2016 Election Prediction

Wenxuan Luo, 5016308 (PSTAT 131)

5/21/2019

Predicting voter behavior is complicated for many reasons despite the tremendous effort in collecting, analyzing, and understanding many available datasets. For our final project, we will analyze the 2016 presidential election dataset, but, first, some background.

Background

The presidential election in 2012 did not come as a surprise. Some correctly predicted the outcome of the election correctly including Nate Silver, and many speculated his approach.

Despite the success in 2012, the 2016 presidential election came as a big surprise to many, and it was a clear example that even the current state-of-the-art technology can surprise us.

Answer the following questions in one paragraph for each.

1. What makes voter behavior prediction (and thus election forecasting) a hard problem? Human is a thoughtful animal. There are many factors that determine a person's choice. First of all, from the perspective of candidates, the candidates' political position, political leadership and speaking ability determine the voters' behavior. Secondly, from the perspective of voters, the performance of candidates during their presidency and whether they bring benefits to themselves will determine voters' election behavior. Many determinants determine the difficulty of voter forecasting.
2. What was unique to Nate Silver's approach in 2012 that allowed him to achieve good predictions? Nate Silver's initial idea was to aggregate the results of several survey companies' sub-state surveys (such as Barack Obama's 36.7 support rate in Alabama) and calculate the error rate based on historical data (such as Alabama's 3.8, which is controversial, because Nate refused to publish the weighted method), multiplying by the number of members of the state's electoral Corps (such as Alabama's 9). This is to defeat the individual with collective wisdom.
3. What went wrong in 2016? What do you think should be done to make future predictions better? Voter data predicted that Hillary Clinton would win the election, but Donald Trump won. To make future predictions better, we should consider more factors, get more voter data, and train better prediction models.

Data

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

Election data

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

county	fips	candidate	state	votes
NA	US	Donald Trump	US	62984825
NA	US	Hillary Clinton	US	65853516

county	fips	candidate	state	votes
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
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

CensusTract	Census.tract.ID	numeric
Service	% employed in service jobs	numeric
Office	% employed in sales and office jobs	numeric
Construction	% employed in natural resources, construction, and maintenance	numeric
Production	% employed in production, transportation, and material movement	numeric
Drive	% commuting alone in a car, van, or truck	numeric
Carpool	% carpooling in a car, van, or truck	numeric
Transit	% commuting on public transportation	numeric
Walk	% walking to work	numeric
OtherTransp	% commuting via other means	numeric
WorkAtHome	% working at home	numeric
MeanCommute	Mean commute time (minutes)	numeric
Employed	% employed (16+)	numeric
PrivateWork	% employed in private industry	numeric
PublicWork	% employed in public jobs	numeric
SelfEmployed	% self-employed	numeric
FamilyWork	% in unpaid family work	numeric
Unemployment	% unemployed	numeric

Data wrangling

4. Remove summary rows from `election.raw` data: i.e.,
    - Federal-level summary into a `election_federal`.
    - State-level summary into a `election_state`.
    - Only county-level data is to be in `election`.

```
election_federal <- election.raw %>% filter(fips == 'US')

election.raw.fix <- election.raw %>% filter( fips != 'US')

election_state <- election.raw.fix %>% filter(is.na(county) == T)

election <- election.raw.fix %>% filter(is.na(county) == F)
```

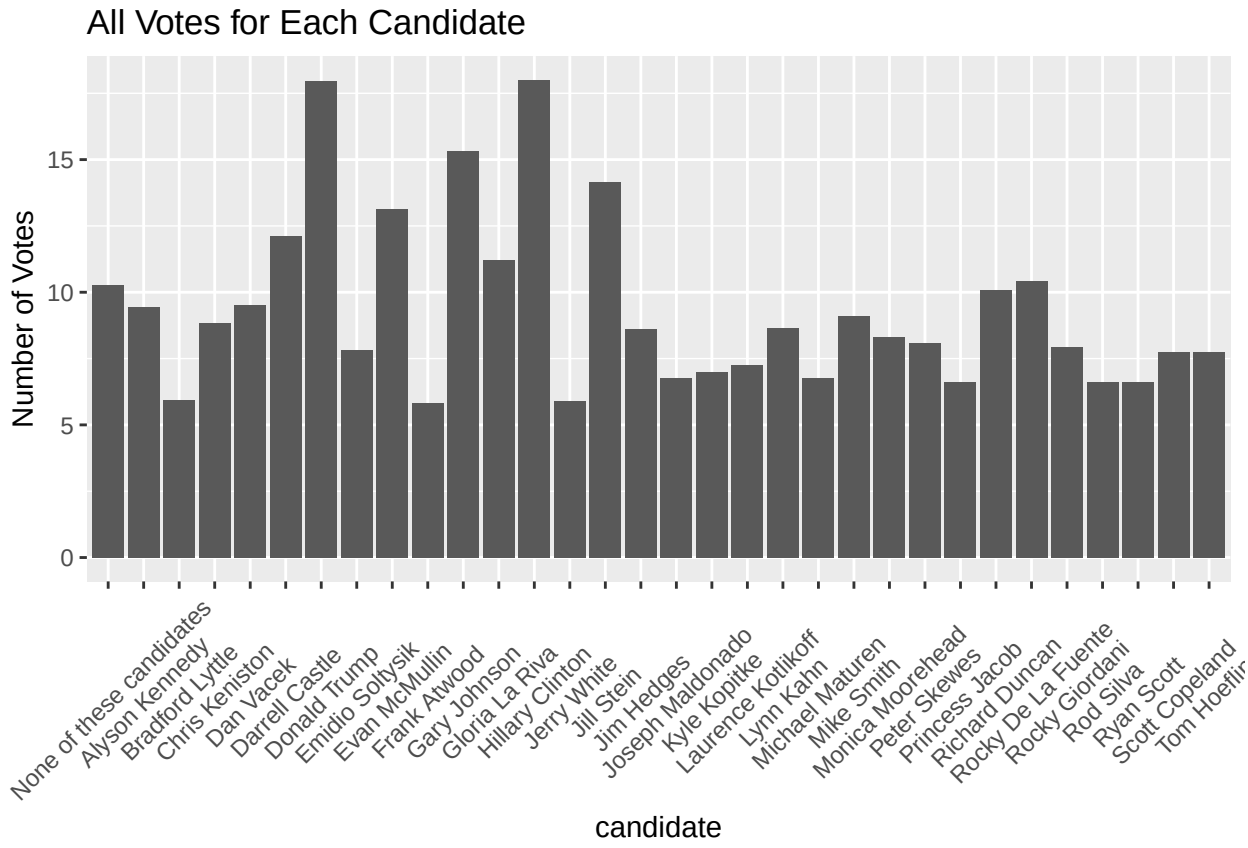
5. How many named presidential candidates were there in the 2016 election? Draw a bar chart of all votes received by each candidate

```
#31
levels(election.raw$candidate)
```

```
##  [1] " None of these candidates" "Alyson Kennedy"
##  [3] "Bradford Lyttle"           "Chris Keniston"
##  [5] "Dan Vacek"                 "Darrell Castle"
##  [7] "Donald Trump"              "Emidio Soltysik"
##  [9] "Evan McMullin"             "Frank Atwood"
## [11] "Gary Johnson"              "Gloria La Riva"
## [13] "Hillary Clinton"           "Jerry White"
## [15] "Jill Stein"                "Jim Hedges"
## [17] "Joseph Maldonado"          "Kyle Kopitke"
## [19] "Laurence Kotlikoff"        "Lynn Kahn"
## [21] "Michael Maturen"           "Mike Smith"
## [23] "Monica Moorehead"          "Peter Skewes"
## [25] "Princess Jacob"            "Richard Duncan"
```

```
## [27] "Rocky De La Fuente"        "Rocky Giordani"
## [29] "Rod Silva"                 "Ryan Scott"
## [31] "Scott Copeland"            "Tom Hoefling"
```

```
names <-unique(election_federal$candidate)
votingnmbrs <-length(names)
ggplot(data=election_federal, aes(x=candidate))+geom_bar(aes(weight=log(votes)))+theme(axis.text.x=eleme
```



Answer:31 presidential candidates were there in the 2016 election.

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

```
county_winner <- election %>% group_by(fips) %>% mutate(pct = votes/sum(votes))%>% top_n(1)
```

```
## Selecting by pct
```

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

```
## Selecting by pct
```

```
kable(county_winner %>% head)
```

county	fips	candidate	state	votes	pct
Los Angeles County	6037	Hillary Clinton	CA	2464364	0.7202514
Cook County	17031	Hillary Clinton	IL	1611946	0.7475189

county	fips	candidate	state	votes	pct
Maricopa County	4013	Donald Trump	AZ	747361	0.4863279
Harris County	48201	Hillary Clinton	TX	707914	0.5422825
San Diego County	6073	Hillary Clinton	CA	735476	0.5696604
Orange County	6059	Hillary Clinton	CA	609961	0.5142130

```
kable(state_winner %>% head)
```

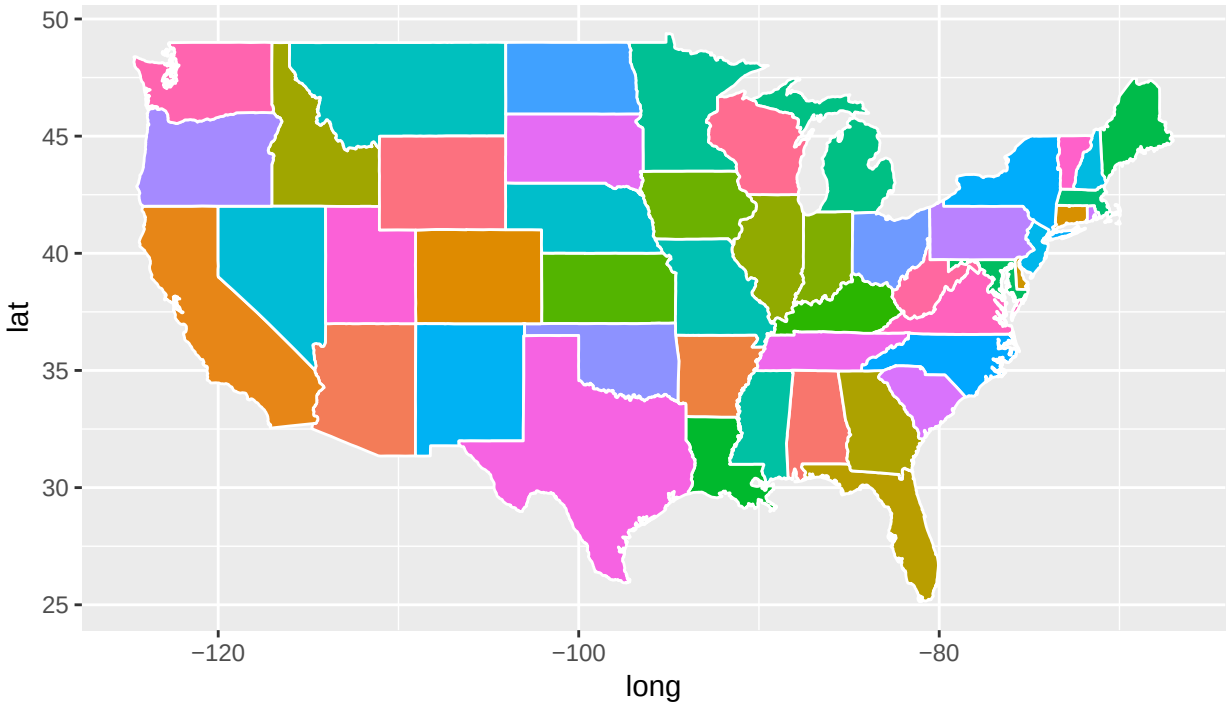
county	fips	candidate	state	votes	pct
NA	CA	Hillary Clinton	CA	8753788	0.6225644
NA	FL	Donald Trump	FL	4617886	0.4902274
NA	TX	Donald Trump	TX	4685047	0.5253493
NA	NY	Hillary Clinton	NY	4556124	0.5947795
NA	PA	Donald Trump	PA	2970733	0.4857789
NA	IL	Hillary Clinton	IL	3090729	0.5595962

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.

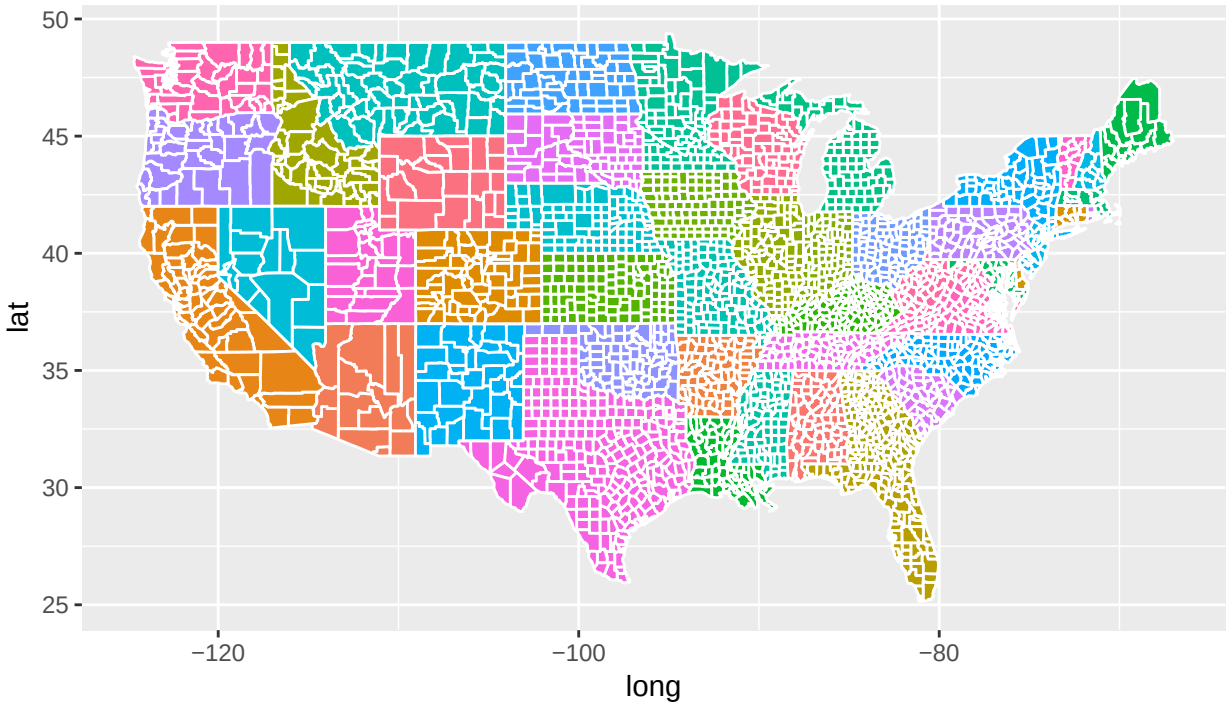
```
states = map_data("state")

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

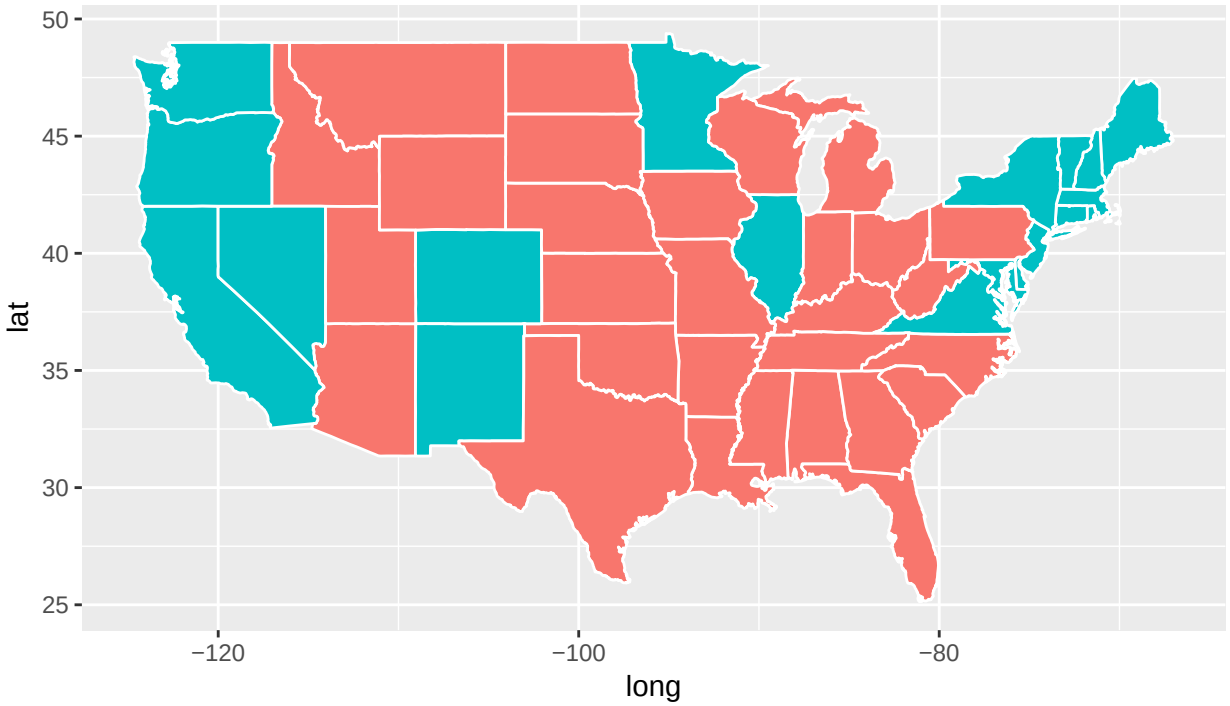


8. Now color the map by the winning candidate for each state. First, combine `states` variable and `state_winner` we created earlier using `left_join()`. Note that `left_join()` needs to match up values of states to join the tables; however, they are in different formats: e.g. `AZ` vs. `arizona`. Before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

```
states$fips <- state.abb[match(states$region,tolower(state.name))]
map_candidate <- left_join(states, state_winner, by = c("fips"="state"))
```

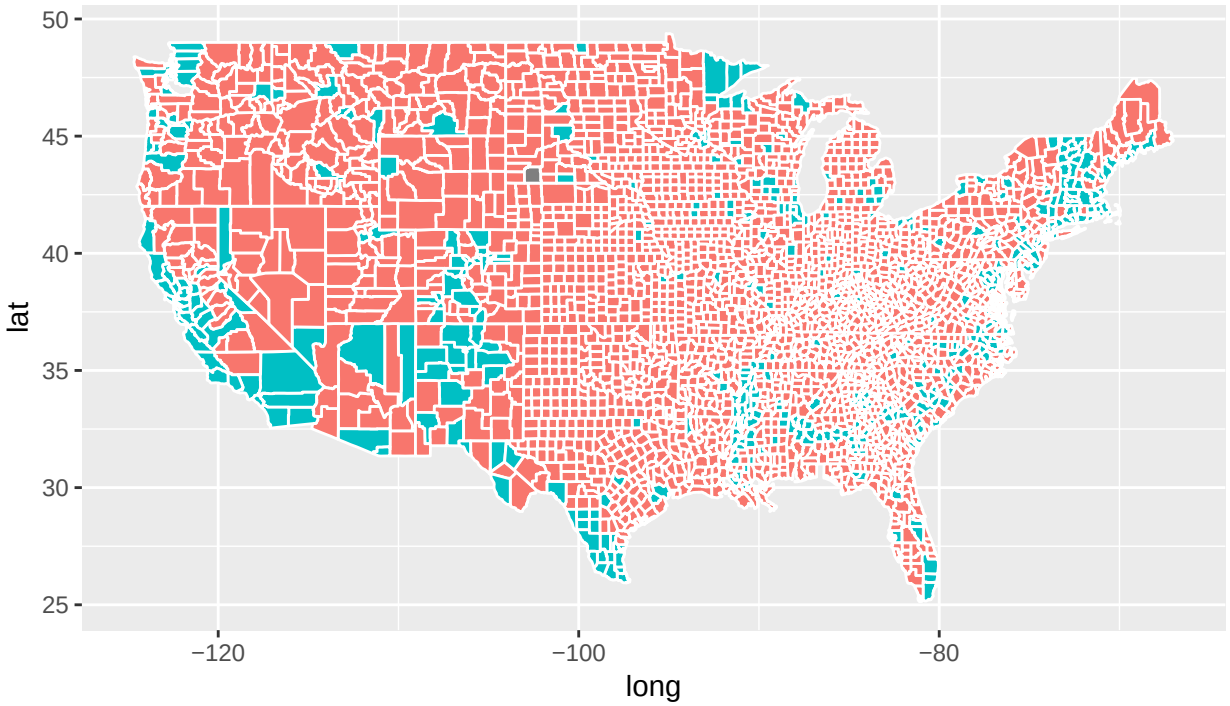
```
## Warning: Column `fips`/`state` joining character vector and factor,
## coercing into character vector
```

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



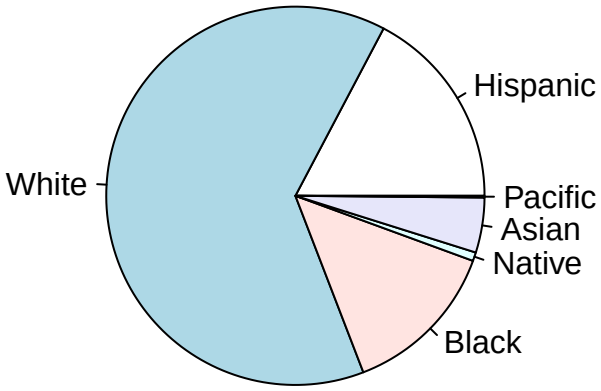
9. The variable `county` does not have `fips` column. So we will create one by pooling information from `maps::county.fips`. Split the `polyname` column to `region` and `subregion`. Use `left_join()` combine `county.fips` into `county`. Also, `left_join()` previously created variable `county_winner`.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

```
t <- str_split_fixed(county.fips$polyname, ",", 2)
county.fips$region <- t[,1]
county.fips$subregion <- t[,2]
county <- left_join(county.fips, counties, by = c("region"="region","subregion"="subregion"))
county$fips <- sapply(county$fips, as.character)
county_winner$fips <- sapply(county_winner$fips, as.character)
map_county_candidate <- left_join(county, county_winner, by = c("fips"="fips"))
ggplot(data = map_county_candidate) +
  geom_polygon(aes(x = long, y = lat, fill = candidate, group = group), color = "white") +
  coord_fixed(1.3) +
  guides(fill=FALSE)
```



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

```
census.tt = census%>% filter(complete.cases(.))
pie(c(sum(census.tt$Hispanic), sum(census.tt$White),sum(census.tt$Black),sum(census.tt$Native),sum(censu
    c('Hispanic','White','Black','Native','Asian','Pacific'))
```



Answer: According to the attached articles, deomogrphics played a hugh part in this election. So we wanted to visualize the voters of this election. As we can see, the white population has a big proportion in the plot, which could indicate the white population is a one of the influential factors in this election.

11. The `census` data contains high resolution information (more fine-grained than county-level).
    In this problem, we aggregate the information into county-level data by computing `TotalPop`-weighted average of each attributes for each county. Create the following variables:

    - *Clean census data* ***`census.del`***: start with `census`, filter out any rows with missing values, convert {`Men`, `Employed`, `Citizen`} attributes to a percentages (meta data seems to be inaccurate), compute `Minority` attribute by combining {Hispanic, Black, Native, Asian, Pacific}, remove {`Walk`, `PublicWork`, `Construction`}.
      Many columns seem to be related, and, if a set that adds up to 100%, one column will be deleted.
    - *Sub-county census data,* ***`census.subct`***: start with `census.del` from above, `group_by()` two attributes {`State`, `County`}, use `add_tally()` to compute `CountyTotal`. Also, compute the weight by `TotalPop/CountyTotal`.
    - *County census data,* ***`census.ct`***: start with `census.subct`, use `summarize_at()` to compute weighted sum
    - *Print few rows of* ***`census.ct`***:

```
census.del = census%>% filter(complete.cases(.)) %>%
  mutate(Men=Men/TotalPop, Employed=Employed/TotalPop, Citizen=Citizen/TotalPop, Minority=Hispanic+Blac
  select(-c(Walk,PublicWork,Construction,Women,Hispanic,Black,Asian,Pacific,Native))

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

```
  mutate(Weight=TotalPop/n)

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

head(census.ct)
```

```
## # A tibble: 6 x 27
## # Groups:   State [1]
##   State County   Men White Citizen Income IncomeErr IncomePerCap
##   <fct> <fct>  <dbl> <dbl>   <dbl>  <dbl>     <dbl>        <dbl>
## 1 Alab~ Autau~ 0.484  75.8   0.737 51696.     7771.       24974.
## 2 Alab~ Baldw~ 0.488  83.1   0.757 51074.     8745.       27317.
## 3 Alab~ Barbo~ 0.538  46.2   0.769 32959.     6031.       16824.
## 4 Alab~ Bibb   0.534  74.5   0.774 38887.     5662.       18431.
## 5 Alab~ Blount 0.494  87.9   0.734 46238.     8696.       20532.
## 6 Alab~ Bullo~ 0.530  22.2   0.755 33293.     9000.       17580.
## # ... with 19 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>
```

```
head(census.subct)
```

```
## # A tibble: 6 x 31
## # Groups:   State, County [1]
##   CensusTract State County TotalPop   Men White Citizen Income IncomeErr
##   <fct>       <fct> <fct>     <int> <dbl> <dbl>   <dbl>  <dbl>     <int>
## 1 1001020100  Alab~ Autau~     1948 0.483  87.4   0.772  61838     11900
## 2 1001020200  Alab~ Autau~     2156 0.491  40.4   0.771  32303     13538
## 3 1001020300  Alab~ Autau~     2968 0.460  74.5   0.787  44922      5629
## 4 1001020400  Alab~ Autau~     4423 0.491  82.8   0.747  54329      7003
## 5 1001020500  Alab~ Autau~    10763 0.457  68.5   0.712  51965      6935
## 6 1001020600  Alab~ Autau~     3851 0.464  72.9   0.686  63092      9585
## # ... with 22 more variables: IncomePerCap <int>, IncomePerCapErr <int>,
## #   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 <int>,
## #   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_all <- prcomp(census.ct[,c(-1,-2)], scale. = TRUE)
ct.pc <- as.data.frame(ct.pc_all$x)
subct.pc_all <- prcomp(census.subct[,c(-1,-2,-3)],scale. = TRUE)
subct.pc <- as.data.frame(subct.pc_all$x)
```

```
ct.pc_all$rotation[,1:2]
```

```
##                          PC1         PC2
## Men                0.006820384 -0.16885207
## White              0.223024821  0.04908398
## Citizen            0.004684404 -0.04142365
## Income             0.320035674  0.13514959
## IncomeErr          0.169870985  0.14221028
## IncomePerCap       0.351555497  0.06754490
## IncomePerCapErr    0.194524741  0.03675009
## Poverty           -0.342307260 -0.07890899
## ChildPoverty      -0.343599697 -0.05160648
## Professional       0.250167858 -0.11150901
## Service           -0.181798119 -0.09586453
## Office            -0.015012712  0.25827588
## Production        -0.118766799  0.18684505
## Drive             -0.094148342  0.36344855
## Carpool           -0.076707928 -0.06564062
## Transit            0.070751017  0.01415007
## OtherTransp       -0.009670695 -0.15308289
## WorkAtHome         0.174889954 -0.37455420
## MeanCommute       -0.058606311  0.22777312
## Employed           0.327837529  0.04410613
## PrivateWork        0.056472415  0.44676537
## SelfEmployed       0.097757106 -0.39825161
## FamilyWork         0.048659860 -0.27473130
## Unemployment      -0.290698344  0.02906326
## Minority          -0.226530572 -0.04636384
```

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

```
##                          PC1          PC2
## TotalPop          -0.03240180  0.011637594
## Men               -0.01730084 -0.049451734
## White             -0.24042491 -0.308539676
## Citizen           -0.16084392 -0.230723964
## Income            -0.30251017  0.155642964
## IncomeErr         -0.19894423  0.222428849
## IncomePerCap      -0.31811986  0.167908325
## IncomePerCapErr   -0.21233534  0.196687738
## Poverty            0.30468855  0.051427811
## ChildPoverty       0.29788665  0.027234819
## Professional      -0.30643659  0.141649910
## Service            0.26882989  0.057629761
## Office             0.01383053 -0.040557958
## Production         0.20682037 -0.192718992
## Drive             -0.07893438 -0.377206032
## Carpool            0.16257632 -0.039890691
## Transit            0.05730847  0.393709073
## OtherTransp        0.04514007  0.133109612
## WorkAtHome        -0.17296089  0.088284469
## MeanCommute       -0.01001204  0.278924637
## Employed          -0.22121053  0.060732007
## PrivateWork        0.04201520  0.052226242
```

```
## SelfEmployed    -0.06972090  0.009883318
## FamilyWork      -0.01518467 -0.044001196
## Unemployment     0.25281744  0.076980036
## Minority         0.24202363  0.305445246
## n                0.02151015  0.283842961
## Weight           0.01194567 -0.228952315
```

```
pc1.ct <- head(sort(abs(ct.pc_all$rotation[,1]),decreasing = TRUE),n=1)
pc2.ct <- head(sort(abs(ct.pc_all$rotation[,2]),decreasing = TRUE),n=1)

pc1.subct <- head(sort(abs(subct.pc_all$rotation[,1]),decreasing = TRUE),n=1)
pc2.subct <- head(sort(abs(subct.pc_all$rotation[,2]),decreasing = TRUE),n=1)

pc1.ct
```

```
## IncomePerCap
##    0.3515555
```

```
pc2.ct
```

```
## PrivateWork
##   0.4467654
```

```
pc1.subct
```

```
## IncomePerCap
##    0.3181199
```

```
pc2.subct
```

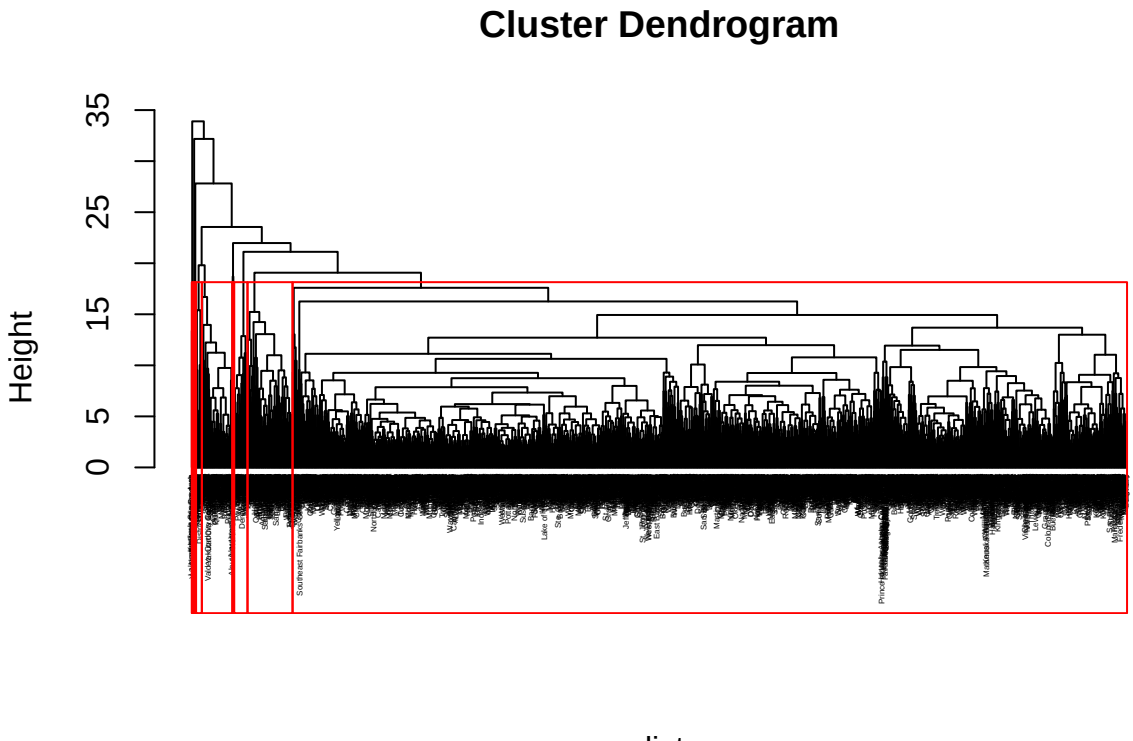
```
##   Transit
## 0.3937091
```

Answer: For county level data, the first two principle components PC1 and PC2 are Per capita income and PrivateWork. For sub-county level data, the first two principle components PC1 and PC2 are Per capita income and Transit.

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?

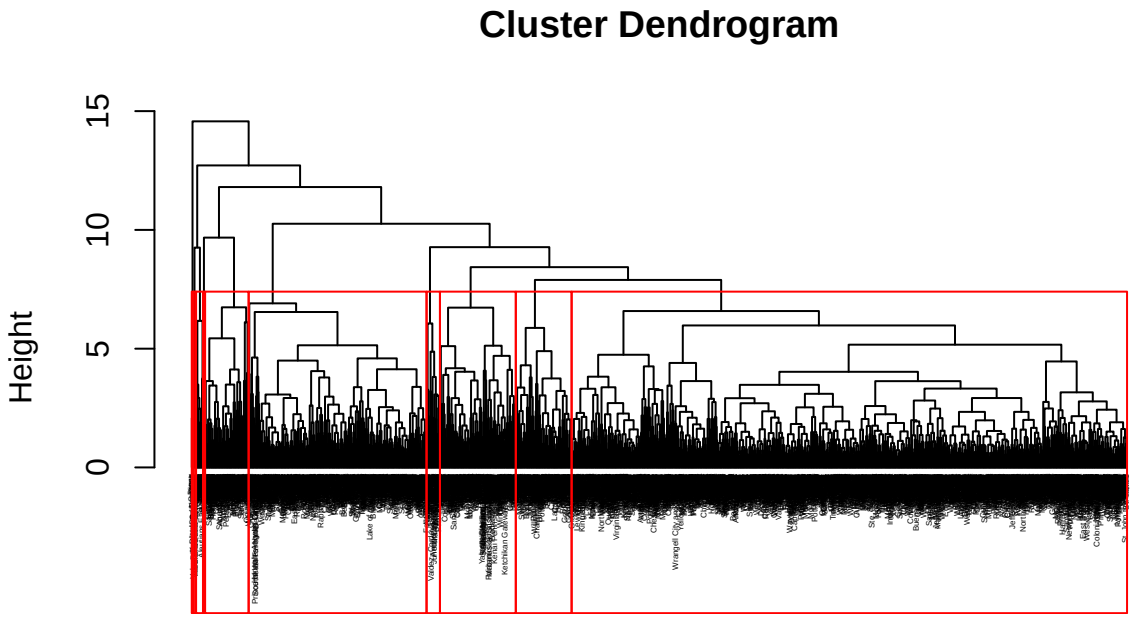
```
labels.census.ct <- paste(census.ct$State,census.ct$County,sep = ",")
scale.census.ct <- scale(census.ct[,c(-1,-2)])
row.names(scale.census.ct) <- labels.census.ct
dist <- dist(scale.census.ct, method = "euclidean")
hc.census.ct <- hclust(dist, method = "complete")
clustersa <- cutree(hc.census.ct, k=10)

plot(hc.census.ct,labels=census.ct$County, hang=-1,cex=0.25)
rect.hclust(hc.census.ct,k=10)
```




```
datacluster <- data.frame(census.ct, clustersa)
SanMateocluster <- datacluster %>% filter(datacluster[2] == "San Mateo")
SanMateogroup <- datacluster %>% filter(clustersa == as.integer(SanMateocluster$clustersa))
group <- nrow(SanMateogroup)
group #the group has 105 obs
```

```
## [1] 105
```

```
#5 PC
ct.pc.cluster <- data.frame(ct.pc_all$x[,1:5])
scale.ct.pc <- scale(ct.pc.cluster)
row.names(scale.ct.pc) <- labels.census.ct
distpc <- dist(scale.ct.pc, method = "euclidean")
hc.pc.ct <- hclust(distpc, method = "complete")
clusterspc <- cutree(hc.pc.ct, k =10)
plot(hc.pc.ct,labels=census.ct$County, hang=-1,cex=0.25)
rect.hclust(hc.pc.ct,k=10)
```




```
dataclusterpca <- data.frame(census.ct, clusterspc)
SanMateocluster <- dataclusterpca %>% filter(dataclusterpca[2] == "San Mateo")
SanMateogroupPCA <- dataclusterpca %>% filter(clusterspc == as.integer(SanMateocluster$clusterspc))
grouppca <- nrow(SanMateogroupPCA)
grouppca #there group has 46 obs
```

```
## [1] 46
```

```
clustersa[which(census.ct$County == "San Mateo")] #in cluster 7
```

```
## California,San Mateo
##                    7
```

```
clusterspc[which(census.ct$County == "San Mateo")] #in cluster 6
```

```
## California,San Mateo
##                    6
```

```
grep("San Mateo",names(clustersa))
```

```
## [1] 227
```

```
grep("San Mateo",names(clusterspc))
```

```
## [1] 227
```

```
clusterspc[227]
```

```
## California,San Mateo
##                    6
```

```
clustersa[227]
```

```
## California,San Mateo
##                    7
```

Answer: As the result above, the San Mateo County was placed in different Clusters. After performing hierarchical clustering using the whole dataset, there are 105 observations in the cluster containing San Mateo. The hierarchical clustering method with the first 5 principal components, we find only 46 observations in the cluster that contains San Mateo. The numbers of observations in the cluster are quite different. By comparing with both cluster data, some counties are shown in both clusters. We assume that those counties are placed in that one cluster since they share some features that are similar. And after using principal component analysis to the data, the hierarchical clustering could find even more features, so that San Mateo is placed in a smaller group that the counties share a more unique feature, that is why San Mateo is placed in different clusters in two methods. However, both sample size is relatively small, it is hard to tell which method perofroms better.

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 %>% ungroup %>% mutate_at(vars(State, County), tolower)

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

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

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

```
set.seed(10)
n = nrow(election.cl)
election.cl <- data.frame(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","Logistic Regression ")
```

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.

```
YTrain = trn.cl$candidate
XTrain = dplyr::select(trn.cl,-candidate)
YTest = tst.cl$candidate
XTest = dplyr::select(tst.cl,-candidate)
```

```
library(ROCR)
```

```
## Loading required package: gplots
```

```
##
## Attaching package: 'gplots'
```

```
## The following object is masked from 'package:stats':
##
##     lowess
```

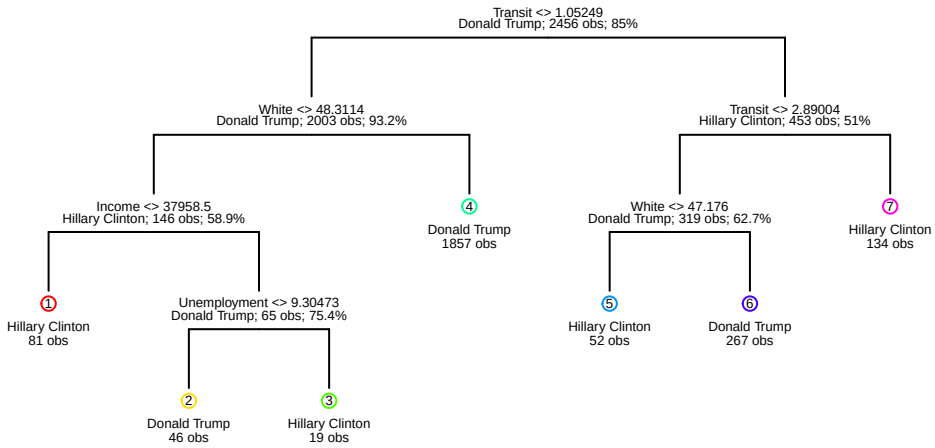
```
set.seed(1)

el.tree <- tree(candidate~. ,data = trn.cl, control = tree.control(nobs = nrow(trn.cl)))
summary(el.tree)
```

```
##
## Classification tree:
## tree(formula = candidate ~ ., data = trn.cl, control = tree.control(nobs = nrow(trn.cl)))
## Variables actually used in tree construction:
## [1] "Transit"      "White"        "Income"       "Unemployment"
## [5] "Production"   "IncomeErr"    "Carpool"
## Number of terminal nodes:  11
## Residual mean deviance:  0.3877 = 947.9 / 2445
## Misclassification error rate: 0.07248 = 178 / 2456
```

```
el.cvtree <- cv.tree(el.tree, folds, FUN = prune.misclass, K = nfold) ##
sizedev <- as.data.frame(cbind(el.cvtree$size, el.cvtree$dev))
sizedev <- sizedev[order(sizedev$V1),]
best.size.cv <- sizedev$V1[which.min(sizedev$V2)]
print(c("The Best tree size is", best.size.cv))
```

```
## [1] "The Best tree size is" "7"
```

```
pruned_el = prune.misclass(el.tree, best=best.size.cv)
draw.tree(pruned_el, cex = 0.4, nodeinfo = TRUE)
```

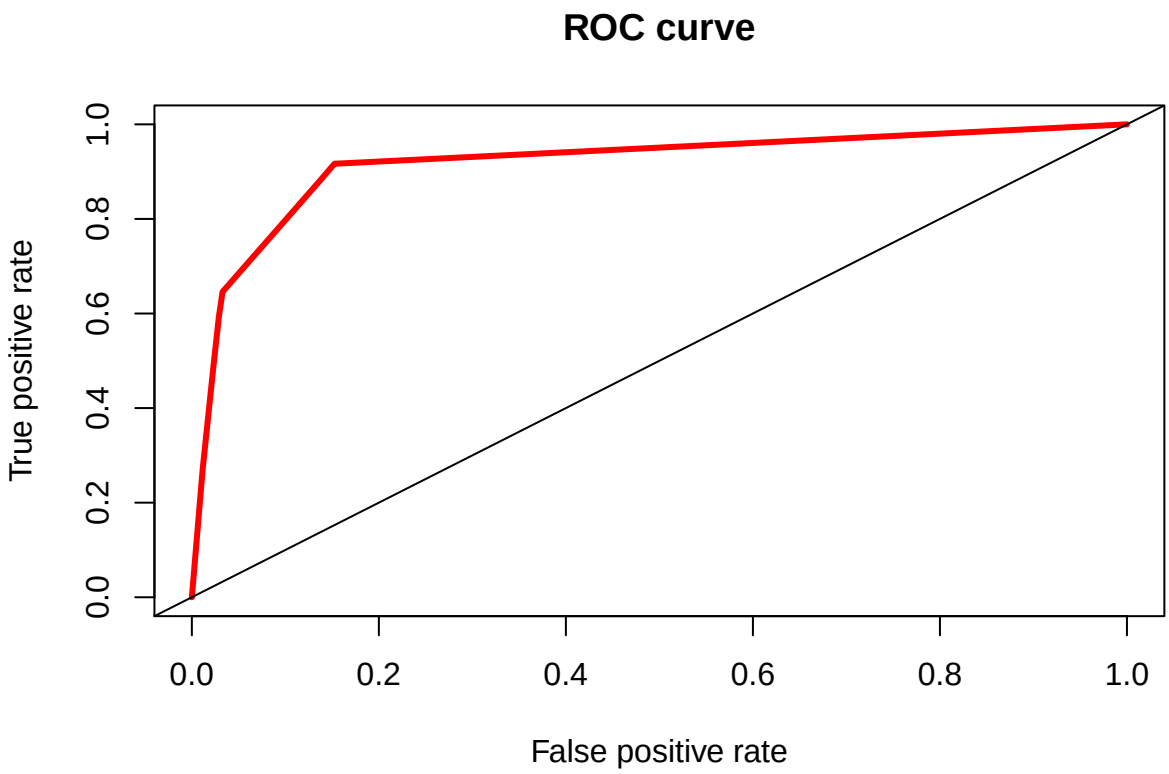


Total classified correct = 92.5 %

```
set.seed(1)
pred.el.Train.tree = predict(pruned_el, trn.cl, type="class")
pred.el.Test.tree = predict(pruned_el, tst.cl, type="class")
train.error.tree = calc_error_rate(pred.el.Train.tree, YTrain)
test.error.tree = calc_error_rate(pred.el.Test.tree, YTest)

records[1,] = c(train.error.tree, test.error.tree)
records
```

```
##                          train.error test.error
## tree                      0.07491857 0.08306189
## knn                              NA         NA
## Logistic Regression              NA         NA
```

```
pred.el.tree <- predict(pruned_el, tst.cl, type="vector")
pred.el.tree <- subset(pred.el.tree, select = c("Hillary Clinton", "Donald Trump"))
colnames(pred.el.tree) <- as.factor(colnames(pred.el.tree))
YTest.tree <- as.factor(YTest)
YTest.tree <- ifelse(YTest.tree == "Donald Trump","Donald Trump","Hillary Clinton")


predtree1 <- prediction(pred.el.tree[,1], YTest.tree)
predtree <- performance(predtree1, measure="tpr", x.measure="fpr")
plot(predtree, col="red", lwd=3, main="ROC curve")
abline(0,1)
```




```
auc_tree <- performance(predtree1, "auc")@y.values
auc_tree
```

```
## [[1]]
## [1] 0.9171493
```

The Decision Tree model AUC is 0.9171493

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)

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

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

  ## get classifications for current training chunks
  predYtr = knn(train = Xtr, test = Xtr, cl = Ytr, k = k)

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

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

```
library(plyr)
```

```
## ------------------------------------------------------------------------------
```

```
## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first, then dplyr:
## library(plyr); library(dplyr)
```

```
## ------------------------------------------------------------------------------
```

```
##
## Attaching package: 'plyr'
```

```
## The following object is masked from 'package:maps':
##
##     ozone
```

```
## The following objects are masked from 'package:dplyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize
```

```
## The following object is masked from 'package:purrr':
##
##     compact
```

```
error.folds = NULL
kvec = c(1, seq(10, 50, length.out=5))

set.seed(1)
for (j in kvec){
  tmp = ldply(1:nfold, do.chunk,
              folddef=folds, Xdat=XTrain, Ydat=YTrain, k=j)
  tmp$neighbors = j
  error.folds = rbind(error.folds, tmp)
}

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

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

```
## # A tibble: 1 x 3
##   neighbors variable  error
##       <dbl> <fct>     <dbl>
## 1        10 val.error 0.138
```

```
numneighbor = max(val.error.means$neighbors)
numneighbor
```

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

```
pred.YTrain.knn = knn(train=XTrain, test=XTrain, cl=YTrain, k=numneighbor)
pred.YTest.knn = knn(train=XTrain, test=XTest, cl=YTrain, k=numneighbor)
train.error.knn = calc_error_rate(pred.YTrain.knn, YTrain)
test.error.knn = calc_error_rate(pred.YTest.knn, YTest)

records[2,] = c(train.error.knn, test.error.knn)
records
```

```
##                       train.error test.error
## tree                   0.07491857 0.08306189
## knn                    0.12866450 0.15146580
## Logistic Regression           NA         NA
```

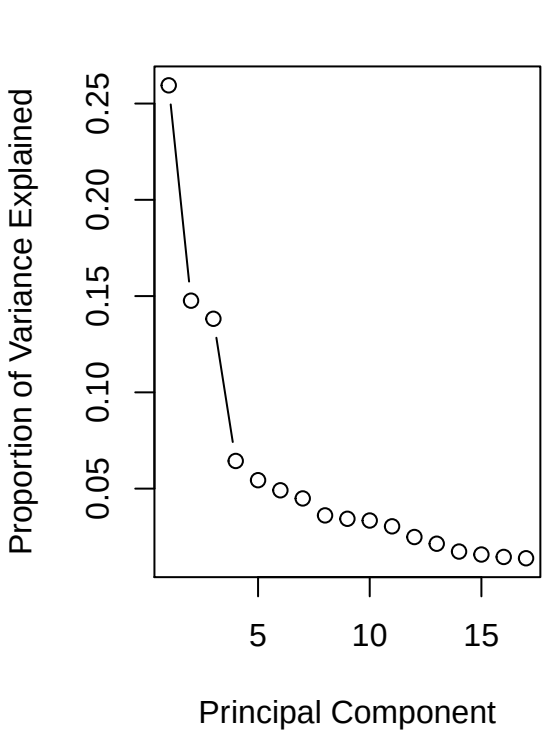
Classification: principal components

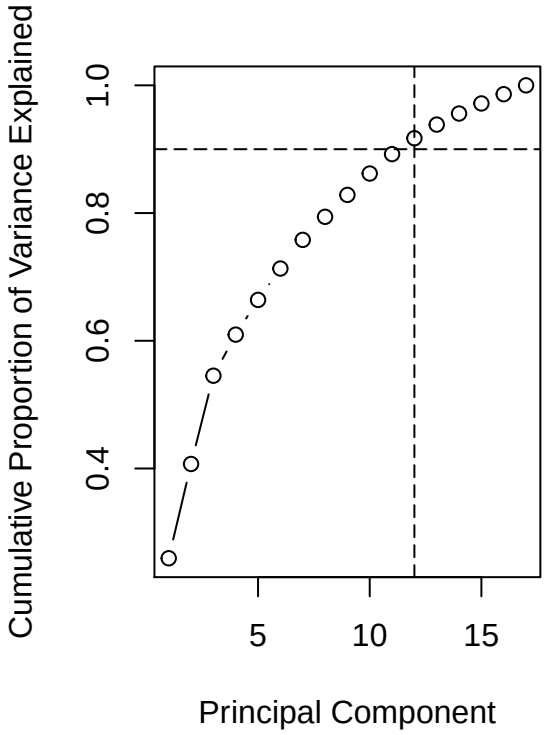
Instead of using the native attributes, we can use principal components in order to train our classification models. After this section, a comparison will be made between classification model performance between using native attributes and principal components.

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

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.

```
set.seed(1)
trn.pc_all <- prcomp(trn.cl[,c(-1)],

                     scale. = TRUE)
trn.pc <- as.data.frame(trn.pc_all$x)
#sum(ct.pc_all$sdev[1:17])/sum(ct.pc_all$sdev)
trnvar <- trn.pc_all$sdev[1:17]^2
par(mfrow=c(1, 2))
plot(trnvar/sum(trnvar), xlab = "Principal Component",
             ylab = "Proportion of Variance Explained",
             type = "b")
plot(cumsum(trnvar/sum(trnvar)), xlab = "Principal Component",
              ylab = "Cumulative Proportion of Variance Explained",
              type = "b") #12  needed
abline(h= .9,v=12,lty=5)
```






```
numpc = which(cumsum(trnvar/sum(trnvar)) >= 0.90)[1]
numpc
```

```
## [1] 12
```

Answer: The number of minimum number of PCs needed to capture 90% of the variance is 12.

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_all <- prcomp(trn.cl[,-1], scale=TRUE)
tr.pca <- as.data.frame(tr.pca_all$x[,1:numpc])
tr.pca <- tr.pca %>% mutate(candidate = trn.cl$candidate)

test.pca <- as.matrix(tst.cl[,-1])
test.pca <- predict(tr.pca_all,newdata = test.pca)
test.pca <- as.data.frame(test.pca[,1:numpc])
test.pca <- test.pca %>% mutate(candidate = tst.cl$candidate)
```

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

```
YTrain.pca = tr.pca$candidate
XTrain.pca = dplyr::select(tr.pca,-candidate)
YTest.pca = test.pca$candidate
XTest.pca = dplyr::select(test.pca,-candidate)
```

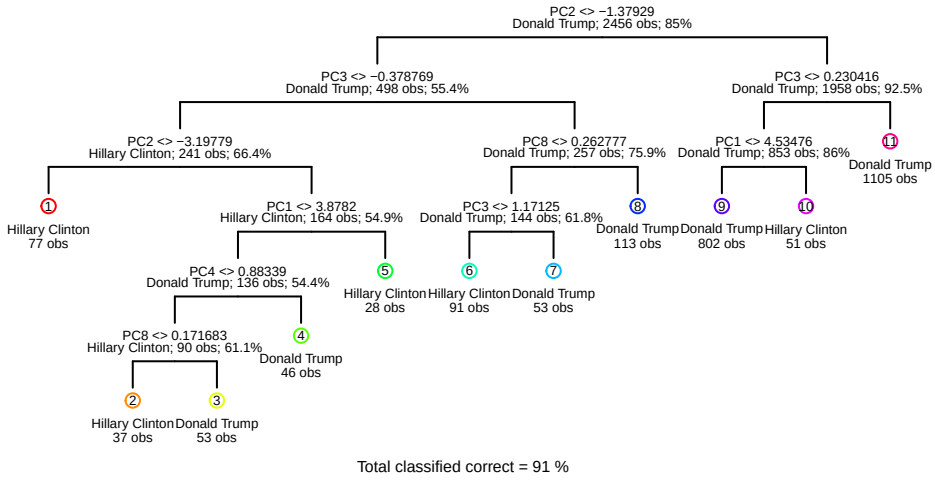
```
set.seed(1)
el.tree.pc <- tree(candidate~.,data = tr.pca, control = tree.control(nobs = nrow(tr.pca)))
summary(el.tree.pc)
```

```
##
## Classification tree:
## tree(formula = candidate ~ ., data = tr.pca, control = tree.control(nobs = nrow(tr.pca)))
## Variables actually used in tree construction:
## [1] "PC2" "PC3" "PC1" "PC4" "PC8" "PC7"
## Number of terminal nodes:  16
## Residual mean deviance:  0.4424 = 1079 / 2440
## Misclassification error rate: 0.09039 = 222 / 2456
```

```
set.seed(1)
el.cvtree.pc <- cv.tree(el.tree.pc, folds, FUN = prune.misclass, K = nfold) ##
sizedev <- as.data.frame(cbind(el.cvtree.pc$size, el.cvtree.pc$dev))
sizedev <- sizedev[order(sizedev$V1),]
best.size.cv <- sizedev$V1[which.min(sizedev$V2)]
print(c("The Best tree size is", best.size.cv))
```

```
## [1] "The Best tree size is" "11"
```

```
pruned_el.pc = prune.misclass(el.tree.pc, best=best.size.cv)
draw.tree(pruned_el.pc, cex = 0.4, nodeinfo = TRUE)
```

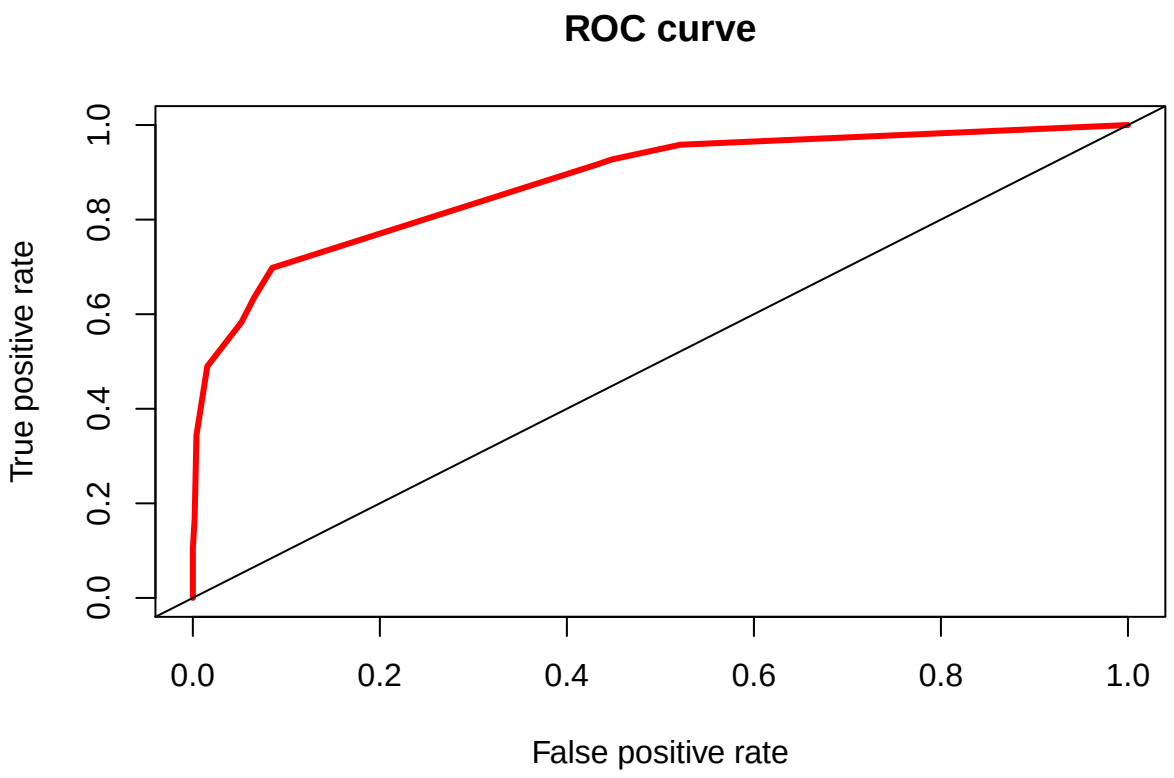



```
pred.el.Train.tree = predict(pruned_el.pc, tr.pca, type="class")
pred.el.Test.tree = predict(pruned_el.pc, test.pca, type="class")
```

```
train.error.tree = calc_error_rate(pred.el.Train.tree, YTrain.pca)
test.error.tree = calc_error_rate(pred.el.Test.tree, YTest.pca)

pca.records [1,] = c(train.error.tree, test.error.tree)
pca.records
```

```
##                       train.error test.error
## tree                   0.09039088  0.1091205
## knn                            NA         NA
## Logistic Regression            NA         NA
```

```
pred.el.tree11 <- predict(pruned_el.pc, test.pca, type="vector")
pred.el.treepca <- subset(pred.el.tree11, select = c("Hillary Clinton", "Donald Trump"))
colnames(pred.el.treepca) <- as.factor(colnames(pred.el.treepca))

predtree11 <- prediction(pred.el.treepca[,1], YTest.tree)
predtreepca <- performance(predtree11, measure="tpr", x.measure="fpr")
plot(predtreepca, col="red", lwd=3, main="ROC curve")
abline(0,1)
```




```
auc_tree <- performance(predtree11, "auc")@y.values
auc_tree
```

```
## [[1]]
## [1] 0.8790722
```

The PCA Decision Tree model AUC is 0.8790722

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

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

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

  train = (folddef!=chunkid)

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

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

  ## get classifications for current training chunks
  predYtr = knn(train = Xtr, test = Xtr, cl = Ytr, k = k)

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

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

```
library(plyr)

error.folds = NULL
kvec = c(1, seq(10, 50, length.out=5))

set.seed(1)
for (j in kvec){
  tmp = ldply(1:nfold , do.chunk,
              folddef=folds, Xdat=XTrain.pca, Ydat=YTrain.pca, k=j)
  tmp$neighbors = j
  error.folds = rbind(error.folds, tmp)
}
errors = melt(error.folds, id.vars='neighbors', value.name='error')
```

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

```
## # A tibble: 1 x 3
##   neighbors variable   error
##       <dbl> <fct>      <dbl>
## 1        10 val.error 0.0774
```

```
numneighbor = max(val.error.means$neighbors)
numneighbor
```

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

```
#the best k is 20
pred.YTrain.knn = knn(train=XTrain.pca, test=XTrain.pca, cl=YTrain.pca, k=numneighbor)
pred.YTest.knn = knn(train=XTrain.pca, test=XTest.pca, cl=YTrain.pca, k=numneighbor)
train.error.knn = calc_error_rate(pred.YTrain.knn, YTrain)
test.error.knn = calc_error_rate(pred.YTest.knn, YTest)

pca.records[2,] = c(train.error.knn, test.error.knn)
pca.records
```

```
##                       train.error test.error
## tree                   0.09039088 0.10912052
## knn                    0.06718241 0.08306189
## Logistic Regression           NA         NA
```

Interpretation & Discussion

19. This is an open question. Interpret and discuss any insights gained and possible explanations. Use any tools at your disposal to make your case: visualize errors on the map, discuss what does/doesn't seems reasonable based on your understanding of these methods, propose possible directions (collecting additional data, domain knowledge, etc)

Answer: Back to 2016, we were exposed to all kinds of news saying that she would win the election, and we thought the same, but the result was a big surprise for us. By looking closer to the election data, we might gain more insight about the election which allows candidates to prepare for 2020. In the dimensionality reduction, we use principal component analysis to gain a better understanding of the county and sub-county level data, which helps us to find the influential predictors. For sub-county data, the two prominent loadings are Per capita income and transition, which surprise us that transition's influence for the data. Although transit factor of sub-counties seems to be less relative to the election, the transition could be the foundation of the area, which helps economic growth, public services and brings resources. Also, when performing the decision tree, the variable got the first split is transit index. We assume the area has a lower transit index is considered as rural area and the urban area would have a higher transit index. The model shows that people living in the area has a lower transit index are more likely to vote for Donald Trump, and for the people living in the urban area, which has a higher transit index, they would vote for Hillary Clinton. The prediction matches the geographical outcome of problem 9. The prediction of decision tree is logical, since the urban area voters, mostly from the Democratic bases, would more favor of Hillary Clinton, and people from rural are likely to vote for Trump. Therefore, voters from the different area could determine who they would vote for, which makes the transit index such an important variable that could determine the voting pattern. From the data, we learn that some factors seem unrelated could have a huge effect on the predicting process, which suggests us to consider more factors for understanding the data. To make the prediction result, we use decision tree and KNN to interpret the data. Since it is almost impossible to know how the data distributed, we should use flexible models that could adapt the data better, making KNN and decision tree become good options. Decision tree and KNN are easy to use and we don't need to make assumptions on our data. From the result we have, decision tree performs the better. In the dimensionality reduction, principal component analysis helps to overcome the overfitting issues and learn the data. However, principal components are hardly readable, and if we don't select the right amount of data, the model will be overfitted. Training with the principal component data, we have KNN as our best model since it has smaller training error and test error comparing to the tree model. Overall, decision tree and KNN are good options in our case. Although our models might not be the most accurate in predicting the outcome of the election, predictions we made are close to the result of the election of 2016, which indicates to study the data is useful for the election.

Taking it further

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

    - 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?

we would perfrom Logistic regression model to find out if it's better than the tree model.

```
#Logistic regression
library(ROCR)

set.seed(1)
lg_train_fit <-  glm(candidate~ ., data=trn.cl, family=binomial)
```

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

```
summary(lg_train_fit)
```

```
##
## Call:
## glm(formula = candidate ~ ., family = binomial, data = trn.cl)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.8086  -0.2685  -0.1124  -0.0400   3.5943
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)     -1.158e+01  9.734e+00  -1.189 0.234244
## Men              8.584e+00  5.406e+00   1.588 0.112329
## White           -2.217e-01  6.478e-02  -3.422 0.000622 ***
## Citizen          1.019e+01  3.010e+00   3.384 0.000715 ***
## Income          -7.579e-05  2.750e-05  -2.756 0.005849 **
## IncomeErr       -3.874e-05  6.283e-05  -0.617 0.537492
## IncomePerCap     2.676e-04  6.728e-05   3.978 6.95e-05 ***
## IncomePerCapErr -2.786e-04  1.309e-04  -2.127 0.033395 *
## Poverty          1.991e-02  4.119e-02   0.483 0.628946
## ChildPoverty    -7.411e-03  2.560e-02  -0.289 0.772213
## Professional     2.817e-01  3.905e-02   7.214 5.44e-13 ***
## Service          3.659e-01  4.923e-02   7.432 1.07e-13 ***
## Office           1.051e-01  4.714e-02   2.230 0.025761 *
## Production       1.845e-01  4.317e-02   4.273 1.93e-05 ***
## Drive           -2.562e-01  5.424e-02  -4.724 2.32e-06 ***
## Carpool         -2.463e-01  6.714e-02  -3.668 0.000244 ***
## Transit         -8.629e-03  1.015e-01  -0.085 0.932251
## OtherTransp     -9.698e-02  1.014e-01  -0.956 0.339094
## WorkAtHome      -2.101e-01  7.961e-02  -2.639 0.008316 **
## MeanCommute      6.325e-02  2.482e-02   2.548 0.010823 *
## Employed         1.653e+01  3.292e+00   5.021 5.13e-07 ***
## PrivateWork      9.254e-02  2.149e-02   4.307 1.66e-05 ***
## SelfEmployed     1.227e-02  4.668e-02   0.263 0.792683
## FamilyWork      -1.191e+00  4.099e-01  -2.907 0.003654 **
## Unemployment     1.838e-01  3.810e-02   4.824 1.41e-06 ***
## Minority        -8.920e-02  6.214e-02  -1.436 0.151136
## ---
## 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:  854.07  on 2430  degrees of freedom
## AIC: 906.07
##
## Number of Fisher Scoring iterations: 7
```

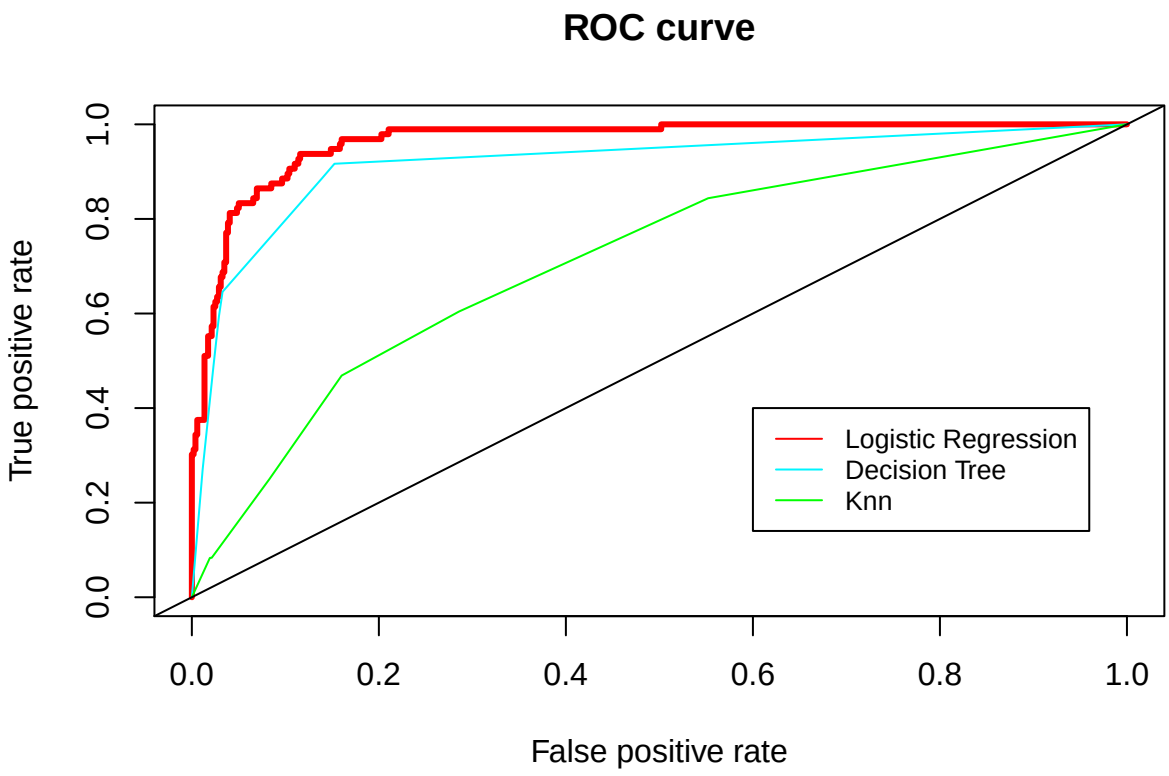
```
lg_train_predict <- predict(lg_train_fit, type = "response")
summary(lg_train_predict)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## 0.0000001 0.0030364 0.0164271 0.1498371 0.1170631 1.0000000
```

```
set.seed(1)
#training and predicting
predict.YTrain.log <- predict(lg_train_fit, data = trn.cl, type = "response")

YTrain.log <- as.factor(as.character(YTrain))
YTest.log <- as.factor(as.character(YTest))

predYTest.log <- predict(lg_train_fit, newdata = tst.cl, type = "response")

predLogistic <- prediction(predYTest.log, YTest.log)
perfLogistic <- performance(predLogistic, measure="tpr", x.measure="fpr")

#knn for ROC
pred.YTest.knn.roc = knn(train=XTrain, test=XTest, cl=YTrain, k=10, prob = TRUE)
prodknn <- attr(pred.YTest.knn.roc, "prob")
predknn <- prediction(1-prodknn, YTest.log)
perfknn <- performance(predknn, measure="tpr", x.measure="fpr")
auc.knn <- performance(predknn, "auc")@y.values
auc.knn
```

```
## [[1]]
## [1] 0.7120837
```

```
{plot(perfLogistic, col="red", lwd=3, main="ROC curve")
plot(predtree,  add=TRUE, col = "turquoise1")
plot(perfknn,  add=TRUE, col = "green")
abline(0,1)
legend(0.6, 0.4, legend=c("Logistic Regression", "Decision Tree", "Knn"), col=c("red", "turquoise1", "g
```



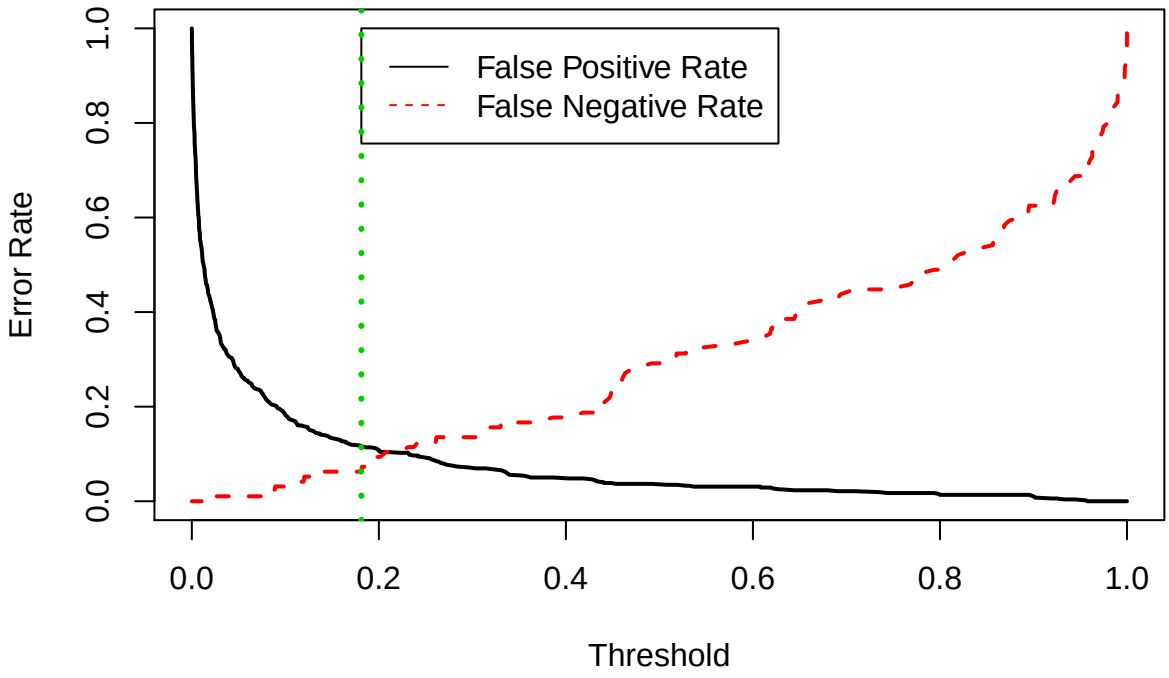

```
auc_logistic <- performance(predLogistic, measure = "auc")
auc_logistic <- auc_logistic@y.values[[1]]
cat("Logistic Regression model AUC is", auc_logistic, "\n")
```

```
## Logistic Regression model AUC is 0.963783
```

```
fpr = performance(predLogistic, "fpr")@y.values[[1]]
cutoff = performance(predLogistic, "fpr")@x.values[[1]]
# FNR
fnr = performance(predLogistic,"fnr")@y.values[[1]]

# Add legend to the plot

rate = as.data.frame(cbind(Cutoff=cutoff, FPR=fpr, FNR=fnr))
rate$distance = sqrt((rate[,2])^2+(rate[,3])^2)
index = which.min(rate$distance)
best = rate$Cutoff[index]
best
```

```
## [1] 0.1812543
```

```
# Plot
matplot(cutoff, cbind(fpr,fnr), type="l",lwd=2, xlab="Threshold",ylab="Error Rate")
# Add legend to the plot
legend( 0.1812543, 1, legend=c("False Positive Rate","False Negative Rate"),
col=c(1,2), lty=c(1,2))
# Add the best value
abline(v=best, col=3, lty=3, lwd=3)
```




```
#with the best threshold
set.seed(1)

predict.YTrain.log.cand <- factor(ifelse(predict.YTrain.log > 0.1812543, "Hillary Clinton", "Donald Tru

predict.YTest.log.cand <- factor(ifelse(predYTest.log > 0.1812543,"Hillary Clinton", "Donald Trump"))

YTrain.log <- as.factor(as.character(YTrain))
train.error.logistic = calc_error_rate(predict.YTrain.log.cand, YTrain.log)

YTest.log <- as.factor(as.character(YTest))
test.error.logistic = calc_error_rate(predict.YTest.log.cand, YTest.log)

records[3,] = c(train.error.logistic, test.error.logistic)
records
```

```
##                      train.error test.error
## tree                  0.07491857 0.08306189
## knn                   0.12866450 0.15146580
## Logistic Regression   0.09934853 0.10749186
```

The ROC for Knn is 0.7120837, The Decision Tree model AUC is 0.9171493.

```
#with pca data
set.seed(1)
lg_train_fit_pca <-  glm(candidate~ ., data=tr.pca, family=binomial)
```

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

```r
summary(lg_train_fit_pca)
```

```
##
## Call:
## glm(formula = candidate ~ ., family = binomial, data = tr.pca)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -5.6028  -0.3628  -0.1956  -0.1007   3.1232
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -2.77735    0.11868 -23.401  < 2e-16 ***
## PC1         -0.05636    0.03298  -1.709   0.0874 .
## PC2         -1.09024    0.06863 -15.885  < 2e-16 ***
## PC3         -0.79386    0.05697 -13.935  < 2e-16 ***
## PC4         -0.42975    0.06771  -6.346 2.20e-10 ***
## PC5         -0.52596    0.10833  -4.855 1.20e-06 ***
## PC6         -0.58034    0.08819  -6.580 4.69e-11 ***
## PC7         -0.32467    0.08123  -3.997 6.42e-05 ***
## PC8         -1.09673    0.12846  -8.537  < 2e-16 ***
## PC9         -0.66382    0.09893  -6.710 1.95e-11 ***
## PC10         0.21246    0.10067   2.110   0.0348 *
## PC11         0.14788    0.13514   1.094   0.2739
## PC12        -0.01105    0.12338  -0.090   0.9286
## ---
## 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: 1103.3  on 2443  degrees of freedom
## AIC: 1129.3
##
## Number of Fisher Scoring iterations: 7
```

```r
lg_train_predict_pac <- predict(lg_train_fit_pca, type = "response")
summary(lg_train_predict_pac)
```

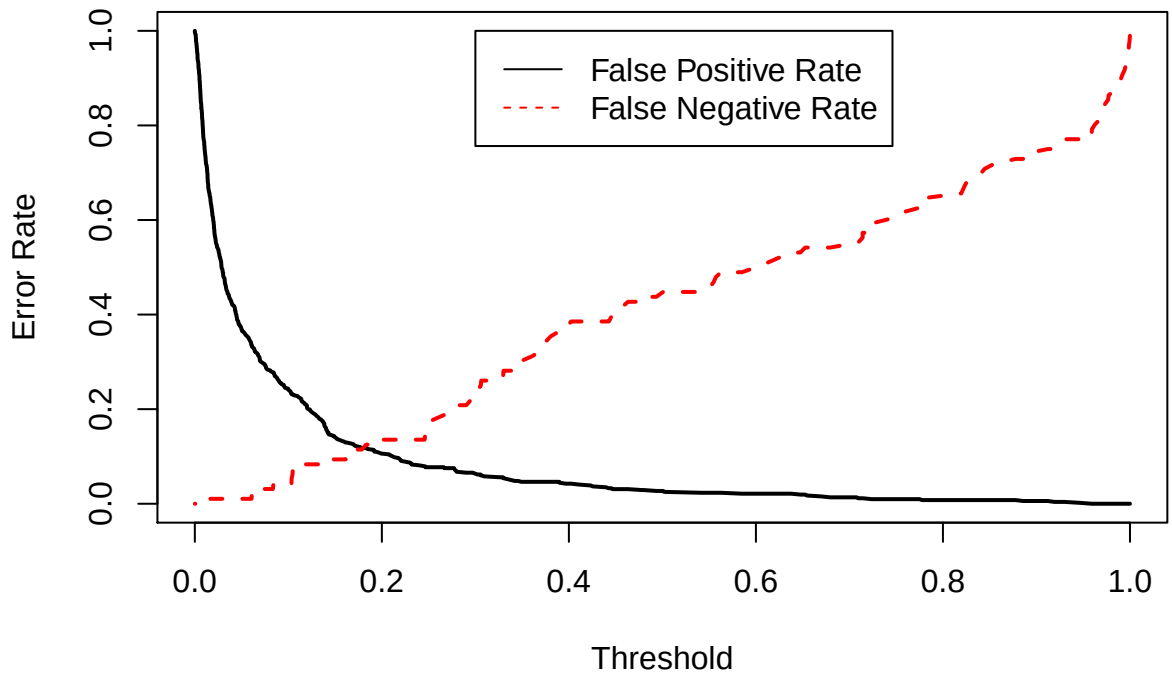
```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## 0.000001 0.012166 0.036390 0.149837 0.144262 1.000000
```

```r
set.seed(1)
#training and predicting
predict.YTrain.logpca <- predict(lg_train_fit_pca, data = trn.pc, type = "response")

YTrain.log.pca <- as.factor(as.character(YTrain.pca))
YTest.log.pca <- as.factor(as.character(YTest.pca))

predYTest.log.pca <- predict(lg_train_fit_pca, newdata = XTest.pca, type = "response")

predLogistic.pca <- prediction(predYTest.log.pca, YTest.log)

perfLogistic1 <- performance(predLogistic.pca, measure="tpr", x.measure="fpr")
```

```r
auc_logisticpca <- performance(predLogistic.pca, measure = "auc")
auc_logisticpca <- auc_logisticpca@y.values[[1]]
cat("PCA Logistic Regression model AUC is",auc_logisticpca,"\n")
```

```
## PCA Logistic Regression model AUC is 0.9447595
```

```r
fpr = performance(predLogistic.pca, "fpr")@y.values[[1]]
cutoff = performance(predLogistic.pca, "fpr")@x.values[[1]]
# FNR
fnr = performance(predLogistic.pca,"fnr")@y.values[[1]]
matplot(cutoff, cbind(fpr,fnr), type="l",lwd=2, xlab="Threshold",ylab="Error Rate")
# Add legend to the plot
legend(0.3, 1, legend=c("False Positive Rate","False Negative Rate"),
col=c(1,2), lty=c(1,2))
```




```r
rate = as.data.frame(cbind(Cutoff=cutoff, FPR=fpr, FNR=fnr))
rate$distance = sqrt((rate[,2])^2+(rate[,3])^2)
index = which.min(rate$distance)
best = rate$Cutoff[index]
best
```

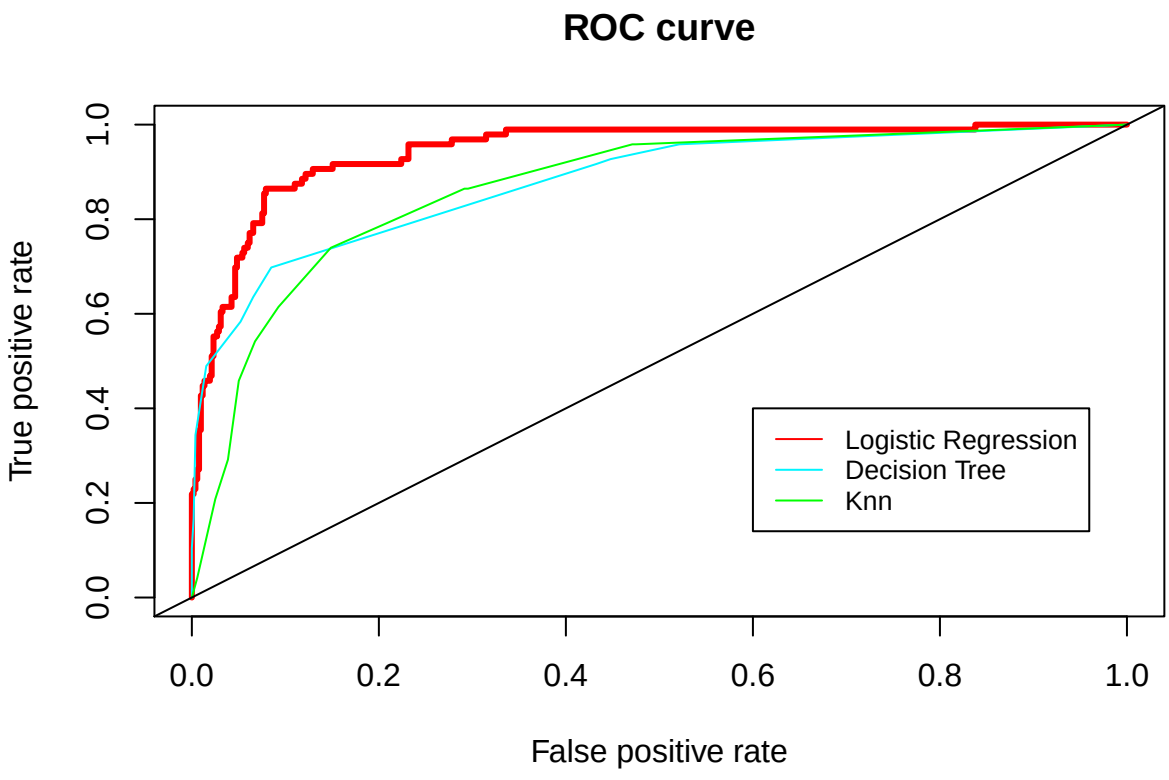
```
## [1] 0.2457648
```

```r
#with the best threshold
set.seed(1)

predict.YTrain.log.candPCA <- factor(ifelse(predict.YTrain.logpca > 0.2457648, "Hillary Clinton", "Dona
```

```
predict.YTest.log.candPCA <- factor(ifelse(predYTest.log.pca > 0.2457648,"Hillary Clinton", "Donald Trum

YTrain.log.pca <- as.factor(as.character(YTrain.pca))
YTest.log.pca <- as.factor(as.character(YTest.pca))


train.error.logistic = calc_error_rate(predict.YTrain.log.candPCA, YTrain.log.pca)
test.error.logistic = calc_error_rate(predict.YTest.log.candPCA, YTest.log.pca)

#PCA knn for ROC
pred.YTest.knn.roc.PCA = knn(train=XTrain.pca, test=XTest.pca, cl=YTrain.pca, k=20, prob = TRUE)
prodknnPCA <- attr(pred.YTest.knn.roc.PCA, "prob")
predknnPCA <- prediction(1-prodknnPCA, YTest.log.pca)
perfknnPCA <- performance(predknnPCA, measure="tpr", x.measure="fpr")
auc.knnPCA <- performance(predknnPCA, "auc")@y.values
auc.knnPCA
```

```
## [[1]]
## [1] 0.8672378
```

```
plot(perfLogistic1, col="red", lwd=3, main="ROC curve")
plot(predtreepca,  add=TRUE, col = "turquoise1")
plot(perfknnPCA,  add=TRUE, col = "green")
abline(0,1)
legend(0.6, 0.4, legend=c("Logistic Regression", "Decision Tree", "Knn"), col=c("red", "turquoise1", "gr
```



```
pca.records[3,] = c(train.error.logistic, test.error.logistic)
pca.records
```

```
##                       train.error test.error
## tree                   0.09039088 0.10912052
## knn                    0.06718241 0.08306189
## Logistic Regression    0.10464169 0.08794788
```

The PCA Decision Tree model AUC is 0.8790722, the PCA Knn AUC is 0.8672378.

Answer: Overall, Logistic regression perfroms the best comparing with area under the curve in both datasets.