Ensemble

Zachary Canoot & Gray Simpson

Ensemble Learning

Introduction

This is a notebook discussing ensemble machine learning with the dataset US Census Income in 1994 from Kaggle.com.

We will be doing classification to see how accurately we can predict IncomeClass, a class that states whether a person with the given socio-economic attributes makes over 50k or not.

When we've worked with this data in the past, we were able to get ~76% accuracy with Naive Bayes, and ~84% accuracy with Logistic Regression. With Ensemble styles of learning, we should be able to get into the 90%+ category, since there was decent success in other ways. In this assignment, we will be doing Random Forest, XGBoost, and caretEnsemble ensemble learning types.

Reading In Data

We read in the data. Since it doesn't have headers, we have to add them manually based on another file that explains them. Then we'll take a quick peek to make sure it's all looking okay.

```
library(RWeka)
df <- read.csv("data/adultdata.csv", header = FALSE)
colnames(df) <- c("Age", "WorkClass", "Weight", "Education", "YearsEdu", "MaritalStatus", "Job", "Relat
head(df)
```

```
##   Age         WorkClass Weight  Education YearsEdu       MaritalStatus
## 1  39         State-gov  77516  Bachelors       13       Never-married
## 2  50  Self-emp-not-inc  83311  Bachelors       13  Married-civ-spouse
## 3  38           Private 215646    HS-grad        9            Divorced
## 4  53           Private 234721       11th        7  Married-civ-spouse
## 5  28           Private 338409  Bachelors       13  Married-civ-spouse
## 6  37           Private 284582    Masters       14  Married-civ-spouse
##                   Job   Relationship   Race     Sex CapitalGain CapitalLoss
## 1       Adm-clerical  Not-in-family  White    Male        2174           0
## 2    Exec-managerial        Husband  White    Male           0           0
## 3  Handlers-cleaners  Not-in-family  White    Male           0           0
## 4  Handlers-cleaners        Husband  Black    Male           0           0
## 5     Prof-specialty           Wife  Black  Female           0           0
## 6    Exec-managerial           Wife  White  Female           0           0
##   HoursWorked  NativeCountry IncomeClass
## 1          40  United-States       <=50K
## 2          13  United-States       <=50K
## 3          40  United-States       <=50K
```

```
## 4          40  United-States       <=50K
## 5          40           Cuba       <=50K
## 6          40  United-States       <=50K
```

Looking into the data, this seems to fit. All is well here.

Data Cleaning

Next, most obviously, we'll want to convert all those different columns to factors so we can look into them deeper.

```
df$WorkClass <- as.factor(df$WorkClass)
df$Education <- as.factor(df$Education)
df$MaritalStatus <- as.factor(df$MaritalStatus)
df$Job <- as.factor(df$Job)
df$Relationship <- as.factor(df$Relationship)
df$Race <- as.factor(df$Race)
df$Sex <- as.factor(df$Sex)
df$NativeCountry <- as.factor(df$NativeCountry)
df$IncomeClass <- as.factor(df$IncomeClass)
summary(df)
```

```
##       Age                    WorkClass         Weight
##  Min.   :17.00    Private         :33906   Min.   :  12285
##  1st Qu.:28.00    Self-emp-not-inc: 3862   1st Qu.: 117551
##  Median :37.00    Local-gov       : 3136   Median : 178145
##  Mean   :38.64    ?               : 2799   Mean   : 189664
##  3rd Qu.:48.00    State-gov       : 1981   3rd Qu.: 237642
##  Max.   :90.00    Self-emp-inc    : 1695   Max.   :1490400
##                  (Other)          : 1463
##          Education        YearsEdu                    MaritalStatus
##   HS-grad     :15784   Min.   : 1.00    Divorced             : 6633
##   Some-college:10878   1st Qu.: 9.00    Married-AF-spouse    :   37
##   Bachelors   : 8025   Median :10.00    Married-civ-spouse   :22379
##   Masters     : 2657   Mean   :10.08    Married-spouse-absent:  628
##   Assoc-voc   : 2061   3rd Qu.:12.00    Never-married        :16117
##   11th        : 1812   Max.   :16.00    Separated            : 1530
##  (Other)      : 7625                    Widowed              : 1518
##                Job                 Relationship                    Race
##   Prof-specialty : 6172    Husband       :19716    Amer-Indian-Eskimo:  470
##   Craft-repair   : 6112    Not-in-family :12583    Asian-Pac-Islander: 1519
##   Exec-managerial: 6086    Other-relative: 1506    Black             : 4685
##   Adm-clerical   : 5611    Own-child     : 7581    Other             :  406
##   Sales          : 5504    Unmarried     : 5125    White             :41762
##   Other-service  : 4923    Wife          : 2331
##  (Other)         :14434
##       Sex          CapitalGain     CapitalLoss      HoursWorked
##   Female:16192   Min.   :    0   Min.   :   0.0   Min.   : 1.00
##   Male  :32650   1st Qu.:    0   1st Qu.:   0.0   1st Qu.:40.00
##                  Median :    0   Median :   0.0   Median :40.00
##                  Mean   : 1079   Mean   :  87.5   Mean   :40.42
##                  3rd Qu.:    0   3rd Qu.:   0.0   3rd Qu.:45.00
```

```
##                  Max.   :99999   Max.   :4356.0   Max.   :99.00
##
##         NativeCountry    IncomeClass
##   United-States:43832    <=50K :24720
##   Mexico        :  951    <=50K.:12435
##   ?             :  857    >50K  : 7841
##   Philippines   :  295    >50K. : 3846
##   Germany       :  206
##   Puerto-Rico   :  184
##  (Other)        : 2517
```

We will not be cleaning out NAs, or the ambiguous "?" factors here. We will consider what was not reported or could not be described as useful data as well in determining results. We can see that the data is skewed towards men, people from the US, and people working with a private group.

Let's clean up the periods on the IncomeClass, and rename the values. For a learning model we will use later, we can't use certain symbols in factor names, so we will go ahead and change those now.

```
levels(df$IncomeClass) <- c("LE50k", "LE50k", "G50k", "G50k")
levels(df$IncomeClass)
```

```
## [1] "LE50k" "G50k"
```

Data Investigation

Preparation

Let's see how many levels some of these have overall.

```
str(df)
```

```
## 'data.frame':    48842 obs. of  15 variables:
##  $ Age          : int  39 50 38 53 28 37 49 52 31 42 ...
##  $ WorkClass    : Factor w/ 9 levels " ?"," Federal-gov",..: 8 7 5 5 5 5 5 7 5 5 ...
##  $ Weight       : int  77516 83311 215646 234721 338409 284582 160187 209642 45781 159449 ...
##  $ Education    : Factor w/ 16 levels " 10th"," 11th",..: 10 10 12 2 10 13 7 12 13 10 ...
##  $ YearsEdu     : int  13 13 9 7 13 14 5 9 14 13 ...
##  $ MaritalStatus: Factor w/ 7 levels " Divorced"," Married-AF-spouse",..: 5 3 1 3 3 3 4 3 5 3 ...
##  $ Job          : Factor w/ 15 levels " ?"," Adm-clerical",..: 2 5 7 7 11 5 9 5 11 5 ...
##  $ Relationship : Factor w/ 6 levels " Husband"," Not-in-family",..: 2 1 2 1 6 6 2 1 2 1 ...
##  $ Race         : Factor w/ 5 levels " Amer-Indian-Eskimo",..: 5 5 5 3 3 5 3 5 5 5 ...
##  $ Sex          : Factor w/ 2 levels " Female"," Male": 2 2 2 2 1 1 1 2 1 2 ...
##  $ CapitalGain  : int  2174 0 0 0 0 0 0 0 14084 5178 ...
##  $ CapitalLoss  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ HoursWorked  : int  40 13 40 40 40 40 16 45 50 40 ...
##  $ NativeCountry: Factor w/ 42 levels " ?"," Cambodia",..: 40 40 40 40 6 40 24 40 40 40 ...
##  $ IncomeClass  : Factor w/ 2 levels "LE50k","G50k": 1 1 1 1 1 1 1 2 2 2 ...
```

Looking good. Since we have so many factors overall, chances are low that the numerical values alone will be helping us much, but instead, it's probably in combinations that they will work effectively.

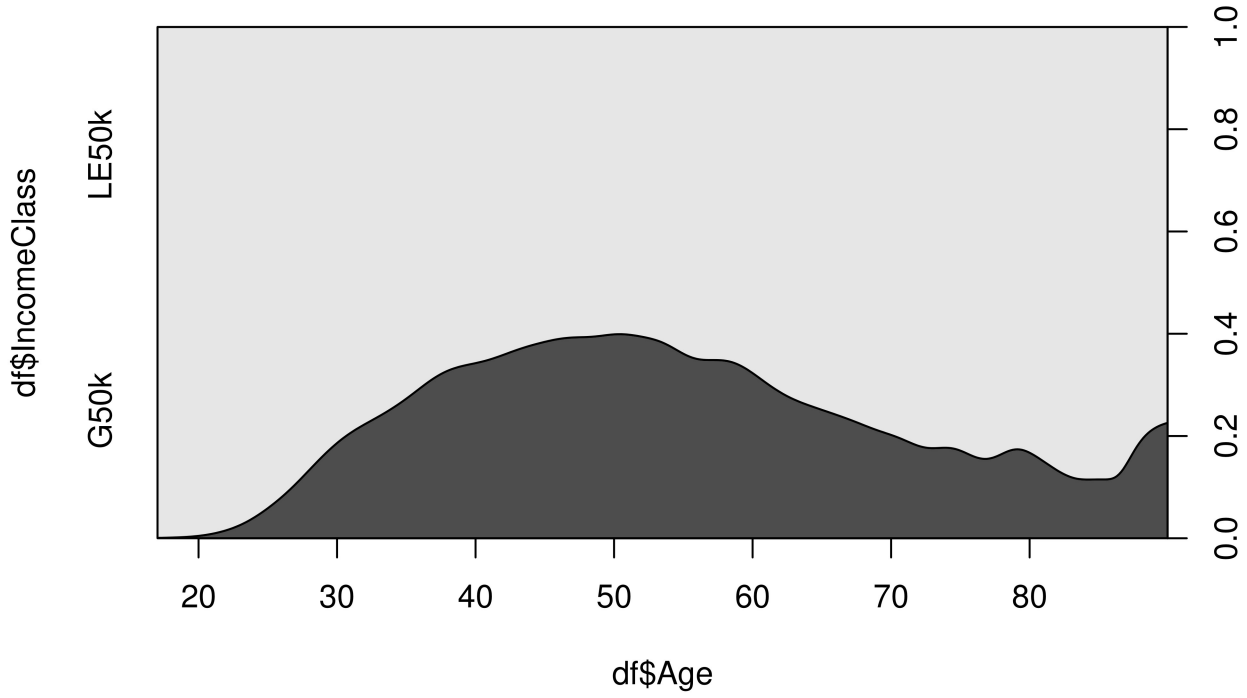
Let's divvy up the data.

```
set.seed(8)
i <- sample(1:nrow(df), nrow(df)*.8, replace = FALSE)
train <- df[i,]
test <- df[-i,]
```

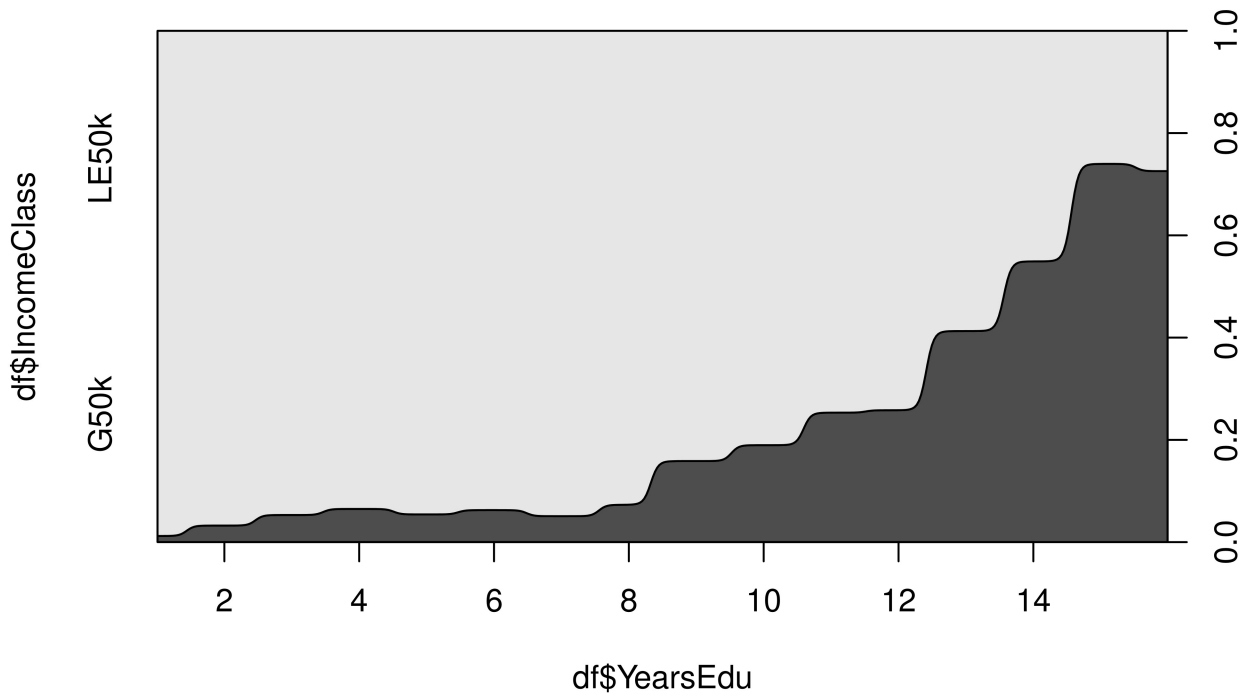
Visual Analysis

Let's look into how income scales with various traits.

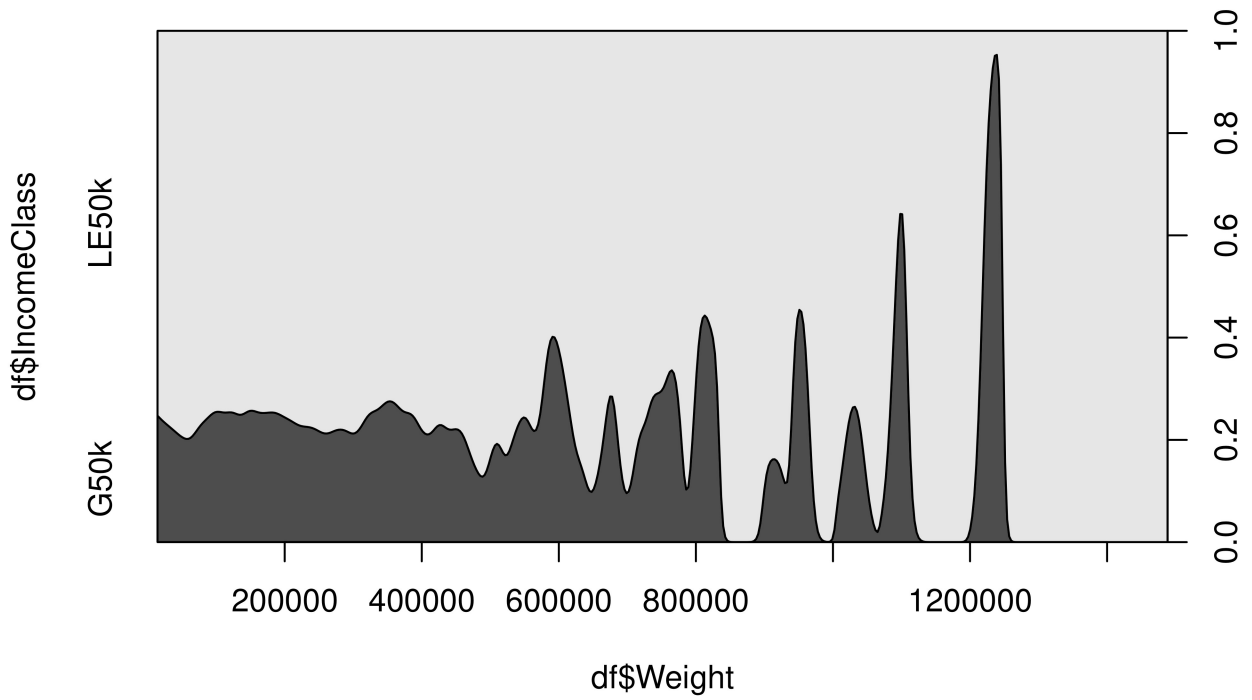
```
cdplot(df$Age,df$IncomeClass)
```




```
cdplot(df$YearsEdu,df$IncomeClass)
```

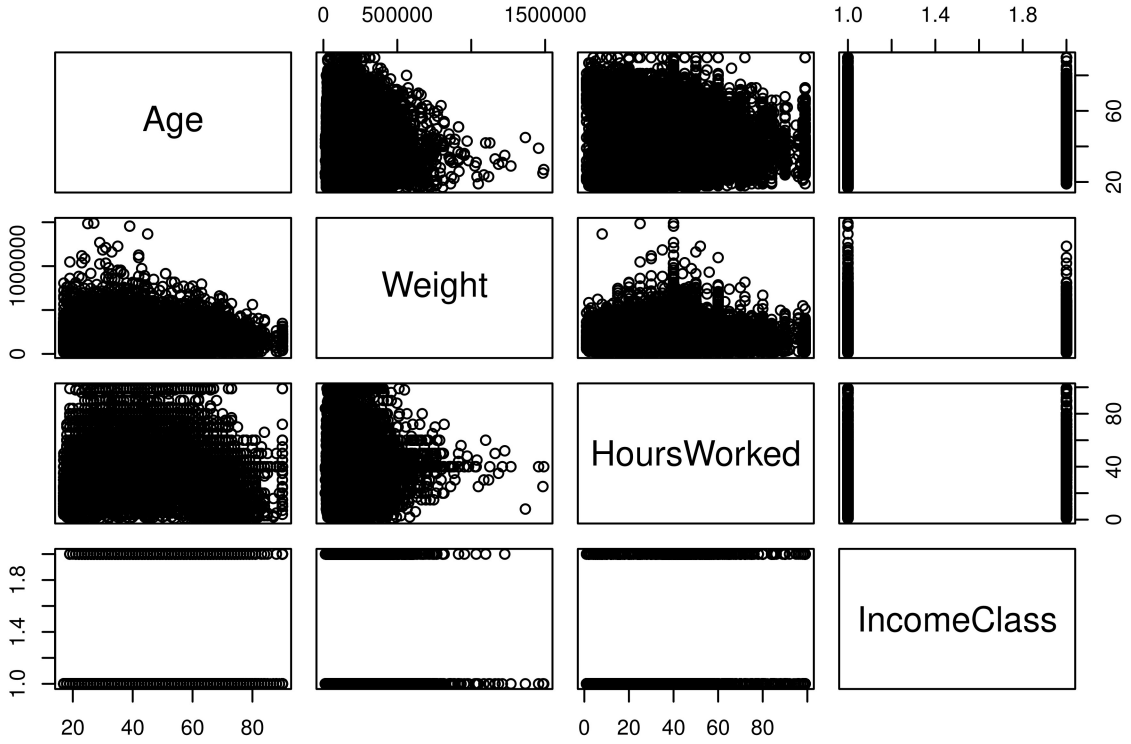



```
cdplot(df$Weight,df$IncomeClass)
```



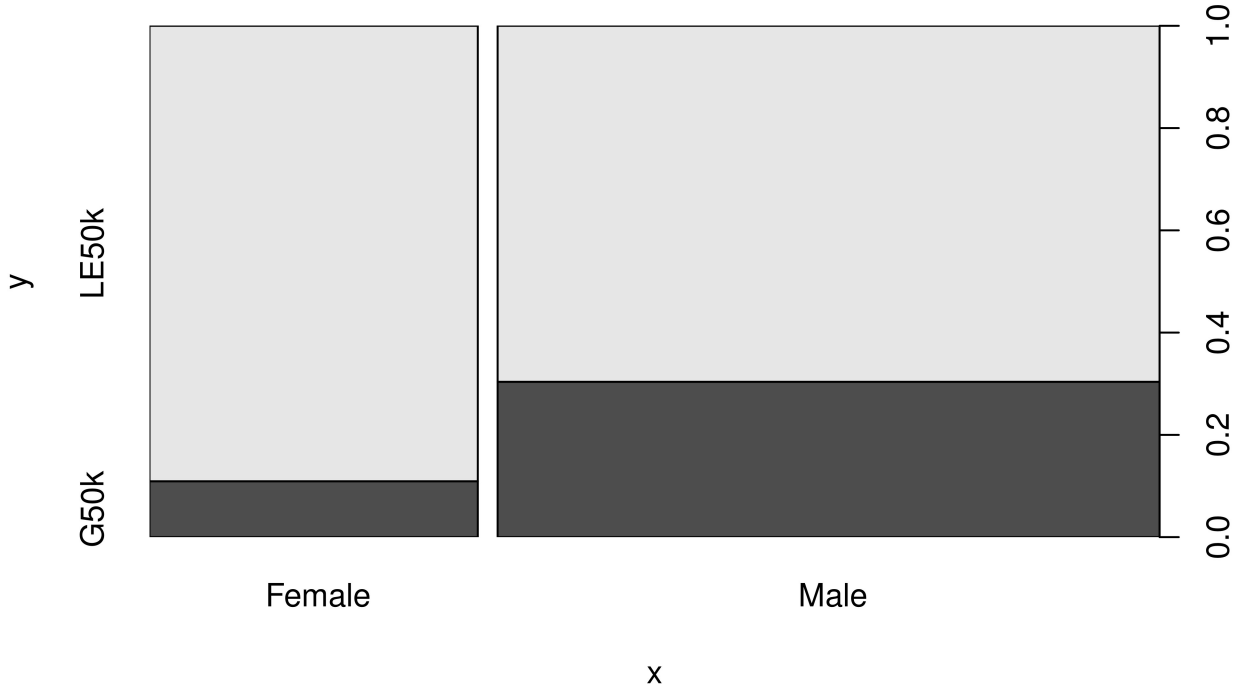
Weight is a bit uncertain in what its telling us, and there are many reasons this could be. If possible, we'll keep it in mind to try and remove it if it proves that it isnt' sufficiently useful.

```
pairs(df[c(1,3,13,15)])
```

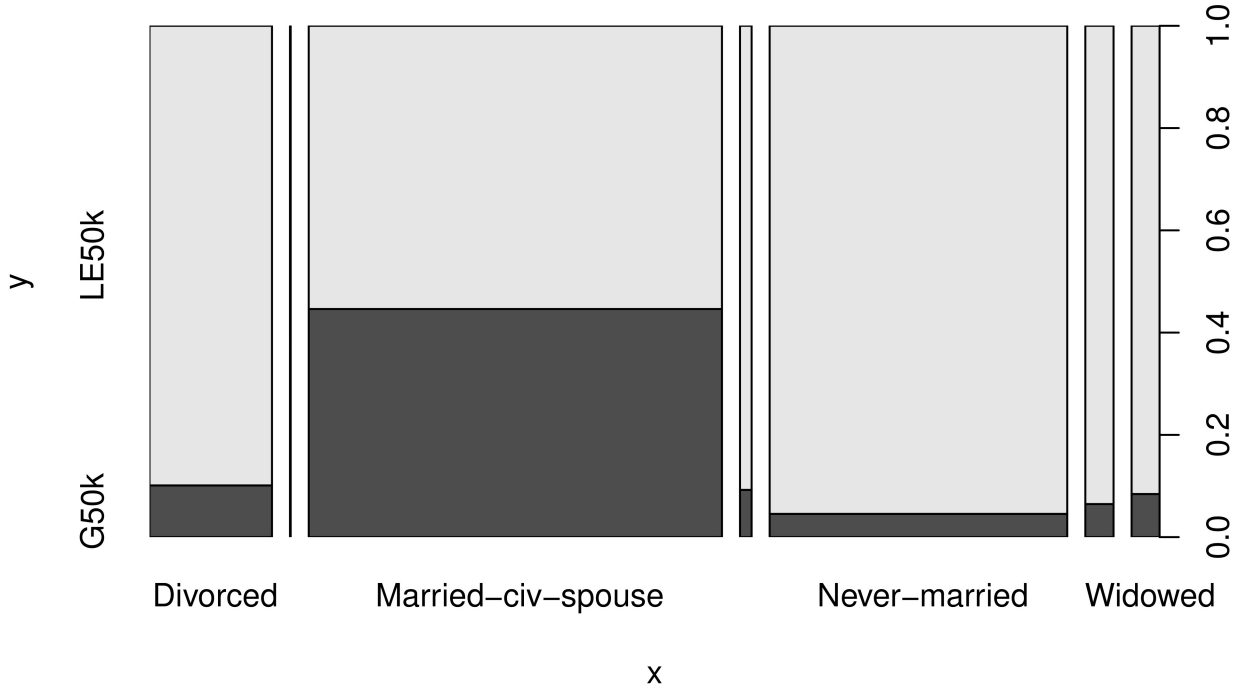


The numerical values do not seem to have much relationship to each other, or too many obvious details towards the income class, but we'll see as we go along.

```
plot(df$Sex,df$IncomeClass)
```

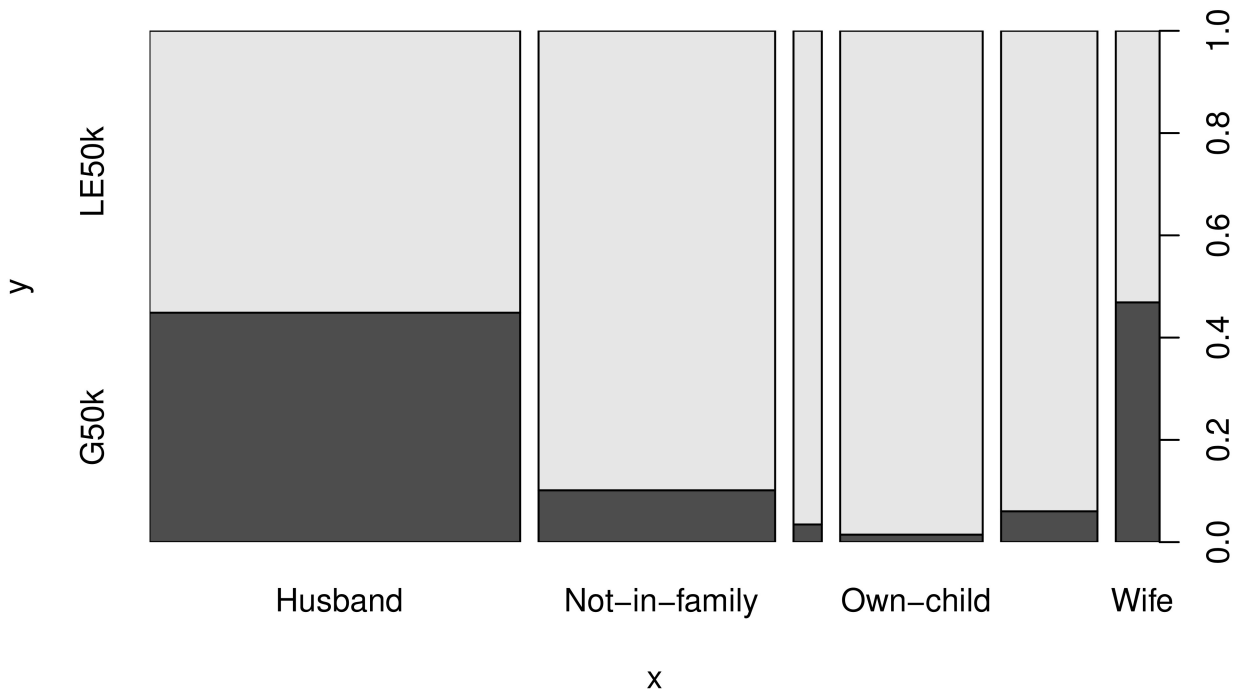



```
plot(df$MaritalStatus,df$IncomeClass)
```




```
plot(df$WorkClass,df$IncomeClass)
```


```
plot(df$Relationship,df$IncomeClass)
```



From other times working with this data, we have already seen a lot about how it relates to various attributes. Now that we've looked a bit more at what we've done in the past with a few new plots, we'll move on.

We see a lot of most likely related data, with a few skews in amount towards certain traits– men, people in the US, and people working for private groups. We can see how the lack of data and the reasons why certain people reported census data and others didn't could cause inaccuracies down the line. While it may look women make larger amounts much less than men, this could be due to a number of factors that aren't specifically indicative of true income of the wider population. This data set does not seem to be perfect. All the same, we can do our best, and it seems that there are still correlations between income and socioeconomic factors to be investigated and predicted.

Let's look into these ensemble learning methods and see what it can do.

Random Forest

Let's start with Random Forest Learning.It is a supervised algorithm that puts together decision trees on differing samples of data and uses that to influence a larger scale vote for what an observation is. It's better with classification than regression, which makes it great for what we're doing.

```
library(randomForest)
```

```
## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
rf <- randomForest(data=train, IncomeClass ~ . ,  importance=TRUE, ntree = 500)
rf
```

```
##
## Call:
##  randomForest(formula = IncomeClass ~ ., data = train, importance = TRUE,      ntree = 500)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 13.27%
## Confusion matrix:
##       LE50k G50k class.error
## LE50k 27931 1828   0.0614268
## G50k   3356 5958   0.3603178
```

Okay, so it is looking alright, but still leaves much to be desired. Let's look at the specific statistics of it working on our test data.

```
library(mltools)
predrf <- predict(rf, newdata=test, type="response")
accrf <- mean(predrf==test$IncomeClass)
mccrf <- mcc(factor(predrf), test$IncomeClass)
print(paste("The accuracy is ",accrf))
```

```
## [1] "The accuracy is  0.864776333299212"
```

```
print(paste("The MCC is ",mccrf))
```

```
## [1] "The MCC is  0.612462167619099"
```

86% is pretty good! It managed to get very specific with all the different values and be surprisingly strong. Let's move on and see what else we can do.

caretEnsemble

Let's try something entirely new: caretEnsemble.

```
library(caret)
```

```
## Loading required package: ggplot2

##
## Attaching package: 'ggplot2'

## The following object is masked from 'package:randomForest':
##
##     margin

## Loading required package: lattice
```

```
library(mlbench)
library(rpart)
library(caretEnsemble)
```

```
##
## Attaching package: 'caretEnsemble'

## The following object is masked from 'package:ggplot2':
##
##     autoplot
```

```
controlCE <- trainControl(method = "boot",number=25, savePredictions = "final", classProbs = TRUE, inde
summary(controlCE)
```

```
##                   Length Class  Mode
## method             1     -none- character
## number             1     -none- numeric
## repeats            1     -none- logical
## search             1     -none- character
## p                  1     -none- numeric
## initialWindow      0     -none- NULL
## horizon            1     -none- numeric
## fixedWindow        1     -none- logical
## skip               1     -none- numeric
## verboseIter        1     -none- logical
## returnData         1     -none- logical
## returnResamp       1     -none- character
## savePredictions    1     -none- character
## classProbs         1     -none- logical
## summaryFunction    1     -none- function
## selectionFunction  1     -none- character
## preProcOptions     6     -none- list
## sampling           0     -none- NULL
## index             25     -none- list
## indexOut           0     -none- NULL
## indexFinal         0     -none- NULL
## timingSamps        1     -none- numeric
## predictionBounds   2     -none- logical
## seeds              1     -none- logical
## adaptive           4     -none- list
## trim               1     -none- logical
## allowParallel      1     -none- logical
```

Let's start working on the model itself now.

```
library(e1071)
```

```
##
## Attaching package: 'e1071'

## The following object is masked from 'package:mltools':
##
##     skewness
```

```r
methods <- c("rpart","glm") #Other models were tried, but even after a while would not produce results
models <- caretList(IncomeClass~., data=train, trControl=controlCE, methodList=methods)
```

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

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

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

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

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

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

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

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

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

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

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

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading
```

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

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

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

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

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

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

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

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

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

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

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

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

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

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

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

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

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

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

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

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

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

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

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

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

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

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading
```

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

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

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

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

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

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

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

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

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

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

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

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

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

We get a lot of warnings, but there are many reasons that they could occur, and there's nothing to be worried about yet.

Now let's really put this to work.

```
predCE <- as.data.frame(predict(models, newdata=test))
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading
```

Nothing stands out yet, so let's see it in a bit more understandable form: statistics.

```
CE <- caretEnsemble(models, metric = "Accuracy", trControl=controlCE)
summary(CE)
```

```
## The following models were ensembled: rpart, glm
## They were weighted:
## 3.2448 -0.9784 -5.4316
## The resulting Accuracy is: 0.8531
## The fit for each individual model on the Accuracy is:
##  method  Accuracy  AccuracySD
##   rpart 0.8404160 0.004609881
##     glm 0.8521954 0.001708711
```

It was able to combine rpart and glm to get 85% accuracy! It isn't anything outstanding, and looks like it may be nearly the same as a general linear model on its own, but it is good.

XGBoost

Now we'll try XGBoost, or Extreme Gradient Boosting. Made by Tianqi Chen, it combines a number of weak models to create something stronger, and has been considered one of the most powerful machine learning algorithms. It does this additively, which is why it's considered "gradient." It uses highly specialized decision trees so specific that they would not be readable or interpretable anymore, where new models "boost" the data, focusing on the errors made in the previous model.

We have to convert factors to integers to work here. So, this is why XGBoost is listed last: we're going to change them as a whole and redistribute.

```
df$WorkClass <- as.numeric(df$WorkClass)
df$Education <- as.numeric(df$Education)
df$MaritalStatus <- as.numeric(df$MaritalStatus)
df$Job <- as.numeric(df$Job)
df$Relationship <- as.numeric(df$Relationship)
df$Race <- as.numeric(df$Race)
df$Sex <- as.numeric(df$Sex)
df$NativeCountry <- as.numeric(df$NativeCountry)
df$IncomeClass <- as.numeric(df$IncomeClass)
summary(df)
```

```
##       Age          WorkClass        Weight          Education
##  Min.   :17.00   Min.   :1.00   Min.   :  12285   Min.   : 1.00
##  1st Qu.:28.00   1st Qu.:5.00   1st Qu.: 117551   1st Qu.:10.00
##  Median :37.00   Median :5.00   Median : 178145   Median :12.00
##  Mean   :38.64   Mean   :4.87   Mean   : 189664   Mean   :11.29
##  3rd Qu.:48.00   3rd Qu.:5.00   3rd Qu.: 237642   3rd Qu.:13.00
##  Max.   :90.00   Max.   :9.00   Max.   :1490400   Max.   :16.00
```

```
##     YearsEdu     MaritalStatus         Job          Relationship
##  Min.   : 1.00   Min.   :1.000   Min.   : 1.000   Min.   :1.000
##  1st Qu.: 9.00   1st Qu.:3.000   1st Qu.: 4.000   1st Qu.:1.000
##  Median :10.00   Median :3.000   Median : 8.000   Median :2.000
##  Mean   :10.08   Mean   :3.619   Mean   : 7.578   Mean   :2.443
##  3rd Qu.:12.00   3rd Qu.:5.000   3rd Qu.:11.000   3rd Qu.:4.000
##  Max.   :16.00   Max.   :7.000   Max.   :15.000   Max.   :6.000
##       Race            Sex          CapitalGain     CapitalLoss
##  Min.   :1.000   Min.   :1.000   Min.   :    0   Min.   :   0.0
##  1st Qu.:5.000   1st Qu.:1.000   1st Qu.:    0   1st Qu.:   0.0
##  Median :5.000   Median :2.000   Median :    0   Median :   0.0
##  Mean   :4.668   Mean   :1.668   Mean   : 1079   Mean   :  87.5
##  3rd Qu.:5.000   3rd Qu.:2.000   3rd Qu.:    0   3rd Qu.:   0.0
##  Max.   :5.000   Max.   :2.000   Max.   :99999   Max.   :4356.0
##    HoursWorked    NativeCountry    IncomeClass
##  Min.   : 1.00   Min.   : 1.00   Min.   :1.000
##  1st Qu.:40.00   1st Qu.:40.00   1st Qu.:1.000
##  Median :40.00   Median :40.00   Median :1.000
##  Mean   :40.42   Mean   :37.75   Mean   :1.239
##  3rd Qu.:45.00   3rd Qu.:40.00   3rd Qu.:1.000
##  Max.   :99.00   Max.   :42.00   Max.   :2.000
```

Now let's redistribute it.

```
set.seed(8)
i <- sample(1:nrow(df), nrow(df)*.8, replace = FALSE)
train <- df[i,]
test <- df[-i,]
```

After running the model initially, the loss flattened out at about round 36, so we only do 40 rounds in the code block below.

```
library(xgboost)
trainlabelXG <- ifelse(as.numeric(train[,15])==1,1,0) #we have to fit it to integers for XGBoost to run
trainmatrixXG <- data.matrix(train[,-15])
XGtrain <- xgb.DMatrix(data = trainmatrixXG, label = trainlabelXG)
xg <- xgboost(data = trainmatrixXG, label = trainlabelXG, nrounds = 100, max.depth = 6, objective="bina
```

```
## [1]  train-logloss:0.541443
## [2]  train-logloss:0.457330
## [3]  train-logloss:0.405242
## [4]  train-logloss:0.370662
## [5]  train-logloss:0.347114
## [6]  train-logloss:0.331148
## [7]  train-logloss:0.318674
## [8]  train-logloss:0.310190
## [9]  train-logloss:0.304467
## [10] train-logloss:0.296857
## [11] train-logloss:0.292901
## [12] train-logloss:0.289260
## [13] train-logloss:0.286101
## [14] train-logloss:0.283711
## [15] train-logloss:0.280617
```

```
## [16] train-logloss:0.277023
## [17] train-logloss:0.274591
## [18] train-logloss:0.272989
## [19] train-logloss:0.271287
## [20] train-logloss:0.269234
## [21] train-logloss:0.266616
## [22] train-logloss:0.265375
## [23] train-logloss:0.263953
## [24] train-logloss:0.262810
## [25] train-logloss:0.261524
## [26] train-logloss:0.260573
## [27] train-logloss:0.259076
## [28] train-logloss:0.257592
## [29] train-logloss:0.257031
## [30] train-logloss:0.256178
## [31] train-logloss:0.255113
## [32] train-logloss:0.253774
## [33] train-logloss:0.253239
## [34] train-logloss:0.252582
## [35] train-logloss:0.251240
## [36] train-logloss:0.250325
## [37] train-logloss:0.249897
## [38] train-logloss:0.249273
## [39] train-logloss:0.248975
## [40] train-logloss:0.248463
## [41] train-logloss:0.248304
## [42] train-logloss:0.247753
## [43] train-logloss:0.247216
## [44] train-logloss:0.246074
## [45] train-logloss:0.244964
## [46] train-logloss:0.244231
## [47] train-logloss:0.243908
## [48] train-logloss:0.243692
## [49] train-logloss:0.242683
## [50] train-logloss:0.242070
## [51] train-logloss:0.241946
## [52] train-logloss:0.241077
## [53] train-logloss:0.240465
## [54] train-logloss:0.240305
## [55] train-logloss:0.239295
## [56] train-logloss:0.238553
## [57] train-logloss:0.237768
## [58] train-logloss:0.237473
## [59] train-logloss:0.236831
## [60] train-logloss:0.236470
## [61] train-logloss:0.236014
## [62] train-logloss:0.235420
## [63] train-logloss:0.234935
## [64] train-logloss:0.234070
## [65] train-logloss:0.233192
## [66] train-logloss:0.233080
## [67] train-logloss:0.231927
## [68] train-logloss:0.231344
## [69] train-logloss:0.230703
```

```
## [70] train-logloss:0.229902
## [71] train-logloss:0.229810
## [72] train-logloss:0.229672
## [73] train-logloss:0.229243
## [74] train-logloss:0.228836
## [75] train-logloss:0.228664
## [76] train-logloss:0.228577
## [77] train-logloss:0.228397
## [78] train-logloss:0.227635
## [79] train-logloss:0.227167
## [80] train-logloss:0.226963
## [81] train-logloss:0.226698
## [82] train-logloss:0.226412
## [83] train-logloss:0.226180
## [84] train-logloss:0.225931
## [85] train-logloss:0.225320
## [86] train-logloss:0.225187
## [87] train-logloss:0.225099
## [88] train-logloss:0.224822
## [89] train-logloss:0.224434
## [90] train-logloss:0.223721
## [91] train-logloss:0.223267
## [92] train-logloss:0.222729
## [93] train-logloss:0.222579
## [94] train-logloss:0.222501
## [95] train-logloss:0.221651
## [96] train-logloss:0.221301
## [97] train-logloss:0.221043
## [98] train-logloss:0.220891
## [99] train-logloss:0.220732
## [100]    train-logloss:0.220171
```

```
summary(xg)
```

```
##                Length Class              Mode
## handle              1 xgb.Booster.handle externalptr
## raw            310690 -none-             raw
## niter               1 -none-             numeric
## evaluation_log      2 data.table         list
## call               15 -none-             call
## params              3 -none-             list
## callbacks           2 -none-             list
## feature_names      14 -none-             character
## nfeatures           1 -none-             numeric
```

Now, let's move on to predicting with it.

```
testlabelXG <- ifelse(as.numeric(test[,15])==1,1,0) #to match with earlier data
testmatrixXG <- data.matrix(test[,-15])
XGtest <- xgb.DMatrix(data = testmatrixXG, label = testlabelXG)
probXG <- predict(xg, testmatrixXG, type="prob")
predXG <- ifelse(probXG>0.5, 1, 0)
summary(probXG)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## 0.0000264 0.6060866 0.9384438 0.7637473 0.9932390 0.9999895
```

Now that we've done the prediction, lets look at the statistics on it.

```
accXG <- mean(predXG==testlabelXG)
mccXG <- mcc(predXG, testlabelXG)
print(paste("Accuracy is ", accXG))
```

```
## [1] "Accuracy is  0.86764254273723"
```

```
print(paste("MCC is ", mccXG))
```

```
## [1] "MCC is  0.623837237072165"
```

86.7% accuracy is not bad! Surprisingly close to Random Forest, and perhaps a bit lower accuracy than that.

Let's look into how they tie into each other.

Conclusion

```
RANDOM FOREST:
  ACC: 86.5%    MCC: .612
CARET ENSEMBLE:
  ACC: 85.3%    MCC couldn't be gathered due to typing complications
XGBOOST:
  ACC: 86.7%    MCC: .624
```

It would seem that, for our data, XGBoost learning is the most successful, but just barely by the side of Random Forest. Perhaps this is due to the large number of factors that decision trees would excel with. They both have similar places where they succeed. XGBoost agrees more fully with the data as well, as we see by the MCC.

CaretEnsemble does gloriously as well, able to pull together multiple different types to refine data in many different ways. Certainly, CaretEnsemble could be refined in the future, perhaps on a different computer that can run faster and use more variant learning methods.

Even then, there are some aspects of someone's income that socioeconomic factors could never completely get– logically, we know that some things in life are simply a matter of luck and elbow grease, no matter background or anything else. There well may be no way to get significantly more accurate with this data set.In the past, 85% accuracy was already achieved with logistic regression. This is in line with what we see here, and perhaps there is more of a hard line that no one could predict, no matter what, by this point.

Overall, ensemble methods seem to be quite wonderful at predicting. It combines what is best from different methods and is able to treat its own weak spots to create a more accurate result. What we have learned here proves that, and we've learned about the data a lot as well in the meantime.