Census and Education Machine Learning/Data Science Project

Braeden Bailey

Data

Census Data

We load in and clean the `census` dataset by transforming the full state names to abbreviations (to match the `education` dataset in later steps). Specifically, `R` contains default global variables `state.name` and `state.abb` that store the full names and the associated abbreviations of the 50 states. However, it does not contain `District of Columbia` (and the associated `DC`). We added it back manually since census contains information in DC. We further remove data from Purto Rico to ease the visualization in later steps.

```
state.name <- c(state.name, "District of Columbia")
state.abb <- c(state.abb, "DC")
## read in census data
census <- read_csv("./acs2017_county_data.csv") %>%
  select(-CountyId, -ChildPoverty, -Income, -IncomeErr, -IncomePerCap, -IncomePerCapErr) %>%
  mutate(State = state.abb[match(`State`, state.name)]) %>%
  filter(State != "PR")
```

Education Data

We also include the `education` dataset, available at Economic Research Service at USDA. The dataset contains county-level educational attainment for adults age 25 and older in 1970-2019. We specifically use educational attainment information for the time period of 2015-2019.

To clean the data, we remove uninformative columns (as in `FIPS Code`, `2003 Rural-urban Continuum Code`, `2003 Urban Influence Code`, `2013 Rural-urban Continuum Code`, and `2013 Urban Influence Code`). To be consistent with `census` data, we exclude data from Purto Rico and we rename `Area name` to `County` in order to match that in the `census` dataset.

```
education <- read_csv("./Education.csv") %>%
  filter(!is.na(`2003 Rural-urban Continuum Code`)) %>%
  filter(State != "PR") %>%
  select(-`FIPS Code`,
         -`2003 Rural-urban Continuum Code`,
         -`2003 Urban Influence Code`,
         -`2013 Rural-urban Continuum Code`,
         -`2013 Urban Influence Code`) %>%
  rename(County = `Area name`)
```

Preliminary Data Analysis

Census Analysis

- Report the dimension of `census`.

```
dim(census)
```

```
## [1] 3142   31
```

There are 3142 observations (or rows) and 31 variables (or columns) in the `census` dataset.

- Are there missing values in the data set?

```
sum(is.na(census))
```

```
## [1] 0
```

There are no missing values in the dataset.

- Compute the total number of distinct values in State in census to verify that the data contains all states and a federal district

```
unique_states <- unique(census$State)
length(unique_states)
```

```
## [1] 51
```

There are 50 states and 1 federal district, which has been verified with the result of 51 given by `length(unique_states)` above.

Education Analysis

- Report the dimension of `education`.

```
dim(education)
```

```
## [1] 3143   42
```

There are 3143 observations (or rows) and 42 variables (or columns) in the `education` dataset.

- Are there missing values in the data set?

```r
# Creates a subset of the data that counts missing values in each column
education_missing <- education %>%
  group_by(County) %>%
  summarise_each(funs(sum(is.na(.))))

# Using the above subset, sums each column within a row to get total missing values
# Then, only considers columns with missing counts and counts how many there are.
cbind(education_missing, total_missing = rowSums(education_missing[, names(education_missing) != "County
  select("County", "total_missing") %>%
  filter(total_missing > 0) %>%
  nrow()
```

```
## [1] 18
```

There are 18 distinct counties that contain missing values in the data set.

- Compute the total number of distinct values in County in education.

```r
unique_county_edu <- unique(education$County)
length(unique_county_edu)
```

```
## [1] 1877
```

There are 1877 distinct values in County.

- Compare the values of total number of distinct county in `education` with that in `census`.

```r
unique_county_cen <- unique(census$County)

# Total distinct counties in census
length(unique_county_cen)
```

```
## [1] 1877
```

```r
# Total distinct counties in education
length(unique_county_edu)
```

```
## [1] 1877
```

Both datasets have the same total number of distinct counties. This is a good thing, as it means it will make using the datasets together in our analysis easier. If they had different number of unique counties, combining the datasets might be more difficult.

Data wrangling/cleaning

NA Values

Remove all NA values in `education`, if there is any.

```
education <- education %>% na.omit()
```

Mutate education data

In `education`, in addition to `State` and `County`, we will start only on the following 4 features: `Less than a high school diploma, 2015-19`, `High school diploma only, 2015-19`, `Some college or associate's degree, 2015-19`, and `Bachelor's degree or higher, 2015-19`. We will then mutate the education dataset by selecting these 6 features only, and create a new feature which is the total population of that county.

```
education <- education %>%
  select("State",
         "County",
         "Less than a high school diploma, 2015-19",
         "High school diploma only, 2015-19",
         "Some college or associate's degree, 2015-19",
         "Bachelor's degree or higher, 2015-19") %>%
   mutate(TotalPop = rowSums(across(where(is.numeric))))
```

Aggregate education statistics

We will construct aggregated data sets from `education` data: i.e., create a state-level summary into a dataset named `education.state`.

```
education.state <- education %>%
  group_by(State) %>%
  summarise_if(is.numeric, mean, na.rm = TRUE)
```

Finding highest education in each state

Create a data set named `state.level` on the basis of `education.state`, where you create a new feature which is the name of the education degree level with the largest population in that state.

```
state.level <- education.state %>%
  mutate(LargestEducationLevel =
           colnames(education.state[, c(2,3,4,5)])
         [max.col(education.state[, c(2,3,4,5)],
                  ties.method = "first")])
```

Visualization

Education data visualzation

We will color the map (on the state level) by the education level with highest population for each state.

First, we need to change the format of the state names in the `states` dataset so that we can combine it with the `state.level` dataset

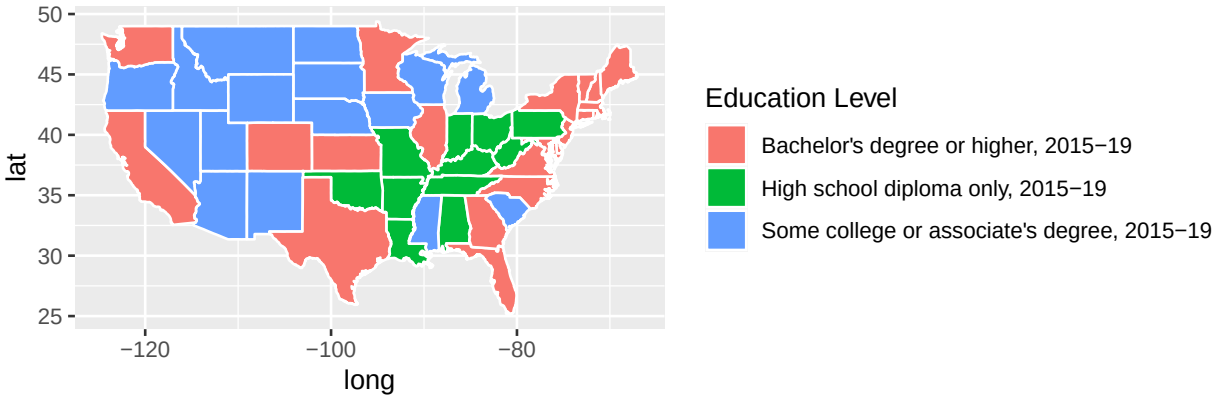
```
states <- map_data("state")
states.region <- states$region
# Creates a vector of `region` column from states dataset

states.abb <- state.abb[match(states.region, tolower(state.name))]
# Matches region names in states to their abbreviations

states <- states %>% mutate(region = states.abb)
# Replace full names in `states` dataset with the abbreviations vector
```

Now that the state names are in the same format, we can combine the data using `left_join()`

```
states.combined <- left_join(states, state.level, by = c("region" = "State"))
```

Using our new `states` dataset, we can plot the map and color each state by the education level with the highest population.

```
ggplot(data = states.combined) +
  geom_polygon(aes(x = long, y = lat, fill = LargestEducationLevel, group = group),
               color = "white") +
  coord_fixed(1.3) +
  guides(fill=guide_legend(title="Education Level"))
```



Census data visualization

Our goal here is to visualize the racial makeup of the 3 most/least populated counties in California using a radar chart.

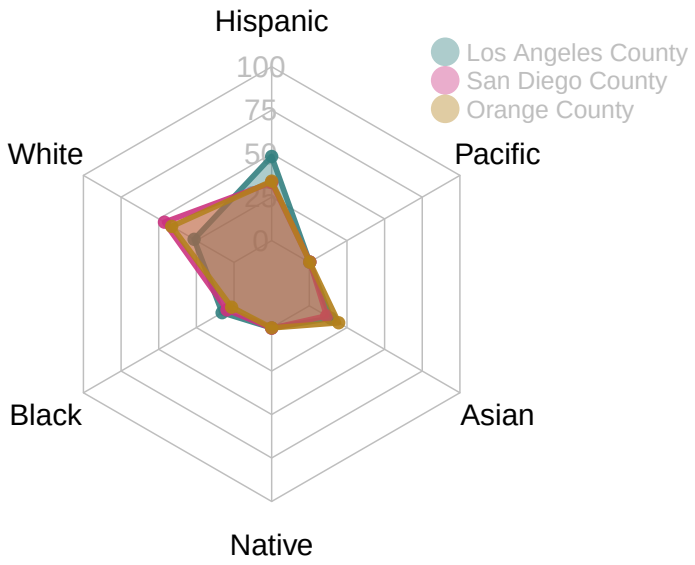
```
# install library needed for graph
library(fmsb)
```

```
## Warning: package 'fmsb' was built under R version 4.1.3
```

```
# Get only the CA counties from the census data
census.CA <- census %>% filter(State == "CA")

# Create a dataset of the three most populated counties
top3pop <- head(census.CA[order(census.CA$TotalPop, decreasing = TRUE),], 3)

# Create a dataset of the three least populated counties
bottom3pop <- head(census.CA[order(census.CA$TotalPop, decreasing = FALSE),], 3)

# More data wrangling
top3pop <- top3pop %>% select(County, c(Hispanic: Pacific))
bottom3pop <- bottom3pop %>% select(County, c(Hispanic: Pacific))
```

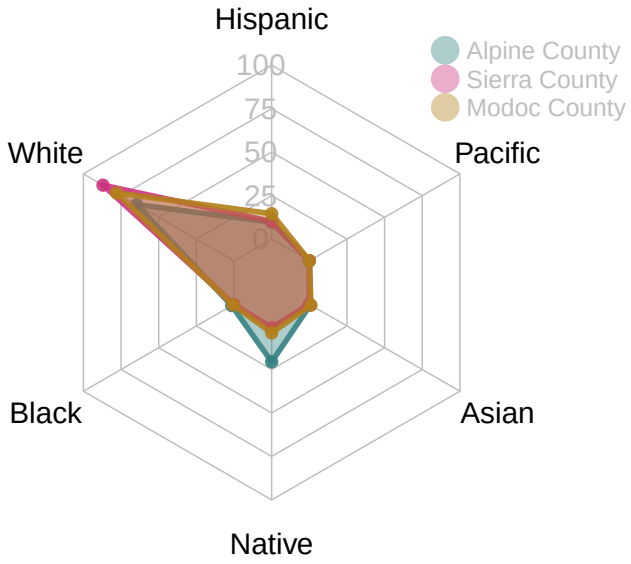
```
top3pop <- top3pop %>% remove_rownames %>% column_to_rownames(var="County")
bottom3pop <- bottom3pop %>% remove_rownames %>% column_to_rownames(var="County")

# To use the fmsb package, I have to add min and max rows
top3pop <- rbind(rep(100.0, 6), rep(0.0, 6), top3pop)
bottom3pop <- rbind(rep(100.0, 6), rep(0.0, 6), bottom3pop)

colors_border=c( rgb(0.2,0.5,0.5,0.9), rgb(0.8,0.2,0.5,0.9) , rgb(0.7,0.5,0.1,0.9) )
colors_in=c( rgb(0.2,0.5,0.5,0.4), rgb(0.8,0.2,0.5,0.4) , rgb(0.7,0.5,0.1,0.4) )
```

```
# Graph of top 3 most populated counties

radarchart(top3pop, axistype=1 ,
    pcol=colors_border , pfcol=colors_in , plwd=3 , plty=1,
    cglcol="grey", cglty=1, axislabcol="grey", caxislabels=seq(0,100,25), cglwd=0.8)

legend(x=0.7, y=1.2, legend = rownames(top3pop[-c(1,2),]), bty = "n", pch=20 ,
       col=colors_in , text.col = "grey", cex=0.8, pt.cex=3)
```




```
# Graph of bottom 3 most populated counties
radarchart(bottom3pop, axistype=1 ,
    pcol=colors_border , pfcol=colors_in , plwd=3 , plty=1,
    cglcol="grey", cglty=1, axislabcol="grey", caxislabels=seq(0,100,25), cglwd=0.8)
```

```
legend(x=0.7, y=1.2, legend = rownames(bottom3pop[-c(1,2),]), bty = "n", pch=20 ,
       col=colors_in , text.col = "grey", cex=0.8, pt.cex=3)
```



These plots by themselves ended up not revealing much individually as the counties in each grouping were quite similar in their racial makeup. However, when compared we can see that the least populated counties in California have a much larger white population and are less diverse in general. Admittedly, this could just be due to their low population thus lower sample size, but it is still important to consider when moving forward with the analysis.

Machine Learning Methods and Model Building

The `census` data contains county-level census information. We will clean and aggregate the information as follows.

We start with `census`, filter out any rows with missing values, convert {`Men`, `Employed`, `VotingAgeCitizen`} attributes to percentages, compute `Minority` attribute by combining {Hispanic, Black, Native, Asian, Pacific}, remove these variables after creating `Minority`, remove {`Walk`, `PublicWork`, `Construction`, `Unemployment`}.

```
census.clean <- census %>% na.omit()

census.clean <- census.clean %>%
  mutate(Men = (Men/TotalPop)*100,
         Employed = (Employed/TotalPop)*100,
         VotingAgeCitizen = (VotingAgeCitizen/TotalPop)*100)

census.clean <- census.clean %>%
  mutate(Minority = Hispanic + Black + Native + Asian + Pacific) %>%
  relocate(Minority, .after = White)

census.clean <- census.clean %>%
  select(-c(Hispanic, Black, Native, Asian, Pacific,
            Walk, PublicWork, Construction, Unemployment))

# Remove columns that are collinear
census.clean <- census.clean %>%
  select(-c(White, Women))
```

Dimensionality reduction methods

- We will run PCA for the cleaned county level census data (with `State` and `County` excluded).

```
census.clean_pr <- select(census.clean, -c(State, County))

# Data is centered and scaled!
pr.out <- prcomp(census.clean_pr, scale = TRUE, center = TRUE)
```

- We will save the first two principle components PC1 and PC2 into a two-column data frame, call it `pc.county`.

```
# This line of code extracts the PC1 and PC2 score vectors.
pc.county <- (pr.out$x)[, c('PC1', 'PC2')]
```

The features were scaled before running PCA. The main reason being that because the `TotalPop` attribute does not share the same units with the rest of the data, which would lead to it have much higher mean and variance if not scaled. The decision was made to center the data because it is generally a good idea to do so before performing PCA.

- We are interested in the three features with the largest absolute values of the first principal component.

```r
loadings <- pr.out$rotation

pc1.loadings <- loadings[, 'PC1']

head(sort(abs(pc1.loadings), decreasing = TRUE), 3)
```

```
##   WorkAtHome SelfEmployed        Drive
##       0.4267       0.3605       0.3578
```

The three features with largest absolute values of the first principal component are `WorkAtHome`, `SelfEmployed`, and `Drive`.

- Which features are inversely correlated?

```r
pc1.loadings < 0
```

```
##         TotalPop              Men         Minority VotingAgeCitizen
##            FALSE            FALSE             TRUE            FALSE
##          Poverty     Professional          Service           Office
##             TRUE            FALSE             TRUE             TRUE
##       Production            Drive          Carpool          Transit
##             TRUE             TRUE             TRUE            FALSE
##      OtherTransp       WorkAtHome      MeanCommute         Employed
##            FALSE            FALSE             TRUE            FALSE
##      PrivateWork     SelfEmployed       FamilyWork
##             TRUE            FALSE            FALSE
```

The features with negative signs in the first principal component are as follows: `Minority`, `Poverty`, `Service`, `Office`, `Production`, `Drive`, `Carpool`, `MeanCommute`, and `PrivateWork`. Their opposite sign says nothing about how much variance is explained, rather it just says that they are negatively correlated with the features with positive signs. The sign just gives the direction that the feature is going in the single-dimension PC vector.

Principal Component Variance Capture %

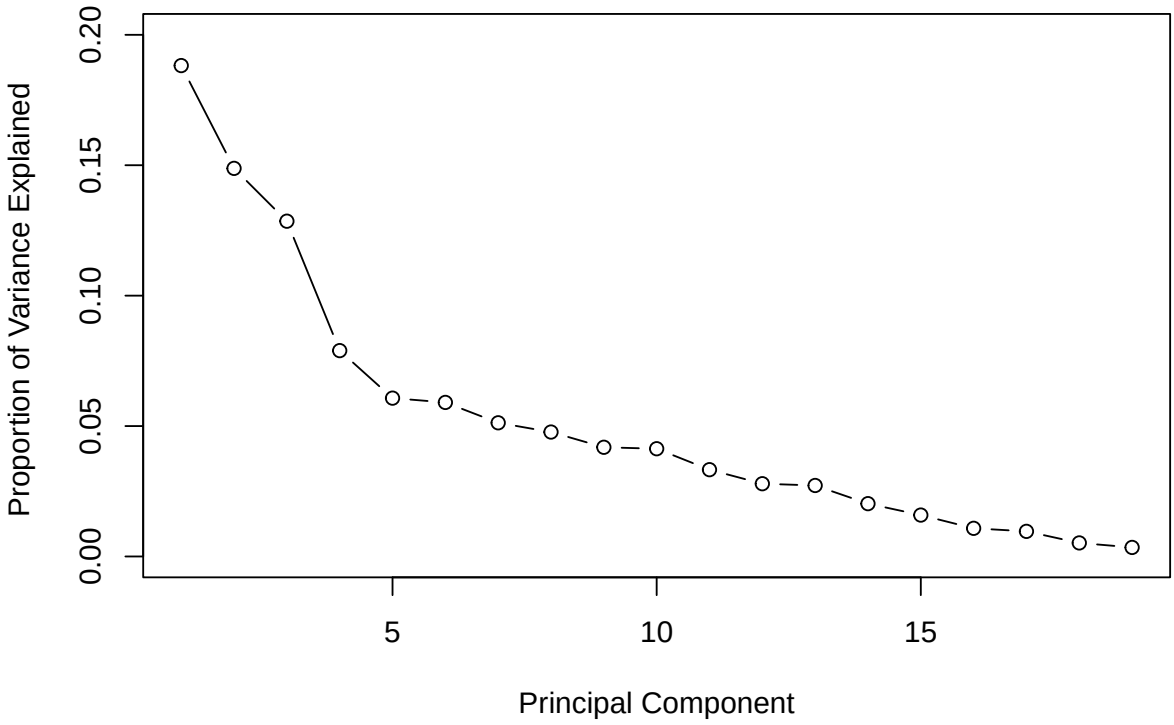
- Determine the number of minimum number of PCs needed to capture 90% of the variance for the analysis.

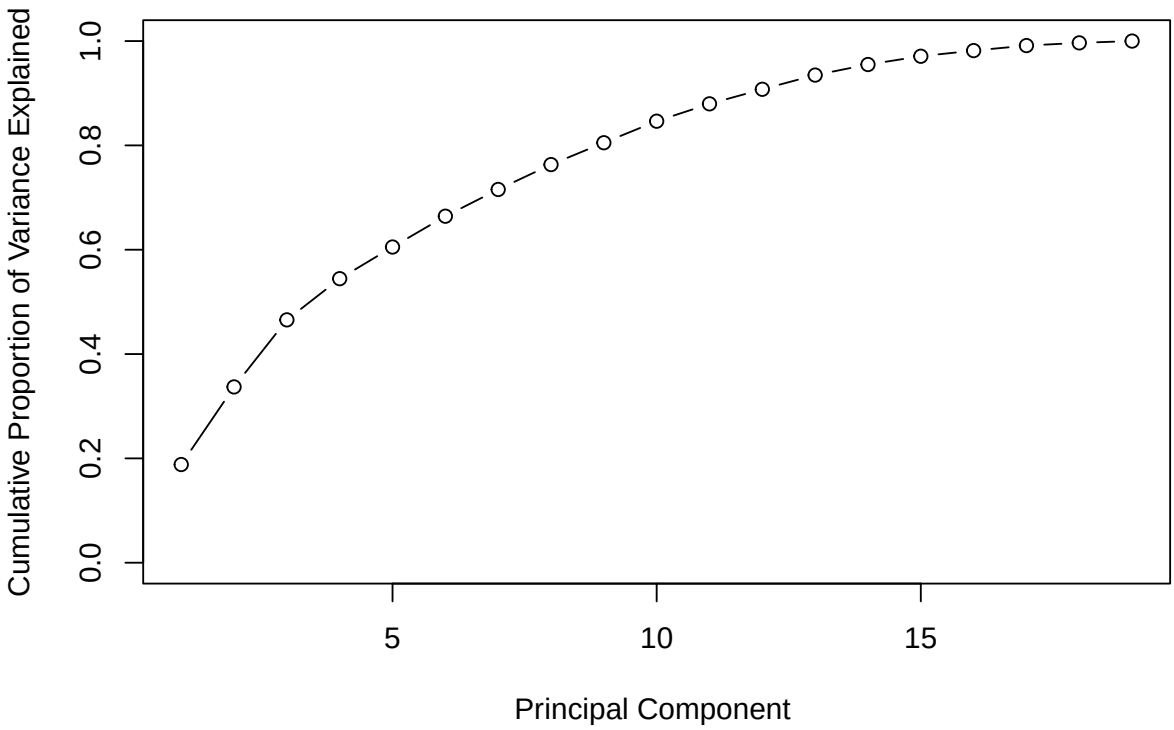
```r
# Calculate PVE for the PCA we ran
pr.var <- pr.out$sdev^2
pve <- pr.var/sum(pr.var)

# Add up the PVE for each PC
pve_cumsum <- cumsum(pve)

# This line finds how many PCs we need to explain 90% of the variance
length(which(pve_cumsum < 0.90)) + 1
```

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

The minimum number of PCs needed to capture 90% of the variance is **12**

- We will plot the proportion of variance explained (PVE) and cumulative PVE.





The first 5 PCs capture about 60% of the variation in the data, as we can see from this cumulative plot.

Clustering

- With `census.clean` (with `State` and `County` excluded), we will now perform hierarchical clustering with complete linkage.

```
census.clean_subset <- select(census.clean, -c(State, County))

# Compute euclidean distance matrix
census.clean.dist <- dist(census.clean_subset)

set.seed(333)
census.hclust <- hclust(census.clean.dist)

census.hclust
```

```
##
## Call:
## hclust(d = census.clean.dist)
##
## Cluster method   : complete
## Distance         : euclidean
## Number of objects: 3142
```

- Cut the tree to partition the observations into 10 clusters.

```
# This line partitions the results of hclust() into 10 clusters
cluster_assignment <- cutree(census.hclust, k=10)
```

- Re-run the hierarchical clustering algorithm using the first 2 principal components from `pc.county` as inputs instead of the original features.

```
pc.county.dist <- dist(pc.county)

set.seed(333)
census.pca_hclust <- hclust(pc.county.dist)

cluster_assignment.pca <- cutree(census.pca_hclust, k=10)
```

- Compare the results. For both approaches we will investigate the cluster that contains Santa Barbara County, the country which I resided in during my university years.

```
# Cluster assignments from hierarchical clustering on original features
table(cluster_assignment)
```

```
## cluster_assignment
##    1    2    3    4    5    6    7    8    9   10
## 3034   69    2    9   12    1    2    5    7    1
```

```
# Cluster assignments from hierarchical clustering on pc.county
table(cluster_assignment.pca)
```

```
## cluster_assignment.pca
##    1    2    3    4    5    6    7    8    9   10
## 1734  272   42   79  772   19  109    1  100   14
```

When the algorithm was run on the first 2 principal components from `pc.county`, we can see from the above tables that it made some noticeably different decisions on what clusters to assign data to. The original features algorithm had a majority of the data fall into Cluster 1. The PC algorithm's biggest cluster was also Cluster 1 to a lesser extent, and in addition the data was more evenly spread across Clusters 1, 2, 5, 7, and 9. In both algorithms run, there are some very small clusters that contain few observations. For example, in the original features algorithm, Cluster 3, 6, 7, and 10 contain only 1 or 2 observations. This suggests that we may be partitioning the data into *too many* clusters, and may try a smaller number of clusters if we were to rerun the algorithm. However, there will be more robust methods used later in this project so it will be left at ten clusters for now.

- Going back to Santa Barbara County, which cluster approach seemed to put it in a more appropriate cluster?

```
# First we have to find the index of Santa Barbara County
# This allows us to find its cluster assignment
which(census.clean$County == "Santa Barbara County")
```

```
## [1] 228
```

```
# The original features algorithm assigned cluster
cluster_assignment[228]
```

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

```
# The PC algorithm assigned cluster
cluster_assignment.pca[228]
```

```
## [1] 5
```

From the above code we can see that *Santa Barbara County* was assigned to Cluster 1 in the original features algorithm and Cluster 5 in the PC algorithm.

```
# Let's look at other counties in Cluster 2 for original features algorithm
cluster1 <- which(cluster_assignment == 1)
cluster1_df <- census.clean[cluster1, ]

# Other counties in Cluster 6 for PC algorithm
cluster5 <- which(cluster_assignment.pca == 5)
cluster5_df <- census.clean[cluster5, ]


mean(cluster1_df$WorkAtHome)
```

```
## [1] 4.797
```

```
mean(cluster5_df$WorkAtHome)
```

```
## [1] 6.185
```

Since Cluster 1 (size of 3034) in the original features algorithm contains so many counties and *Santa Barbara County* is in that cluster, intuitively it makes sense to say that the original features algorithm did a bad job of assigning it to a cluster. The PC algorithm assigned *Santa Barbara County* to Cluster 5 (size of 772), so it did a better job of assigning it to a cluster since the clusters are more well defined in that algorithm. We look at the mean values of `WorkAtHome` for the two clusters, since this was the most important feature in the first PC. There was a difference between the two, with Cluster 1 having a mean of 4.797 and Cluster 5 having a mean of 6.185. This might be due to the first PC placing more importance on this feature, thus it was possibly grouped as such when sorted into clusters. This is just a possible explanation, it is hard to tell definitively which algorithm did a better job assigning, but still interesting nonetheless.

Modeling

We start considering supervised learning tasks now. Our goal is to figure out *can we use census information as well as the education information in a county to predict the level of poverty in that county?*

We are interested in a binary classification problem. Specifically, we will transform `Poverty` into a binary categorical variable: high and low, and conduct its classification.

In order to build classification models, we first need to combine `education` and `census.clean` data (and removing all NAs).

```
# we join the two datasets
all <- census.clean %>%
  left_join(education, by = c("State"="State", "County"="County")) %>%
  na.omit
```

- We will transform the variable `Poverty` into a binary categorical variable with two levels: 1 if Poverty is greater than 20, and 0 if `Poverty` is smaller than or equal to 20. Features that are uninformative in classification tasks will be removed.

```
all <- all %>%
  mutate(Poverty=as.factor(ifelse(Poverty>20, 1, 0))) %>%
  select(-c(State, County))
```

- The commas and format of some column names from `education` set are causing issues with `tree()` and other modeling functions. We will rename them here so they are easier to work with as we continue.

```
colnames(all)
all <- setNames(all, c(colnames(all)[1:19], "LessThanHS", "HSonly", "SomeDegree",
                       "BachelorsOrHigher", "TotalPop.y"))
all
```

- Partitioning the dataset (80% training, 20% test)

```
set.seed(123)
n <- nrow(all)
idx.tr <- sample.int(n, 0.8*n)
all.tr <- all[idx.tr, ]
all.te <- all[-idx.tr, ]
```

- We will define 10 cross-validation folds:

```
set.seed(123)
nfold <- 10
folds <- sample(cut(1:nrow(all.tr), breaks=nfold, labels=FALSE))
```

- Error rate function setup shown here. The object `records` is used to record the classification performance of each method in the subsequent analysis.

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

Decision Trees

Decision tree: train a decision tree by cv.tree(). Prune tree to minimize misclassification error.

```
tree.all <- tree(Poverty~.,data=all.tr)

set.seed(420)
cv = cv.tree(tree.all, rand = folds, FUN=prune.misclass)


cv$size
```

```
## [1] 8 3 1
```

```
cv$dev
```

```
## [1] 414 414 552
```

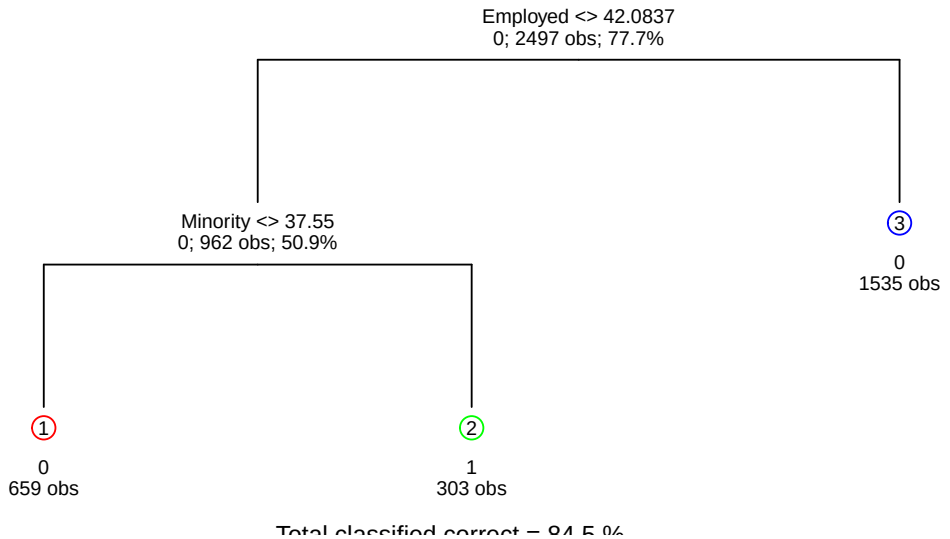
```
best.cv = min(cv$size[cv$dev == min(cv$dev)])
best.cv
```

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

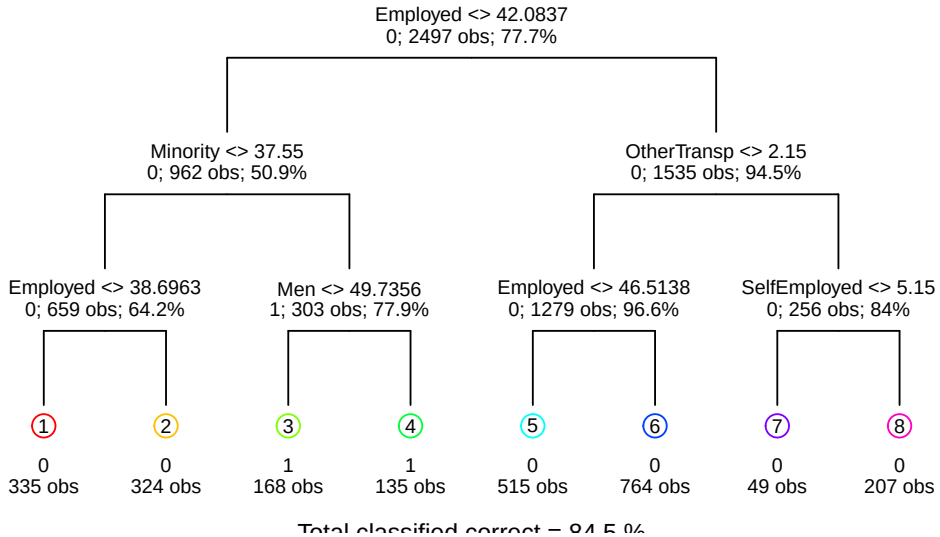
```
# Prune tree.all
pt.cv = prune.misclass(tree.all, best=best.cv)
```

- Visualize the trees before and after pruning.

```
library(maptree)

# Plot pruned tree
draw.tree(pt.cv, nodeinfo=TRUE, cex = 0.7)
title("Pruned Tree of Size 3")
```




```
# Plot the unpruned tree
draw.tree(tree.all, nodeinfo=TRUE, cex = 0.7)
title("Unpruned Tree")
```



- Save training and test errors to `records` object.

```
# Predict on test set
pred.pt.cv.te = predict(pt.cv, all.te, type="class")
# Test error rate (Classification Error)
tree.error.te <- calc_error_rate(pred.pt.cv.te, all.te$Poverty)


# Predict on training set
pred.pt.cv.tr = predict(pt.cv, all.tr, type="class")
# Training error rate (Classification Error)
tree.error.tr <- calc_error_rate(pred.pt.cv.tr, all.tr$Poverty)
```

```
records[1, 1] <- tree.error.tr
records[1, 2] <- tree.error.te
```

- Interpretation of decision trees:

[During this discussion, note that "in Poverty" means classified as a 1 or Poverty > 20.] The pruning for the decision tree proved very effective in this case, cutting down our terminal nodes from 8 to 3. This makes the tree more interpretable. Notice that the pruned and unpruned tree both first split on `Employed`, suggesting that this is the most important feature when classifying `Poverty` in a county. This makes sense because how much of a population is employed will obviously have a significant impact on that population being

in poverty or not. The only other feature the pruned tree considers for classification is `Minority`, where populations with higher `Minority` population have more `Poverty` (as they are classified as a 1 more often). Thus, the pruned tree is suggesting that populations who have low employment and high minority population are **more** likely to have high rates of Poverty and can be classified as such with the calculated error rates. On the other hand, the pruned tree automatically classifies any population with high employment (greater than 42%) as **not** in poverty.

- Quick Summary of Pruned Tree:
- High Employment Rate = Low Poverty Rate
- Low Employment Rate, Low Minority Population = Low Poverty Rate
- Low Employment Rate, High Minority Population = High Poverty Rate

Logistic Regression

- Run a logistic regression to predict `Poverty` in each county.

```
glm.fit <- glm(Poverty~., data=all.tr, family=binomial)
```

- Save training and test errors to records variable.

```
pred.labels.te <- as.numeric(predict(glm.fit, all.te, type="response") > 0.5)
pred.labels.tr <-as.numeric(predict(glm.fit, all.tr, type="response") > 0.5)

# Test error rate (Classification Error)
log_reg.error.te <- calc_error_rate(pred.labels.te, all.te$Poverty)

# Training error rate (Classification Error)
log_reg.error.tr <- calc_error_rate(pred.labels.tr, all.tr$Poverty)
```

```
records[2, 1] <- log_reg.error.tr
records[2, 2] <- log_reg.error.te
```

- Are the significant variables consistent with the results from the decision tree?

```
summary(glm.fit)
```

```
##
## Call:
## glm(formula = Poverty ~ ., family = binomial, data = all.tr)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -4.312  -0.419  -0.158  -0.003   3.422
##
## Coefficients: (1 not defined because of singularities)
##                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)        2.75e+01   4.41e+00    6.24  4.4e-10 ***
## TotalPop.x         1.58e-04   1.99e-05    7.93  2.2e-15 ***
## Men               -3.47e-01   2.98e-02  -11.65  < 2e-16 ***
## Minority           3.74e-02   4.84e-03    7.72  1.1e-14 ***
## VotingAgeCitizen   4.44e-02   1.98e-02    2.25  0.02465 *
## Professional       5.26e-02   2.54e-02    2.07  0.03819 *
## Service            9.23e-02   2.89e-02    3.19  0.00141 **
## Office             9.22e-03   3.07e-02    0.30  0.76397
## Production         8.08e-02   2.32e-02    3.49  0.00049 ***
## Drive             -5.08e-02   2.98e-02   -1.70  0.08842 .
## Carpool           -1.51e-03   3.74e-02   -0.04  0.96775
## Transit            9.69e-02   6.46e-02    1.50  0.13352
## OtherTransp       -1.17e-01   6.76e-02   -1.73  0.08335 .
## WorkAtHome        -1.29e-01   4.83e-02   -2.68  0.00739 **
## MeanCommute       -2.79e-02   1.64e-02   -1.71  0.08795 .
## Employed          -2.97e-01   1.98e-02  -15.05  < 2e-16 ***
## PrivateWork       -2.59e-02   1.67e-02   -1.55  0.12174
```

```
## SelfEmployed      -4.02e-02   3.20e-02   -1.26  0.20865
## FamilyWork        -1.31e-01   1.82e-01   -0.72  0.47037
## LessThanHS        -1.93e-04   3.71e-05   -5.20  2.0e-07 ***
## HSonly            -2.05e-04   3.04e-05   -6.75  1.5e-11 ***
## SomeDegree        -3.54e-04   4.58e-05   -7.74  1.0e-14 ***
## BachelorsOrHigher -2.11e-04   3.20e-05   -6.59  4.4e-11 ***
## TotalPop.y               NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2650.6  on 2496  degrees of freedom
## Residual deviance: 1366.3  on 2474  degrees of freedom
## AIC: 1412
##
## Number of Fisher Scoring iterations: 9
```

In above results, `TotalPop.x`, `Men`, `Minority`, `Service`, `Production`, `WorkAtHome`, `Employed`, `LessThanHS`, `HSonly`, `SomeDegree`, and `BachelorsOrHigher` are all significant at level 0.01. This is somewhat inconsistent with what we saw in the decision tree analysis. While both methods recognized `Employed` and `Minority` as an significant predictor, that is their only similarity in variables when comparing the pruned tree variables to the logistic regression significant variables. The only significant logistic regression variables that appear in the *unpruned* tree are `Employmed`, `Minority`, and `Men`. The logistic regression includes all the education-based variables, which would seem important when trying to predict `Poverty`, so that makes sense. The unpruned tree instead opts to focus on `SelfEmployed` and `OtherTransp`, which are not significant in the logistic regression.

- A couple helpful interpretations on logistic regressions for the reader:

The variable `Service` has a coefficient 0.0923. For every one unit change in `Service`, the log odds of a county being in poverty (meaning they have a `Poverty` rate greater than 20%) *increases* by 0.0923, holding other variables fixed.

The variable `BachelorsOrHigher` has a coefficient -0.000211. For every one unit change in `BachelorsOrHigher`, the log odds of a county being in poverty (meaning they have a `Poverty` rate greater than 20%) *decreases* by -0.000211, holding other variables fixed.

Cross-Validation and LASSO

Use the `cv.glmnet` function from the `glmnet` library to run a 10-fold cross validation and select the best regularization parameter for the logistic regression with LASSO penalty. Set `lambda = seq(1, 20) * 1e-5` in `cv.glmnet()` function to set pre-defined candidate values for the tuning parameter λ.

```
set.seed(123)

dat <- model.matrix(Poverty~., all)

nrow <- nrow(all)
train = sample.int(n, 0.8*n)

x.train = dat[train, ]
y.train = all[train, ]$Poverty

x.test = dat[-train, ]
y.test = all[-train, ]$Poverty


lambda.list.lasso <- seq(1, 20) * 1e-5

cv.lasso.lambda <- cv.glmnet(x.train, y.train, family = "binomial", alpha = 1, nfolds = 10,
                             lambda = lambda.list.lasso)

all.lasso.fit <- glmnet(x.train, y.train, alpha = 1, family = "binomial", lambda = cv.lasso.lambda$lambd
```

- We need to know the optimal value of λ in cross validation.

```
# Display and store optimal value of lambda
bestlam <- cv.lasso.lambda$lambda.min
cv.lasso.lambda$lambda.min
```

```
## [1] 1e-05
```

The optimal value of λ in cross validation is 0.00001.

- What are the non-zero coefficients in the LASSO regression for the optimal value of λ? How do they compare to the unpenalized logistic regression?

```
# Displays lasso coefficients corresponding to chosen value of lambda
coef(cv.lasso.lambda, bestlam)
```

```
## 25 x 1 sparse Matrix of class "dgCMatrix"
##                              s1
## (Intercept)       27.7009593
## (Intercept)        .
## TotalPop.x         0.0001467
## Men               -0.3449103
## Minority           0.0372842
## VotingAgeCitizen   0.0427594
## Professional       0.0532179
```

```
## Service            0.0917317
## Office             0.0089543
## Production         0.0810699
## Drive             -0.0525401
## Carpool           -0.0030537
## Transit            0.0870285
## OtherTransp       -0.1159912
## WorkAtHome        -0.1310122
## MeanCommute       -0.0281208
## Employed          -0.2958994
## PrivateWork       -0.0265168
## SelfEmployed      -0.0430042
## FamilyWork        -0.1322283
## LessThanHS        -0.0001759
## HSonly            -0.0001928
## SomeDegree        -0.0003317
## BachelorsOrHigher -0.0001940
## TotalPop.y         .
```

All of the coefficients are non-zero in the LASSO regression for optimal λ value except for the feature called `TotalPop.y`, which is the total population we created earlier in the `education` dataset. This is similar to the unpenalized logistic regression, where `TotalPop.y` was also ignored in the model output, which is likely due to us already having a TotalPop column and values from the `census` dataset. Many of the coefficients in the LASSO regression are very close to zero, suggesting that they were close to being feature selected by the model. This comparison between the two models tells us that there is at least a little bit of information to be gained from all variables we included.

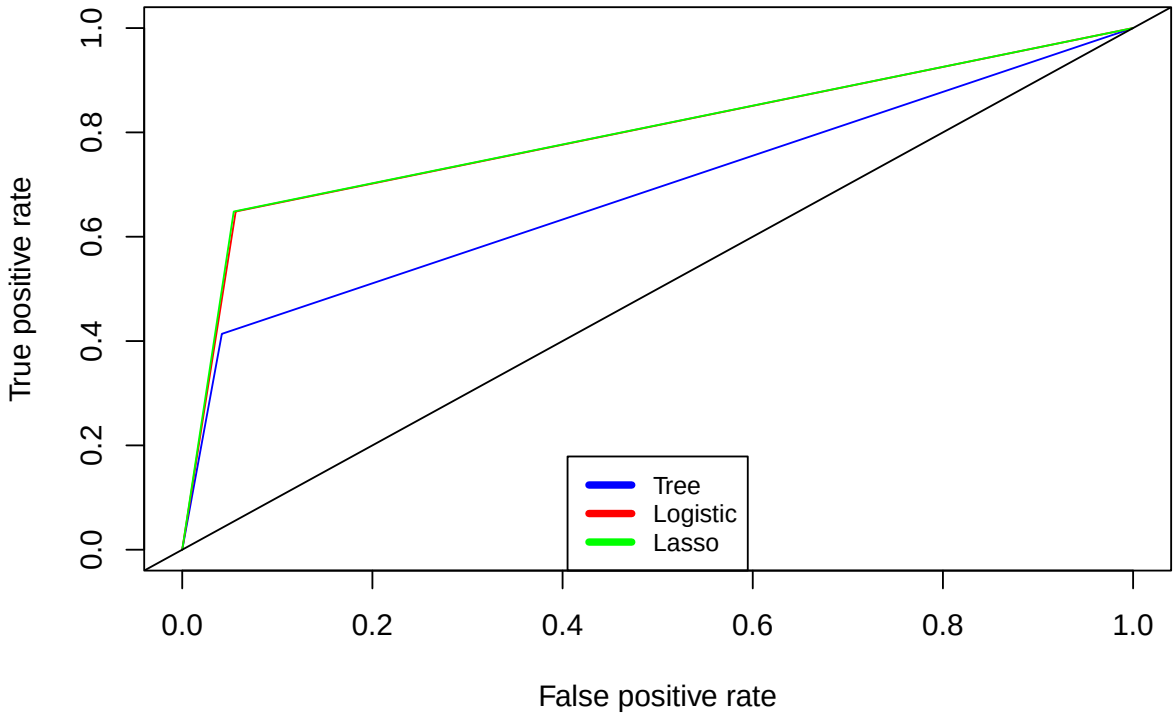
```
# Make predictions using training and test set data.
lasso.pred.train = as.numeric(predict(all.lasso.fit, s = bestlam, newx = dat[train,], type = "class"))
lasso.pred.test = as.numeric(predict(all.lasso.fit, s = bestlam, newx = dat[-train,], type = "class"))

# Test error rate (Classification Error)
lasso.error.te <- calc_error_rate(lasso.pred.test, all.te$Poverty)

# Training error rate (Classification Error)
lasso.error.tr <- calc_error_rate(lasso.pred.train, all.tr$Poverty)
```

```
records[3, 1] <- lasso.error.tr
records[3, 2] <- lasso.error.te

# View the now completed records table
records
```

```
##          train.error test.error
## tree          0.1554     0.1680
## logistic      0.1233     0.1248
## lasso         0.1233     0.1232
```

Performance Comparison (Decision Trees, Logistic Regression, LASSO regression)

- They will be compared using ROC curves and AUC values.




```
# Compute AUC values to see performance of model
auc_tree <- performance(pred_tree, "auc")@y.values
auc_log <- performance(pred_log, "auc")@y.values
auc_lasso <- performance(pred_lasso, "auc")@y.values

# Decision Tree AUC
auc_tree
```

```
## [[1]]
## [1] 0.6861
```

```
# Logistic Regression AUC
auc_log
```

```
## [[1]]
## [1] 0.796
```

```
# LASSO AUC
auc_lasso
```

```
## [[1]]
## [1] 0.7971
```

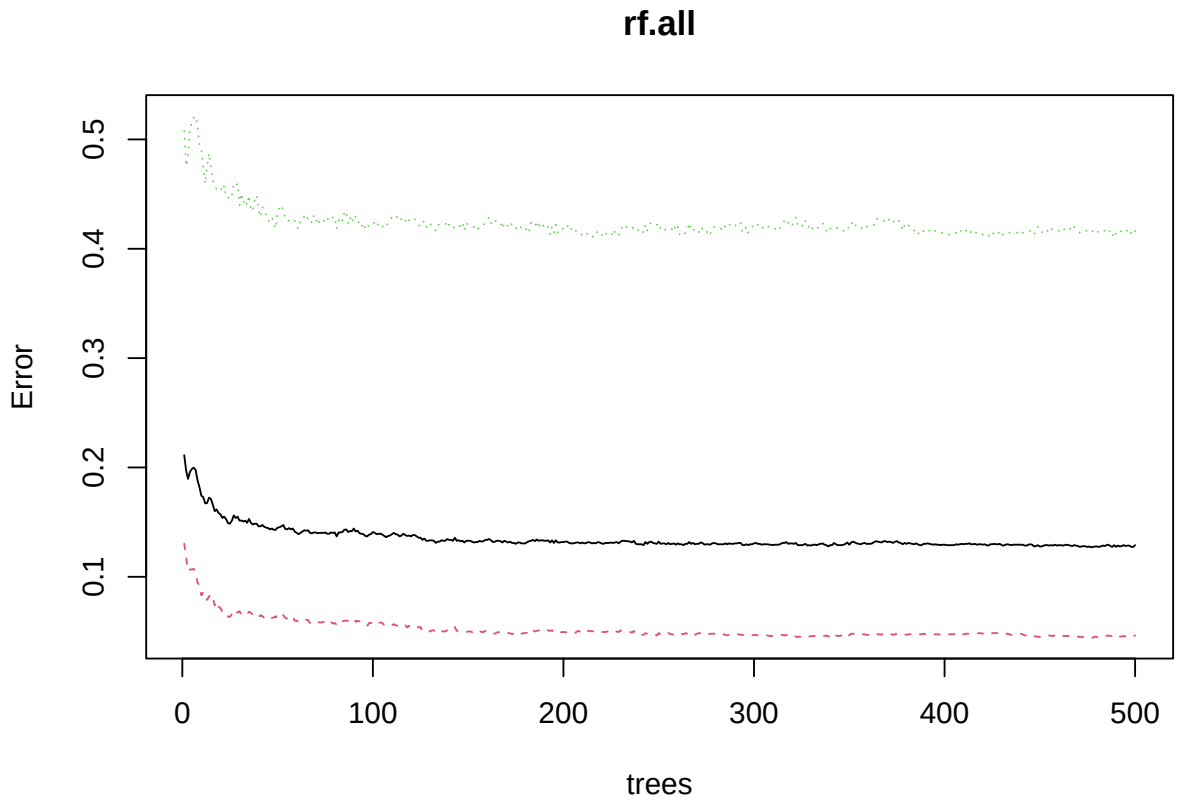
The LASSO logistic regression has the highest AUC out of all the methods, which means it is the best at classifying a county as in Poverty or not in Poverty. The logistic regression is just barely behind the LASSO logistic regression in terms of performance, meaning it is nearly as good at classifying. The logistic regression trades a small amount of performance for easier computation and less coding than the LASSO logistic regression required. The major con of the decision tree is that it had the lowest AUC by a large margin, and is noticeably lower in performance when looking at the ROC curves above. This suggests that decision trees are less appropriate for classifying a county as in Poverty or not. This is likely because decision trees suffer from the *curse of dimensionality*, and our dataset has many dimensions with most of them being non-binary. If we had a dataset with fewer predictor variables, it may be possible for decision trees to perform better in answering questions about Poverty levels in a county.

(Reminder: "In Poverty" refers to a binary value of 1, which is a county with poverty rate higher than 20%)

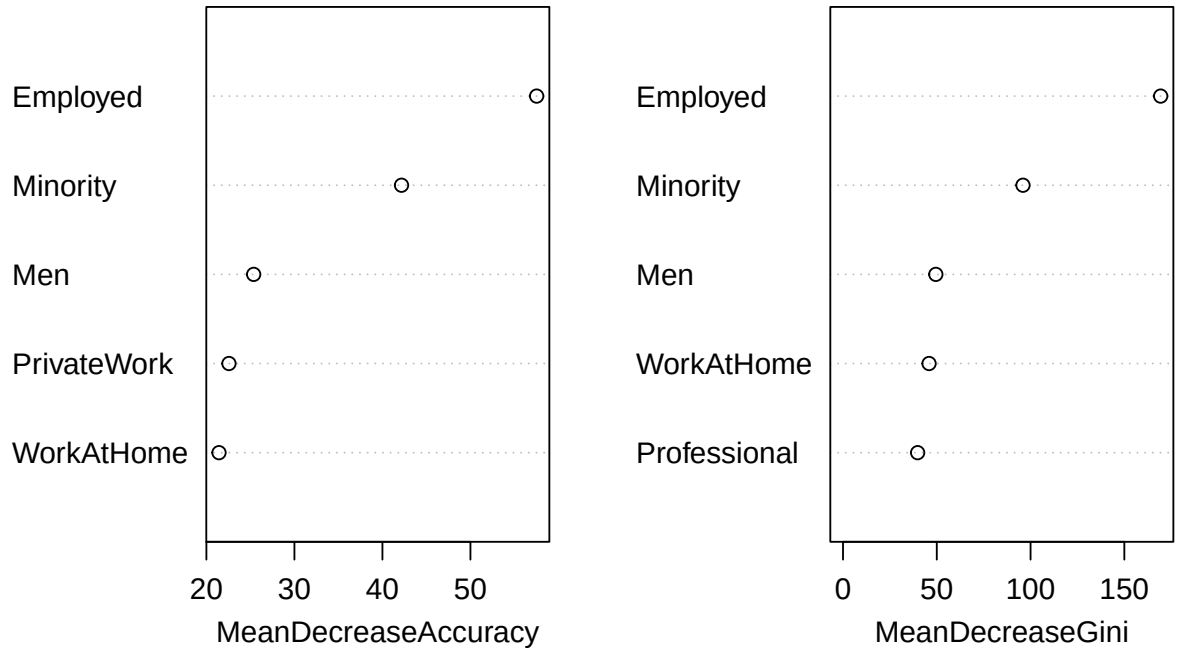
Random Forests and Boosting Methods

- We will perform *random forests* method on the dataset.

```
set.seed(333)
# Fit a random forest model on training data
rf.all <- randomForest(Poverty ~ ., data=all.tr, importance=TRUE)
```




```
# Plot and check variable importance in rf.all
varImpPlot(rf.all, sort=T,
           main="Variable Importance for rf.all", n.var=5)
```



It is important to notice that random forests, across all trees considered, also places a high importance on the `Employed` and `Minority` variables, just like our decision tree, logistic regression, and LASSO logistic regression did. These variables are clearly very important when trying to assign a `Poverty` classifier due to their continued significance in all of our models. It is interesting that the random forest considers `PrivateWork` and `Professional` as important variables, as these were not considered significant in any of the previous models discussed.

```
print(rf.all)
```

```
##
## Call:
##  randomForest(formula = Poverty ~ ., data = all.tr, importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 12.9%
## Confusion matrix:
##      0   1 class.error
## 0 1850  90     0.04639
## 1  232 325     0.41652
```

This summary tells us a few things. First, that 500 trees were used to fit the data with 4 variables randomly considered at each split in the trees. The OOB estimate of error rate for our random forest model is 12.9%,

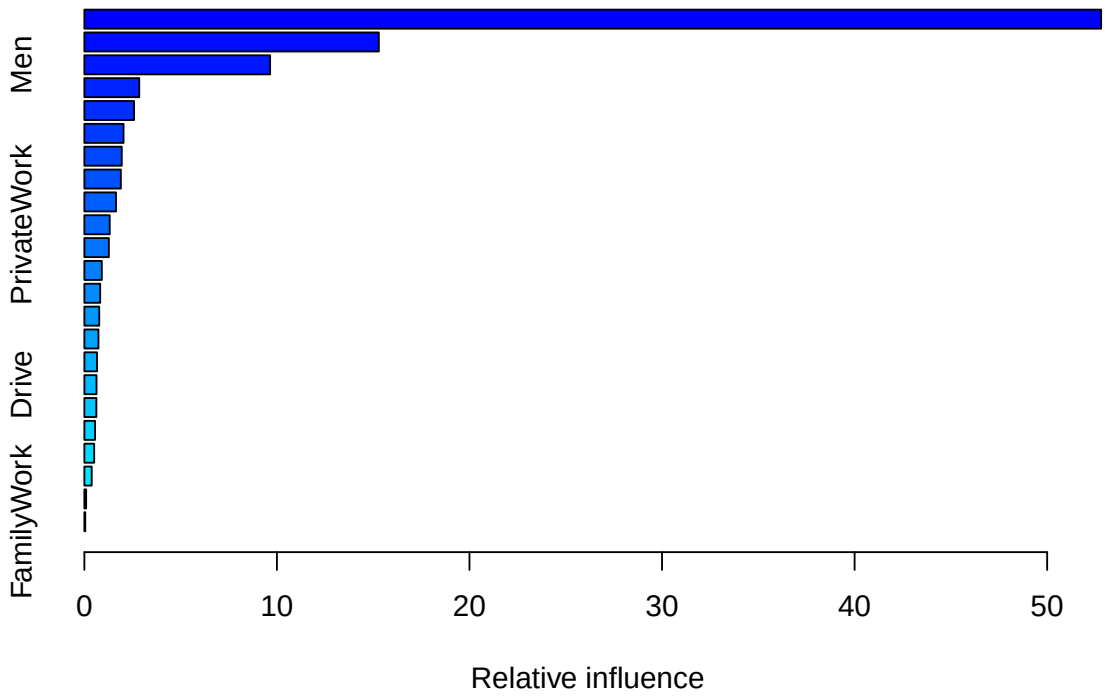
which is comparable to our previous models so the parameters used during model fitting are likely fine. Knowing this, we can continue with the analysis and make predictions.

```
# Make predictions using our random forest model
pred.rf.te = predict(rf.all, newdata = all.te)
pred.rf.tr = predict(rf.all, newdata = all.tr)

# Calculate error rates for random forest
test.rf.err <- calc_error_rate(pred.rf.te, all.te$Poverty)
train.rf.err <- calc_error_rate(pred.rf.tr, all.tr$Poverty)
```

- Secondly, we will run a boosting classification method on the data. Boosting was chosen because it is a good comparison to add, as it does not use bootstrap sampling like random forests. This could potentially provide new insight or better performance than previous methods.

```
set.seed(714)

# Fit the model to training data
# interaction.depth parameter is set to 2 to catch interactions

boost.all <- gbm(ifelse(Poverty==1,1,0)~., data=all.tr, distribution="bernoulli", n.trees=1000,
                 shrinkage=0.01, interaction.depth = 2)
```

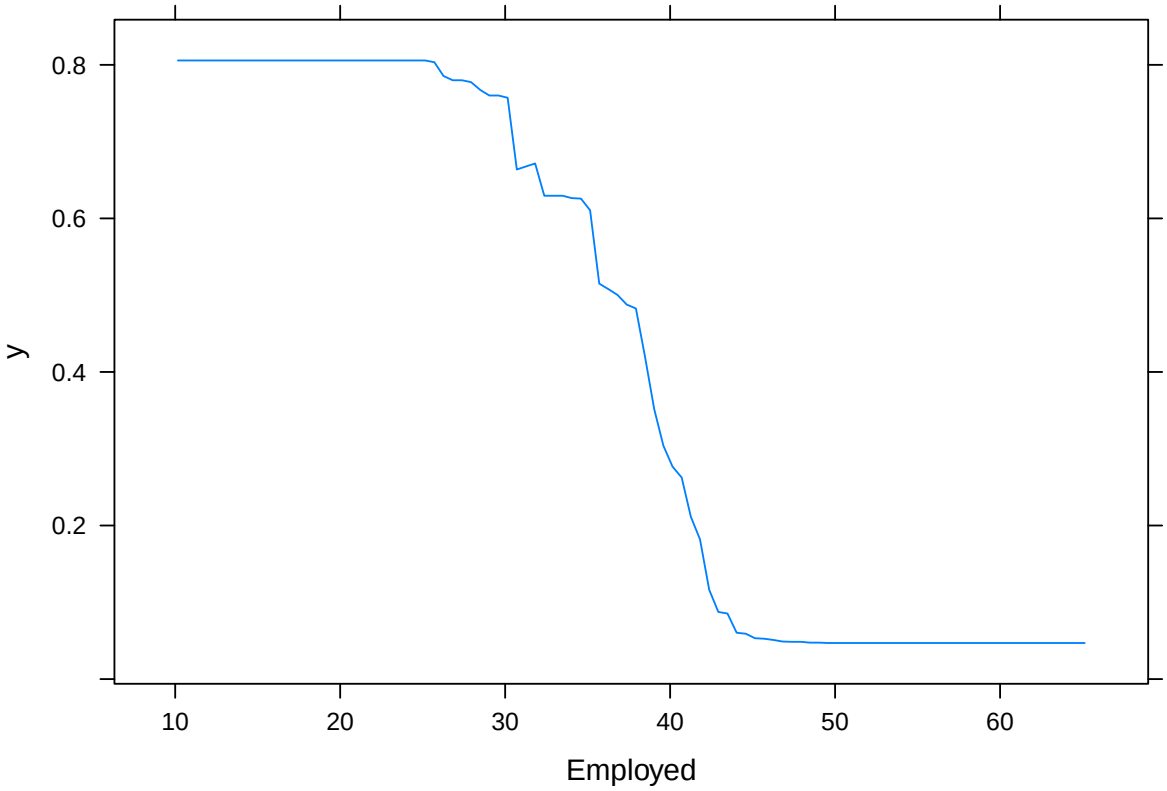
```
# View summary of our model and see what we can learn
summary(boost.all)
```



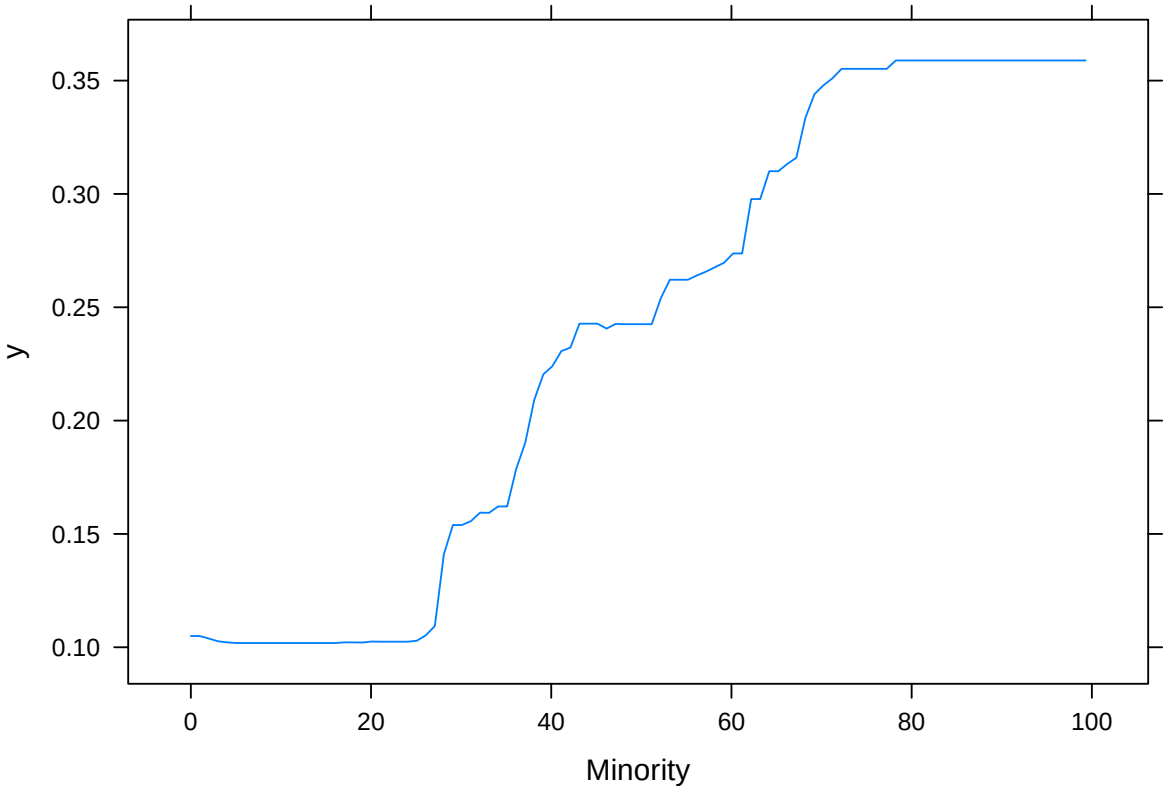
```
##                                   var  rel.inf
## Employed                     Employed 52.80359
## Minority                     Minority 15.29201
## Men                               Men  9.64648
## VotingAgeCitizen   VotingAgeCitizen  2.84967
## WorkAtHome               WorkAtHome  2.58080
## Service                     Service  2.03428
## Professional           Professional  1.94137
## MeanCommute             MeanCommute  1.89366
## Production               Production  1.64340
## PrivateWork             PrivateWork  1.31657
## SelfEmployed           SelfEmployed  1.27119
## OtherTransp             OtherTransp  0.90506
## SomeDegree               SomeDegree  0.82156
## Transit                     Transit  0.77323
## TotalPop.x               TotalPop.x  0.73217
## Office                       Office  0.65531
## Drive                         Drive  0.63079
## LessThanHS               LessThanHS  0.62350
## Carpool                     Carpool  0.55602
## BachelorsOrHigher BachelorsOrHigher  0.50987
## HSonly                       HSonly  0.38090
## TotalPop.y               TotalPop.y  0.09247
## FamilyWork               FamilyWork  0.04610
```

The boosting model places the most priority on the `Employed` and the `Minority` variables, with `Employed` being far more important than the other variables. This is a good thing as it shows consistency with our other models and their variable priority. The third most influential variable in this model is `Men`, with the rest of the variables all falling into similar levels of influence. We have seen `Men` show up as an important variable in other models too, notably placing 3^{rd} in our random forest model.

```
# Now we will observe partial dependence plots

par(mfrow = c(1,2))

# Partial dependence plot for #1 variable, Employed
plot(boost.all ,i="Employed", type = "response")
```




```
# Partial dependence plot for #2 variable, Minority
plot(boost.all ,i="Minority", type = "response")
```



While we knew before that `Employed` and `Minority` were important, these plots are interesting because they show us just how important they are in predicting `Poverty`. In the `Employed` partial dependence plot, we can see once employment rate of a county passes around 30%, the prediction nosedives and quickly flattens out. This means that past that point the county is more likely to be classified as a 0, or "not in Poverty". The `Minority` partial dependence plot shows the opposite trend, where it gradually climbs, which makes sense because as `Minority` increases, so should `Poverty`.

```
# Make predictions using our boosting model
pred.boost.te <- as.numeric(predict(boost.all, newdata = all.te, n.trees=1000, type = "response") > 0.5)
pred.boost.tr <- as.numeric(predict(boost.all, newdata = all.tr, n.trees=1000, type = "response") > 0.5)

# Calculate error rates boosting
test.boost.err <- calc_error_rate(pred.boost.te, all.te$Poverty)
train.boost.err <- calc_error_rate(pred.boost.tr, all.tr$Poverty)
```

```
# Let's make a new records table to compare to our previous methods
records2 = matrix(NA, nrow=5, ncol=2)
colnames(records2) = c("train.error","test.error")
rownames(records2) = c("tree","logistic","lasso", "random forest", "boosting")
```

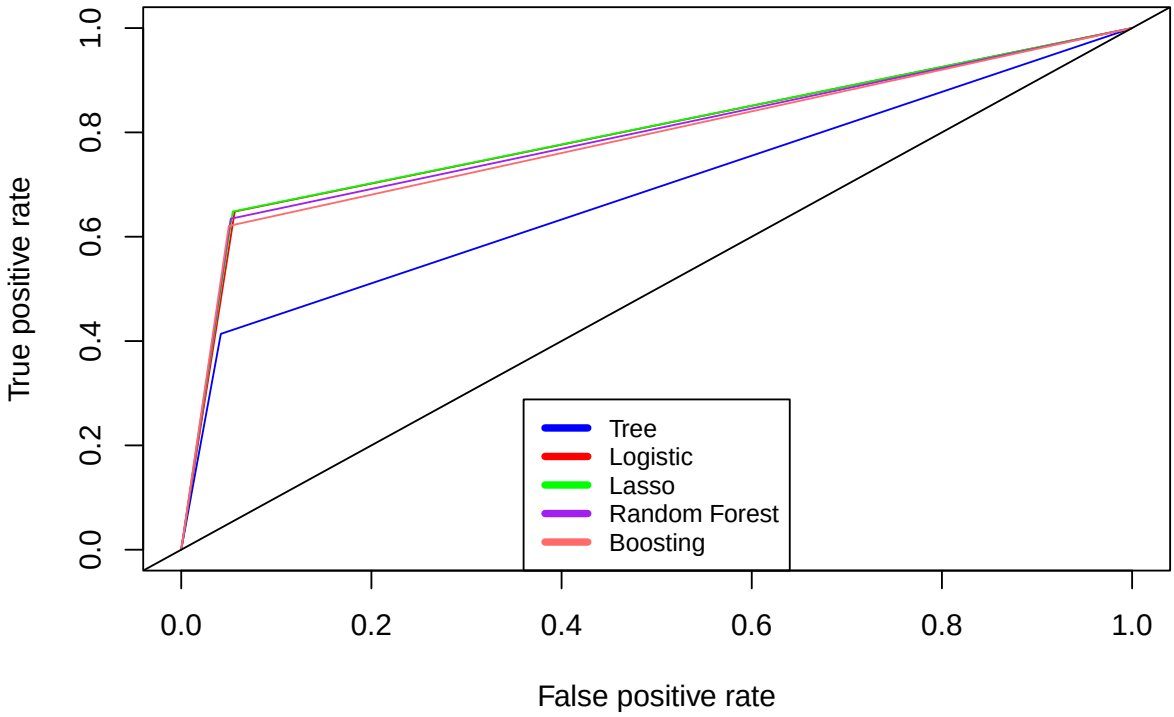
```
# Observe error comparison of all models in the report
records2
```

```
##               train.error test.error
## tree               0.1554     0.1680
```

```
## logistic            0.1233     0.1248
## lasso               0.1233     0.1232
## random forest       0.0000     0.1248
## boosting            0.1061     0.1264
```

As we can see from the table, LASSO logistic regression still has the lowest test error rate out of all the methods. The two methods tried in this part of the report did not improve upon our best, but they still both performed better than decision tree model. Random forest had the exact same test error rate as logistic regression and actually performed better than boosting did.

```
# First arument is the predicted labels, second is true labels
pred_tree = prediction(as.numeric(pred.pt.cv.te), all.te$Poverty)
pred_log  = prediction(pred.labels.te, all.te$Poverty)
pred_lasso = prediction(lasso.pred.test, all.te$Poverty)
pred_rf = prediction(as.numeric(pred.rf.te)-1, all.te$Poverty)
pred_boost = prediction(pred.boost.te, all.te$Poverty)

# We want TPR on the y axis and FPR on the x axis
perf_tree = performance(pred_tree, measure="tpr", x.measure="fpr")
perf_log = performance(pred_log, measure="tpr", x.measure="fpr")
perf_lasso = performance(pred_lasso, measure="tpr", x.measure="fpr")
perf_rf = performance(pred_rf, measure="tpr", x.measure="fpr")
perf_boost = performance(pred_boost, measure="tpr", x.measure="fpr")

plot(perf_tree, col='blue')
plot(perf_log, add = TRUE, col = 'red')
plot(perf_lasso, add = TRUE, col = 'green')
plot(perf_rf, add = TRUE, col = 'purple')
plot(perf_boost, add = TRUE, col = 'indianred1')

legend("bottom",
       legend=c("Tree", "Logistic", "Lasso", "Random Forest", "Boosting"),
       col=c("blue", "red", "green", 'purple', 'indianred1'),
       lwd=4, cex =0.8, xpd = TRUE, horiz = FALSE)

abline(0,1)
```



This ROC plot just shows what was discussed above, where we can visually see Random Forest and Boosting slightly underperforming when compared to LASSO. As a side note, running a kNN classifier was considered but was quickly discarded when seeing how the curse of dimensionality affect the decision tree method.

We will now use regression models to predict the actual value of `Poverty` (before we transformed `Poverty` to a binary variable) by county. It would be of interest to compare and contrast these results with the classification models.

We will perform a linear regression as a baseline test for predicting `Poverty`. we will also apply a regularized regression method, LASSO, since we already performed one earlier and it can be used for both classification and regression problems. Lastly, I will perform a decision tree to solve this regression problem. Since decision tree performed the worst out of all of our classification models, it will be interesting to see if when given a regression problem it improves or performs even worse. Note that much of the model fitting code will be hidden from the knitted file, with a greater emphasis on model results being shown.

Analysis of each model individually will be lightly analyzed, only showing the code and results with a brief analysis. There will be a heavier focus on a comparison of the *performance* differences between the regression and classification versions of each model at the end.

```
# First, create new dataset all_reg
# Same as all but with Poverty as a value
all_reg <- census.clean %>%
  left_join(education, by = c("State"="State", "County"="County")) %>%
  na.omit

# Remove uninformative features
all_reg <- all_reg %>%
  select(-c(State, County))

# Fix column names
all_reg <- setNames(all_reg, c(colnames(all)[1:19], "LessThanHS", "HSonly", "SomeDegree",
                        "BachelorsOrHigher", "TotalPop.y"))
```

```
# 80% training, 20% test data partition
set.seed(123)
n <- nrow(all_reg)
idx.tr <- sample.int(n, 0.8*n)
all_reg.tr <- all_reg[idx.tr, ]
all_reg.te <- all_reg[-idx.tr, ]
```

- **linear regression**

```
# Fit a linear model
fit.linear <- lm(Poverty~., data=all_reg.tr)
```

```
# Check coefficients
summary(fit.linear)
```

```
##
## Call:
## lm(formula = Poverty ~ ., data = all_reg.tr)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -16.70  -2.24  -0.24   1.93  20.25
##
## Coefficients: (1 not defined because of singularities)
```

```
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)        8.17e+01   4.61e+00   17.72  < 2e-16 ***
## TotalPop.x         3.63e-05   6.80e-06    5.34  1.0e-07 ***
## Men               -6.64e-01   3.39e-02  -19.61  < 2e-16 ***
## Minority           8.25e-02   5.70e-03   14.47  < 2e-16 ***
## VotingAgeCitizen   6.28e-02   2.05e-02    3.07  0.00217 **
## Professional       2.15e-02   2.59e-02    0.83  0.40683
## Service            1.87e-01   3.20e-02    5.85  5.7e-09 ***
## Office            -1.52e-02   3.47e-02   -0.44  0.66255
## Production         1.97e-01   2.68e-02    7.37  2.3e-13 ***
## Drive             -9.76e-02   3.02e-02   -3.23  0.00127 **
## Carpool           -3.57e-02   4.16e-02   -0.86  0.39062
## Transit            1.18e-02   4.78e-02    0.25  0.80432
## OtherTransp       -1.15e-01   7.25e-02   -1.59  0.11239
## WorkAtHome        -6.83e-02   5.07e-02   -1.35  0.17807
## MeanCommute       -1.35e-01   1.68e-02   -8.00  1.9e-15 ***
## Employed          -6.28e-01   1.75e-02  -35.89  < 2e-16 ***
## PrivateWork       -7.45e-02   1.76e-02   -4.24  2.3e-05 ***
## SelfEmployed      -1.13e-01   3.31e-02   -3.41  0.00066 ***
## FamilyWork         2.75e-01   1.87e-01    1.47  0.14093
## LessThanHS        -4.74e-05   1.40e-05   -3.39  0.00072 ***
## HSonly            -5.07e-05   1.28e-05   -3.97  7.4e-05 ***
## SomeDegree        -7.24e-05   1.40e-05   -5.17  2.6e-07 ***
## BachelorsOrHigher -4.34e-05   9.27e-06   -4.69  2.9e-06 ***
## TotalPop.y               NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.77 on 2474 degrees of freedom
## Multiple R-squared:  0.669,  Adjusted R-squared:  0.666
## F-statistic:  227 on 22 and 2474 DF,  p-value: <2e-16
```

```
summary(fit.linear)$coef[summary(fit.linear)$coef[,4] <= .01, 4]
```

```
##       (Intercept)        TotalPop.x               Men          Minority
##         2.987e-66         1.013e-07         1.114e-79         1.346e-45
##  VotingAgeCitizen           Service        Production             Drive
##         2.172e-03         5.668e-09         2.302e-13         1.266e-03
##       MeanCommute          Employed       PrivateWork      SelfEmployed
##         1.861e-15        2.000e-227         2.307e-05         6.614e-04
##        LessThanHS            HSonly        SomeDegree BachelorsOrHigher
##         7.216e-04         7.407e-05         2.550e-07         2.925e-06
```

Let's compare the variables significant at a 0.01 level in the linear regression model to the ones in the logistic regression model.

```
# Linear Regression significant variables
summary(fit.linear)$coef[summary(fit.linear)$coef[,4] <= .01, 1]
```

```
##       (Intercept)        TotalPop.x               Men          Minority
##         8.171e+01         3.631e-05        -6.644e-01         8.249e-02
##  VotingAgeCitizen           Service        Production             Drive
##         6.282e-02         1.873e-01         1.974e-01        -9.755e-02
```

```
##       MeanCommute          Employed       PrivateWork      SelfEmployed
##        -1.346e-01        -6.280e-01         -7.448e-02        -1.128e-01
##         LessThanHS            HSonly        SomeDegree BachelorsOrHigher
##        -4.737e-05        -5.072e-05         -7.240e-05        -4.344e-05
```

```
# Logistic Regression significant variables
summary(glm.fit)$coef[summary(glm.fit)$coef[,4] <= .01, 1]
```

```
##       (Intercept)        TotalPop.x               Men          Minority
##        27.5079904         0.0001579        -0.3467878         0.0373603
##           Service        Production        WorkAtHome          Employed
##         0.0922989         0.0807531        -0.1292717        -0.2974525
##         LessThanHS            HSonly        SomeDegree BachelorsOrHigher
##        -0.0001926        -0.0002048        -0.0003545        -0.0002111
```

The linear regression model identified 16 significant variables while the logistic regression model identified 12. The only significant variable unique to the regression model is `WorkAtHome`.

The variable `Service` has a coefficient 0.1873 in the linear model. For every one unit change in `Service`, the predicted percentage of impoverished population in a county *increases* by 0.1873, holding other variables fixed. Important to note that this **does not** mean that the percentage increases by 18.73%, but rather 0.1873% as stated.

- **lasso regression model**

```
# Display and store optimal value of lambda
bestlam2 <- cv.lasso.lambda2$lambda.min
cv.lasso.lambda2$lambda.min
```

```
## [1] 2e-04
```

```
# Displays lasso coefficients corresponding to chosen value of lambda
coef(cv.lasso.lambda2, bestlam2)
```

```
## 25 x 1 sparse Matrix of class "dgCMatrix"
##                              s1
## (Intercept)        8.194e+01
## (Intercept)        .
## TotalPop.x         3.096e-05
## Men               -6.640e-01
## Minority           8.246e-02
## VotingAgeCitizen   6.003e-02
## Professional       2.320e-02
## Service            1.878e-01
## Office            -1.434e-02
## Production         1.987e-01
## Drive             -9.820e-02
## Carpool           -3.651e-02
## Transit            9.498e-03
## OtherTransp       -1.177e-01
## WorkAtHome        -6.991e-02
## MeanCommute       -1.346e-01
```

```
## Employed          -6.274e-01
## PrivateWork       -7.550e-02
## SelfEmployed      -1.146e-01
## FamilyWork         2.751e-01
## LessThanHS        -5.835e-05
## HSonly            -6.424e-05
## SomeDegree        -8.446e-05
## BachelorsOrHigher -5.707e-05
## TotalPop.y         2.055e-05
```

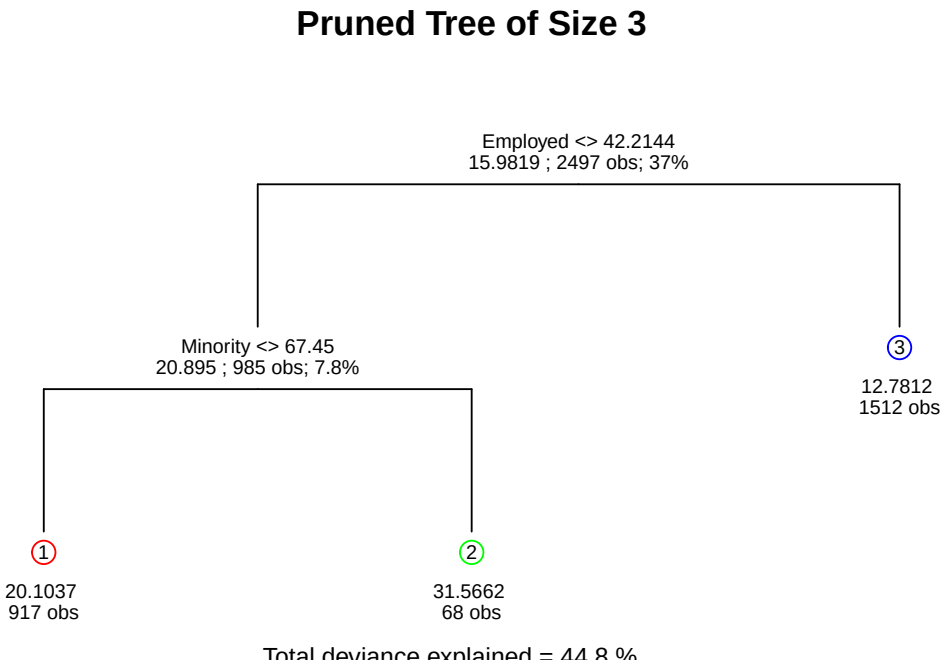
There are no coefficients set to zero this time when using LASSO to solve this regression problem. When done with classification, `TotalPop.y` was set to zero, while it remains a small number here. It might have been wise to remove one of the population variables at an earlier stage in data analysis, so we are not dealing with two variables with different values that represent the same thing.

- **decision tree (regression)**

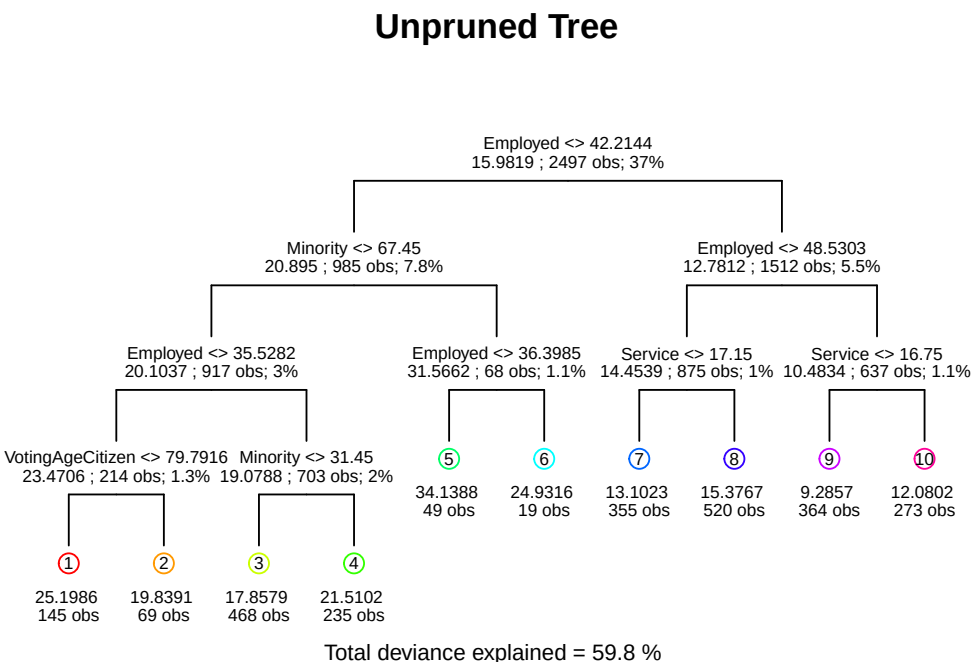
```
## [1] 3
```

- Visualize the trees before and after pruning.

```
library(maptree)

# Plot pruned tree
draw.tree(pt.cv_reg, nodeinfo=TRUE, cex = 0.7)
title("Pruned Tree of Size 3")
```



```
# Plot the unpruned tree
draw.tree(tree.all_reg, nodeinfo=TRUE, cex = 0.6)
title("Unpruned Tree")
```



In terms of the variables included in the pruned tree, the regression and classification versions look quite similar, both opting for a 3 terminal node tree using only `Employed` and `Minority`. The unpruned tree is a little different, as it has 10 terminal nodes compared to the 8 we saw in the classification decision tree. This classiciation tree also opts to include `VotingAgeCitizen` and `Service`, which have both shown up in our other regression problem models consistently.

Let's look at and form a table of MSEs for the methods above, similar to how was done for the classification methods.

```
# LINEAR REGRESSION MSE
# training mse
linear_training_mse <- mean(summary(fit.linear)$residuals^2)

# test mse
linear_test_mse <- mean((all_reg.te$Poverty - predict.lm(fit.linear, all_reg.te))^2)

# Display both values
print(linear_training_mse)
```

```
## [1] 14.07
```

```
print(linear_test_mse)
```

```
## [1] 15.23
```

Our linear regression model has a training MSE of 14.07 and a test MSE of 15.23.

```
# LASSO MSE
# Make predictions using training and test set data.
lasso.pred.train_reg = predict(all_reg.lasso.fit, s = bestlam2 , newx = x.train2)
lasso.pred.test_reg = predict(all_reg.lasso.fit, s = bestlam2 , newx = x.test2)

# Calculate MSE for training and test set
lasso.train.mse = mean((lasso.pred.train_reg - y.train2)^2)
lasso.test.mse = mean((lasso.pred.test_reg - y.test2)^2)

print(lasso.train.mse)
```

```
## [1] 14.07
```

```
print(lasso.test.mse)
```

```
## [1] 15.24
```

Our LASSO regression model has a training MSE of 14.07 and a test MSE of 15.24.

```
# Predict on test set
pred.pt.cv_reg.te = predict(pt.cv_reg, all_reg.te, method="anova")
# Test MSE
tree_reg.mse.te <- mean((pred.pt.cv_reg.te - all_reg.te$Poverty)^2)


# Predict on training set
pred.pt.cv_reg.tr = predict(pt.cv_reg, all_reg.tr, method="anova")
# Training MSE
tree_reg.mse.tr <-mean((pred.pt.cv_reg.tr - all_reg.te$Poverty)^2)

print(tree_reg.mse.te)
```

```
## [1] 27.61
```

```
print(tree_reg.mse.tr)
```

```
## [1] 64.77
```

Our decision tree regression model has a training MSE of 64.77 and a test MSE of 27.61

Let's now look again at our original classification methods records table and our new regression methods records table and compare within and between the performance of the models.

```
# Display classification problem performance
records
```

```
##          train.error test.error
## tree          0.1554     0.1680
## logistic      0.1233     0.1248
## lasso         0.1233     0.1232
```

```
# Display regression problem performance
records_reg
```

```
##        train.error test.error
## tree         64.77      27.61
## linear       14.07      15.24
## lasso        14.07      15.23
```

From these tables, we can see that our previous models did not perform any better relative to eachother when given a regression problem. Decision Tree still performed the worst this time around, with a high test MSE of 27.61. Just like when given a classifcation problem, linear regression *(which we will consider as a comparison to logistic regression)* only performed slightly worse than a LASSO regularized method. In the regression problem they were even closer to each other, with LASSO only doing better than linear regression by 0.01 test MSE.

We cannot compare the models directly across problems, such as being able to say if the classification or regression decision tree performed better based on MSE or error rate, since they are different scales and metrics. But, we can compare the models on other things, such as interpretability and preference. As for the decision tree, the classification problem seems more interpretable, as it is easier to follow to a terminal node and arrive at a clear answer of 0 or 1. A regression decision tree is less preferred and not as interpretable because with 3 terminal nodes it can only return 3 possible predictions for poverty level, which means a lot of the time its prediction will be way off from the true value. However, when it comes to LASSO and logistic/linear regression, I think the regression problem models are preferred. This is because there are so many predictor variables that it is able to arrive at many different prediction values, while the classification problem can only return probabilities. Even further, the classification problem is forced to make a split at 0.5, deciding whether to classify as 0 or 1 based on this broad range of probabilites. For example, say the LASSO classification model returned 0.53. This would be classified as a 1, even though the value is quite close to the middle; this is likely a large portion of the error rate in the logistic and LASSO models. Meanwhile, a linear/LASSO regression problem returns an actual percentage, such as 24%, which is easily interpretable and tells you immediately what a county's predicted poverty level is.

Wrapping Up (Best and Worst Methods)

Just for further analysis, let's see what the decision tree model would predict for `Santa Barbara County` and `Fresno County`, my hometown, since it was the worst performing in both classification and regression we will see the importance of having a well fitted model.

```
# Find index of SB County in census.clean
which(census.clean$County == "Santa Barbara County")
```

```
## [1] 228
```

```
which(census.clean$County == "Fresno County")
```

```
## [1] 196
```

```
# Fresno is idx 196, SB County is 228
```

```
# Predict using regression decision tree
predict(pt.cv_reg, all_reg[c(196, 228),], method="anova")
```

```
## [1] 12.78 20.10
```

```
# Predict using classification decision tree
predict(pt.cv, all[c(196, 228),], type="class")
```

```
## [1] 0 0
## Levels: 0 1
```

```
# Check true values
all_reg[c(196, 228),]$Poverty
```

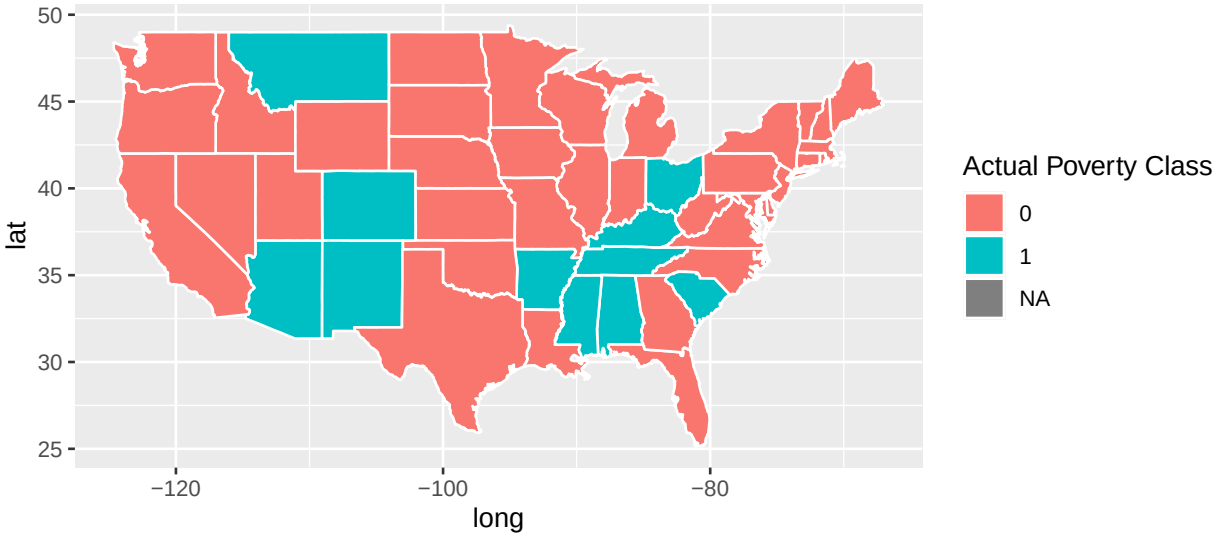
```
## [1]  8.1 19.9
```

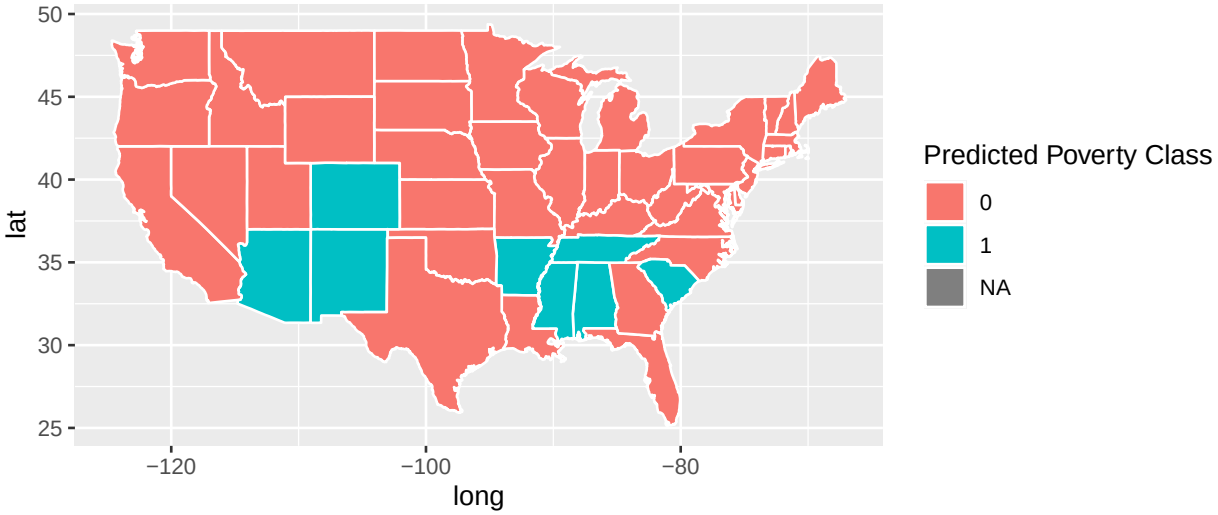
```
all[c(196, 228),]$Poverty
```

```
## [1] 0 0
## Levels: 0 1
```

We can see from these predictions that the decision tree actually made the correct classification in both counties, correctly predicting 0. While the regression value was quite close for Santa Barbara County, only being off by $20.10 - 19.9 = 0.2$, the prediction for Fresno was a little farther off, ending up at a difference of $12.78 - 8.10 = 4.68$. There isn't really much further analysis to do here, this was just to show that the models developed in this project can be used on real life examples that are easier to put into perspective because I live/used to live in these areas. It is also entirely possible that the either of these counties was in the training data, which means we shouldn't be be predicting on those values anyways.

Let's visualize the true classifications of `Poverty` on the map and compare it to the predictions made by the LASSO model, since it was the best performing model.


50
45
40
35
30
25
lat
−120
−100
−80
long
Actual Poverty Class
0
1
NA



These two maps are very interesting, as it shows that if we map out our test data (since we can't reliably use predictions from the training data set), and then average the a state's counties `Poverty` classes, the maps are fairly similar. The only difference between the two occurs in three states: Kentucky, Ohio, and Montana. The LASSO regression has marked more counties in these states as a 1 than not, resulting in these states overall being predicted as 1's. However, according to our true test data these states should actually be marked as 0, meaning they have a `Poverty` value less than 20%.

All the models performed as expected, however I think there were too many variables and it often made the models hard to interpret and follow during the duration of the project. Looking back, it would have been smart upon seeing so many predictor variables to employ a dimension reduction method, such as PCA, SVD, or LDA. This way, the results and models are easier to work with from the very start, and non-informative variables can be weeded out before they begin to get in the way of good predictions.

As for possible directions in the future, obviously with more time and manpower it would be a good idea to test even more models on the same dataset. This way we can continue to improve our results and end up with the best possible model to predict `Poverty` in a county. Also, whenever a new census is collected and the data is made public it would be wise to rerun all the models and tests and compare them to the old census. We could even take the new census data entirely as test data and see if it performs well on the models we created using the old census. The predictions made on the new census data could provide new insights, such as if counties are changing over time in `Poverty` levels and what is causing it. This could be done by looking at the coefficients/variable importance in different models. For example ,say suddenly `Employment` jumps up as an important predictor variable, signifying that jobs are suddenly paying more so when a county has high `Employment` its `Poverty` will be especially low.

If you've made it this far, thank you for your time and reading through this project. This project is on my portfolio to showcase my understanding of the R programming language and the applications of machine learning methods in data analysis. If you have any questions contact me at: braeden077@gmail.com.