Modeling and Predicting the 2016 US Presidential Election

Christopher Nguyen | *Kendall Brown*

Winter 2018

Introduction to the US 2016 Presidential Election

1.

Hillary Clinton vs Donald Trump. Democrat vs Republican. Red vs Blue. Good versus Evil(depending on who you ask). The 2016 US election proved to be one of the most polarizing and consequential elections of our time. The results struck as a horrific surprise to some, and a resurgence to others. Indeed, this came as a surprise to even the most robust polling systems available. In addition to irreducible error, what things could we potentially not measure? Many cite voter apathy is large contributor for Clintonâ□□s stunning loss. Nearly half our all eligible voters did not vote. This is no more apparent than in swing states that cost Clinton dearly. Republican voter turnout, has been consistent, not really changing. Democrat voter turnout, has been on decline ever since 2008, declining until 2016, when swing states that voted blue for Obama, now had majority victory because of consistency. Other intangibles that polls did could not predict was the so called â□□silent majority.â□□ Like Nixon before him, Trump called upon the silent majority, blue collar Americans mostly forgotten in the era of globalization. Again, these voters would prove to be invaluable in swing states that Trump would clinch by.

Introduction to the US 2016 Presidential Election

2.

Although the exact details of his methods are kept secret, Nate Silverâ□□s method eerily is reminiscent of what we are currently learning. Silver, like us, he acquires data from sources such as pollsters and trains it to new existing data as they are created. Silver, however, adds one step further and conducts a time series of sorts, simulating it and going forward in time. He weights these predicted points with the original point, the true value assumption, thus allowing to forecast. But comically enough, Silver wrangles with the same problem we do, the bias variance trade off. Data acquisition could be biased depending on sources of the survey, poll, etc.

Introduction to the US 2016 Presidential Election

3.

The bulk of these errors are bias variance trade off related. Nearly every pollster that had significant errors, they all pointed in the same direction as each other, a strong indication of an underlying systematic error. Admittedly yes, polls can be wrong, thatâ□□s the nature of statistics and forecasting but Trump devastated the predictions, outperforming forecasts by an average of 7%. Whatâ□□s more shocking was Trumpâ□□s winning of states pollsters were confident he could not win in, but did so anyway. We should be wary of the intangibles mentioned above and be cautious of overfitting our model, lest we fall for the same mistake as they did. Furthermore, pollsters fail to realize the weight that opinion polls carried, overvaluing that of demographics, as the election had become a fiercely racial standoff.

Data Wrangling: Making Sense of the Data

4.

```
election.raw = read.csv("election.csv") %>% as.tbl
census_meta = read.csv2("metadata.csv") %>% as.tbl
census = read.csv("census.csv") %>% as.tbl
census$CensusTract = as.factor(census$CensusTract)
```

```
election_federal=filter(election.raw,state=="US",is.na(county))
election_state=filter(election.raw,state!="US",is.na(county))
election=filter(election.raw,is.na(county)==0)
totalvotes  = election_federal %>%
              select(candidate, votes) %>%
              group_by(candidate)
```

Data Wrangling: Making Sense of the Data

5.Number of candidates running in 2016

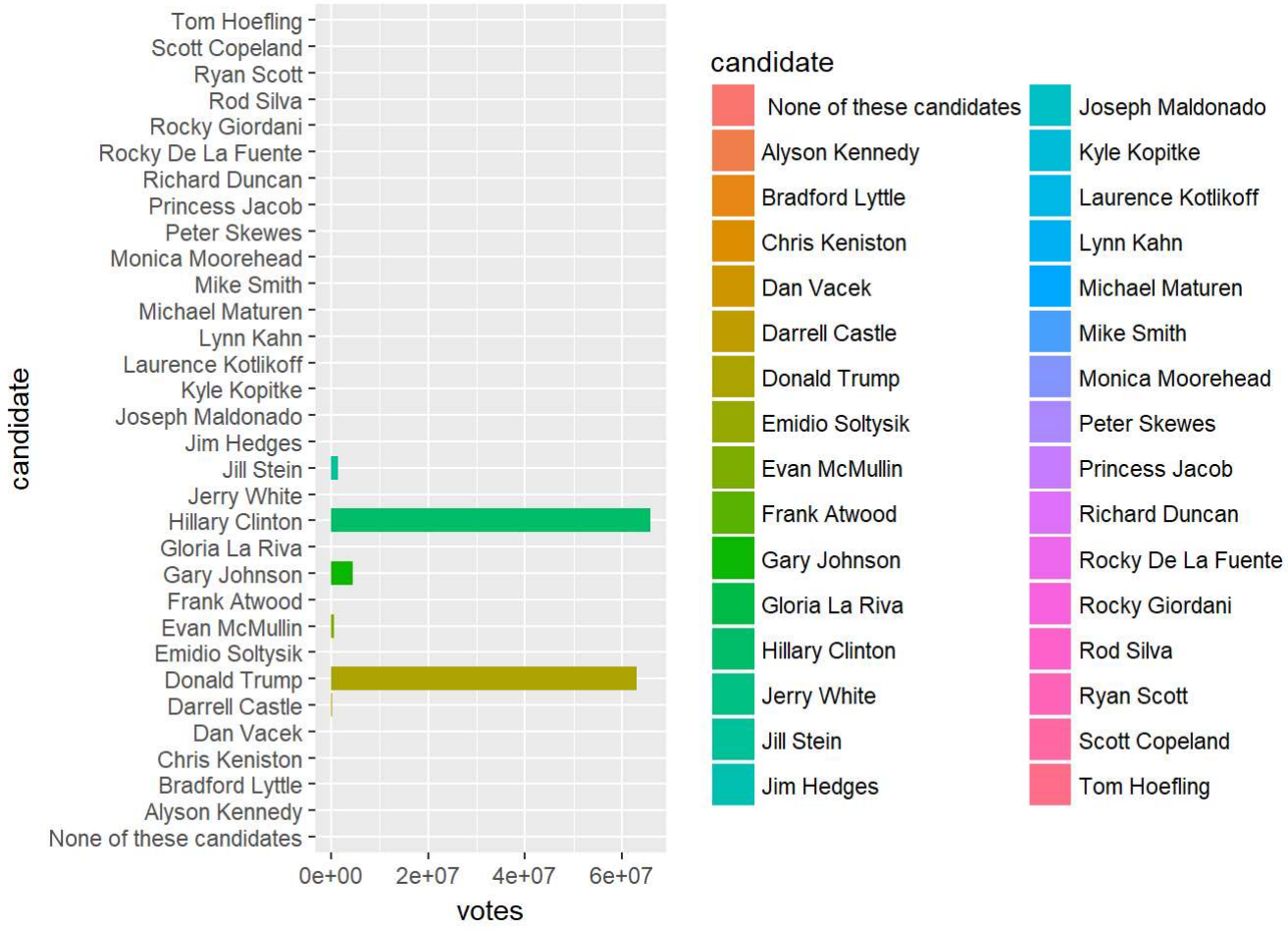
```
nrow(totalvotes)-1
```

```
## [1] 31
```

According to this data set there were 31 different candidates running in 2016.

Visualizing votes per candidate in a bar chart.

```
ggplot(totalvotes)+
  geom_bar(mapping=aes(x=candidate,y=votes,fill=candidate),stat="identity")+
  coord_flip()
```



Data Wrangling: Making Sense of the Data

6.Creating county_winner and state_winner variables by calculating vote proportion.

```
county_votes=election%>%
  select(fips,county,candidate,votes,state)%>%
  group_by(fips)
county_votes$County_Votes=ave(county_votes$votes,county_votes$fips,FUN=sum)
county_votes$pct=round(county_votes$votes/county_votes$County_Votes,4)
county_votes$Win_Prediction=county_votes$pct==ave(county_votes$pct,county_votes$fips,FUN=max)
county_winner=county_votes%>%filter(Win_Prediction==1)
county_winner=county_winner[,c(1:5,7)]
head(county_winner)
```

fips <fctr>	county <fctr>	candidate <fctr>	votes <int>	state <fctr>	pct <dbl>
6037	Los Angeles County	Hillary Clinton	2464364	CA	0.7203
17031	Cook County	Hillary Clinton	1611946	IL	0.7475
4013	Maricopa County	Donald Trump	747361	AZ	0.4863
48201	Harris County	Hillary Clinton	707914	TX	0.5423
6073	San Diego County	Hillary Clinton	735476	CA	0.5697
6059	Orange County	Hillary Clinton	609961	CA	0.5142

6 rows

Data Wrangling: Making Sense of the Data

6.Creating county_winner and state_winner variables by calculating vote proportion.

```
state_votes=election_state%>%
  select(state,candidate,votes)%>%
  group_by(state)
state_votes$state_votes=ave(state_votes$votes,state_votes$state,FUN=sum)
state_votes$pct=round(state_votes$votes/state_votes$state_votes,4)
state_votes$Win_Prediction=state_votes$pct==ave(state_votes$pct,state_votes$state,FUN=max)
state_winner=state_votes%>%filter(Win_Prediction==1)
state_winner=state_winner[,c(1:3,5)]
head(state_winner)
```

state <fctr>	candidate <fctr>	votes <int>	pct <dbl>
CA	Hillary Clinton	8753788	0.6226
FL	Donald Trump	4617886	0.4902
TX	Donald Trump	4685047	0.5253

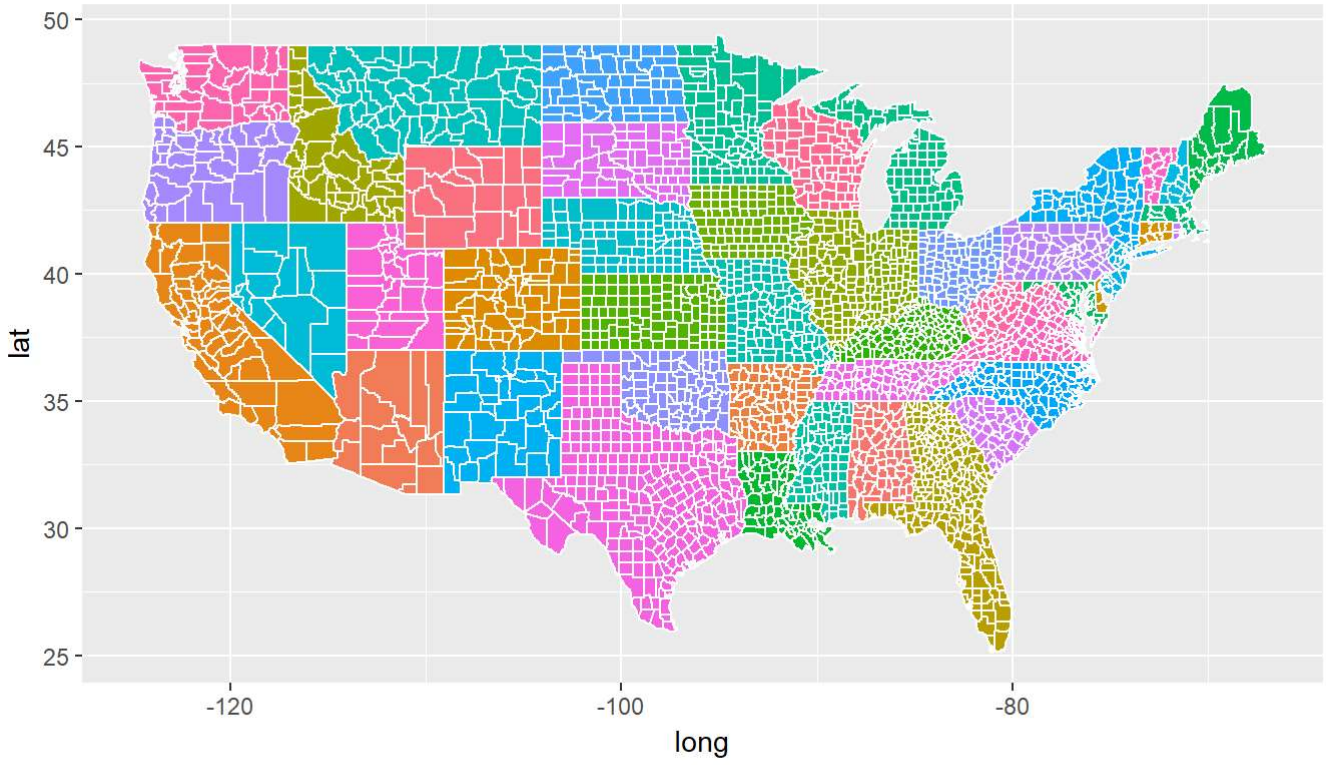
state <fctr>	candidate <fctr>	votes <int>	pct <dbl>
NY	Hillary Clinton	4556124	0.5948
PA	Donald Trump	2970733	0.4858
IL	Hillary Clinton	3090729	0.5596

6 rows

Visualization

7.Drawing a county level map of the United States.

```
counties=map_data("county")
ggplot(data=counties)+
  geom_polygon(aes(x = long, y = lat, fill = region, group = group), color = "white") +
  coord_fixed(1.3) +
  guides(fill=FALSE)
```

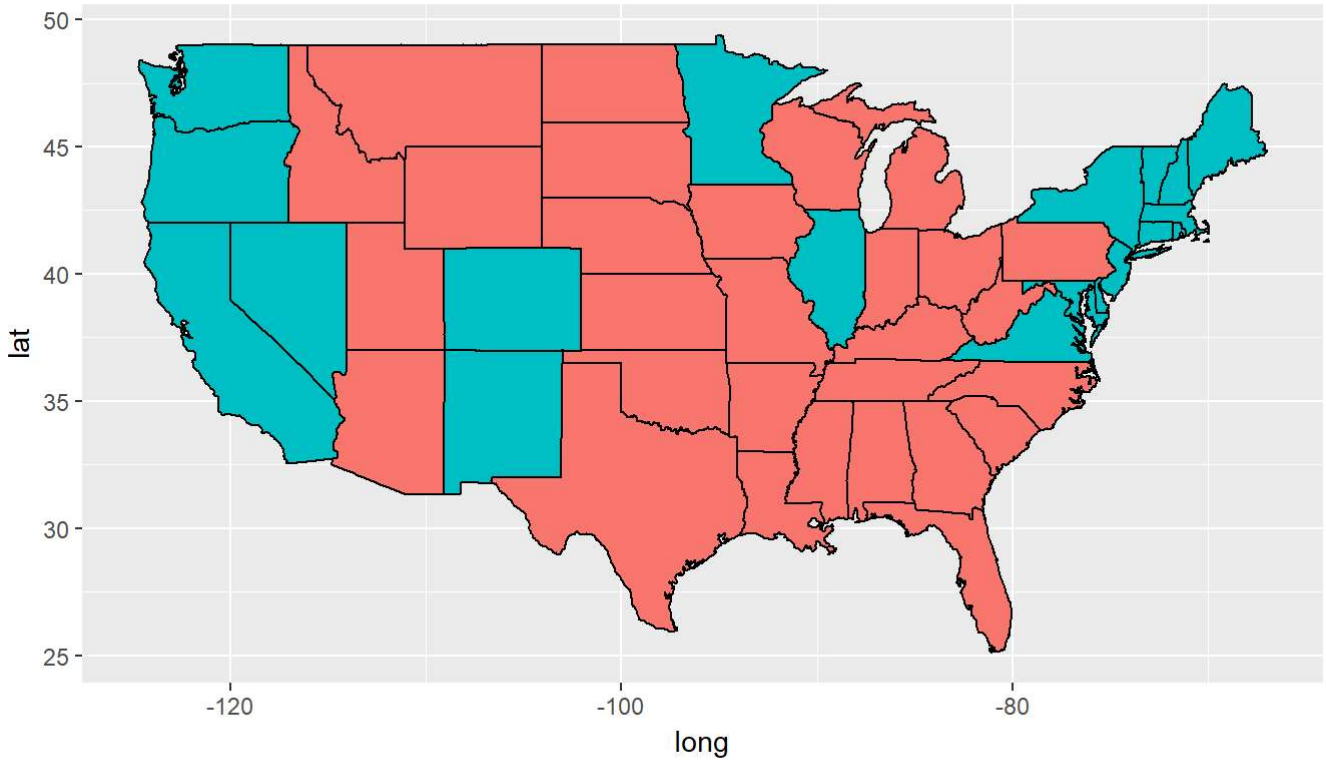


Visualization

8. Matching each county to their winning candidate.

```
state_winner$region=tolower(state.name[match(state_winner$state,state.abb)])
```

```
states=map_data("state")
states.comb=left_join(state_winner,states)
ggplot(data=states.comb)+
  geom_polygon(aes(x = long, y = lat, fill = states.comb$candidate, group = group), color = "bla
ck") +
  coord_fixed(1.3) +
  guides(fill=FALSE)
```



Visualization

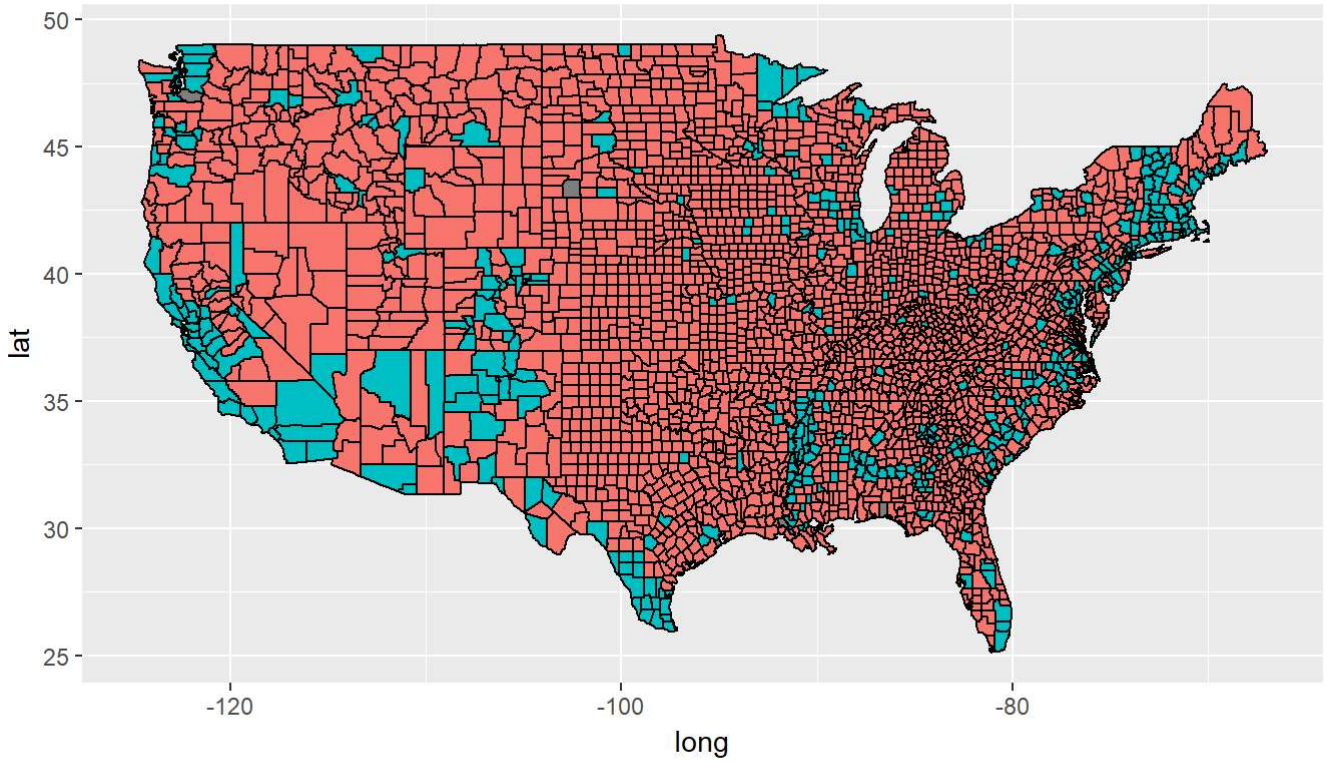
9. Adding fips column to counties data set and plotting a map according to county winner.

```
counties.fips=counties
x=maps::county.fips
x$region=gsub(",.*","",x$polyname)
x$subregion=gsub(".*,","",x$polyname)
counties.fips=left_join(counties,x)
counties.fips$fips=as.factor(counties.fips$fips)
head(counties.fips)
```

	long	lat	group	order	region	subregion	fips	polyname
	<dbl>	<dbl>	<dbl>	<int>	<chr>	<chr>	<fctr>	<chr>
1	-86.51	32.35	1	1	alabama	autauga	1001	alabama,autauga
2	-86.53	32.35	1	2	alabama	autauga	1001	alabama,autauga
3	-86.55	32.37	1	3	alabama	autauga	1001	alabama,autauga
4	-86.56	32.38	1	4	alabama	autauga	1001	alabama,autauga
5	-86.58	32.38	1	5	alabama	autauga	1001	alabama,autauga
6	-86.59	32.38	1	6	alabama	autauga	1001	alabama,autauga

6 rows

```
county.comb=left_join(counties.fips,county_winner,by="fips")
ggplot(data=county.comb)+
  geom_polygon(aes(x = long, y = lat, fill = county.comb$candidate, group = group), color = "bla
ck") +
  coord_fixed(1.3) +
  guides(fill=FALSE)
```



Visualization

10. Plotting a bubble graph of income per capita in each state, where bubble size is proportional to the state's minority population.

```
census.plus=census%>%
  filter(complete.cases(census))
census.plus$Men.per=(census.plus$Men/census.plus$TotalPop)
census.plus$Women.per=(census.plus$Women/census.plus$TotalPop)
census.plus$Employment.Popluation.ratio=census.plus$Employed/census.plus$TotalPop
```

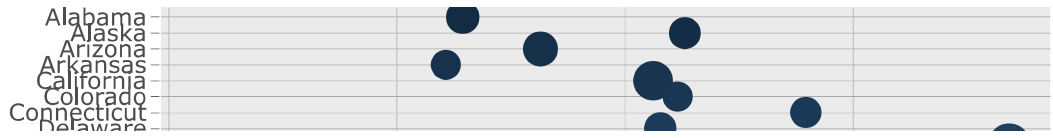
```
census.plus$state.pop=(ave(census.plus$TotalPop,census.plus$State,FUN=sum))
census.plus$hisp.per=(ave(census.plus$Hispanic,census.plus$State,FUN=mean)/100)*census.plus$stat
e.pop
census.plus$white.per=(ave(census.plus$White,census.plus$State,FUN=mean)/100)*census.plus$state.
pop
census.plus$black.per=(ave(census.plus$Black,census.plus$State,FUN=mean)/100)*census.plus$state.
pop
census.plus$asian.per=(ave(census.plus$Asian,census.plus$State,FUN=mean)/100)*census.plus$state.
pop
census.plus$native.per=(ave(census.plus$Native,census.plus$State,FUN=mean)/100)*census.plus$stat
e.pop
census.plus$pacific.per=(ave(census.plus$Pacific,census.plus$State,FUN=mean)/100)*census.plus$st
ate.pop
census.plus$men.per=(ave(census.plus$Men.per,census.plus$State,FUN=mean))*census.plus$state.pop
census.plus$women.per=(ave(census.plus$Women.per,census.plus$State,FUN=mean))*census.plus$state.
pop
census.plus$StateIncomePerCap=(ave(census.plus$IncomePerCap,census.plus$State,FUN=mean))
```

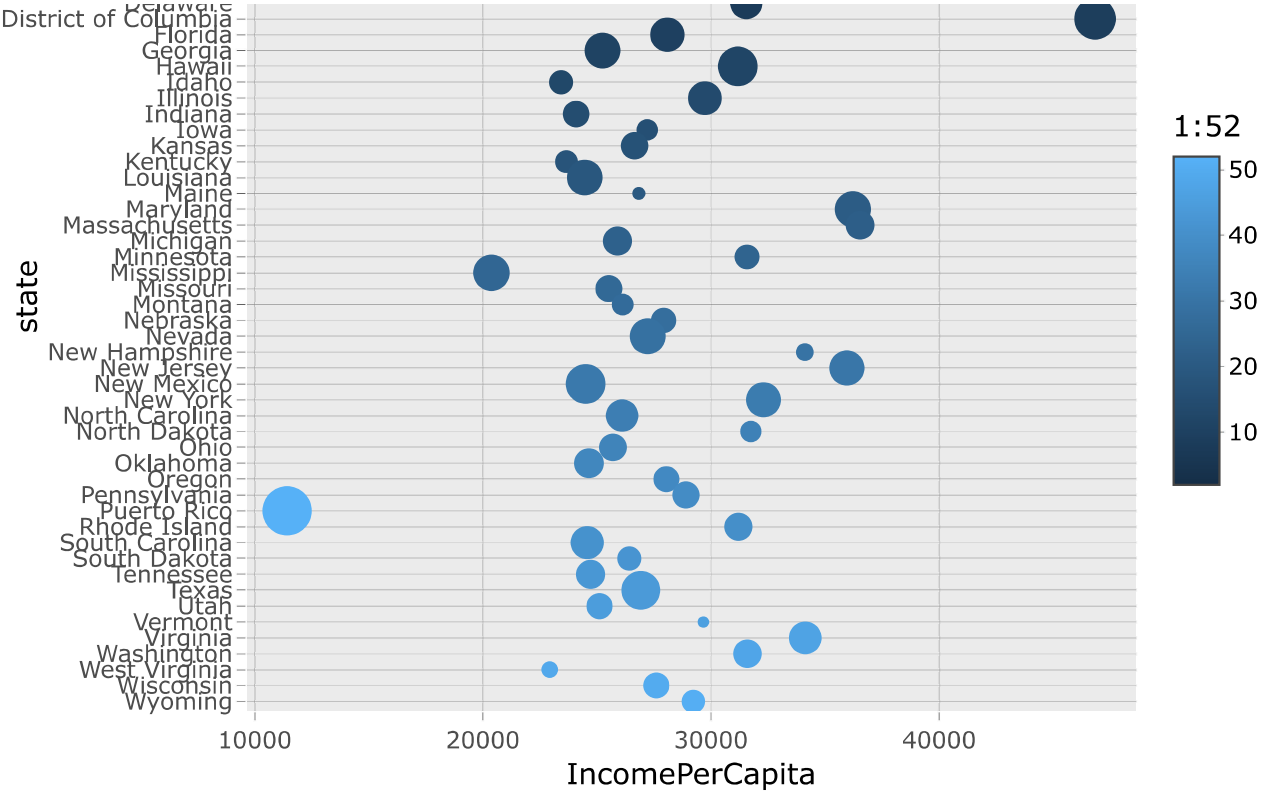
```
census.plus.unique=data.frame(c(1:52))
census.plus.unique=as.tibble(census.plus.unique)
```

```
census.plus.unique$state=unique(census.plus$State)
census.plus.unique$pop=unique(census.plus$state.pop)
census.plus.unique$hisp=unique(census.plus$hisp.per)
census.plus.unique$white=unique(census.plus$white.per)
census.plus.unique$black=unique(census.plus$black.per)
census.plus.unique$asian=unique(census.plus$asian.per)
census.plus.unique$native=unique(census.plus$native.per)
census.plus.unique$pacific=unique(census.plus$pacific.per)
census.plus.unique$men=unique(census.plus$men.per)
census.plus.unique$women=unique(census.plus$women.per)
census.plus.unique$minority=(census.plus.unique$hisp+census.plus.unique$black+census.plus.unique
$native+
  census.plus.unique$asian+census.plus.unique$pacific)/census.plus.unique$pop
census.plus.unique$IncomePerCapita=unique(census.plus$StateIncomePerCap)
```

```
census.plus.unique$state=
  factor(census.plus.unique$state,levels=rev(levels(census.plus.unique$state)))
p=ggplot(mapping=aes(state,IncomePerCapita,size=minority,color=1:52),data=census.plus.unique)+
  geom_point()+
  coord_flip()
ggplotly(p)
```

```
## We recommend that you use the dev version of ggplot2 with `ggplotly()`
## Install it with: `devtools::install_github('hadley/ggplot2')`
```



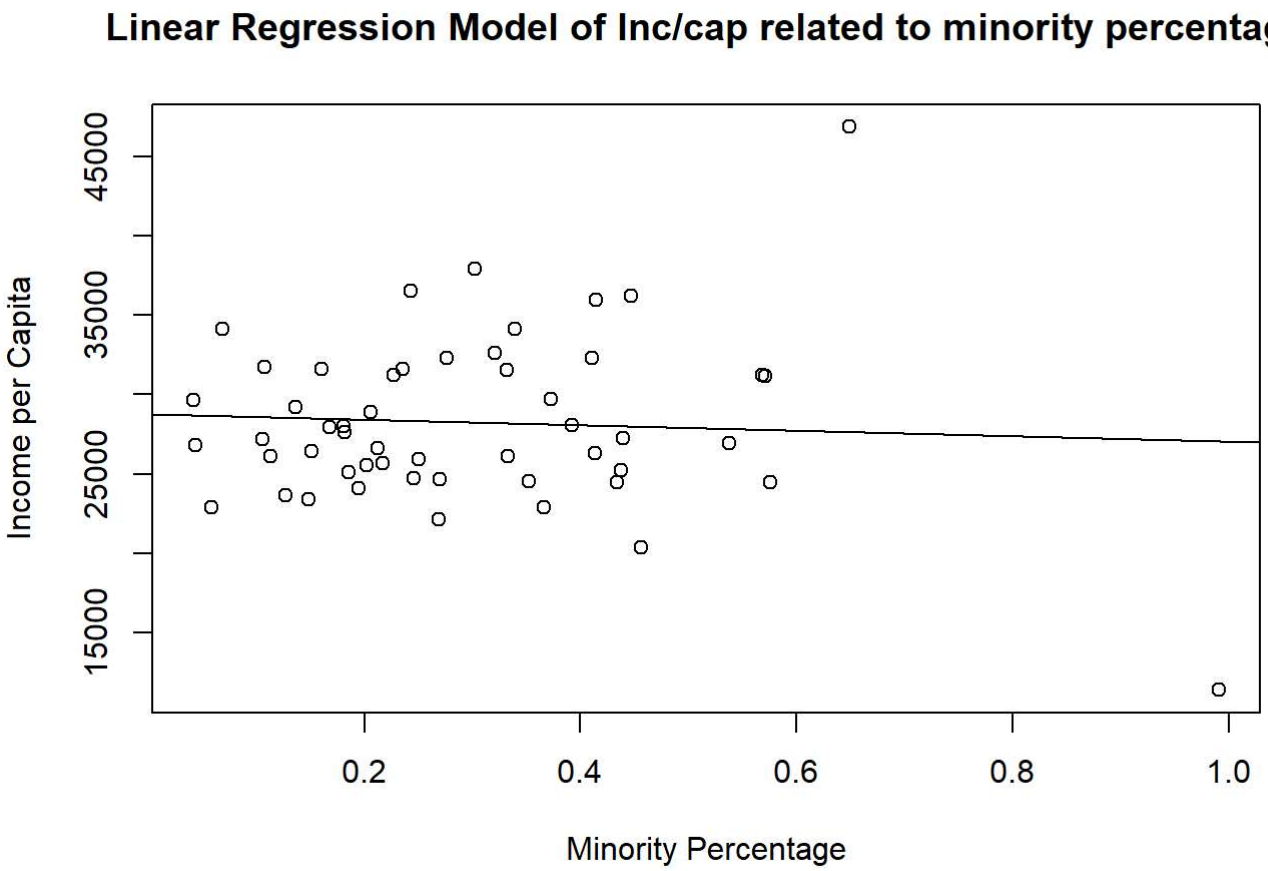


As we can see the percentage of minorities in a given state has little impact on the states income per capita. To further illustrate this point a basic linear regression model is to be drawn mapping income per cap to minority percentage.

```
fit.income=lm(IncomePerCapita~minority,data=census.plus.unique)
summary(fit.income)
```

```
##
## Call:
## lm(formula = IncomePerCapita ~ minority, data = census.plus.unique)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -15631  -3285   -885   3261  19191
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    28775       1461   19.70   <2e-16 ***
## minority       -1741       4212   -0.41     0.68
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5400 on 50 degrees of freedom
## Multiple R-squared:  0.0034, Adjusted R-squared:  -0.0165
## F-statistic: 0.171 on 1 and 50 DF,  p-value: 0.681
```

```
plot(census.plus.unique$minority,census.plus.unique$IncomePerCapita,
     main="Linear Regression Model of Inc/cap related to minority percentage",
     xlab="Minority Percentage",ylab="Income per Capita")
abline(fit.income)
```



From the summary report and the plot, we clearly see income per capita is indipendant of a states minority percentage.

Visualization

11.Cleaning census data to be converted to weighted county level data.

```
census.del=census%>%
  filter(complete.cases(census))
census.del$Men=(census.del$Men/census.plus$TotalPop)*100
census.del$Women=(census.del$Women/census.plus$TotalPop)
census.del$Employed=(census.del$Employed/census.del$TotalPop)*100
census.del$Citizen=(census.del$Citizen/census.del$TotalPop)*100
census.del$Minority=(census.del$Hispanic+census.del$Black+census.del$Native+
  census.del$Asian+census.del$Pacific)
census.del=census.del%>%select(-c(Walk,PublicWork,Construction,Hispanic,Asian,Black,Native,Pacif
ic,Women,White))
```

Creating subcounty census data.

```
census.subct=census.del%>%group_by(State,County)
census.subct$CountyTotal=ave(census.subct$TotalPop,census.subct$County,census.subct$State,FUN=su
m)
CountyWeight=census.subct$TotalPop/census.subct$CountyTotal
```

Creating county census data and printing the first 6 rows of census.ct using the head() function.

```
census.ct=census.subct[1:3]
census.ct[4:29]=census.subct[4:29]*CountyWeight
census.ct=census.ct%>%
  summarise_at(vars(TotalPop:CountyTotal),sum)
census.ct$TotalPop=census.ct$CountyTotal
census.ct=census.ct%>%select(-c(CountyTotal))
head(census.ct)
```

State <fctr>	County <fctr>	TotalPop <dbl>	Men <dbl>	Citizen <dbl>	Inco… <dbl>	IncomeErr <dbl>	IncomePerC… <dbl>	IncomePerCapErr <dbl>	P
Alabama	Autauga	55221	48.43	73.75	51696	7771	24974	3434	
Alabama	Baldwin	195121	48.85	75.69	51074	8745	27317	3804	
Alabama	Barbour	26932	53.83	76.91	32959	6031	16824	2430	
Alabama	Bibb	22604	53.41	77.40	38887	5662	18431	3074	
Alabama	Blount	57710	49.41	73.38	46238	8696	20532	2052	
Alabama	Bullock	10678	53.01	75.45	33293	9000	17580	3111	

6 rows | 1-10 of 27 columns

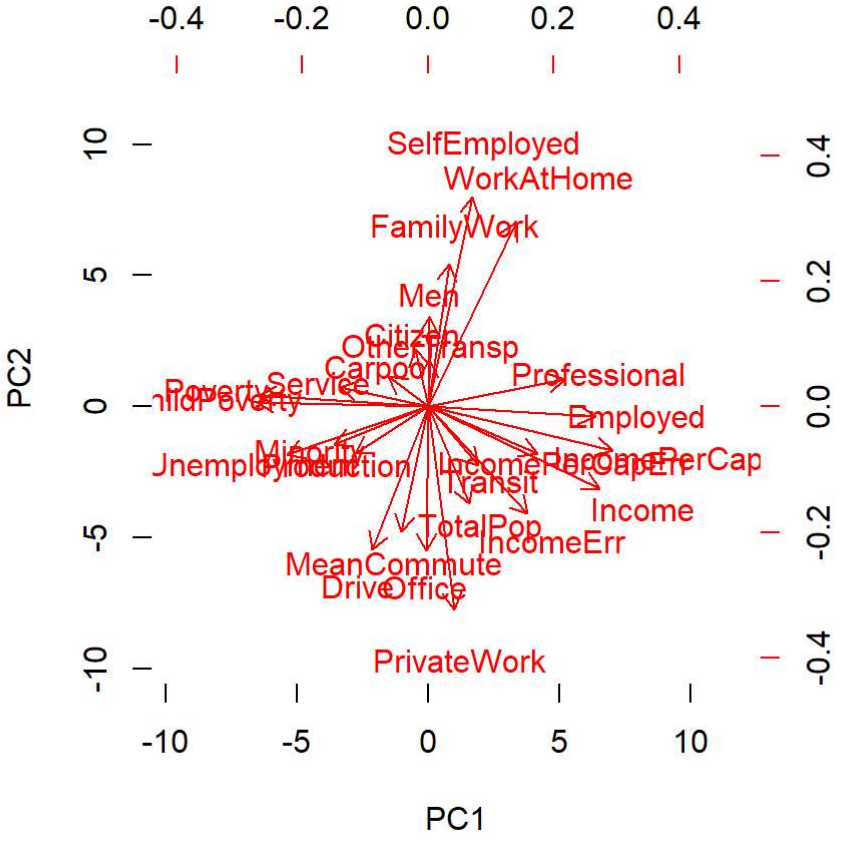
Dimensionality Reduction

12. Gathering principle components of census.ct data, saved in ct.pc data set.

```
ct.pc.temp=census.ct[,3:27]
row.names(ct.pc.temp)=paste(census.ct$County,census.ct$State)
ct.pc=prcomp(ct.pc.temp,scale=T)
subct.pc.temp=census.subct[,4:29]
row.names(subct.pc.temp)=paste(census.subct$County,census.subct$State,census.subct$CensusTract)
subct.pc=prcomp(subct.pc.temp,scale=T)
```

Comparing the loadings of the principal components.

```
biplot(ct.pc,scale=0,col=c(0,2))
```



We see employment related variables to be particularly impactful, as are poverty related variables. Ethnicity and Sex appear to be less impactul than expected.

Clustering

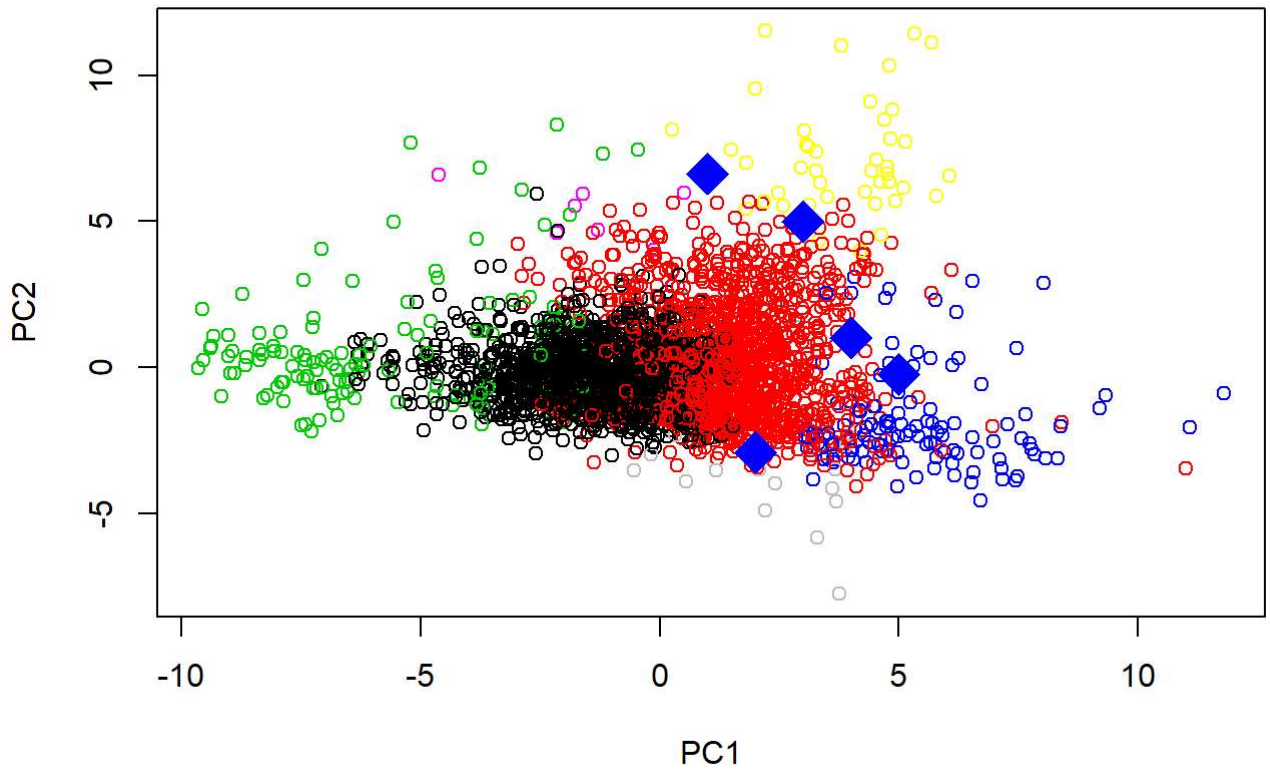
13. Preforming hierarchical clustering across 10 clusters using census.ct data and its first 5 PCs. Additionally comparing the clustering of California, San Mateo County.

```
census.ct.dist=dist(census.ct)
cen.ct.hc=hclust(census.ct.dist,method = "complete")
cen.ct.hc.cut=cutree(cen.ct.hc,k=10)
cen.ct.hc.cut[2575]
```

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

We observe San Mateo County to be in cluster 1 when we cluster census.ct directly.

```
ct.pc.5=ct.pc$x[,1:5]
dist.ct.pc.5=dist(ct.pc.5)
ct.hc=hclust(dist.ct.pc.5,method="complete")
ct.hc.cut=cutree(ct.hc,k=10)
plot(ct.pc.5,col=ct.hc.cut)
points(ct.pc.5["San Mateo California",1:5],pch=18,cex=3,col=ct.hc.cut["San Mateo California"])
```




```
ct.hc.cut["San Mateo California"]
```

```
## San Mateo California
##                    4
```

We observe San Mateo county to be clustered into cluster 4. This can be explained by the fact that the first 5 PCs do not account for enough of the variance explained in the data set.

```
summary(ct.pc)$importance[3,]
```

```
##    PC1    PC2    PC3    PC4    PC5    PC6    PC7    PC8    PC9   PC10
## 0.2572 0.3949 0.5131 0.5808 0.6287 0.6748 0.7181 0.7546 0.7882 0.8210
##   PC11   PC12   PC13   PC14   PC15   PC16   PC17   PC18   PC19   PC20
## 0.8515 0.8763 0.8982 0.9174 0.9331 0.9463 0.9590 0.9702 0.9786 0.9852
##   PC21   PC22   PC23   PC24   PC25
## 0.9906 0.9945 0.9970 0.9988 1.0000
```

In fact they only account for <63% of all variance explained. Thus leading to quite drastic misclassification errors.

```
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
tmpcensus$State=tolower(tmpcensus$State)
tmpcensus$County=tolower(tmpcensus$County)

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))
```

```
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,]
```

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

```
calc_error_rate = function(predicted.value, true.value){
  return(mean(true.value!=predicted.value))
}
records = matrix(NA, nrow=5, ncol=2)
colnames(records) = c("train.error","test.error")
rownames(records) = c("tree","knn","lda","log","qda")
```

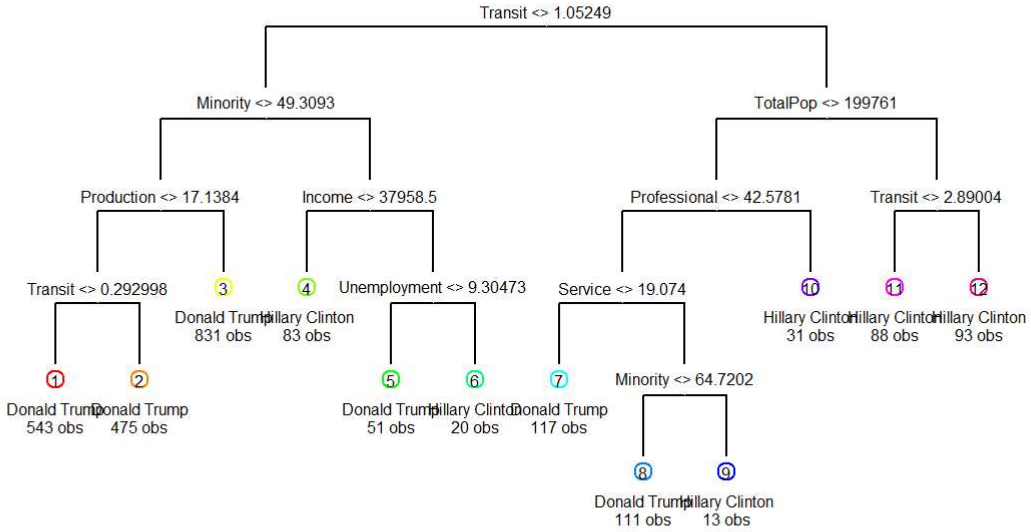
Classification: Native Attributes

13.2. Creating a decision tree trained by the trn.cl. Cross-Validated across 10 folds, and pruned to minimize test error.

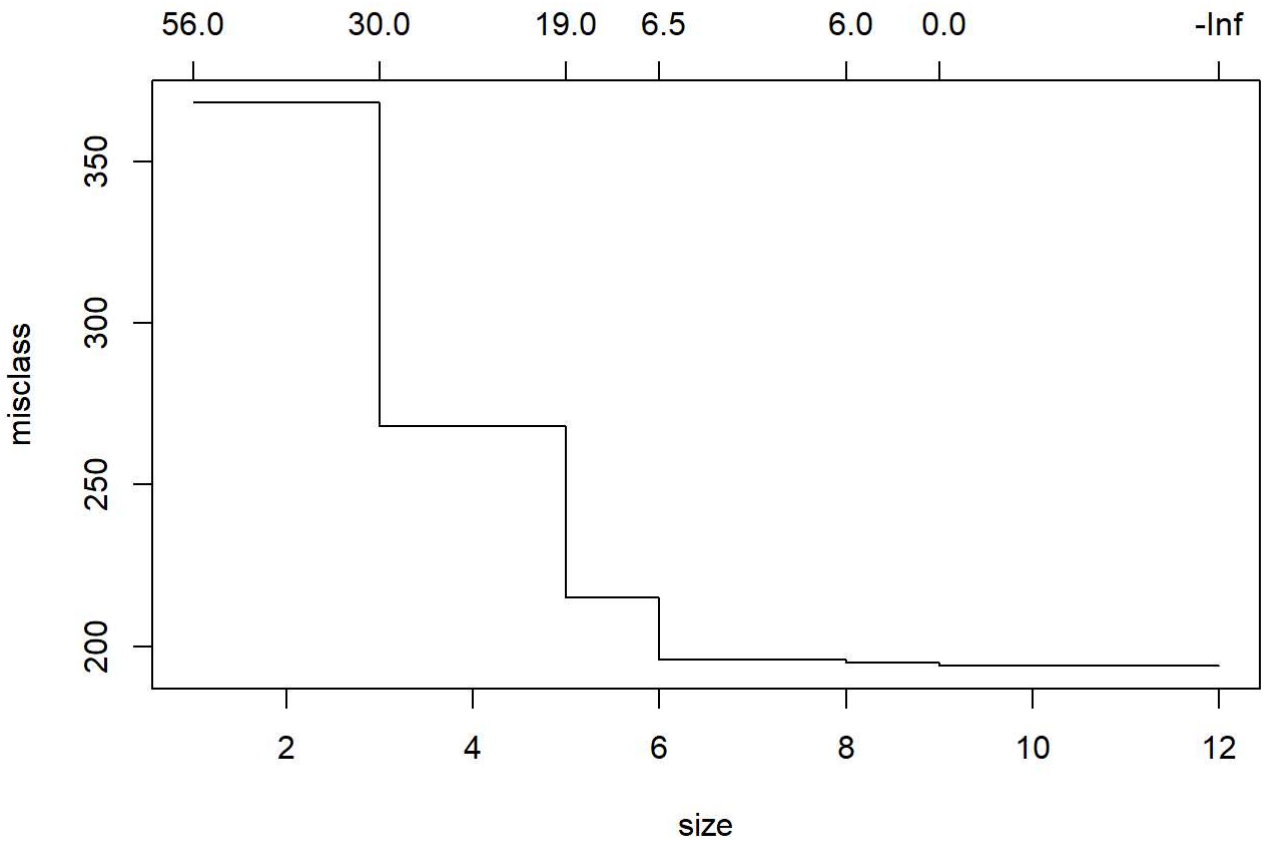
```
tcont=tree.control(nobs=nrow(trn.cl))
election.tree=tree(candidate~.,data=trn.cl,method="class",control=tcont)
summary(election.tree)
```

```
##
## Classification tree:
## tree(formula = candidate ~ ., data = trn.cl, control = tcont,
##     method = "class")
## Variables actually used in tree construction:
## [1] "Transit"      "Minority"     "Production"   "Income"
## [5] "Unemployment" "TotalPop"     "Professional" "Service"
## Number of terminal nodes:  12
## Residual mean deviance:  0.361 = 883 / 2440
## Misclassification error rate: 0.0651 = 160 / 2456
```

```
draw.tree(election.tree,cex=.5)
```



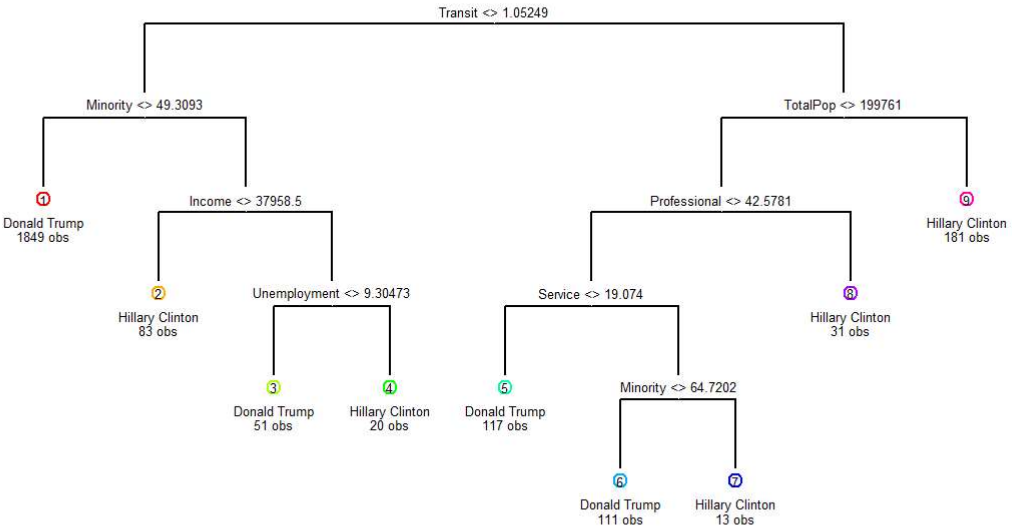

```
cv.election.tree=cv.tree(election.tree,FUN=prune.misclass,rand=folds,K=10)
plot(cv.election.tree)
```




```
best.size.cv=min(cv.election.tree$size[which(cv.election.tree$dev==min(cv.election.tree$dev))])
election.tree.prune=prune.tree(election.tree,best = best.size.cv,method = "misclass")
summary(election.tree.prune)
```

```
##
## Classification tree:
## snip.tree(tree = election.tree, nodes = c(4L, 7L))
## Variables actually used in tree construction:
## [1] "Transit"      "Minority"     "Income"       "Unemployment"
## [5] "TotalPop"     "Professional" "Service"
## Number of terminal nodes:  9
## Residual mean deviance:  0.404 = 987 / 2450
## Misclassification error rate: 0.0651 = 160 / 2456
```

```
draw.tree(election.tree.prune,cex=.4)
```




```
tree.train=predict(election.tree.prune,trn.cl,type="class")
tree.test=predict(election.tree.prune,tst.cl,type="class")
records[1,1]=calc_error_rate(tree.train,trn.cl$candidate)
records[1,2]=calc_error_rate(tree.test,tst.cl$candidate)
records
```

```
##      train.error test.error
## tree     0.06515    0.08306
## knn           NA         NA
## lda           NA         NA
## log           NA         NA
## qda           NA         NA
```

We observe a training and test error of .065 and .083 respectively when data is trained to fit a decision tree.

Classification: Native Attributes

14. Preforming a 10-fold CV for K-nearest neighbors heirarchical clustering.

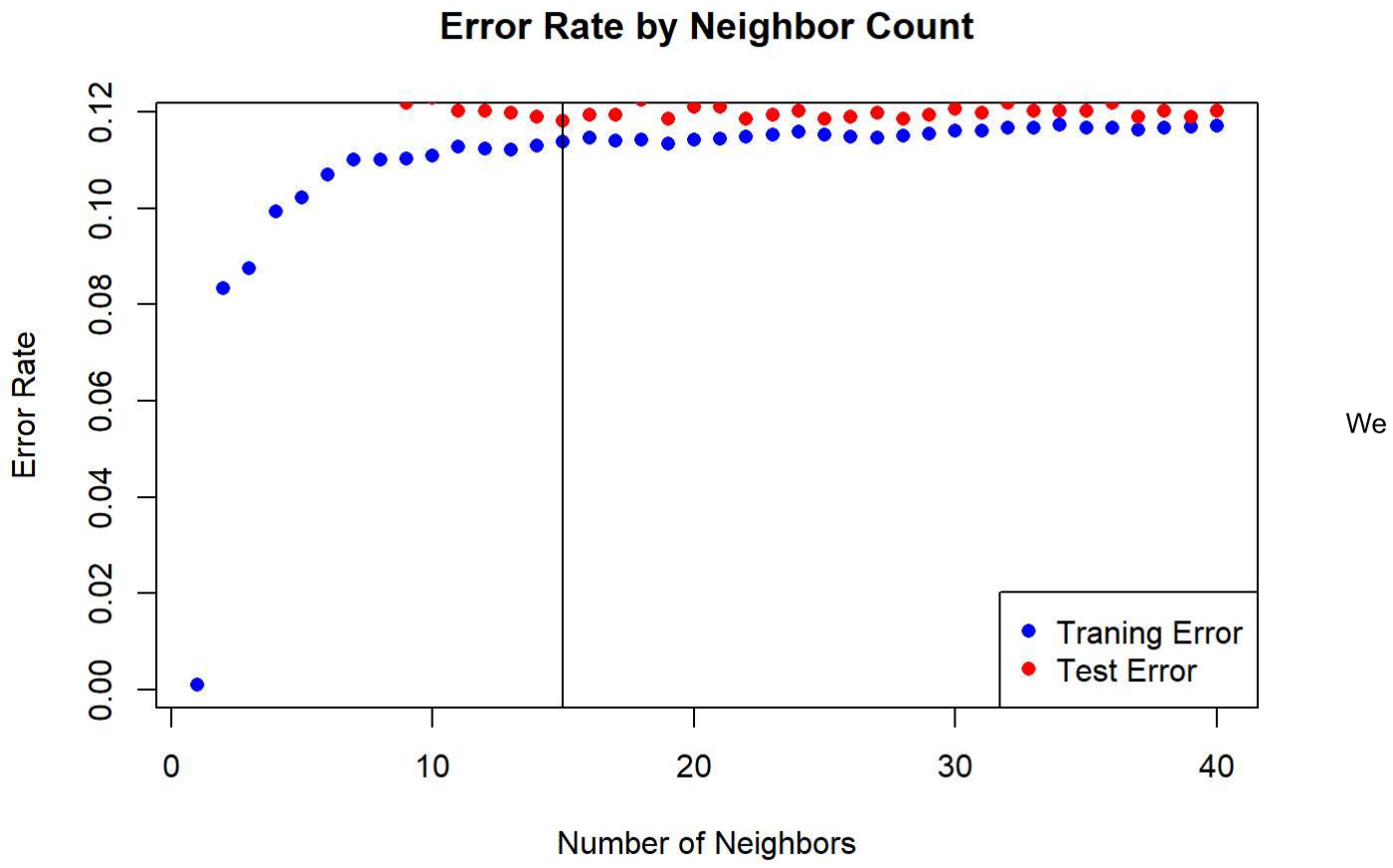
```
kvec = seq(1,40)
error.mat=matrix(c(rep(0,80)),nrow=40,ncol=2)
colnames(error.mat)=c("Training Error","Test Error")
rownames(error.mat)=kvec
for(j in 1:40){
  error=ldply(1:10,do.chunk,folds,select(trn.cl,-candidate),trn.cl$candidate,k=kvec[j])
  error.mat[j,1]=mean(error$train.error)
  error.mat[j,2]=mean(error$val.error)
}
```

Computing optimal number for k achieved via cross-validation

```
#k value that gives minimum test error
best.k=kvec[match(min(error.mat[,2]),error.mat[,2])]
best.k
```

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

```
plot(error.mat[,1],xlab="Number of Neighbors", ylab="Error Rate",col="blue",pch=16,main="Error R
ate by Neighbor Count")
points(error.mat[,2],col="red",pch=16)
legend("bottomright",legend=c("Traning Error","Test Error"),pch=c(16,16),col=c("blue","red"))
abline(v=best.k)
```



We choose k=15 as CV tells us that it should lead to minimum test error.

```
knn.train=knn(train=trn.cl%>%select(-candidate),
              test=trn.cl%>%select(-candidate),
              cl=trn.cl$candidate,
              k=best.k)
knn.test=knn(train=trn.cl%>%select(-candidate),
              test=tst.cl%>%select(-candidate),
              cl=trn.cl$candidate,
              k=best.k)
```

```
records[2,1]=calc_error_rate(knn.train,trn.cl$candidate)
records[2,2]=calc_error_rate(knn.test,tst.cl$candidate)
records
```

```
##      train.error test.error
## tree     0.06515    0.08306
## knn      0.11319    0.12378
## lda           NA         NA
## log           NA         NA
## qda           NA         NA
```

Surprisingly we have comparitively high traing and test error when we opt to train a knn-model for heirarchical clustering.

Classification: Principal Components

15. Determining how many PCs are needed to account for at least 90% of observed variance.

```
trn.cl.pca=prcomp(~.-candidate,data=trn.cl,scale=T)
min(which(summary(trn.cl.pca)$importance[3,]>=.9))
```

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

We see that at least 14 principal components are needed to account for 90% of observed variance in the data set.

Classification: Principal Components

16. Computing PCs of training and test data to train predictive models. saved into datasets trn.pca and tst.pca.

```
election.pca=data.frame(candidate=election.cl$candidate, prcomp(~.-candidate,data=election.cl,sc
ale=T)$x[,1:15])
set.seed(10)
n.pca = nrow(election.pca)
in.trn= sample.int(n.pca, 0.8*n.pca)
trn.pca = election.pca[ in.trn,]
tst.pca = election.pca[-in.trn,]
```

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

```
pca.records = matrix(NA, nrow=5, ncol=2)
colnames(pca.records) = c("train.error","test.error")
rownames(pca.records) = c("tree","knn","lda","log","qda")
```

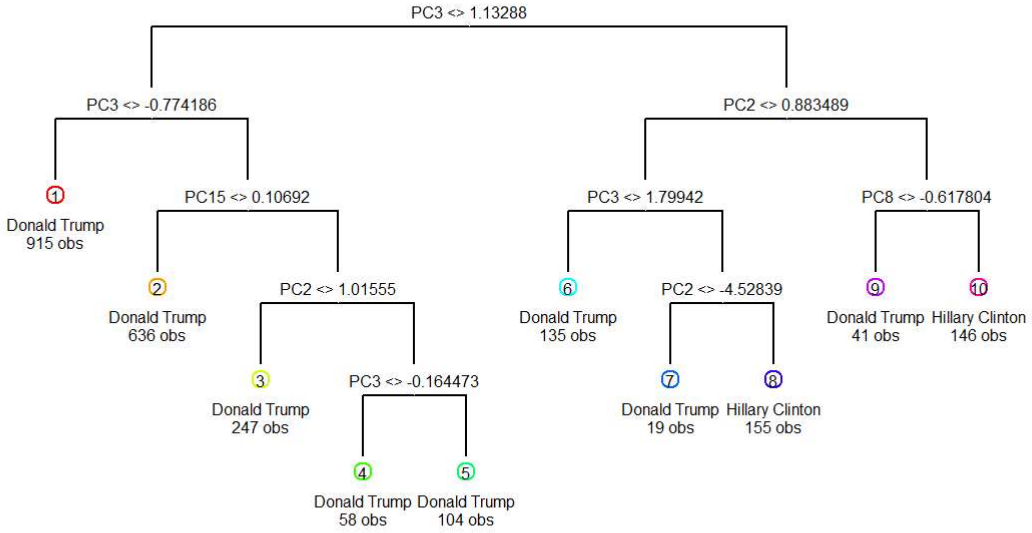
Classification: Principal Components

17. Training, cross-validating, and plotting a decision tree using PCs of training data.

```
tcont.pca=tree.control(nobs=nrow(trn.pca))
election.tree.pca=tree(candidate~.,data=trn.pca,method="class",control=tcont.pca)
summary(election.tree.pca)
```

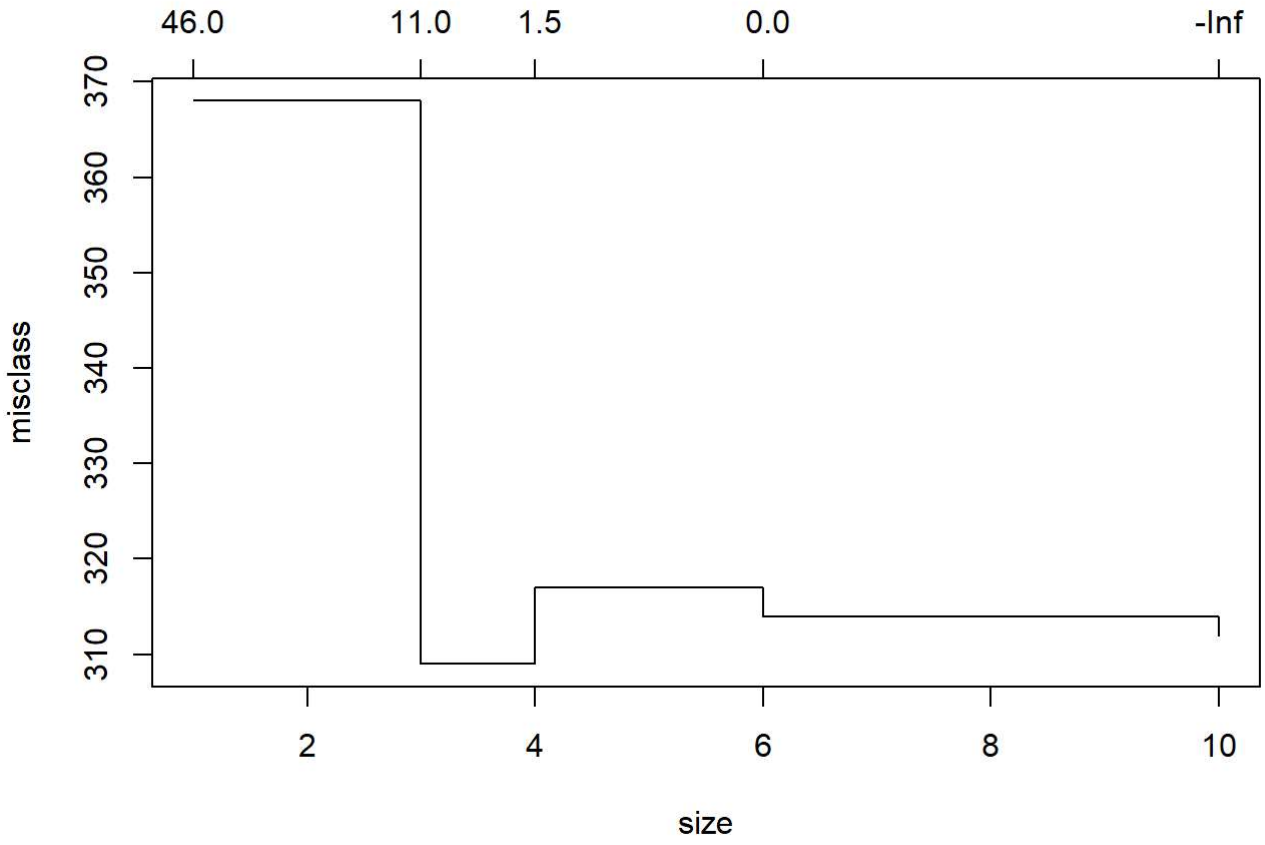
```
##
## Classification tree:
## tree(formula = candidate ~ ., data = trn.pca, control = tcont.pca,
##     method = "class")
## Variables actually used in tree construction:
## [1] "PC3"  "PC15" "PC2"  "PC8"
## Number of terminal nodes:  10
## Residual mean deviance:  0.519 = 1270 / 2450
## Misclassification error rate: 0.107 = 263 / 2456
```

```
draw.tree(election.tree.pca,cex=.5)
```



Pruning then plotting PCA trained decision tree to minimalize test error.

```
cv.election.tree.pca=cv.tree(election.tree.pca,FUN=prune.misclass,rand=folds.pca,K=10)
plot(cv.election.tree.pca)
```



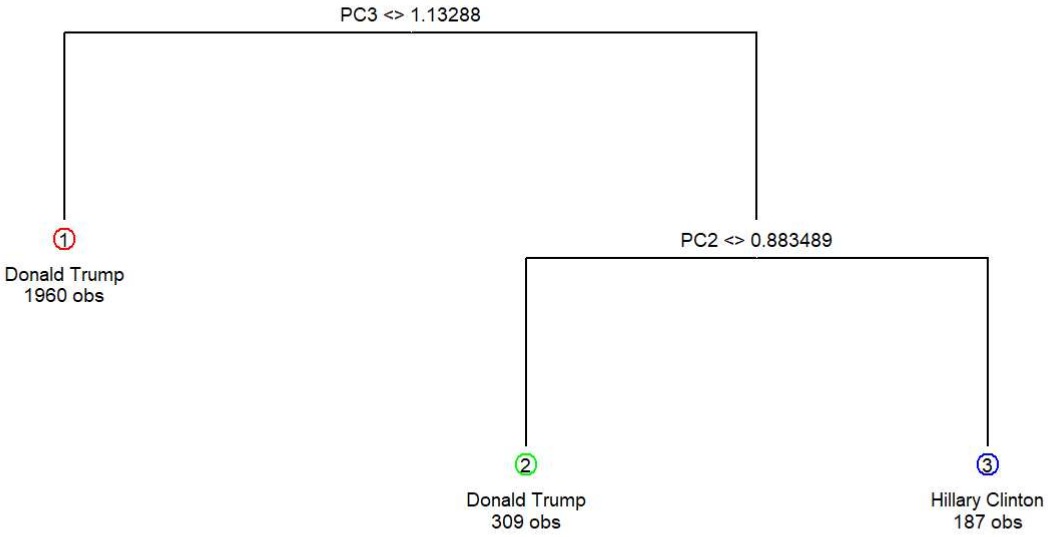

```
best.size.cv.pca=min(cv.election.tree.pca$size[which(cv.election.tree.pca$dev==min(cv.election.t
ree.pca$dev))])
best.size.cv.pca
```

```
## [1] 3
```

```
election.tree.prune.pca=prune.tree(election.tree.pca,best = best.size.cv.pca,method = "misclass"
)
summary(election.tree.prune.pca)
```

```
##
## Classification tree:
## snip.tree(tree = election.tree.pca, nodes = c(2L, 6L, 7L))
## Variables actually used in tree construction:
## [1] "PC3" "PC2"
## Number of terminal nodes:  3
## Residual mean deviance:  0.638 = 1560 / 2450
## Misclassification error rate: 0.113 = 277 / 2456
```

```
draw.tree(election.tree.prune.pca,cex=.6)
```




```
tree.train.pca=predict(election.tree.prune.pca,trn.pca,type="class")
tree.test.pca=predict(election.tree.prune.pca,tst.pca,type="class")
pca.records[1,1]=calc_error_rate(tree.train.pca,trn.pca$candidate)
pca.records[1,2]=calc_error_rate(tree.test.pca,tst.pca$candidate)
pca.records
```

```
##      train.error test.error
## tree      0.1128      0.114
## knn           NA         NA
## lda           NA         NA
## log           NA         NA
## qda           NA         NA
```

We observe a traing and test eror of 11.27% and 11.4%, respective, when training a decision tree with PCAs.

Classification: Principal Components

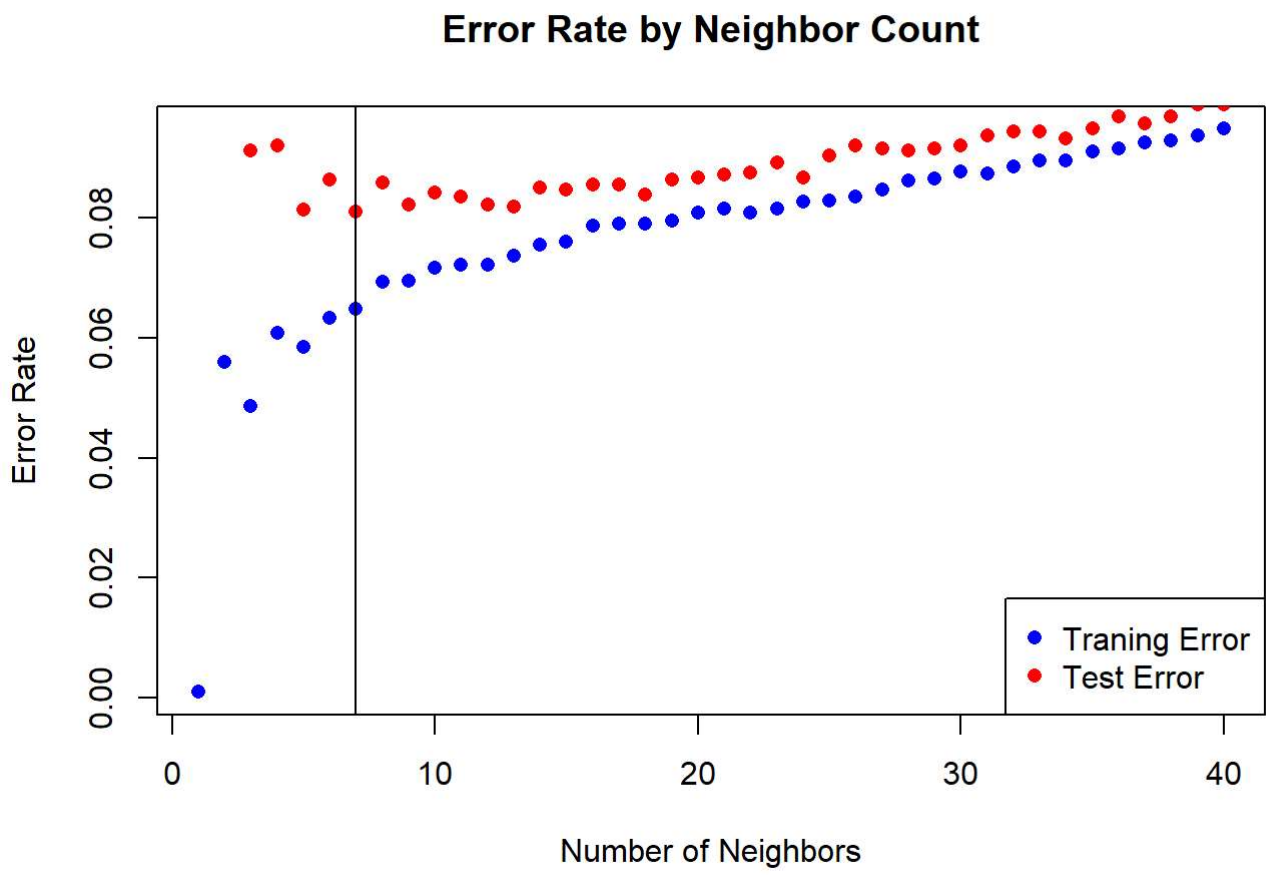
18. training a knn model using priciple components.

```
kvec.pca = seq(1,40)
error.mat.pca=matrix(c(rep(0,80)),nrow=40,ncol=2)
colnames(error.mat.pca)=c("Training Error","Test Error")
rownames(error.mat.pca)=kvec.pca
for(j in 1:40){
  error=ldply(1:10,do.chunk.pca,folds.pca,select(trn.pca,-candidate),trn.pca$candidate,k=kvec.pc
a[j])
  error.mat.pca[j,1]=mean(error$train.error)
  error.mat.pca[j,2]=mean(error$val.error)
}
```

```
#k value that gives minimum test error
best.k.pca=kvec.pca[match(min(error.mat.pca[,2]),error.mat.pca[,2])]
best.k.pca
```

```
## [1] 7
```

```
plot(error.mat.pca[,1],xlab="Number of Neighbors", ylab="Error Rate",col="blue",pch=16,main="Err
or Rate by Neighbor Count")
points(error.mat.pca[,2],col="red",pch=16)
legend("bottomright",legend=c("Traning Error","Test Error"),pch=c(16,16),col=c("blue","red"))
abline(v=best.k.pca)
```



```
knn.train.pca=knn(train=trn.pca%>%select(-candidate),
              test=trn.pca%>%select(-candidate),
              cl=trn.pca$candidate,
              k=best.k.pca)
knn.test.pca=knn(train=trn.pca%>%select(-candidate),
              test=tst.pca%>%select(-candidate),
              cl=trn.pca$candidate,
              k=best.k.pca)
```

```
pca.records[2,1]=calc_error_rate(knn.train.pca,trn.pca$candidate)
pca.records[2,2]=calc_error_rate(knn.test.pca,tst.pca$candidate)
pca.records
```

```
##      train.error test.error
## tree      0.1128    0.11401
## knn       0.0623    0.09772
## lda           NA         NA
## log           NA         NA
## qda           NA         NA
```

We observe a respective training and test error of 6.2% and 9.8% when we train a knn-model with PCAs. # Interpretation & Discussion ## 19. Insights into the Project During this project we created several prediction models to best describe the 2016 election results on a county level. Although the statistical justification behind the methods used is sound, the goal of accurately predicting the presidential nomination using models similar to these is quite a challenge. Firstly, there were a number of predictor variables which could have yielded more accurate predictions at the county level. Measurements such as median age, median education level, and opinion based statistics regarding political topics could have proven to be much more important than the percentage of carpools in a county. Secondly, county winners do not translate directly to state winners. The Electoral College of the United States minimizes the voter participation importance of a fair number of states, playcing all real weight on a handful of swing states. Thus focusing the analysis to the most impactful swing states would be a more efficient way to predict the presidential nomination. To be fair, our models are strong enoough to accurately predict county winners in a given state. However, if the goal is to predict the presidential nomination we must weigh individual counties according to their importance to the electoral college. As many liberals as there are in Austin, Texas, its contribution to the electoral college is practically zero due to the fact that Texas, at least for the foreseable future, will always vote red.

Taking It Further

20. Preforming LDA, QDA, and Logistic Regression

```
library(MASS)
```

Setting factor levels to a binary level.

```
trn.cl$candidate=factor(trn.cl$candidate)
tst.cl$candidate=factor(tst.cl$candidate)
trn.pca$candidate=factor(trn.pca$candidate)
tst.pca$candidate=factor(tst.pca$candidate)
```

Preforming LDA with “raw” data. Calculating error rates for LDA model.

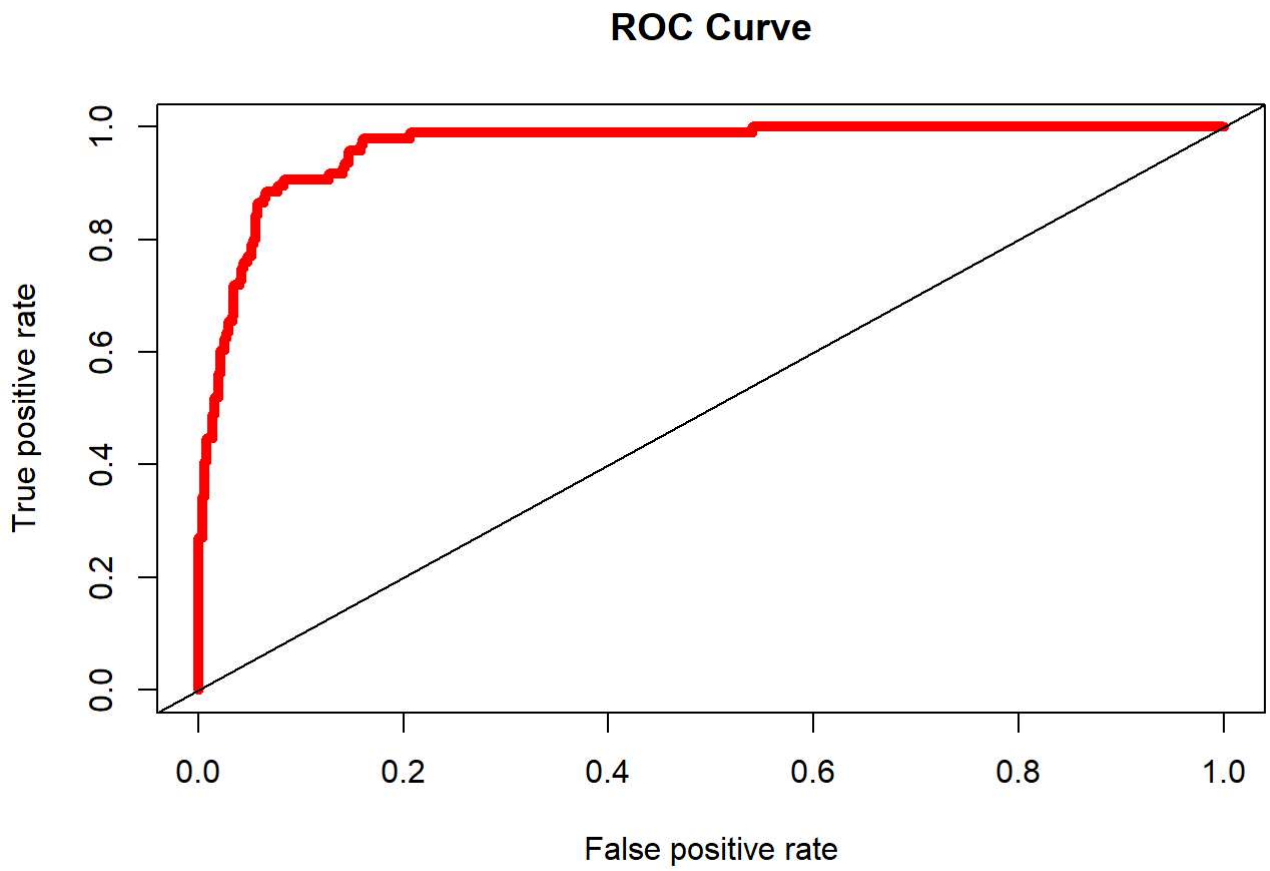
```
lda.cl=lda(candidate~.,data=trn.cl)
lda.cl.pred.trn=predict(lda.cl,newdata=trn.cl)
lda.cl.pred.tst=predict(lda.cl,newdata=tst.cl)
records[3,1]=calc_error_rate(lda.cl.pred.trn$class,trn.cl$candidate)
records[3,2]=calc_error_rate(lda.cl.pred.tst$class,tst.cl$candidate)
records
```

```
##      train.error test.error
## tree     0.06515    0.08306
## knn      0.11319    0.12378
## lda      0.06718    0.07980
## log           NA         NA
## qda           NA         NA
```

We observe a respective training and test error of 6.7% and 7.9% when we train a model with raw data.

Plotting ROC curve and computing AUC for LDA model.

```
pred.cl.lda=prediction(lda.cl.pred.tst$posterior[,2],tst.cl$candidate)
pref.cl.lda=performance(pred.cl.lda,measure = "tpr",x.measure="fpr")
plot(pref.cl.lda,col=2,main="ROC Curve",lwd=5)
abline(0,1)
```




```
auc.lda.cl=performance(pred.cl.lda,measure = "auc")@y.values[[1]]
auc.lda.cl
```

```
## [1] 0.9638
```

Ploting the ROC curve we see that the prediction model achieves an AUC score of of ~.964, making it more than satisfacotry for prediction purposes.

Training logstic model with raw data, setting classification threshold to .5. Reason being, when calculated to “optimal” value we achieved a test error of over 50%. As such we decided on the binary decision boundry for our classifications.

```
election.glm=glm(candidate~.,data=trn.cl,family = binomial())
summary(election.glm)
```

```
##
## Call:
## glm(formula = candidate ~ ., family = binomial(), data = trn.cl)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -3.975  -0.276  -0.119  -0.045   3.533
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)     -3.22e+01   7.53e+00   -4.28  1.9e-05 ***
## TotalPop         5.01e-07   4.23e-07    1.18  0.23686
## Men              7.68e-02   5.39e-02    1.42  0.15424
## Citizen          9.85e-02   3.06e-02    3.22  0.00128 **
## Income          -6.73e-05   2.74e-05   -2.45  0.01418 *
## IncomeErr       -5.40e-05   6.25e-05   -0.86  0.38731
## IncomePerCap     2.58e-04   6.73e-05    3.83  0.00013 ***
## IncomePerCapErr -2.82e-04   1.31e-04   -2.15  0.03122 *
## Poverty          1.70e-02   4.13e-02    0.41  0.68067
## ChildPoverty    -1.53e-04   2.54e-02   -0.01  0.99520
## Professional     2.82e-01   3.99e-02    7.07  1.5e-12 ***
## Service          3.72e-01   4.97e-02    7.49  6.7e-14 ***
## Office           1.12e-01   4.84e-02    2.31  0.02067 *
## Production       1.85e-01   4.33e-02    4.28  1.8e-05 ***
## Drive           -2.45e-01   5.34e-02   -4.59  4.5e-06 ***
## Carpool         -2.24e-01   6.59e-02   -3.39  0.00069 ***
## Transit          1.94e-02   1.01e-01    0.19  0.84847
## OtherTransp     -9.81e-02   1.00e-01   -0.98  0.32706
## WorkAtHome      -2.06e-01   7.86e-02   -2.62  0.00889 **
## MeanCommute      5.83e-02   2.48e-02    2.35  0.01883 *
## Employed         1.59e-01   3.26e-02    4.87  1.1e-06 ***
## PrivateWork      7.50e-02   2.16e-02    3.46  0.00053 ***
## SelfEmployed     2.33e-04   4.66e-02    0.00  0.99601
## FamilyWork      -1.15e+00   4.05e-01   -2.84  0.00452 **
## Unemployment     1.83e-01   3.84e-02    4.77  1.8e-06 ***
## Minority         1.23e-01   9.29e-03   13.22  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2074.96  on 2455  degrees of freedom
## Residual deviance:  862.18  on 2430  degrees of freedom
## AIC: 914.2
##
## Number of Fisher Scoring iterations: 7
```

```
election.glm.train=(predict(election.glm,trn.cl,type="response"))
election.glm.test=(predict(election.glm,tst.cl,type="response"))
elec.glm.train.pred=trn.cl%>%
  mutate(predcand=(ifelse(election.glm.train<=.5,"Donald Trump","Hillary Clinton")))
elec.glm.test.pred=tst.cl%>%
  mutate(predcand=(ifelse(election.glm.test<=.5,"Donald Trump","Hillary Clinton")))
```

Calculating error rates for logistic model.

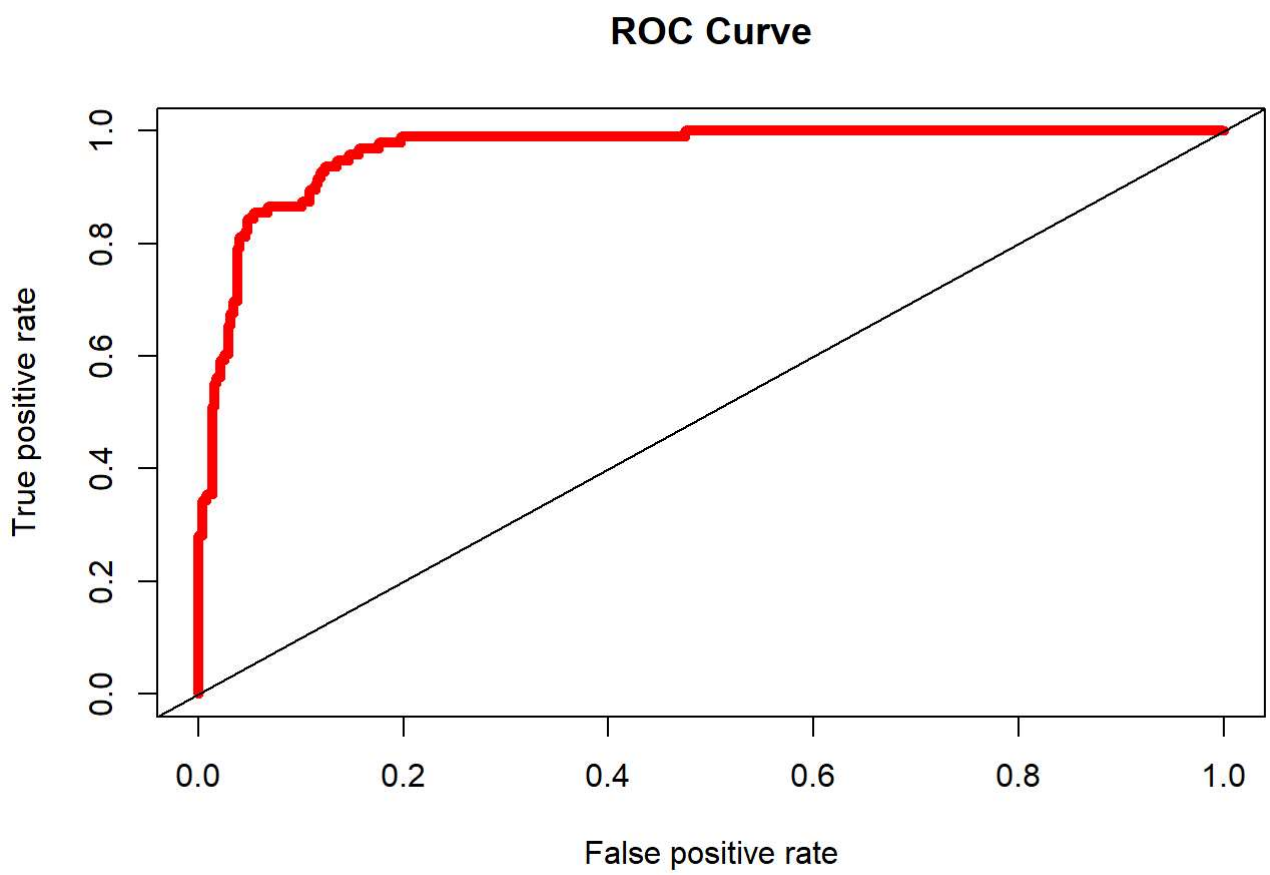
```
records[4,1]=calc_error_rate(elec.glm.train.pred$predcand,trn.cl$candidate)
records[4,2]=calc_error_rate(elec.glm.test.pred$predcand,tst.cl$candidate)
records
```

```
##      train.error test.error
## tree     0.06515    0.08306
## knn      0.11319    0.12378
## lda      0.06718    0.07980
## log      0.06596    0.07818
## qda           NA         NA
```

The logistic model has a respective training and test error of 6.59% and 7.81%.

Plotting and evaluating ROC curve for logistic model.

```
pred.cl.glm=prediction(election.glm.test,tst.cl$candidate)
pref.cl.glm=performance(pred.cl.glm,measure = "tpr",x.measure="fpr")
plot(pref.cl.glm,col=2,main="ROC Curve",lwd=5)
abline(0,1)
```




```
auc.glm.cl=performance(pred.cl.glm,measure = "auc")@y.values[[1]]
auc.glm.cl
```

```
## [1] 0.9641
```

Ploting the ROC curve we see that the prediction model achieves an AUC score of of ~.964, putting it slightly better with the LDA model for prediction purposes.

Training QDA prediction model with raw data, and computing error rates.

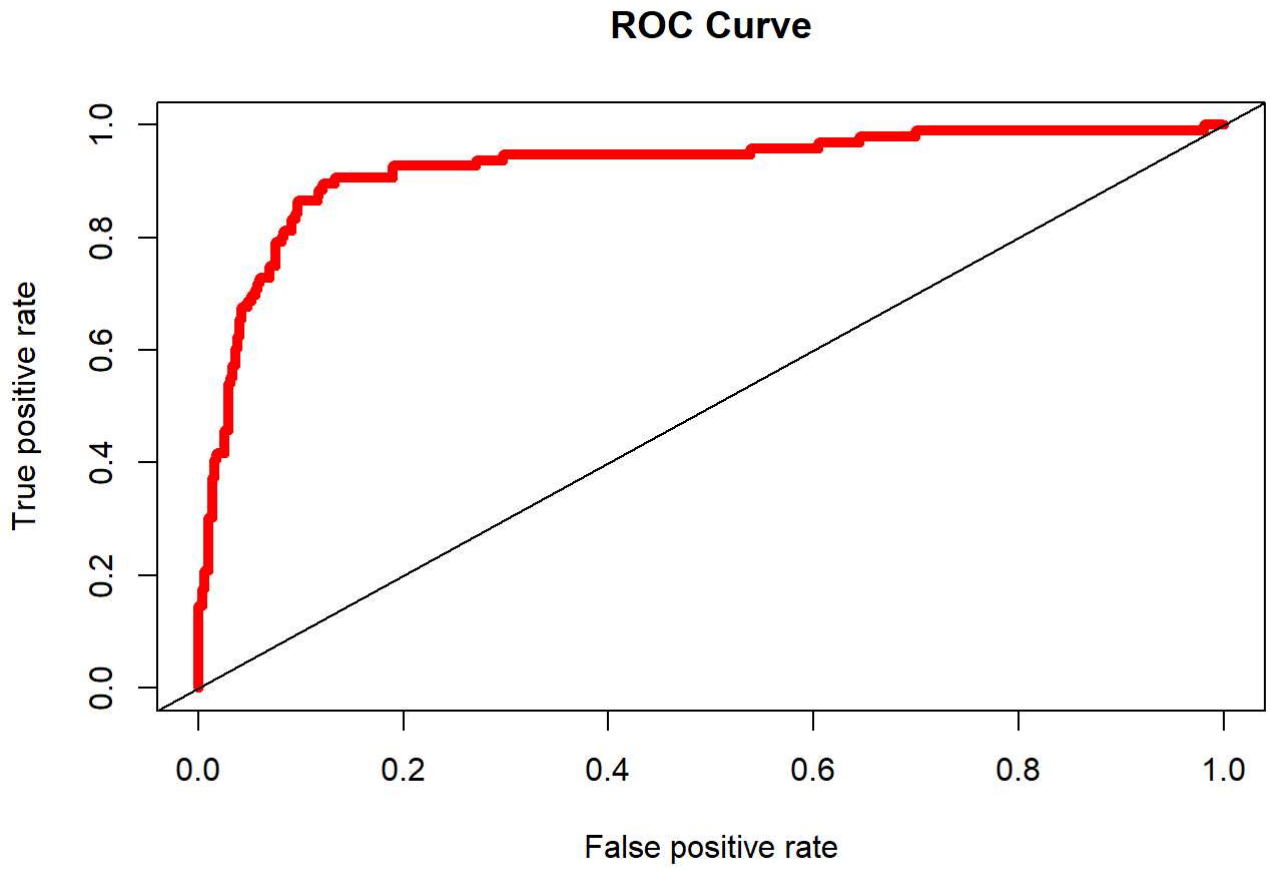
```
qda.cl=qda(candidate~.,data=trn.cl)
qda.cl.pred.trn=predict(qda.cl,newdata=trn.cl)
qda.cl.pred.tst=predict(qda.cl,newdata=tst.cl)
records[5,1]=calc_error_rate(qda.cl.pred.trn$class,trn.pca$candidate)
records[5,2]=calc_error_rate(qda.cl.pred.tst$class,tst.pca$candidate)
records
```

```
##      train.error test.error
## tree     0.06515    0.08306
## knn      0.11319    0.12378
## lda      0.06718    0.07980
## log      0.06596    0.07818
## qda      0.07410    0.08958
```

The QDA model achieves a respective training and test error of 7.41% and 8.96%.

Plotting ROC curve and calculating AUC

```
pred.cl.qda=prediction(qda.cl.pred.tst$posterior[,2],tst.pca$candidate)
pref.cl.qda=performance(pred.cl.qda,measure = "tpr",x.measure="fpr")
plot(pref.cl.qda,col=2,main="ROC Curve",lwd=5)
abline(0,1)
```




```
auc.cl.qda=performance(pred.cl.qda,measure = "auc")@y.values[[1]]
auc.cl.qda
```

```
## [1] 0.9235
```

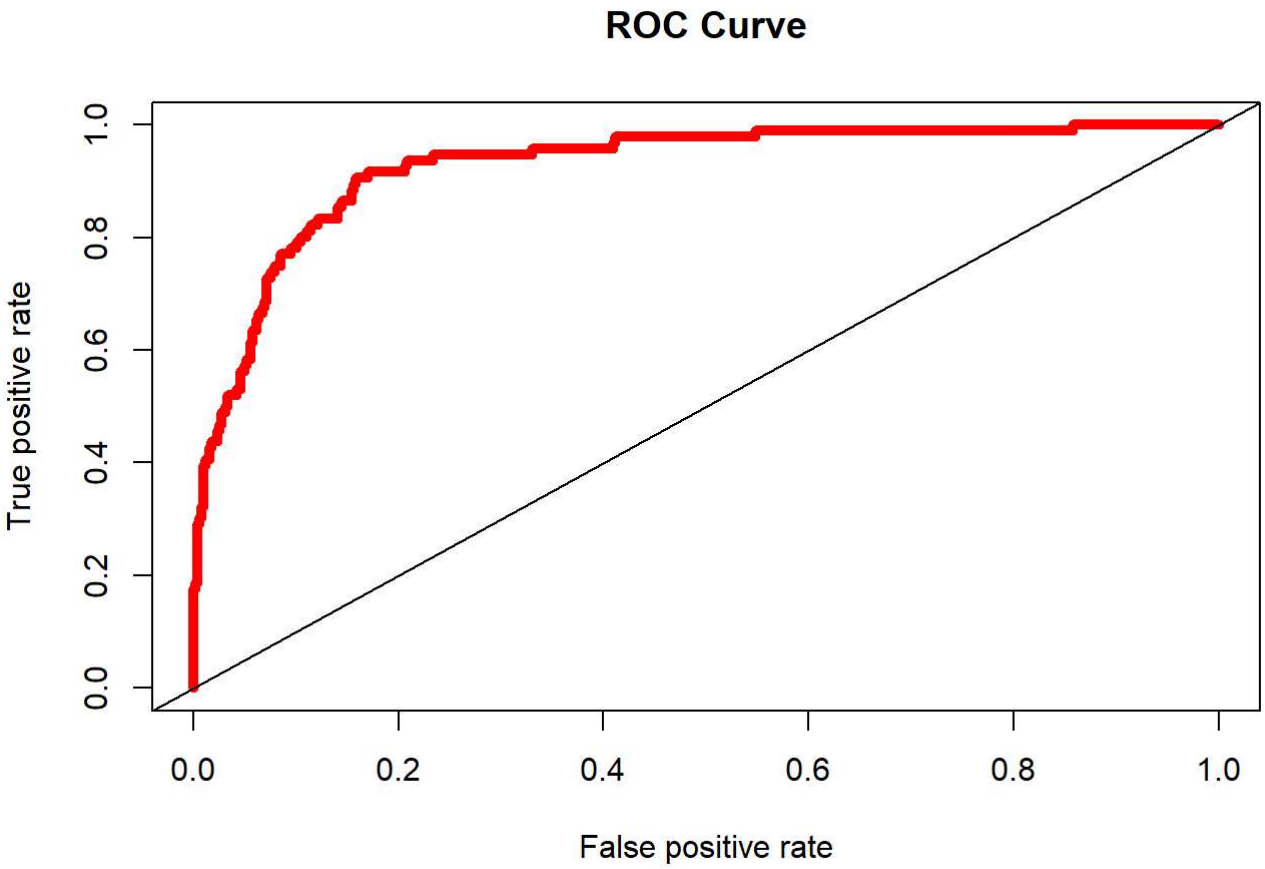
From the ROC curve we see that the QDA model preforms predictions adequatley with a AUC of ~.91. However this does not match up to the logistic or LDA models.

Preforming LDA with principlal components.

```
lda.pca=lda(candidate~.,data=trn.pca)
lda.pca.pred.trn=predict(lda.pca,newdata=trn.pca)
lda.pca.pred.tst=predict(lda.pca,newdata=tst.pca)
pca.records[3,1]=calc_error_rate(lda.pca.pred.trn$class,trn.pca$candidate)
pca.records[3,2]=calc_error_rate(lda.pca.pred.tst$class,tst.pca$candidate)
pca.records
```

```
##      train.error test.error
## tree     0.11279    0.11401
## knn      0.06230    0.09772
## lda      0.08673    0.10423
## log           NA         NA
## qda           NA         NA
```

```
pred.pca.lda=prediction(lda.pca.pred.tst$posterior[,2],tst.pca$candidate)
pref.pca.lda=performance(pred.pca.lda,measure = "tpr",x.measure="fpr")
plot(pref.pca.lda,col=2,main="ROC Curve",lwd=5)
abline(0,1)
```




```
auc.lda.pca=performance(pred.pca.lda,measure = "auc")@y.values
auc.lda.pca
```

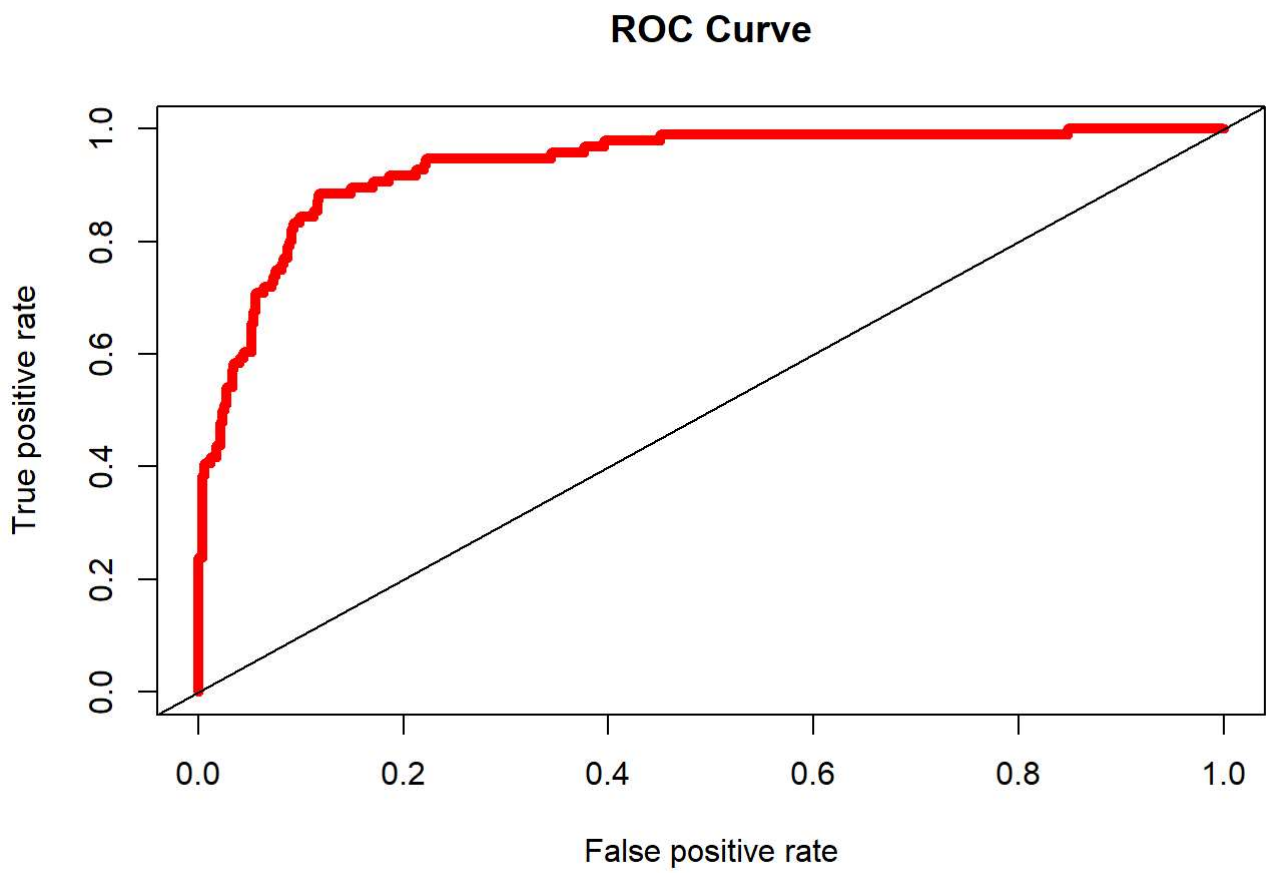
```
## [[1]]
## [1] 0.9273
```

```
election.glm.pca=glm(candidate~.,data=trn.pca,family = binomial)
summary(election.glm.pca)
```

```
##
## Call:
## glm(formula = candidate ~ ., family = binomial, data = trn.pca)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -4.449  -0.369  -0.199  -0.098   3.329
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -2.8070     0.1227  -22.88  < 2e-16 ***
## PC1          -0.1145     0.0322   -3.56  0.00038 ***
## PC2           0.6567     0.0640   10.26  < 2e-16 ***
## PC3           1.1491     0.0682   16.86  < 2e-16 ***
## PC4           0.1676     0.0685    2.45  0.01435 *
## PC5          -0.4715     0.1045   -4.51  6.4e-06 ***
## PC6           0.2145     0.0805    2.66  0.00770 **
## PC7           0.3041     0.0871    3.49  0.00048 ***
## PC8           0.5748     0.1069    5.38  7.6e-08 ***
## PC9           0.3023     0.0945    3.20  0.00138 **
## PC10         -0.7024     0.1281   -5.48  4.2e-08 ***
## PC11         -0.1887     0.1408   -1.34  0.18005
## PC12         -0.4155     0.1241   -3.35  0.00081 ***
## PC13          0.0385     0.1465    0.26  0.79281
## PC14         -0.0299     0.1162   -0.26  0.79658
## PC15          0.6648     0.1372    4.84  1.3e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2075.0  on 2455  degrees of freedom
## Residual deviance: 1123.4  on 2440  degrees of freedom
## AIC: 1155
##
## Number of Fisher Scoring iterations: 7
```

```
election.glm.train.pca=(predict(election.glm.pca,trn.pca,type="response"))
election.glm.test.pca=(predict(election.glm.pca,tst.pca,type="response"))
elec.glm.train.pred.pca=trn.pca%>%
  mutate(predcand=(ifelse(election.glm.train.pca<=.5,"Donald Trump","Hillary Clinton")))
elec.glm.test.pred.pca=tst.pca%>%
  mutate(predcand=(ifelse(election.glm.test.pca<=.5,"Donald Trump","Hillary Clinton")))
```

```
pred.pca.glm=prediction(election.glm.test.pca,tst.cl$candidate)
pref.pca.glm=performance(pred.pca.glm,measure = "tpr",x.measure="fpr")
plot(pref.pca.glm,col=2,main="ROC Curve",lwd=5)
abline(0,1)
```




```
auc.glm.pca=performance(pred.pca.glm,measure = "auc")@y.values
auc.glm.pca
```

```
## [[1]]
## [1] 0.9352
```

Calculating error rate for pca trained logistic model

```
pca.records[4,1]=calc_error_rate(elec.glm.train.pred.pca$predcand,trn.pca$candidate)
pca.records[4,2]=calc_error_rate(elec.glm.test.pred.pca$predcand,tst.pca$candidate)
pca.records
```

```
##      train.error test.error
## tree     0.11279    0.11401
## knn      0.06230    0.09772
## lda      0.08673    0.10423
## log      0.08225    0.09772
## qda           NA         NA
```

When trained with PCAs the respective training and test error of the logistic model were 8.22% and 9.77%.

Training QDA model with PCAs and computing training/test errors.

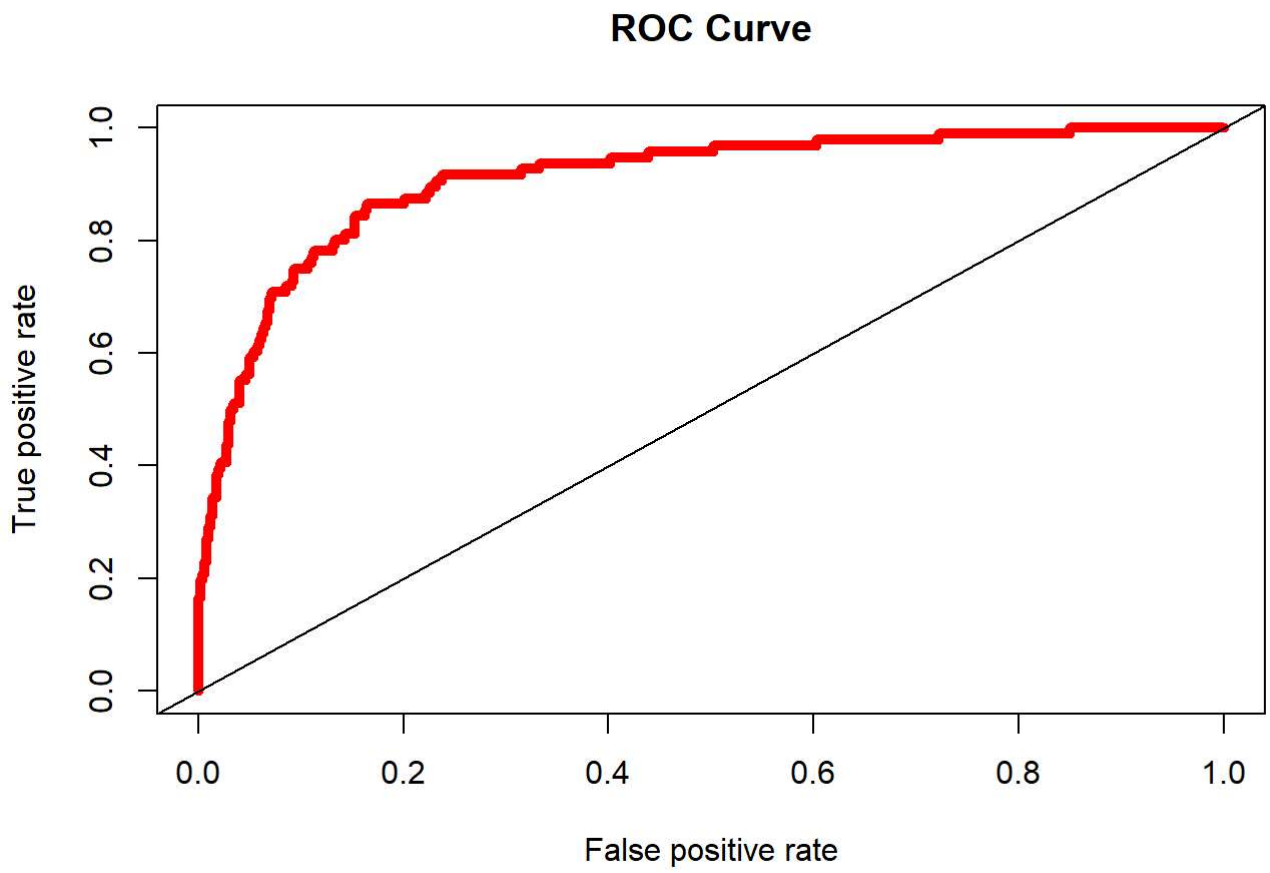
```
qda.pca=qda(candidate~.,data=trn.pca)
qda.pca.pred.trn=predict(qda.pca,newdata=trn.pca)
qda.pca.pred.tst=predict(qda.pca,newdata=tst.pca)
pca.records[5,1]=calc_error_rate(qda.pca.pred.trn$class,trn.pca$candidate)
pca.records[5,2]=calc_error_rate(qda.pca.pred.tst$class,tst.pca$candidate)
pca.records
```

```
##      train.error test.error
## tree     0.11279    0.11401
## knn      0.06230    0.09772
## lda      0.08673    0.10423
## log      0.08225    0.09772
## qda      0.09487    0.10912
```

When trained with PCAs the QDA model has respective training and test error of the logistic model were 9.48% and 10.9%.

Plotting ROC curve for PCA trained QDA model.

```
pred.pca.qda=prediction(qda.pca.pred.tst$posterior[,2],tst.pca$candidate)
pref.pca.qda=performance(pred.pca.qda,measure = "tpr",x.measure="fpr")
plot(pref.pca.qda,col=2,main="ROC Curve",lwd=5)
abline(0,1)
```



```
auc.pca.qda=performance(pred.pca.qda,measure = "auc")@y.values[[1]]
auc.pca.qda
```

```
## [1] 0.9087
```

From the ROC curve and AUC score of ~.91 we can say that the model is capable of making sound predictions but it is not the best one we've computed.

From the above models error rates shown below

```
#Clustered Raw Data trained model error rates
records
```

```
##      train.error test.error
## tree     0.06515    0.08306
## knn      0.11319    0.12378
## lda      0.06718    0.07980
## log      0.06596    0.07818
## qda      0.07410    0.08958
```

```
#PCA Data trained model error rates
pca.records
```

```
##      train.error test.error
## tree     0.11279    0.11401
## knn      0.06230    0.09772
## lda      0.08673    0.10423
## log      0.08225    0.09772
## qda      0.09487    0.10912
```

We can see that the model that gave the lowest test error was the logistic model trained under the raw, clustered data. From this we can conclude our report by stating that if we want to predict county level winners based off of these statistics, we should use a logistic regression model based off of raw data measurements.

```
detach("package:MASS", unload=TRUE)
```