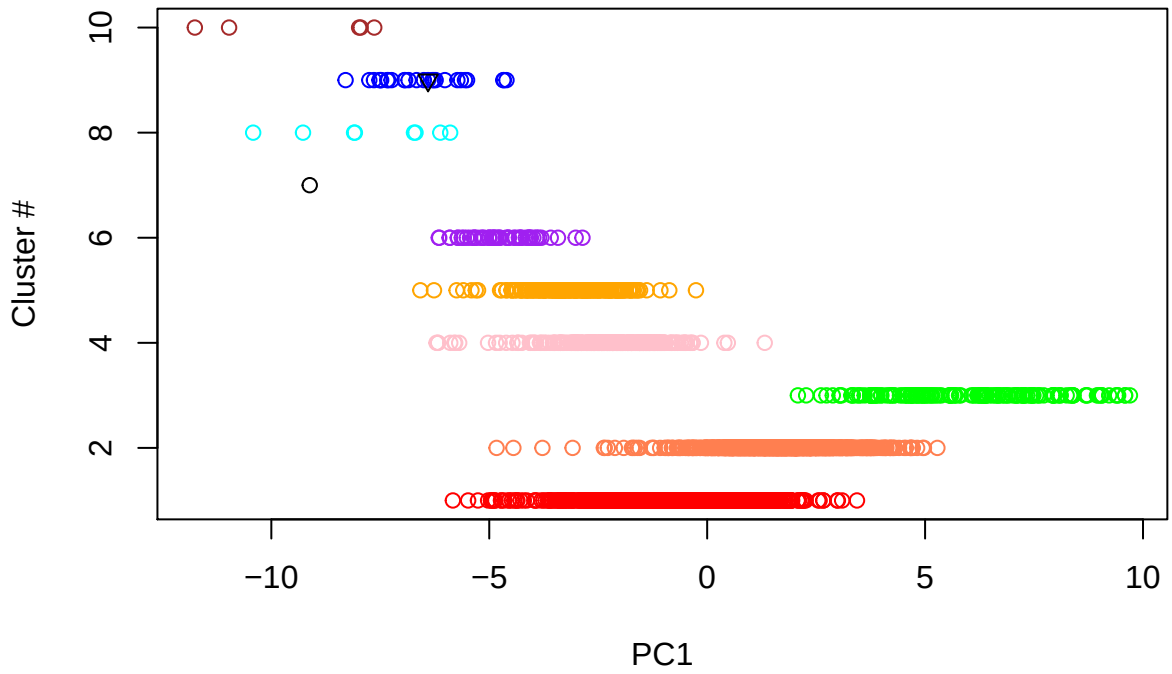
```
set.seed(10)

dist.ct = dist(census.ct[3:25], method = "euclidean")
hc.ct = hclust(dist.ct, method = "complete")
clust.ct = cutree(hc.ct, 10)

ct.pc5 = as.data.frame(ct.pc$x[,1:5])
dist.ct.pc5 = dist(ct.pc5, method = "euclidean")
```

```
hc.ct5 = hclust(dist.ct.pc5, method = "complete")
clust.ct5 = cutree(hc.ct5,10)
```

```
clust.sanmateo = clust.ct[227]
clust.sanmateo
```

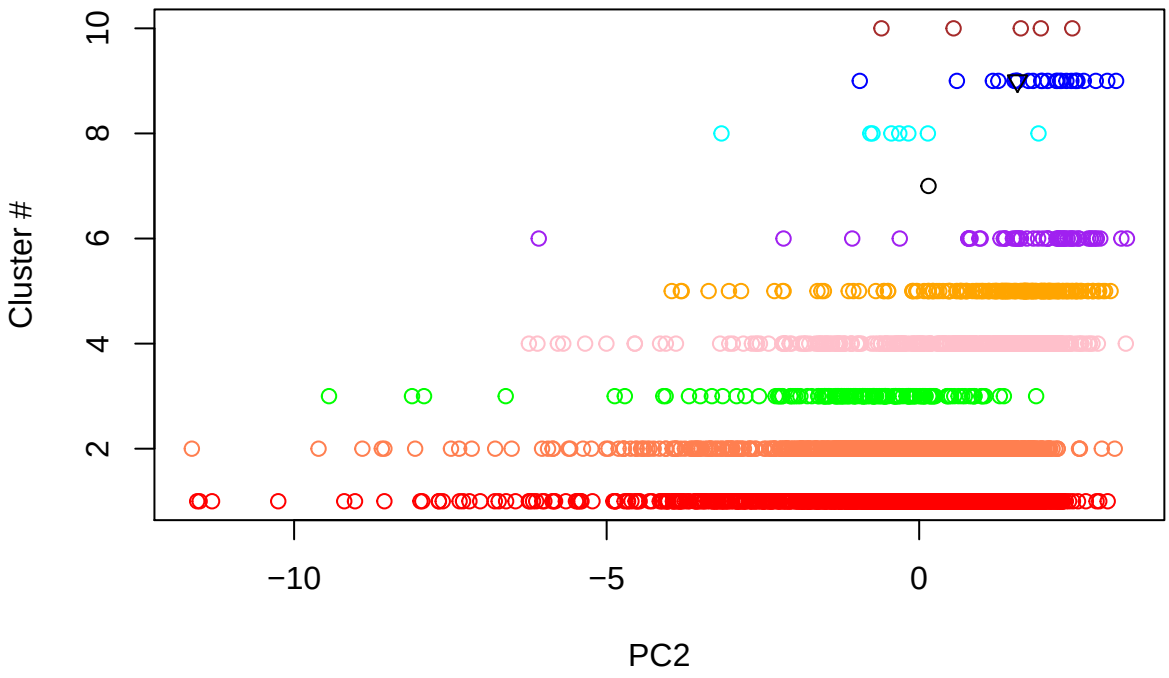
```
## [1] 9
```

```
plot(ct.pc2[,1], clust.ct, xlab = "PC1", ylab = "Cluster #",
     col = c( "red", "coral", "green", "pink", "orange", "purple", "black", "cyan", "blue", "brown")[clu
points(ct.pc2[,1][227],clust.ct[227], pch = 25)
```

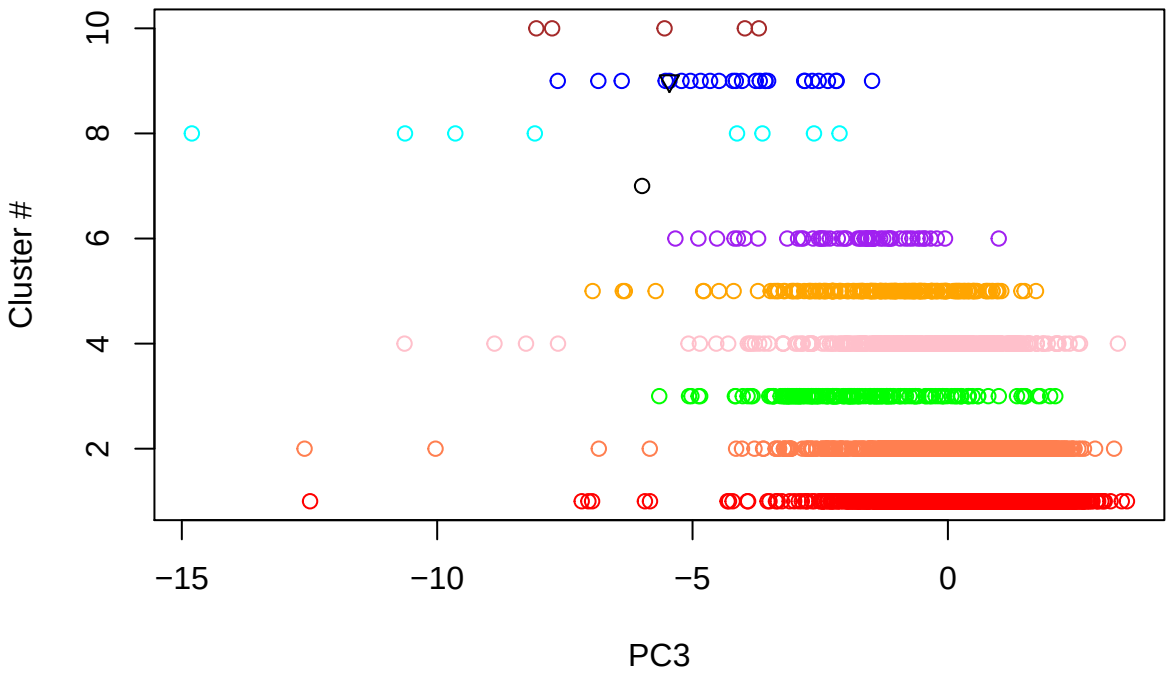


If we look at graph of PC1 and sort them according to cluster, we see that all values in Cluster # 9(San Mateo's Cluster, the blue points on the graph) have very similar PC1 values.

```
plot(ct.pc2[,2], clust.ct, xlab = "PC2", ylab = "Cluster #",
     col = c( "red", "coral", "green", "pink", "orange", "purple", "black", "cyan", "blue", "brown")[clu
points(ct.pc2[,2][227],clust.ct[227], pch = 25)
```




```
plot(ct.pc5[,3], clust.ct, xlab = "PC3", ylab = "Cluster #",
     col = c( "red", "coral", "green", "pink", "orange", "purple", "black", "cyan", "blue", "brown")[clu
points(ct.pc5[,3][227],clust.ct[227], pch = 25)
```



The same can be said when comparing San Mateo County to other cluster 9 observations in the graph for PC2 and PC3. Since most variance is observed in the largest PC values. It is safe to say that San Mateo County is well fitted to the cluster the observation is in.

Classification

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

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

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

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

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

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

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

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

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

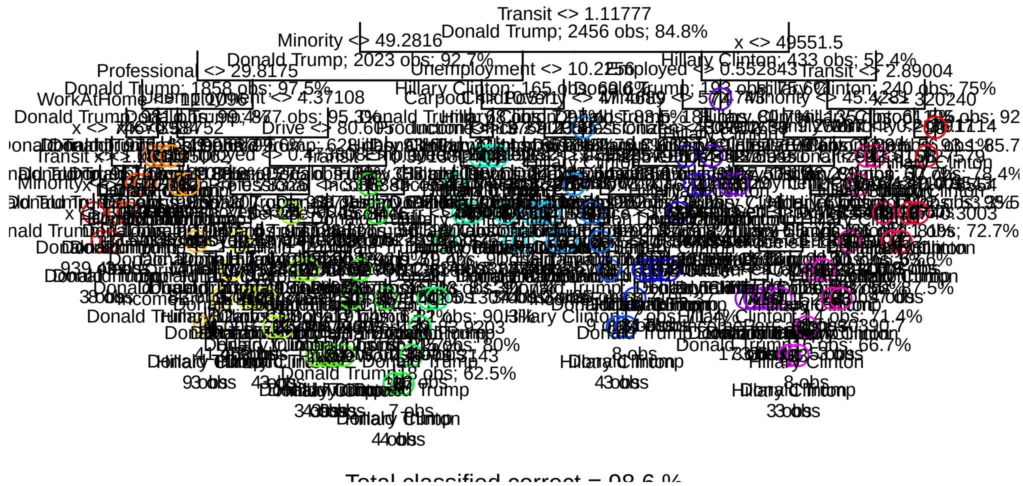
Using the following error rate function:

```
calc_error_rate = function(predicted.value, true.value){
  return(mean(true.value!=predicted.value))
}
records = matrix(NA, nrow=3, ncol=2)
colnames(records) = c("train.error","test.error")
rownames(records) = c("tree","knn","glm")
```

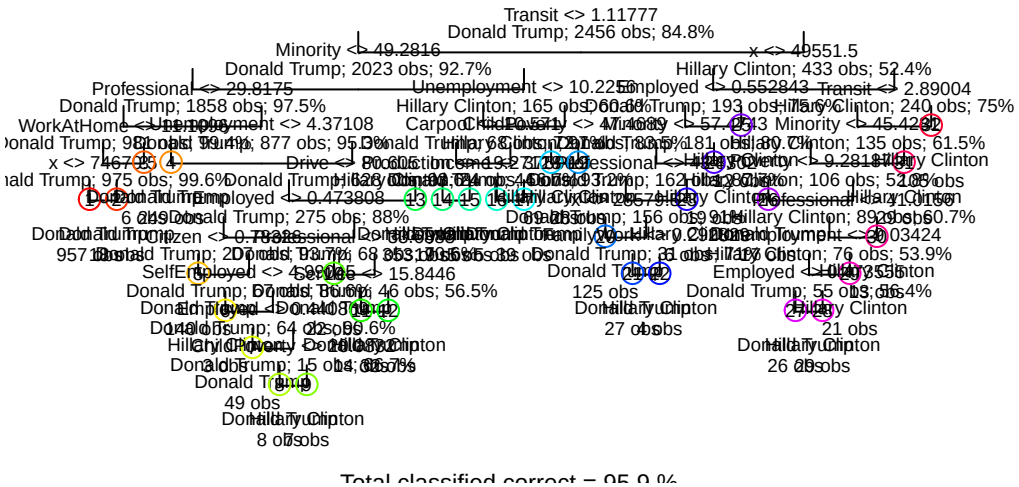
Classification: native attributes

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.

```
election.tree = tree(candidate~., data = as.data.frame(trn.cl), control = tree.control(nobs = nrow(trn.
draw.tree(election.tree, cex = 0.6, nodeinfo=TRUE)
```




```
election.cv = cv.tree(election.tree, rand = folds, FUN = prune.misclass, K = nfold)
nleaves = min(election.cv$size[which(election.cv$dev == min(election.cv$dev))])
election.pruned = prune.tree(election.tree, best = nleaves)
draw.tree(election.pruned,cex = 0.6, nodeinfo = TRUE)
```




```
trn.pred = predict(election.pruned, trn.cl, type = 'class')
tst.pred = predict(election.pruned, tst.cl, type = 'class')

error.trn <- calc_error_rate(trn.pred, trn.cl$candidate)
error.tst <- calc_error_rate(tst.pred, tst.cl$candidate)

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

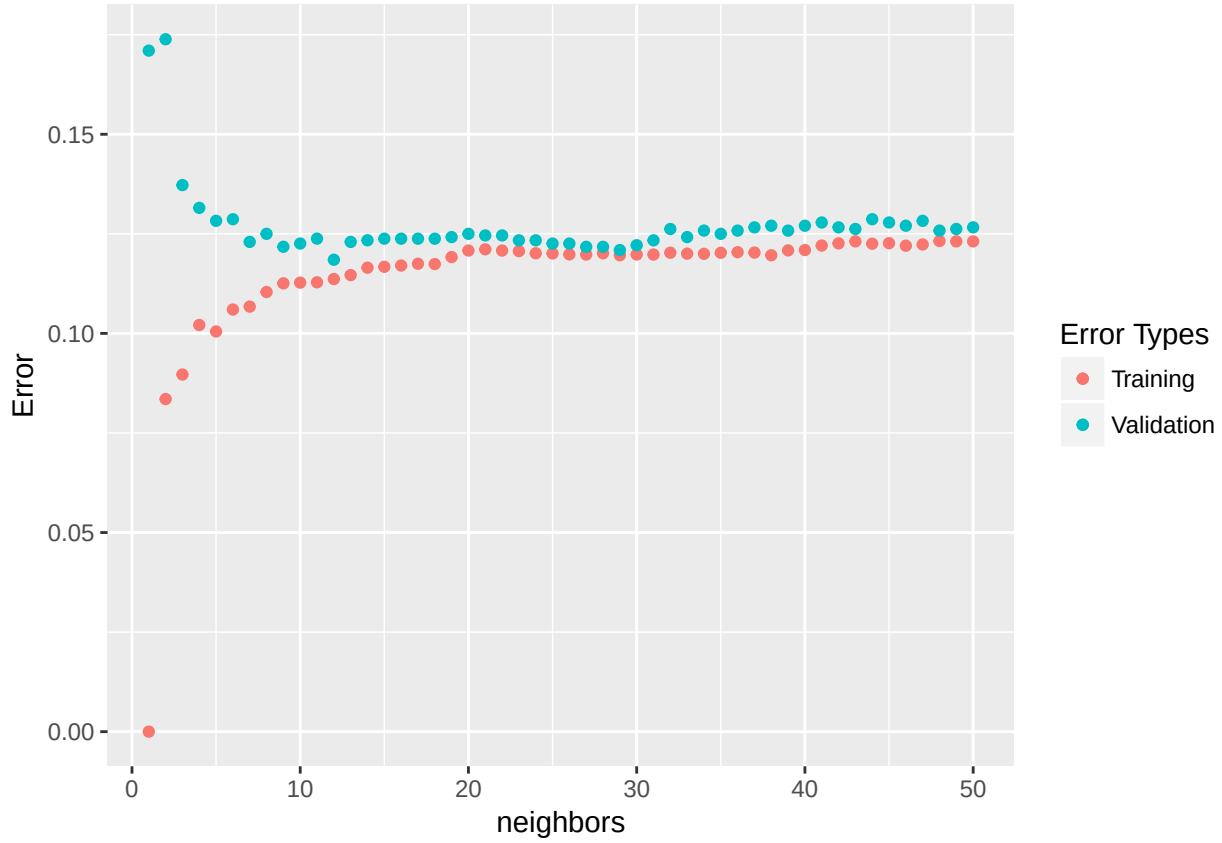
14. K-nearest neighbor: train a KNN model for classification. Use cross-validation to determine the best number of neighbors, and plot number of neighbors vs. resulting training and validation errors. Compute test error and save to `records`.

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

  train = (folddef!=chunkid) #obtain the training set indices, chunkid defines the test group

  Xtr = Xdat[train,] #obtain the predictors of the training set
  Ytr = Ydat[train] #obtain the true values of the training set

  Xvl = Xdat[!train,] #obtain the predictors of the validation set
  Yvl = Ydat[!train] #obtain the true values of the validation set
  ## get classifications for current training chunks
  predYtr = knn(train = Xtr, test = Xtr, cl = Ytr, k = k) #obtain the predicted classifications for the

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

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

```
set.seed(10)
Y = trn.cl$candidate
X = trn.cl %>% select(-candidate)

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

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

errors = reshape2::melt(error.folds, id.vars = c('neighbors'), value.name = 'error')
```

```
res = error.folds %>% group_by(neighbors) %>% summarise_all(mean)

ggplot(res, aes(neighbors))+
  geom_point(aes(y = train.error, col = 'blue')) +
  geom_point(aes(y = val.error, col = 'red')) +
  scale_colour_discrete(name = "Error Types", labels=c("Training", "Validation")) +
  labs(ylab('Error'))
```




```
val.errors.means = errors %>%
  filter(variable == 'val.error') %>%
  group_by(neighbors, variable) %>%
  summarise_all(funs(mean)) %>%
  ungroup() %>%
  filter(error == min(error))
best.neighbors = max(val.errors.means$neighbors)
```

```
Ytrain = trn.cl$candidate
Xtrain = trn.cl %>% select(-candidate)
Ytest = tst.cl$candidate
Xtest = tst.cl %>% select(-candidate)

pred.Ytrain = knn(train = Xtrain, test = Xtrain, cl = Ytrain, k = best.neighbors)
pred.Ytest = knn(train = Xtrain, test = Xtest, cl = Ytrain, k = best.neighbors)

error.knn.trn <- calc_error_rate(pred.Ytrain, Ytrain)
error.knn.tst <- calc_error_rate(pred.Ytest, Ytest)

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

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","glm")
```

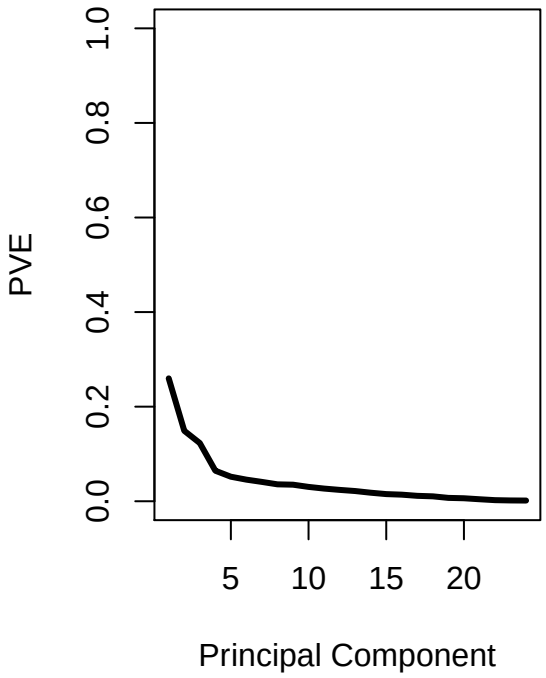
15. Compute principal components from the independent variables in training data. Then, determine the number of minimum number of PCs needed to capture 90% of the variance. Plot proportion of variance explained.

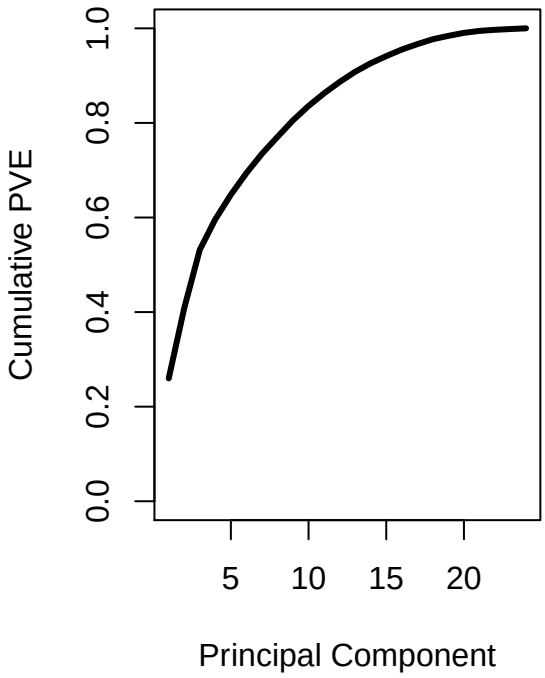
```
Xtrain.pc = prcomp(Xtrain, center = TRUE, scale = TRUE)
Xtrain.summ = summary(Xtrain.pc)
Xtrain.pc.cve = Xtrain.summ$importance[3,]
nPC = first(which(Xtrain.pc.cve > 0.9))
nPC
```

```
## [1] 13
```

Therefore, in order to capture 90% of variance with our PC's. We must use 13 PC's.

```
sdev = Xtrain.pc$sdev
pve = sdev^2/sum(sdev^2)
cum_pve = cumsum(pve)
par(mfrow=c(1,2))
plot(pve, type="l", lwd=3, xlab="Principal Component", ylab="PVE", ylim=c(0,1))
plot(cum_pve, type="l", xlab="Principal Component", ylab="Cumulative PVE", ylim=c(0, 1), lwd=3)
```



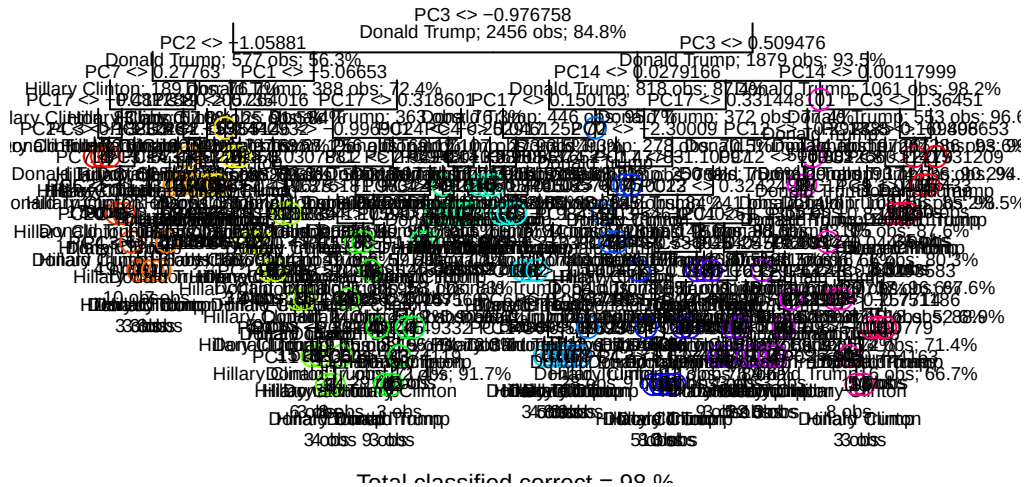


16. Create a new training data by taking class labels and principal components. Call this variable `tr.pca`. Create the test data based on principal component loadings: i.e., transforming independent variables in test data to principal components space. Call this variable `test.pca`.

```
#tr.pca = as.matrix(Xtrain) %*% Xtrain.pc$rotation
#test.pca = as.matrix(Xtest) %*% Xtrain.pc$rotation
Xtest.pc = prcomp(Xtest, center = TRUE, scale = TRUE)

tr.pca = Xtrain.pc$x
test.pca = Xtest.pc$x

tr.pca = as.data.frame(tr.pca)
test.pca = as.data.frame(test.pca)

tr.pca = cbind(as.data.frame(Ytrain), tr.pca)
test.pca = cbind(as.data.frame(Ytest), test.pca)
```

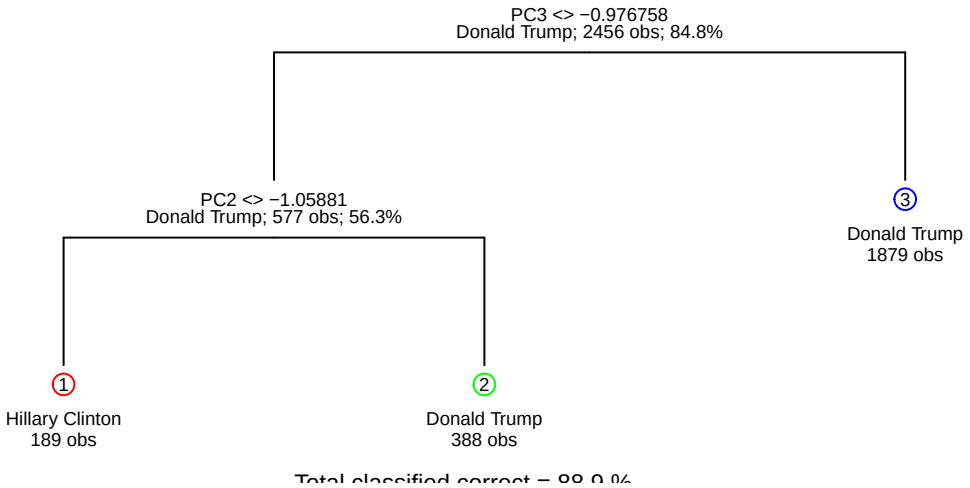
17. Decision tree: repeat training of decision tree models using principal components as independent variables. Record resulting errors.

```
tree.pc = tree(Ytrain~., data = tr.pca, control = tree.control(nobs = nrow(tr.pca), minsize = 6, mindev
draw.tree(tree.pc, cex = 0.6, nodeinfo=TRUE)
```



Total classified correct = 98 %

```
tree.pc.cv = cv.tree(tree.pc, rand = folds, FUN = prune.misclass, K = nfold)
nleaves = min(tree.pc.cv$size[which(tree.pc.cv$dev == min(tree.pc.cv$dev))])
tree.pc.pruned = prune.tree(tree.pc, best = nleaves)
draw.tree(tree.pc.pruned,cex = 0.6, nodeinfo = TRUE)
```




```
trn.pca.pred = predict(tree.pc.pruned, tr.pca, type = 'class')
tst.pca.pred = predict(tree.pc.pruned, test.pca, type = 'class')

error.trn <- calc_error_rate(trn.pca.pred, tr.pca$Ytrain)
error.tst <- calc_error_rate(tst.pca.pred, test.pca$Ytest)

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

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

```
set.seed(10)
Y = tr.pca$Ytrain
X = tr.pca %>% select(-Ytrain)

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

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

errors = reshape2::melt(error.folds, id.vars = c('neighbors'), value.name = 'error')
```

```
val.errors.means = errors %>%
  filter(variable == 'val.error') %>%
  group_by(neighbors, variable) %>%
  summarise_all(funs(mean)) %>%
  ungroup() %>%
  filter(error == min(error))
best.neighbors = max(val.errors.means$neighbors)
```

```
Ytrain = tr.pca$Ytrain
Xtrain = tr.pca %>% select(-Ytrain)
Ytest = test.pca$Ytest
Xtest = test.pca %>% select(-Ytest)

pred.Ytrain = knn(train = Xtrain, test = Xtrain, cl = Ytrain, k = best.neighbors)
pred.Ytest = knn(train = Xtrain, test = Xtest, cl = Ytrain, k = best.neighbors)

error.knn.trn <- calc_error_rate(pred.Ytrain, Ytrain)
error.knn.tst <- calc_error_rate(pred.Ytest, Ytest)

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

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 our investigation of the census data, we can see that we are able to accurately predict who would win the county according to county census. In our decision tree model, we were able to successfully predict approximately 90% of the counties in our test data set. For future elections, polls by county may be far more indicative of the winner of an election.

Taking it further

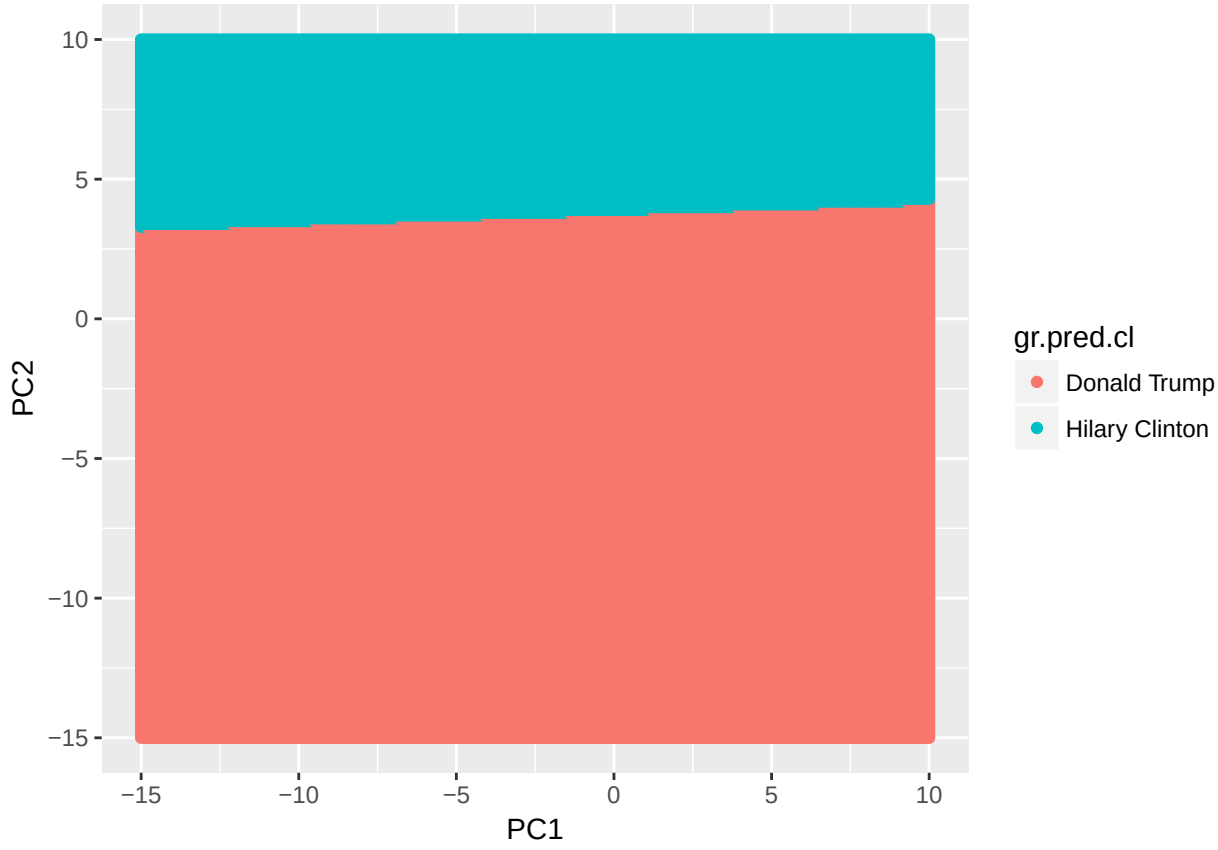
20. Propose and tackle at least one interesting question. Be creative! Some possibilities are:
    - Data preprocessing: we aggregated sub-county level data before performing classification. Would classification at the sub-county level before determining the winner perform better? What implicit assumptions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

Let's take a look at how logistic regression performs. First, let's look at some decision boundaries starting with principle components.

```
Y = election.cl$candidate
X = election.cl %>% select(-candidate)

Y = as.data.frame(Y)
X = as.data.frame(X)

X.pc = prcomp(X, center = TRUE, scale = TRUE)

election.pc = cbind(Y,X.pc$x)

ggplot(election.pc, aes(PC1,PC2, color = Y)) + geom_point()
```

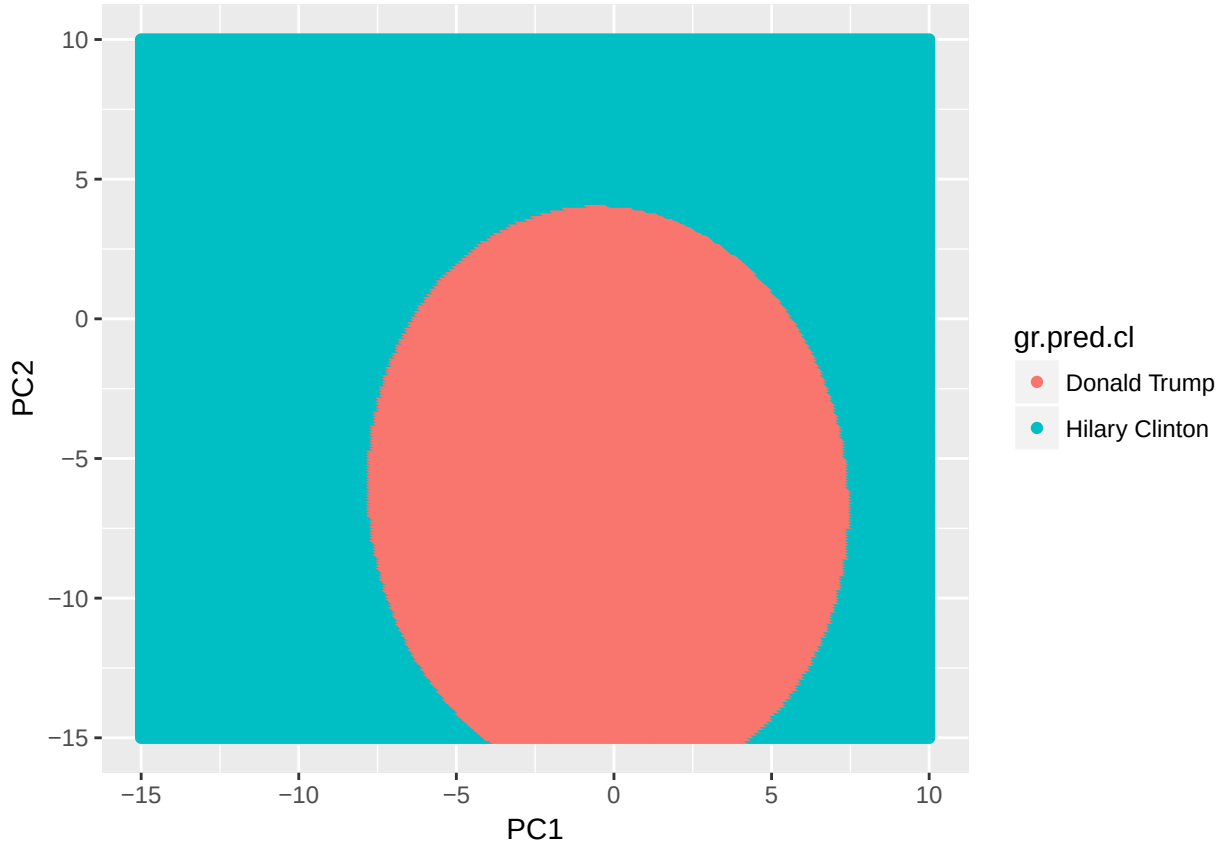

```
gr <- expand.grid(PC1=seq(-15, 10, by=0.1), # sample points in PC1
PC2=seq(-15, 10, by=0.1)) # sample points in PC2
```

```
election.glm = glm(Y~PC1 + PC2, family = binomial('logit'), data = election.pc)
gr.pred = predict(election.glm, gr, type = 'response')
gr.pred.cl = ifelse(gr.pred > 0.5, 'Hilary Clinton', 'Donald Trump')
ggplot(gr, aes(PC1,PC2)) + geom_point(aes(color = gr.pred.cl))
```



This is a decision boundary with a probability threshold set at 50%. For now the threshold is totally arbitrary (this will change later). The decision boundary of the linear logistic regression is very rough. It looks like it is representing a distribution other than the one shown in the scatterplot.

```
election.poly = glm(Y~poly(PC1, degree = 2, raw = "false") + poly(PC2, degree = 2, raw = "false") + PC1

gr.pred = predict(election.poly, gr, type = 'response')
gr.pred.cl = ifelse(gr.pred > 0.5, 'Hilary Clinton', 'Donald Trump')
ggplot(gr, aes(PC1,PC2)) + geom_point(aes(color = gr.pred.cl))
```

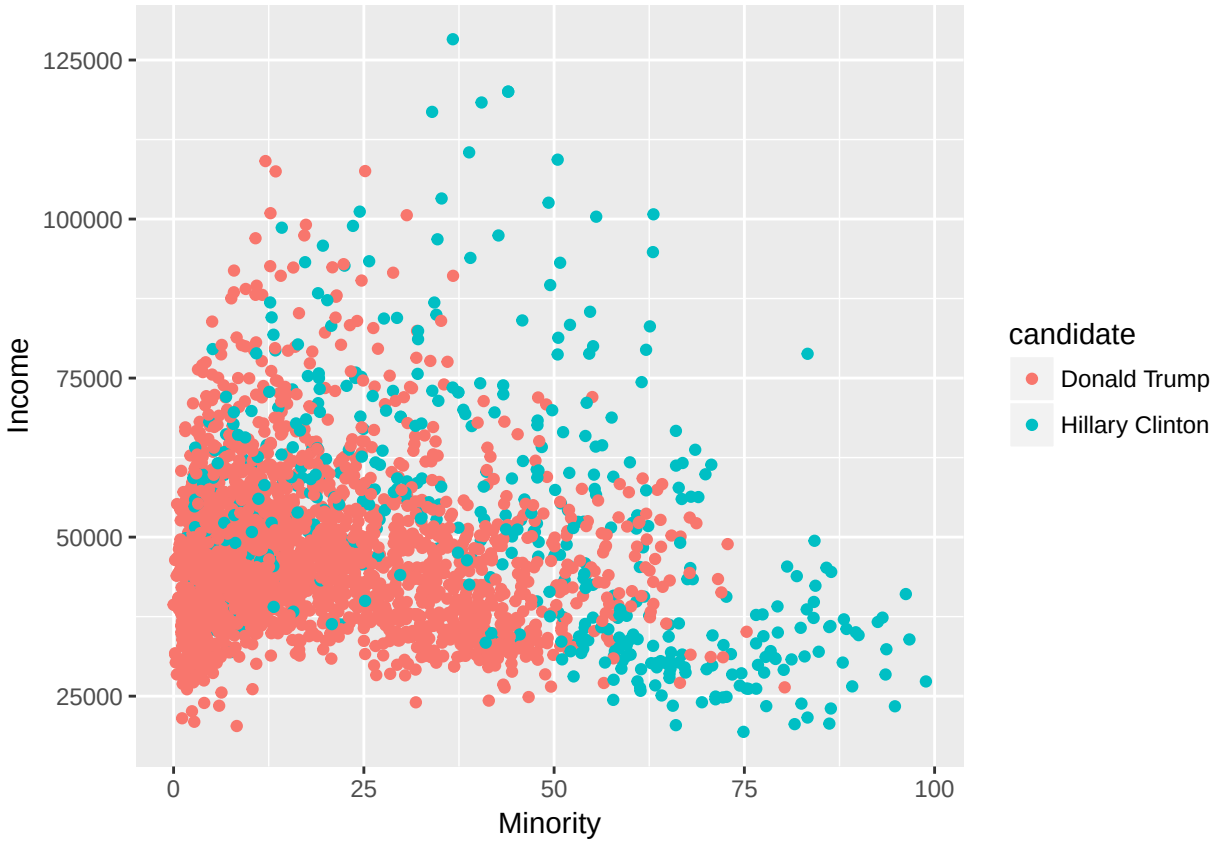


The decision boundary looks better than that of the linear logistic regression, however it still doesn't look like it is representing the scatterplot distribution. It is for this reason that it is reasonable to conclude that using only the first two principle components yields very high bias. We are going to have to use more principle components if we are going to be relying on components over native features.

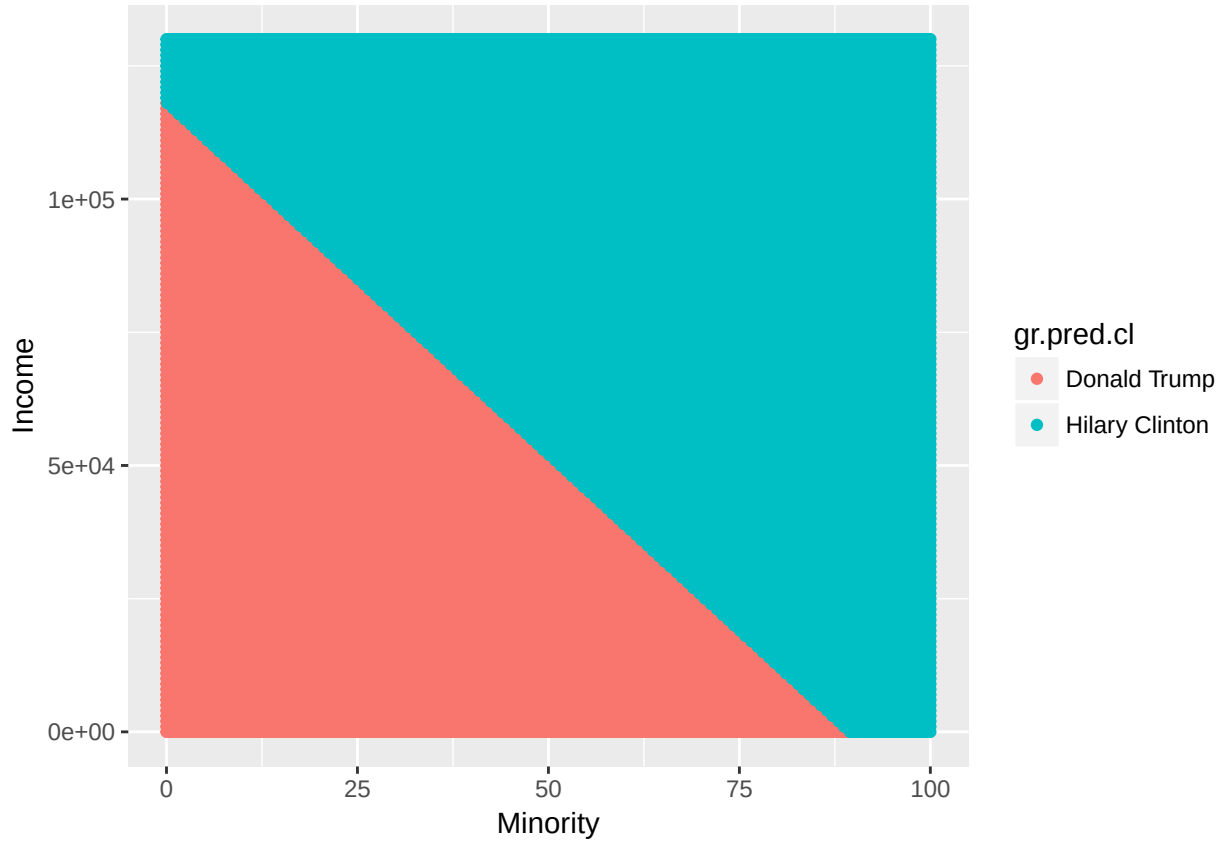
Next, let's look at decision boundaries made using native features.

```
gr <- expand.grid(Minority=seq(0, 100, by=0.1), # sample points in PC1
Income=seq(0, 130000, by=1000)) # sample points in PC2
```

```
ggplot(election.cl, aes(Minority,Income, color = candidate)) + geom_point()
```

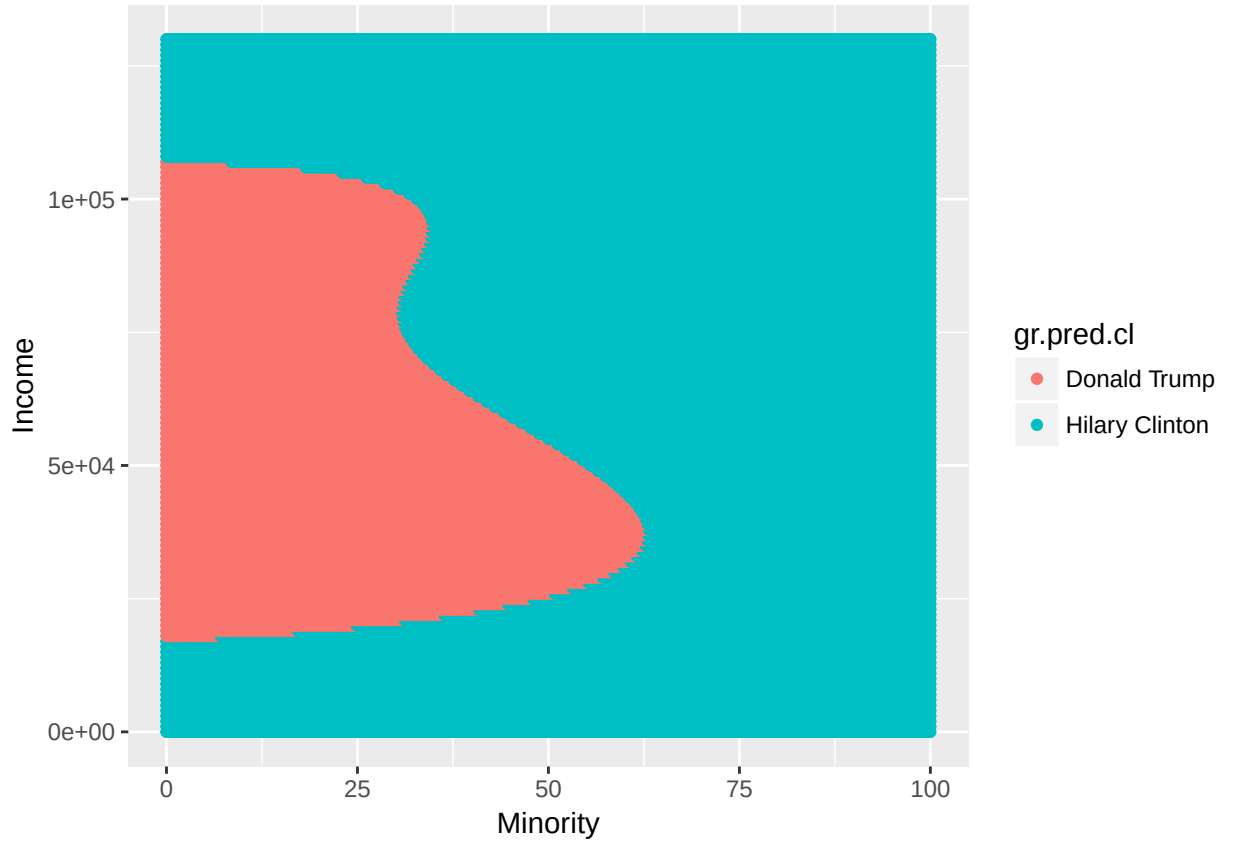



```
election.glm = glm(candidate~Minority + Income, family = binomial('logit'), data = election.cl)
gr.pred = predict(election.glm, gr, type = 'response')
gr.pred.cl = ifelse(gr.pred > 0.5, 'Hilary Clinton', 'Donald Trump')
ggplot(gr, aes(Minority,Income)) + geom_point(aes(color = gr.pred.cl))
```



This decision boundary looks like it does a decent job of representing the scatter plot. However, it still looks like it is representing a different scatter plot than the previous one.

```
election.poly = glm(candidate~poly(Minority, degree = 2, raw = "false") + poly(Income, degree = 5, raw =
gr.pred = predict(election.poly, gr, type = 'response')
gr.pred.cl = ifelse(gr.pred > 0.5, 'Hilary Clinton', 'Donald Trump')
ggplot(gr, aes(Minority,Income)) + geom_point(aes(color = gr.pred.cl))
```



This is the decision boundary created by polynomial logistic regression of degree 2. There is now curvature that isn't apparent in the above scatterplot. This is a sign that this model is overfitting and has high variance compared to the previous one. Because of this, we are going to simply include more predictors in our model instead of relying on a polynomial of Income and Minority.

Let's determine a good amount of principle components to use. We will settle for 90% cumulative variance explained.

```
Ytrain = trn.cl$candidate
Xtrain = trn.cl %>% select(-candidate)
Ytest = tst.cl$candidate
Xtest = tst.cl %>% select(-candidate)

Ytrain = as.data.frame(Ytrain)
Xtrain = as.data.frame(Xtrain)
Ytest = as.data.frame(Ytest)
Xtest = as.data.frame(Xtest)

Xtrain.pc = prcomp(Xtrain, center = TRUE, scale = TRUE)
Xtest.pc = prcomp(Xtest, center = TRUE, scale = TRUE)
```

```
Xtrain.summ = summary(Xtrain.pc)
Xtrain.pc.cve = Xtrain.summ$importance[3,]
nPC = first(which(Xtrain.pc.cve > 0.9))
nPC
```

```
## [1] 13
```

So, in order to account for 90% of the variance of the dataset, we need to include the first 13 principle components. Although we are only using 13 out of 25 possible principle components, the fact that they account for 90% of the variance in the dataset gives us reason to believe that the resulting model will be similar to one utilizing all of the predictors/principle components.

```
tr.pca = Xtrain.pc$x[,1:nPC]
test.pca = Xtest.pc$x [,1:nPC]

tr.pca = as.data.frame(tr.pca)
test.pca = as.data.frame(test.pca)

tr.pca = cbind(as.data.frame(Ytrain), tr.pca)
test.pca = cbind(as.data.frame(Ytest), test.pca)
```

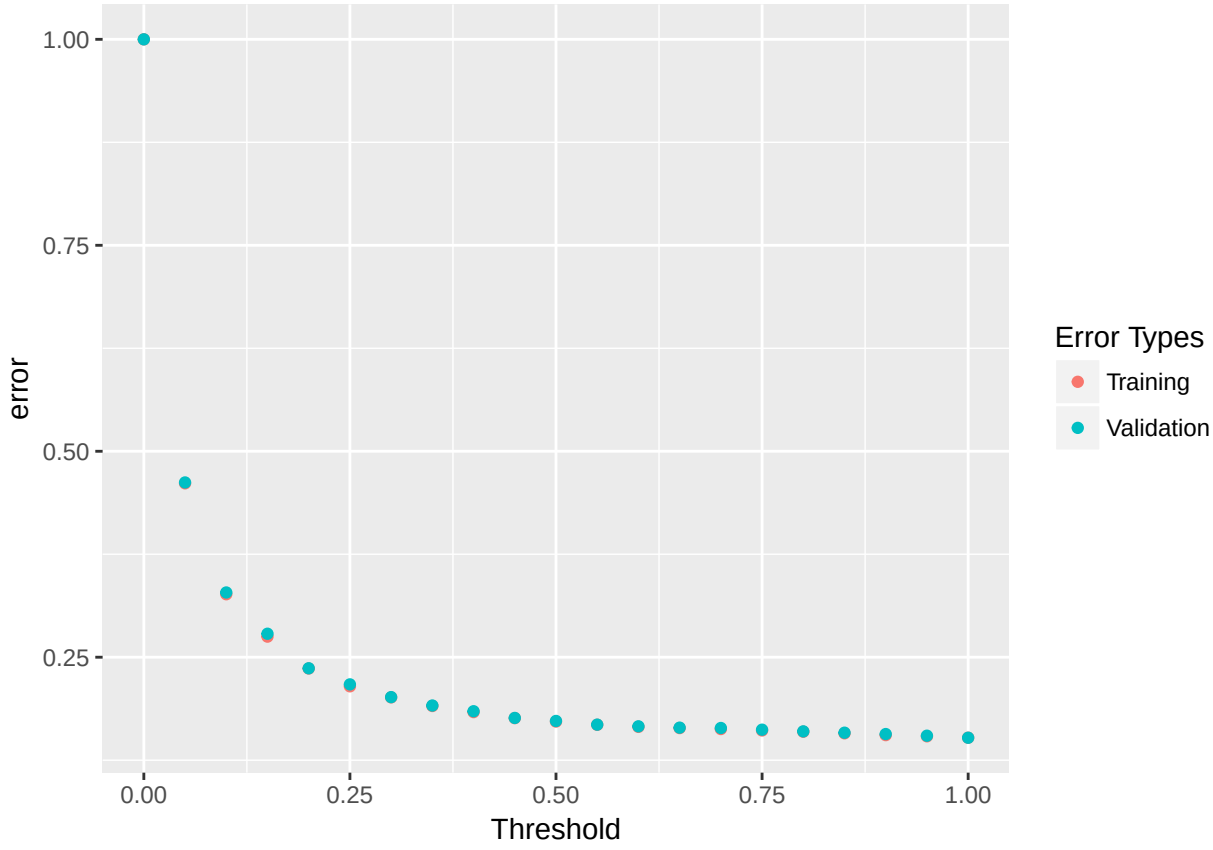
Previously the probability threshold was arbitrary. Let's use cross validation to figure out what threshold is best for our model.

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

  train = (folddef!=chunkid) #obtain the training set indices, chunkid defines the test group
  Xtr = Xdat[train,] #obtain the predictors of the training set
  Ytr = Ydat[train] #obtain the true values of the training set

  tr = cbind(as.data.frame(Ytr), Xtr)

  Xvl = Xdat[!train,] #obtain the predictors of the validation set
  Yvl = Ydat[!train] #obtain the true values of the validation set

  vl = cbind(as.data.frame(Yvl),Xvl)

  pred.glm = glm(Ytr~.,family = binomial('logit'), data = tr) #obtain the logistic regression model bas
  predYtr.probs = predict(pred.glm, tr,type = 'response') #obtain the classification probabilities for
  predYtr = ifelse(predYtr.probs > k, 'Hilary Clinton', 'Donald Trump')

  predYvl.probs = predict(pred.glm, vl, type = 'response')
  predYvl = ifelse(predYvl.probs > k, 'Hilary Clinton', 'Donald Trump')

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

```
set.seed(10)
X = tr.pca %>% select(-Ytrain)
Y = tr.pca$Ytrain

tr.pca = as.data.frame(tr.pca)
test.pca = as.data.frame(test.pca)

kvec = seq(0,1, by = 0.05)
error.folds = NULL

for(i in kvec){
  tmp = plyr::ldply(1:nfold, do.chunk, folddef = folds, Xdat = X, Ydat = Y, k = i)
  tmp$perc = i
  error.folds = rbind(error.folds, tmp)
```

```
}
errors = reshape2::melt(error.folds, id.vars = c('perc'), value.name = 'error')
```

```
val.errors.means = errors %>%
  filter(variable == 'val.error') %>%
  group_by(perc, variable) %>%
  summarise_all(funs(mean)) %>%
  ungroup() %>%
  filter(error == min(error))

errors.means = errors %>%
  group_by(perc, variable) %>%
  summarise_all(funs(mean))

ggplot(errors.means, aes(perc))+
  geom_point(aes(y = error, col = variable)) +
  scale_colour_discrete(name = "Error Types", labels=c("Training", "Validation")) +
  labs(xlab('Threshold'), ylab('Error'))
```




```
best.perc = max(val.errors.means$perc)
best.perc
```

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

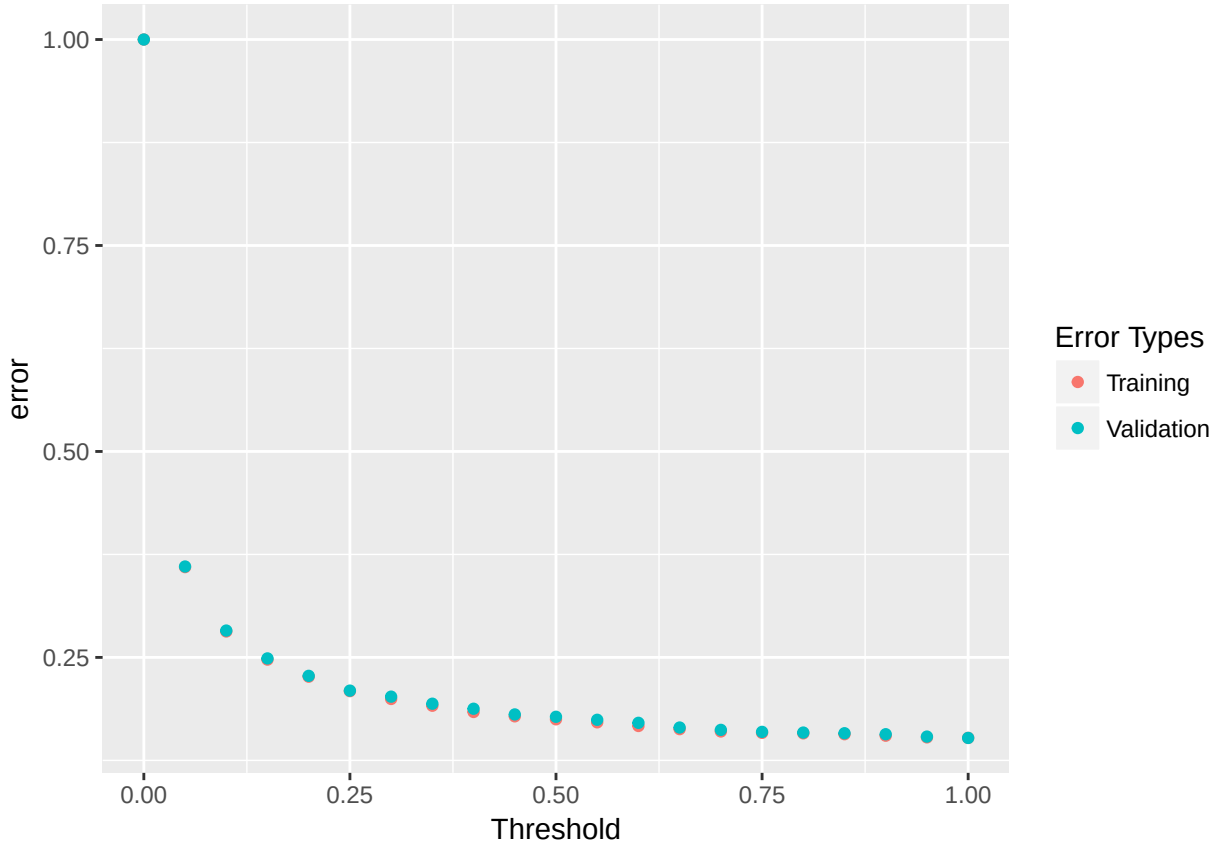
According to 10-fold cross validation, the percentage threshold to minimize validation error is 1.

Now, we'll perform Logistic Regression using 13 principle components and a probability threshold of 1.

```
Ytrain = tr.pca$Ytrain
Xtrain = tr.pca %>% select(-Ytrain)
Ytest = test.pca$Ytest
Xtest = test.pca %>% select(-Ytest)

tr.glm = glm(Ytrain~., family = binomial('logit'), data = tr.pca)
pred.Ytrain = predict(tr.glm, tr.pca,type = 'response')
pred.Ytest = predict(tr.glm, test.pca, type = 'response')

pred.Ytrain = ifelse(pred.Ytrain > best.perc, 'Hilary Clinton', 'Donald Trump')
pred.Ytest = ifelse(pred.Ytest > best.perc, 'Hilary Clinton', 'Donald Trump')

error.glm.trn <- calc_error_rate(pred.Ytrain, Ytrain)
error.glm.tst <- calc_error_rate(pred.Ytest, Ytest)

pca.records[3,1]<-error.glm.trn
pca.records[3,2]<-error.glm.tst
```

```
set.seed(10)
Y = trn.cl$candidate
X = trn.cl %>% select(-candidate)

kvec = seq(0,1, by = 0.05)
error.folds = NULL

for(i in kvec){
  tmp = plyr::ldply(1:nfold, do.chunk, folddef = folds, Xdat = X, Ydat = Y, k = i)
  tmp$perc = i
  error.folds = rbind(error.folds, tmp)
}
errors = reshape2::melt(error.folds, id.vars = c('perc'), value.name = 'error')
```

```
val.errors.means = errors %>%
  filter(variable == 'val.error') %>%
  group_by(perc, variable) %>%
  summarise_all(funs(mean)) %>%
  ungroup() %>%
  filter(error == min(error))

errors.means = errors %>%
  group_by(perc, variable) %>%
  summarise_all(funs(mean))

ggplot(errors.means, aes(perc))+
  geom_point(aes(y = error, col = variable)) +
  scale_colour_discrete(name = "Error Types", labels=c("Training", "Validation")) +
  labs(xlab('Threshold'), ylab('Error'))
```




```
best.perc = max(val.errors.means$perc)
best.perc
```

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

Coincidentally, the best probability threshold for native logistic regression is also 1.

```
Ytrain = trn.cl$candidate
Xtrain = trn.cl %>% select(-candidate)
Ytest = tst.cl$candidate
Xtest = tst.cl %>% select(-candidate)

tr.glm = glm(Ytrain~., family = binomial('logit'), data = trn.cl)
pred.Ytrain = predict(tr.glm, trn.cl,type = 'response')
pred.Ytest = predict(tr.glm, tst.cl, type = 'response')

pred.Ytrain = ifelse(pred.Ytrain > best.perc, 'Hilary Clinton', 'Donald Trump')
pred.Ytest = ifelse(pred.Ytest > best.perc, 'Hilary Clinton', 'Donald Trump')

error.glm.trn <- calc_error_rate(pred.Ytrain, Ytrain)
error.glm.tst <- calc_error_rate(pred.Ytest, Ytest)

records[3,1]<-error.glm.trn
records[3,2]<-error.glm.tst
```

Now that we have performed all of the logistic regressions, we can evaluate how they performed.

```
records
```

```
##      train.error test.error
## tree  0.04112378  0.1058632
## knn   0.11319218  0.1254072
## glm   0.15228013  0.1465798
```

```
pca.records
```

```
##      train.error test.error
## tree  0.11115635  0.2052117
## knn   0.05415309  0.2524430
## glm   0.15228013  0.1465798
```

As we suspected, logistic regression on 13 principle components gives you the same result as running logistic regression on all predictors. The strange thing is that the test error is slightly lower than the training error, however that can come the way that we split election.cl into the training and testing sets. We also see that among all of the classification methods using principle components, Logistic Regression outperformed all of the other methods. Actually, all of the native feature classification methods outperform their PCA counterparts. Logistic regression is the only exception as it is essentially the same for both predictor types. So, we see that it doesn't seem to make a difference whether or not you use principle components or native features for this dataset, especially if you are going to use as many principle components as we did.