Classifying Census Data

This is an R Markdown document containing the attempted classification of US census data into high (>$50k) and low (<$50k) incomes.

Let's start by loading the data, which comes from the UCI machine leaning repository:

```
library(readr)
census <- read.csv(
  "http://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data",
  header = FALSE)
names(census) <- c("age", "workclass", "fnlwgt", "education",
                   "education.num", "marital.status", "occupation", "relationship",
                   "race", "sex", "capital.gain", "capital.loss", "hours.per.week",
                   "native.country", "income")
```

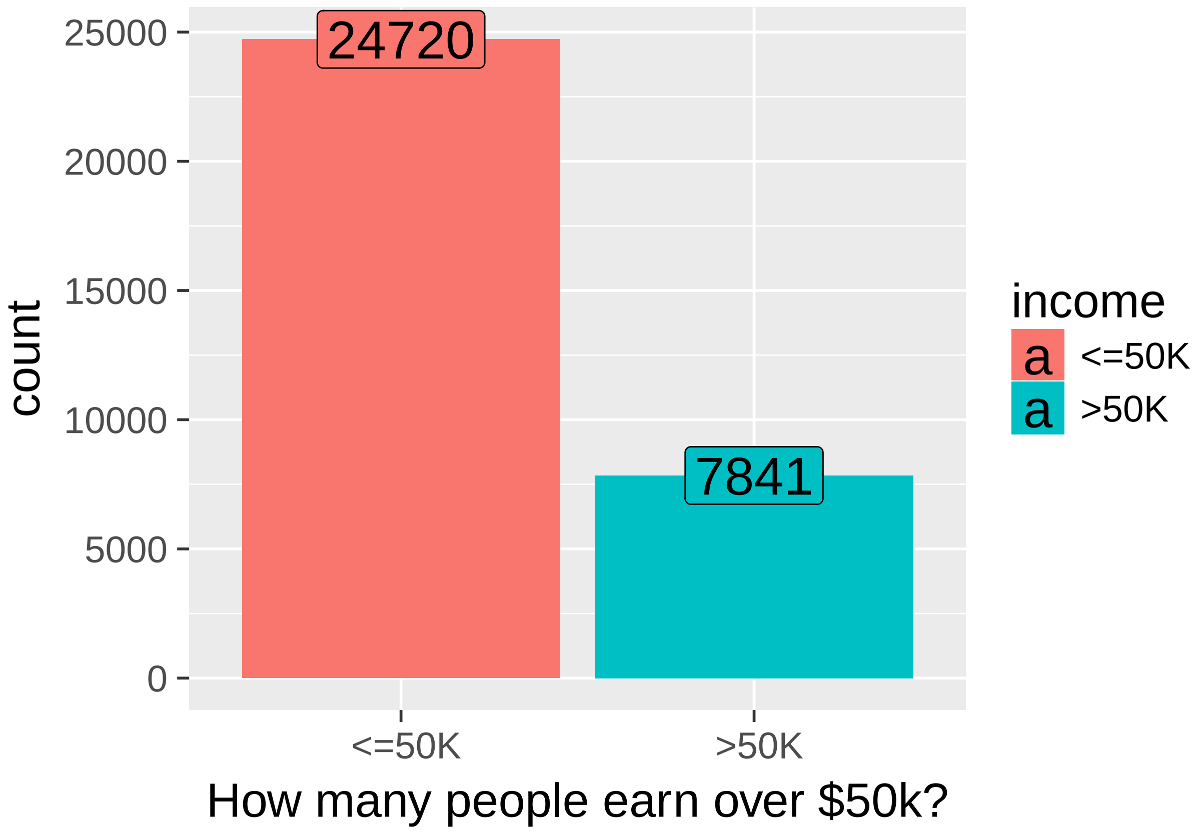
Exploratory Analysis

Let's start by exploring the data using ggplot2. Firstly, the dependent variable:

```
# Let's start by exploring the dependent
library(tidyverse)
```

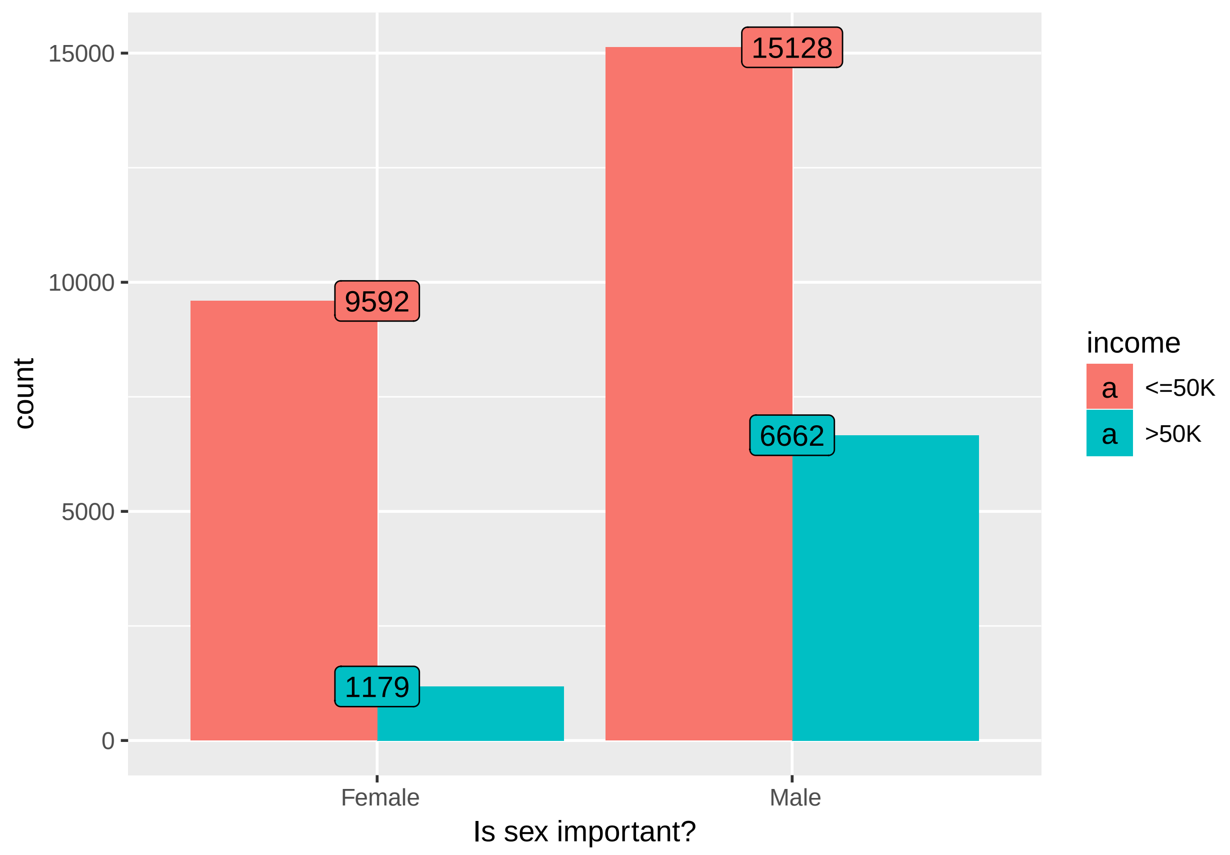
```
## Loading tidyverse: ggplot2
## Loading tidyverse: tibble
## Loading tidyverse: tidyr
## Loading tidyverse: purrr
## Loading tidyverse: dplyr

## Conflicts with tidy packages ----------------------------------------------

## filter(): dplyr, stats
## lag():    dplyr, stats
```

```
p1 <- ggplot(census, aes(x = income, fill = income)) +
  geom_bar(stat='count') +
  labs(x = 'How many people earn over $50k?') +
  geom_label(stat='count',aes(label=..count..), size=7) +
  theme_grey(base_size = 18)
p1
```



There is a fairly large class imbalance. This will examined later. How about we first examine sex as a predictor of income over $50k:

```r
# How about sex as a predictor?
p2 <- ggplot(census, aes(x = sex, fill = income)) +
  geom_bar(stat='count', position='dodge') + theme_grey() +
  labs(x = 'Is sex important?') +
  geom_label(stat='count', aes(label=..count..))
p2
```



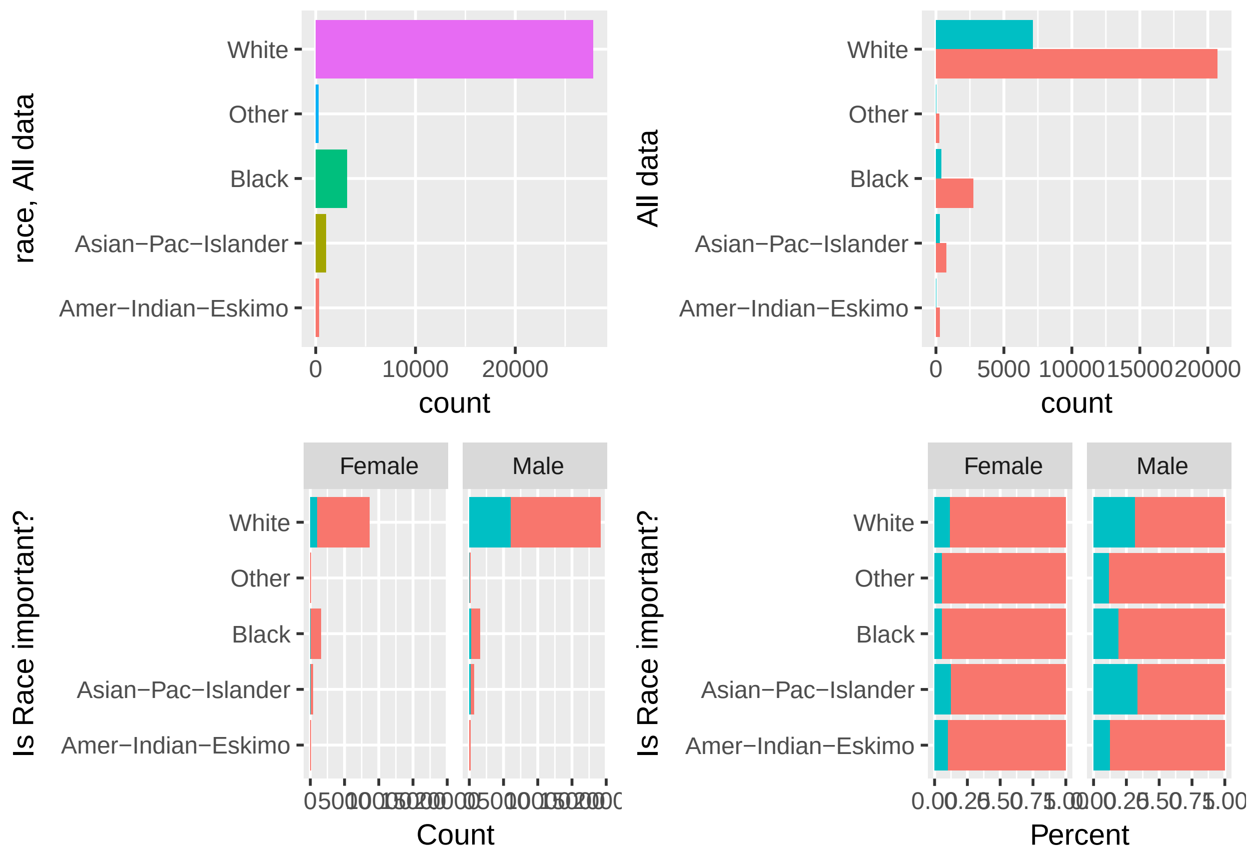
Sex appears to be important. Those who are male are considerably more likely to earn greater than 50k, when proportionally compared to women. There are far more males than females in the dataset. We can also examine race as a predictor, arranging the plots using the package *gridExtra*:

```r
# How about race?
p3 <- ggplot(census, aes(x = race, fill = race)) +
  geom_bar(stat='count', position='dodge', show.legend = FALSE) +
  labs(x = 'race, All data') +
  theme(legend.position="none") + theme_grey() + coord_flip()
p4 <- ggplot(census, aes(x = race, fill = income)) +
  geom_bar(stat='count', position='dodge', show.legend = FALSE) + labs(x = 'All data') +
  theme(legend.position="none") + theme_grey()+ coord_flip()
p5 <- ggplot(census, aes(x = race, fill = income)) +
  geom_bar(stat='count', position='stack', show.legend = FALSE) +
  labs(x = 'Is Race important?', y= "Count") + facet_grid(.~sex) +
  theme(legend.position="race") + theme_grey()+ coord_flip()
p6 <- ggplot(census, aes(x = race, fill = income)) +
  geom_bar(stat='count', position='fill', show.legend = FALSE) +
  labs(x = 'Is Race important?', y= "Percent") + facet_grid(.~sex) +
  theme(legend.position="none") + theme_grey()+ coord_flip()

library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'

## The following object is masked from 'package:dplyr':
```

```
##
##     combine
```

```r
grid.arrange(p3, p4, p5, p6, ncol=2)
```

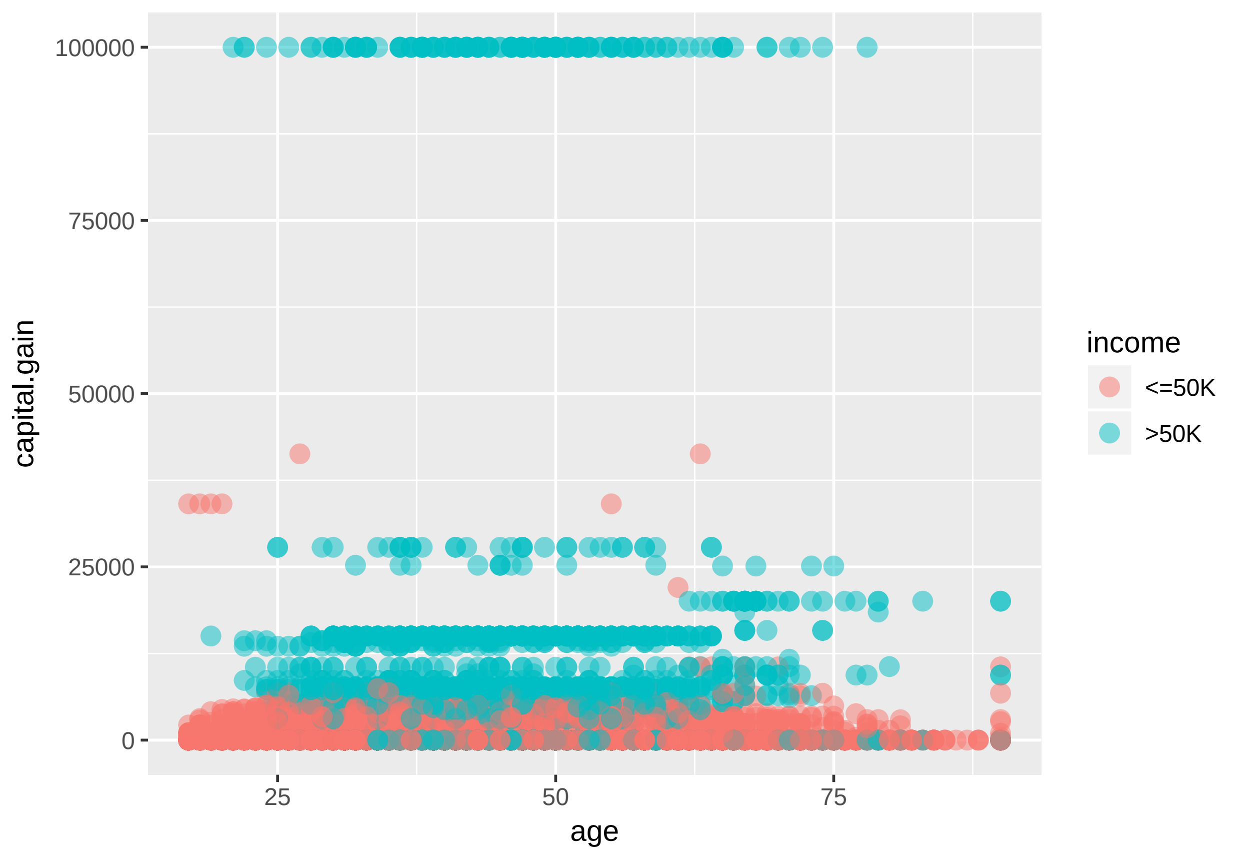


It seems a disproportionate number of white males earn greater than 50k. We could insert a dummy variable for if the person is a white male:

```r
census$WhiteMale <- ifelse(census$sex == " Male" &
                              census$race == " White", "Yes", "No")
census$WhiteMale <- as.factor(census$WhiteMale)
```

We can also do a simple scatter for numerical values. For example capital gains:

```r
p7 <- ggplot(census, aes(x = age, y = capital.gain, colour = income)) +
  geom_point(size = 3, alpha = 0.5)
p7
```



In this simple scatter, we can see a bunch of people paying $100,000 in capital gains. This appears to be a lot of people but a quick google tells me this is the maximum capital gains which can be paid. We do not have the exact income data, only the knowledge that they earn over $50k.

Missing data

I am not interested in the weights, so I remove this. Education is the same as the education number, so I remove this. Workclass is extremely similar to occupation, so I will remove workclass.

```
census$fnlwgt <- NULL
census$education <- NULL
census$workclass <- NULL
```

For the missing data, I will first convert all the question marks to NAs and find the resulting NAs:

```
# find elements
idx <- census == " ?"
# replace elements with NA
is.na(census) <- idx
# Find NAs
sapply(census, function(x) {sum(is.na(x))})
```

```
##            age  education.num marital.status     occupation   relationship
##              0              0              0           1843              0
##           race            sex   capital.gain   capital.loss hours.per.week
##              0              0              0              0              0
## native.country         income
##            583              0
```

There are 1843 observations missing from occupation, and 583 from native country. Native country has 42 categories and is therefore unhelpful for analysis. I will remove this variable after coding for the missingness. I also think that whether the person is from the US may have an effect, so I will add a dummy for this.

```
# Is Native country present? Then convert to factor.
census$native_country_present <- ifelse(is.na(census$native.country),"No", "Yes")
census$native_country_present <- as.factor(census$native_country_present)

# Add a variable denoting whether the native country is USA:
census$US_native <- ifelse(census$native.country == " United-States", "Yes", "No")
census$US_native <- as.factor(census$US_native)

# Finally, remove the native country:
census$native.country <- NULL
```

I think occupation is likely to be a useful feature for predicting income, so I will attempt to impute these values. While nothing is gained in terms of predictive power of the model, I think it a useful technique to be aware of. I will code for the missingness first.

```
# Is Occupation present? Then convert to factor
census$occupation_present <- ifelse(is.na(census$occupation),"No", "Yes")
census$occupation_present <- as.factor(census$occupation_present)

# Now, impute using MICE
library(mice)
tempData <- mice(census,m=5,maxit=50,meth='pmm',seed=500)
summary(tempData)
```

The missing data has now been imputed and can be changed back to census. We can summarise the data using *skimr*

```
census <- complete(tempData,1)

library(skimr)
skimmed <- skim_to_wide(census)
```

And we can view the summary in a tidy format:

```
skimmed
```

```
## # A tibble: 15 x 17
##       X1    type               variable missing complete     n n_unique
##    <int>   <chr>                  <chr>   <int>    <int> <int>    <int>
##  1     1  factor                 income       0    32561 32561        2
##  2     2  factor         marital.status       0    32561 32561        7
##  3     3  factor native_country_present       0    32561 32561        2
##  4     4  factor             occupation       0    32561 32561       14
##  5     5  factor     occupation_present       0    32561 32561        2
##  6     6  factor                   race       0    32561 32561        5
##  7     7  factor           relationship       0    32561 32561        6
##  8     8  factor                    sex       0    32561 32561        2
##  9     9  factor              US_native       0    32561 32561        2
## 10    10  factor              WhiteMale       0    32561 32561        2
## 11    11 integer                    age       0    32561 32561       NA
## 12    12 integer           capital.gain       0    32561 32561       NA
## 13    13 integer           capital.loss       0    32561 32561       NA
## 14    14 integer          education.num       0    32561 32561       NA
## 15    15 integer         hours.per.week       0    32561 32561       NA
## # ... with 10 more variables: top_counts <chr>, ordered <lgl>, mean <dbl>,
## #   sd <dbl>, p0 <int>, p25 <int>, p50 <int>, p75 <int>, p100 <int>,
## #   hist <chr>
```

From p1 it's clear there is a large imbalance in the data. There are several ways to deal with this. The models will be built using the *Caret* package so I will focus on carets built in methods. Briefly, they are downsampling the major class, upsampling the minor class, or using a hybrid of the 2 (SMOTE and ROSE). For more information see this website.

For simplicity, at this stage I will also convert the income variable to "High" and "Low", because some machine leanring algorithms are uncomfortbale with "=" in the algorithms. I will rename the data trainData.

```r
# Convert to high/low
levels(census$income) <- c("Low", "High")
# Remove census
trainData <- census
rm(census)
```

Model Setup

The next step is to set up the running of the models. I will attempt a range of well-known machine learning algorithms in an attempt to see which can obtain the best results. Since they are all done in *Caret*, they can be easily compared. I think the easiest way of tuning models is to do as I go along, so I will add different tuning parameters for each. *Caret* automatically picks the best model based on what I choose. I will focus on the ROC curve.

I will first define the formula used for the algorithms. This will be the same for each algorithm.

```r
# Add in the formula
form <- as.formula(income ~ age + education.num + marital.status +
                     occupation + relationship + race + sex + capital.gain +
                     capital.loss + hours.per.week + WhiteMale +
                     native_country_present + occupation_present + US_native)
```

I will define train control with normal, down and smote, and assess which method is best. These algorithms are implemented within the repeated cross validation.Repeated cross validation avoids the need for manually splitting a training and test set. It divides the data into k folds, holding 1 set for testing and the others for training. The resampling using "up" or "smote" is done to the training data only. This is repeated and the result is an average. This avoids over fitting. The diagram below shows how this works.

```r
# And train control with "normal", "down" and "smote"
library(DMwR)
ctrl <- trainControl(method= "repeatedcv",
                     number = 5,
                     repeats= 5,
                     summaryFunction = twoClassSummary,
                     classProbs = TRUE)

ctrl_down <- trainControl(method= "repeatedcv",
                     number = 5,
                     repeats= 5,
                     summaryFunction = twoClassSummary,
                     classProbs = TRUE,
                     sampling = "down")
```

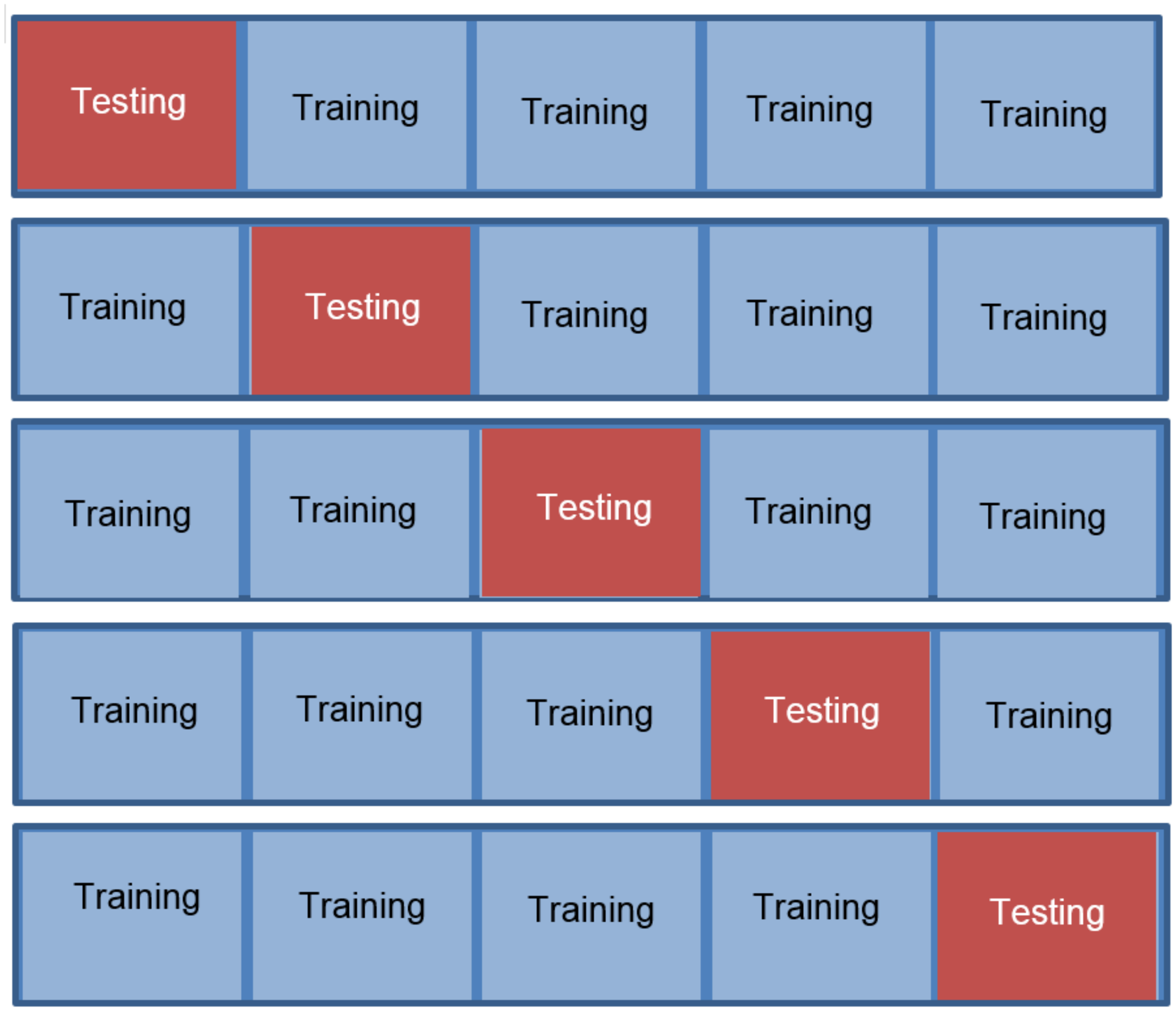

Testing
Training
Training
Training
Training
Training
Testing
Training
Training
Training
Training
Training
Testing
Training
Training
Training
Training
Training
Testing
Training
Training
Training
Training
Training
Testing

Figure 1:

```r
ctrl_smote <- trainControl(method= "repeatedcv",
                     number = 5,
                     repeats= 5,
                     summaryFunction = twoClassSummary,
                     classProbs = TRUE,
                     sampling = "smote")
```

I will configure multicore:

```r
# configure multicore
library(doParallel)
cl <- makeCluster(detectCores()-1)
registerDoParallel(cl)
```

And now train each model. I will avoid printing the result of each, as this will be clear in model evaluation. In each algorithm the formula defined is used, the data is the same, the numerical variables are normalised to fall between 0 and 1 (to make all variables comparable) and the tuning is unique for each algorithm.

The easiest way to run these models is to start off by defining a simple function:

```r
train_models <- function(method, ctrl, tuning) {
  model.train <- train(form,
                 data = trainData,
                 method = method,
                 trControl = ctrl,
                 tuneGrid = tuning,
                 maxit = 1000000,
                 preProcess = c('range'))

  return(model.train)

}
```

And now to implement the function, after defining the algorithms tuning parameters

Firstly, the extreme gradient bosoting algorithm:

```r
# 1. Extreme gradient boosting model
set.seed(2018)
tune.grid.xgb <- expand.grid(eta = c(0.05, 0.075, 0.1),
                             nrounds = c(50, 75, 100),
                             max_depth = 6:8,
                             min_child_weight = c(2.0, 2.25, 2.5),
                             colsample_bytree = c(0.3, 0.4, 0.5),
                             gamma = 0,
                             subsample = 1)

# Normal non-sampled
model_xgb <- train_models("xgbTree", ctrl, tune.grid.xgb)

# Down Sampled
model_xgb_down <- train_models("xgbTree", ctrl_down, tune.grid.xgb)

# Smote Sampled
model_xgb_smote <- train_models("xgbTree", ctrl_smote, tune.grid.xgb)
```

The random Forest:

```r
set.seed(2018)
tune.grid.rf <- expand.grid(.mtry = sqrt(ncol(trainData)))

# Normal non-sampled
model_rf <- train_models("rf", ctrl, tune.grid.rf)

# Down Sampled
model_rf_down <- train_models("rf", ctrl_down, tune.grid.rf)

# Smote Sampled
model_rf_smote <- train_models("rf", ctrl_smote, tune.grid.rf)
```

The Naive Bayes Model:

```r
set.seed(2018)
tune.grid.nb <- data.frame(fL=c(0,0.5,1.0), usekernel = TRUE,
                           adjust=c(0,0.5,1.0))

# Normal non-sampled
model_nb <- train_models("nb", ctrl, tune.grid.xgb)

# Down Sampled
model_nb_down <- train_models("nb", ctrl_down, tune.grid.xgb)

# Smote Sampled
model_nb_smote <- train_models("nb", ctrl_smote, tune.grid.xgb)
```

A model averaged neural net model

```r
set.seed(2018)
library(nnet)
tune.grid.nn <-  expand.grid(size=c(10), decay=c(0.1))

# Normal non-sampled
model_net <- train_models("nnet", ctrl, tune.grid.nn)

# Down Sampled
model_net_down <- train_models("nnet", ctrl_down, tune.grid.nn)

# Smote Sampled
model_net_smote <- train_models("nnet", ctrl_smote, tune.grid.nn)
```

The logistic regression with elastic net regularisation:

```r
set.seed(2018)
lamda.grid <- 10^seq(2, -2, length = 100)
alpha.grid <- seq(0,1, length = 10)
srchGrd <-  expand.grid(.alpha = alpha.grid,
                        .lambda = lamda.grid)

# Normal non-sampled
model_xgb <- train_models("glmnet", ctrl, srchGrd)

# Down Sampled
model_xgb_down <- train_models("glmnet", ctrl_down, srchGrd)
```

```r
# Smote Sampled
model_xgb_smote <- train_models("glmnet", ctrl_smote, srchGrd)
```

Evaluating Models

Now the models can be evaluated. First, use carets resamples function to group the models, and then create the scales for plotting the graphs.

```r
# Group the models
models_compare <- resamples(list(RF = model_rf,
                                 XGB = model_xgb,
                                 ENet = model_glm,
                                 NNet = model_net,
                                 NB = model_nb))

models_compare_down <- resamples(list(RF = model_rf_down,
                                 XGB = model_xgb_down,
                                 ENet = model_glm_down,
                                 NNet = model_net_down,
                                 NB = model_nb_down))

models_compare_smote <- resamples(list(RF = model_rf_smote,
                                       XGB = model_xgb_smote,
                                       ENet = model_glm_smote,
                                       NNet = model_net_smote,
                                       NB = model_nb_smote))

# Create the scales for the plotting
scales <- list(x=list(relation="free"), y=list(relation="free"))
```

Now the plots can be generated. Firstly, I will look at the normal (non-resampled) results.

```r
# Box plot
boxplot <- bwplot(models_compare, scales=scales)
# Density Plot
denplot <- densityplot(models_compare, scales=scales, pch = "|")
# Dotplot
dotplot <- dotplot(models_compare, scales=scales)
# Parallel plot
paraplot <- parallelplot(models_compare)
# Scatterplot
scatter <- splom(models_compare)
```

And View them:

```r
boxplot
```

```r
denplot
```

```r
dotplot
```

```r
paraplot
```

```r
scatter
```

And for the down sampled models:

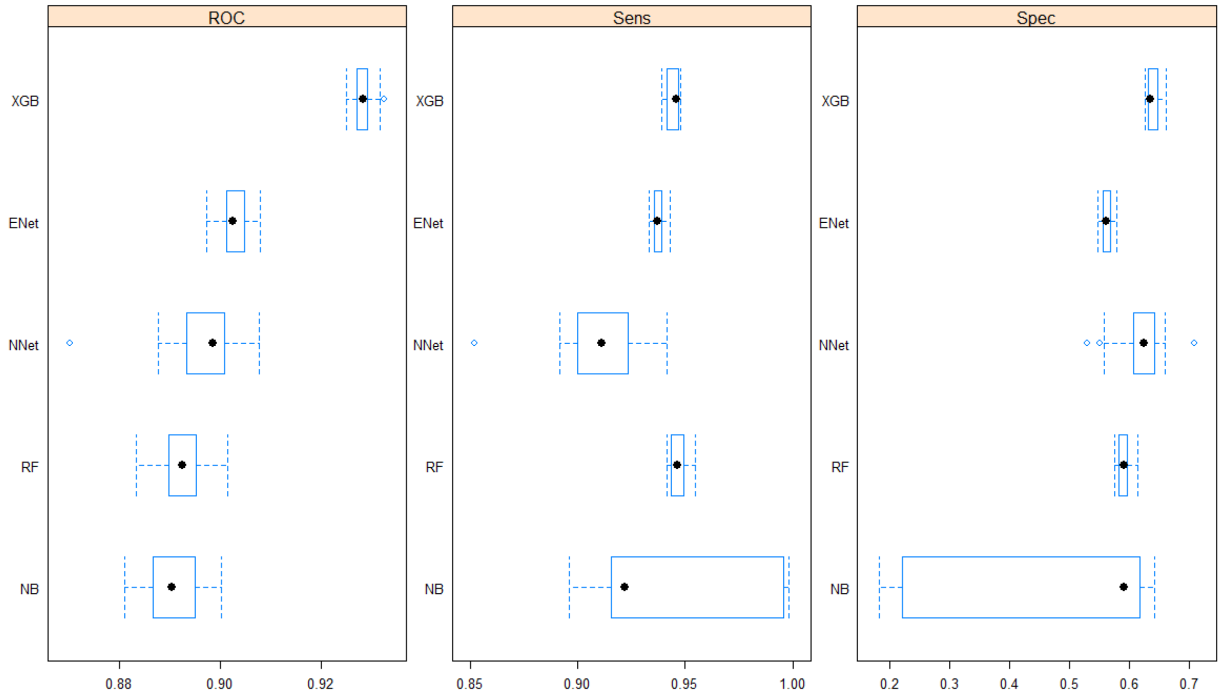


Figure 2:

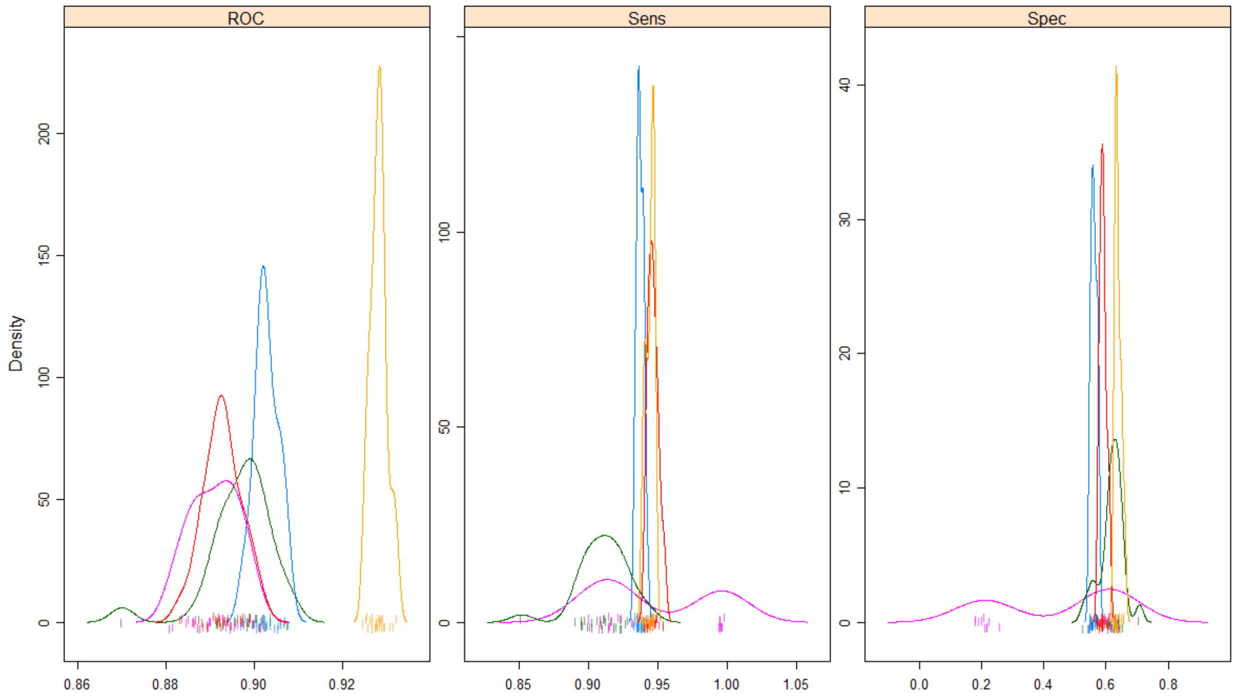


Figure 3:

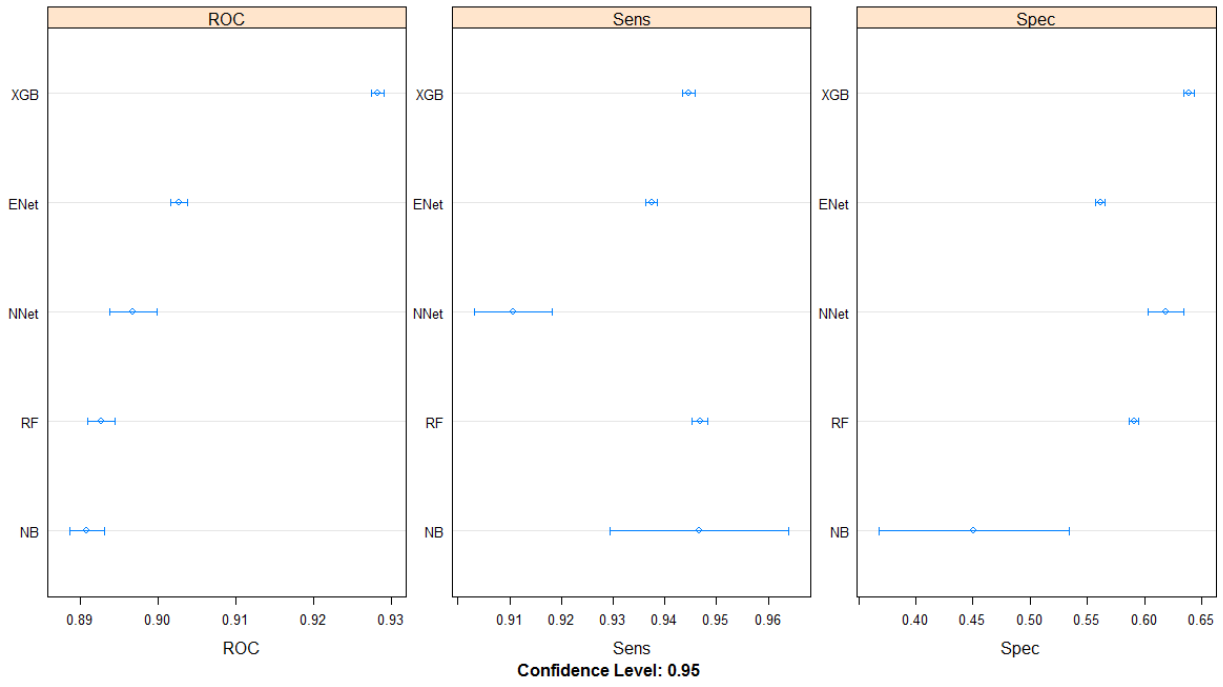


Figure 4:

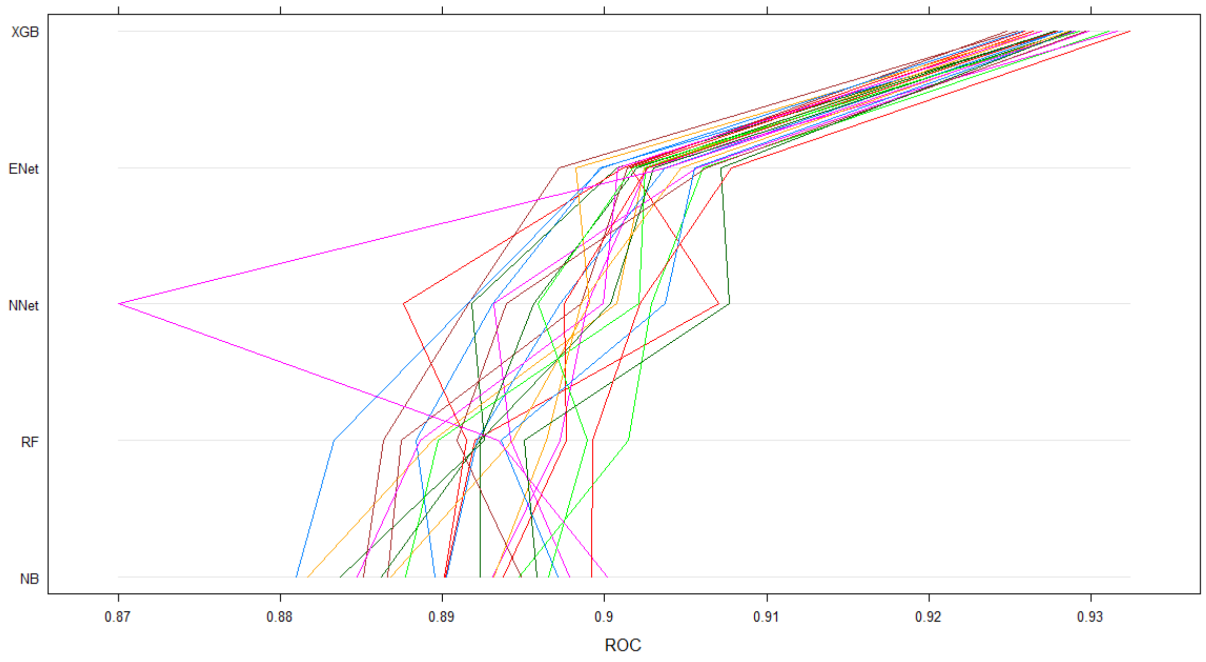


Figure 5:

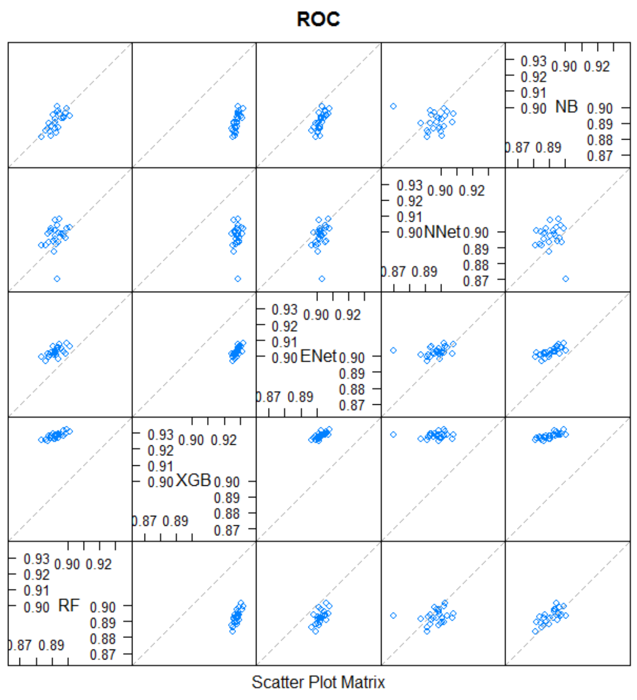


Figure 6:

```r
# Box plot
boxplot_down <- bwplot(models_compare_down, scales=scales)

# Density Plot
denplot_down <- densityplot(models_compare_down, scales=scales, pch = "|")

# Dotplot
dotplot_down <- dotplot(models_compare_down, scales=scales)

# Parallel plot
paraplot_down <- parallelplot(models_compare_down)

# Scatterplot
scatter_down <- splom(models_compare_down)
```

And View them

```r
boxplot_down
```

```r
denplot_down
```

```r
dotplot_down
```

```r
paraplot_down
```

```r
scatter_down
```

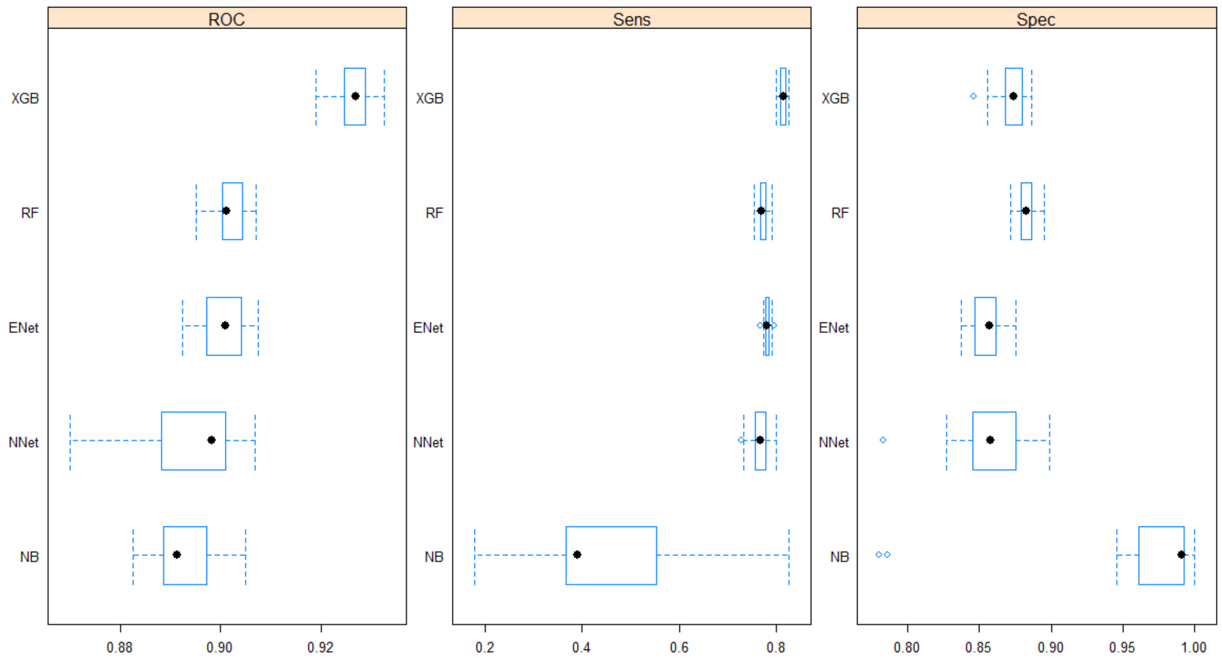


Figure 7:

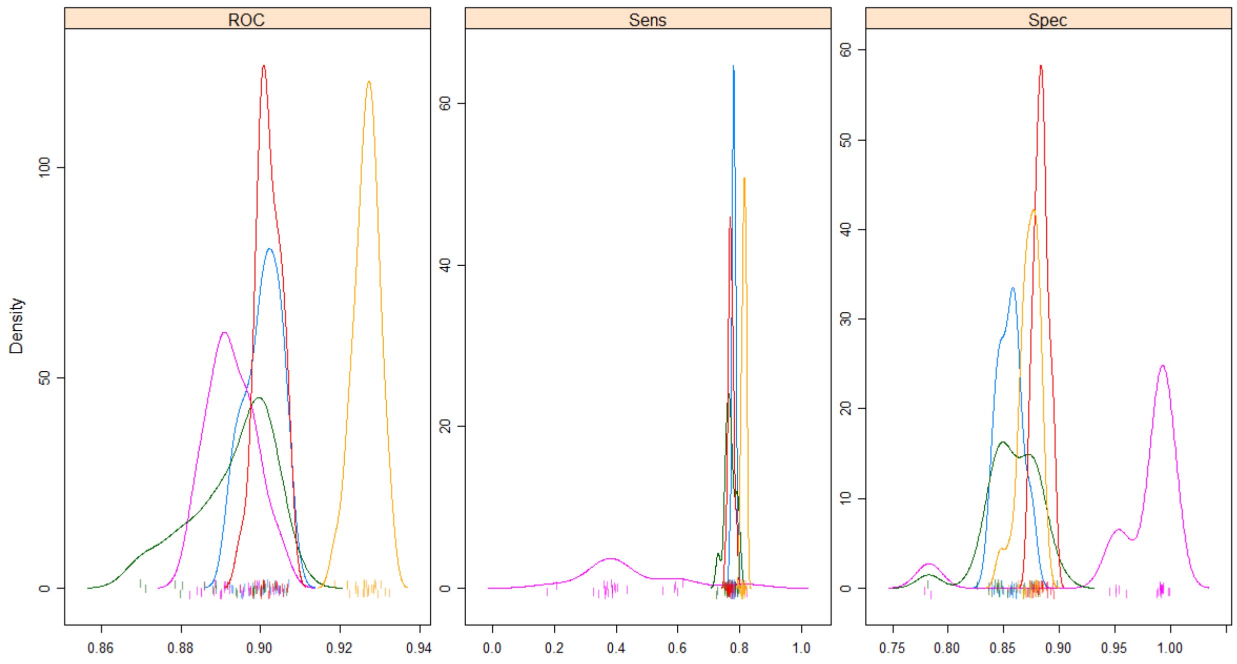


Figure 8:

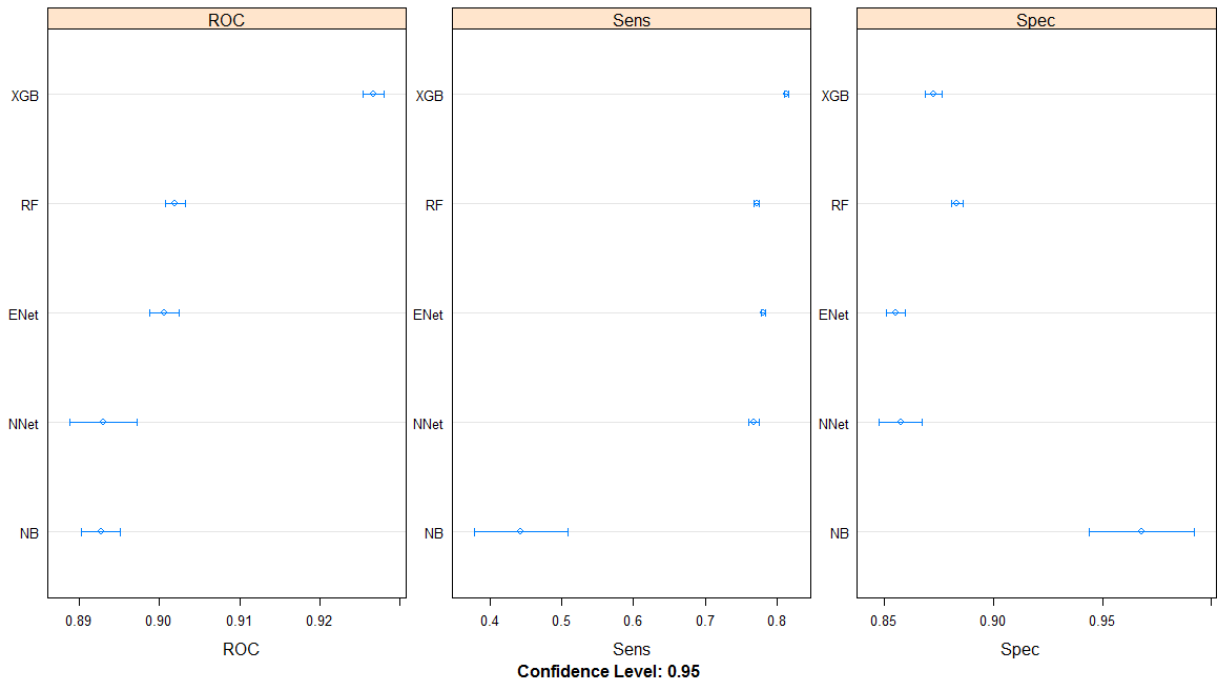


Figure 9:

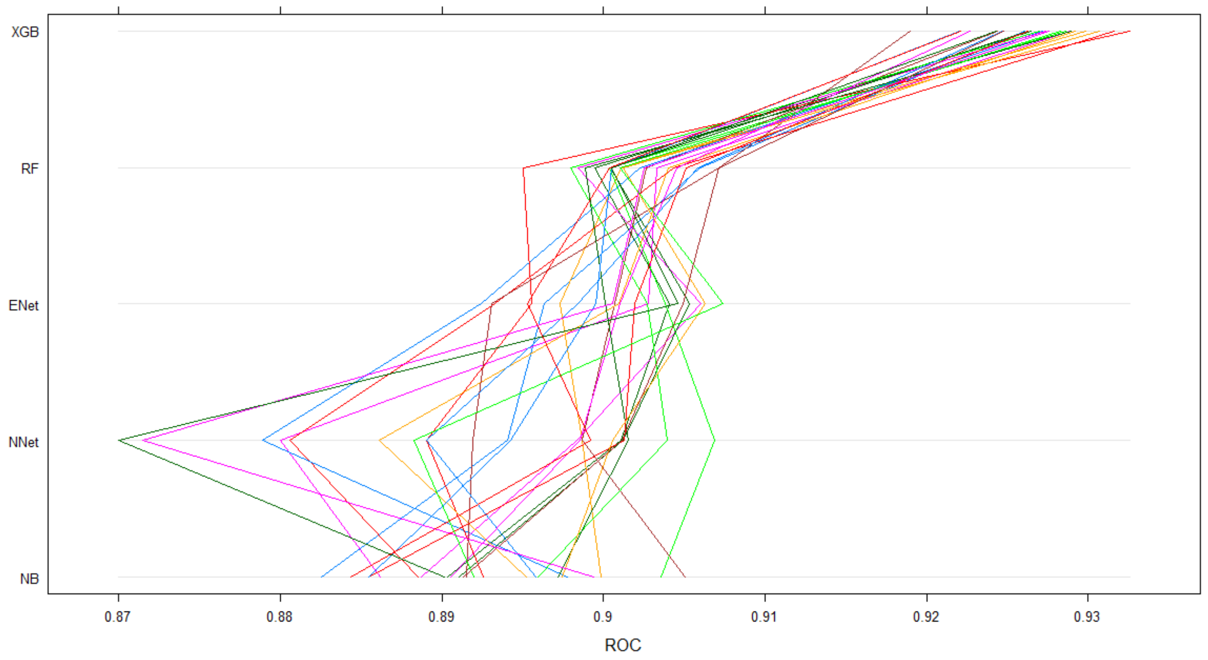
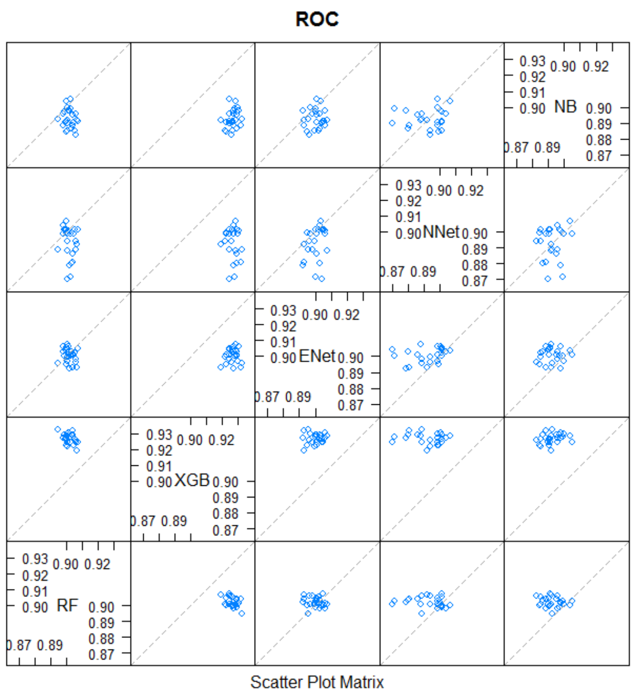


Figure 10:



And for the samples with SMOTE:

```
# Box plot
boxplot_smote <- bwplot(models_compare_smote, scales=scales)
# Density Plot
denplot_smote <- densityplot(models_compare_smote, scales=scales, pch = "|")
# Dotplot
dotplot_smote <- dotplot(models_compare_smote, scales=scales)
# Parallel plot
paraplot_smote <- parallelplot(models_compare_smote)
# Scatterplot
scatter_smote <- splom(models_compare_smote)
```

```
boxplot_smote
```

```
denplot_smote
```

```
dotplot_smote
```

```
paraplot_smote
```

```
scatter_smote
```

Feature Importance

Now we can look at the feature importance of each of the models.

For each model we can use the *varImp* function from caret to rank the relative importance and scale it between 0 and 100. THey can then be viewed as combined plots for each resampled set of models. For example, by defining a function which can be used for each plot:

```
# XGB
# Define Function
plot_imp <- function(FI_plot, model, title) {
```

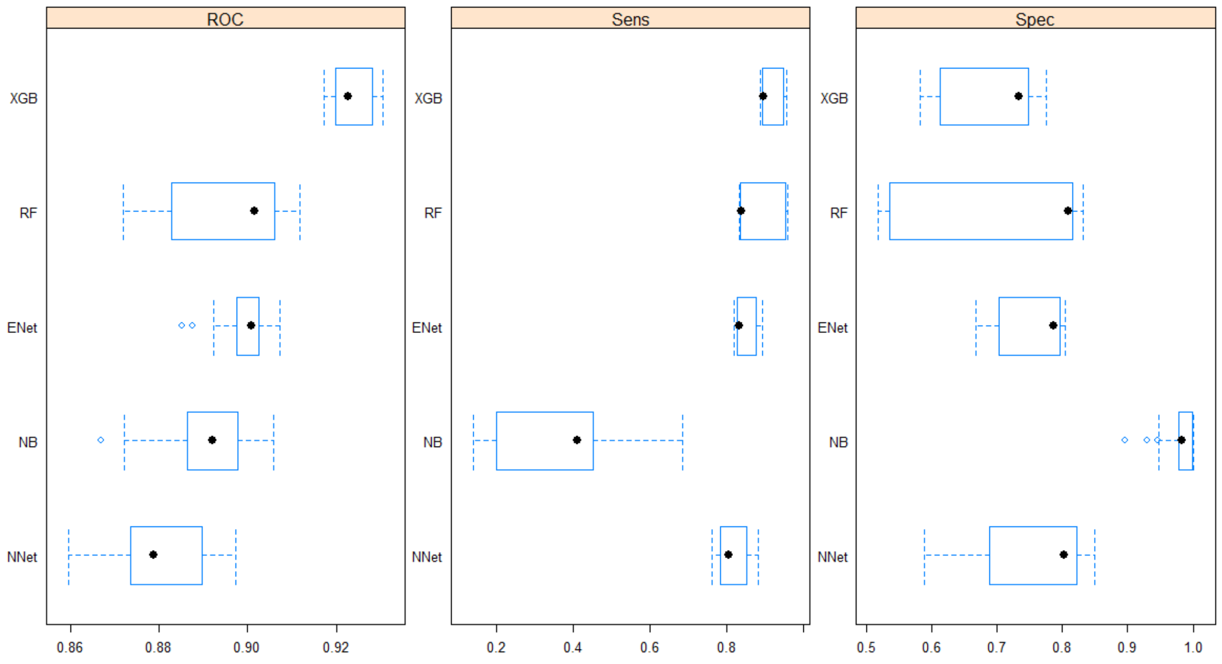


Figure 11:

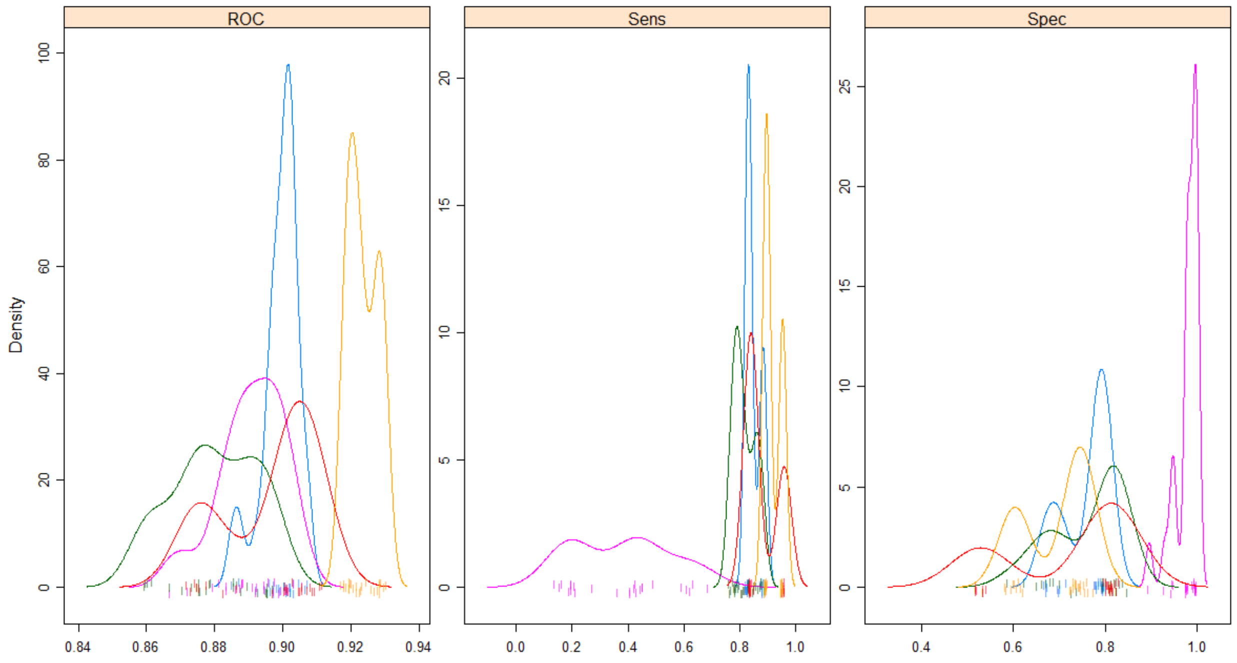


Figure 12:

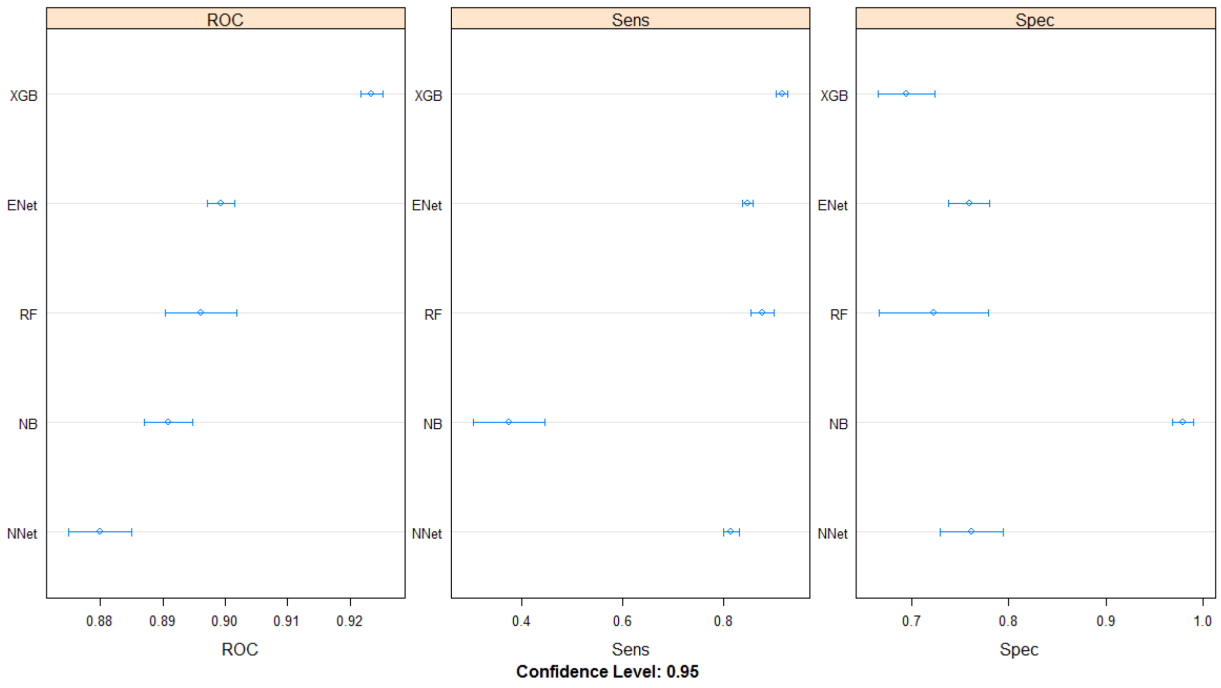


Figure 13:

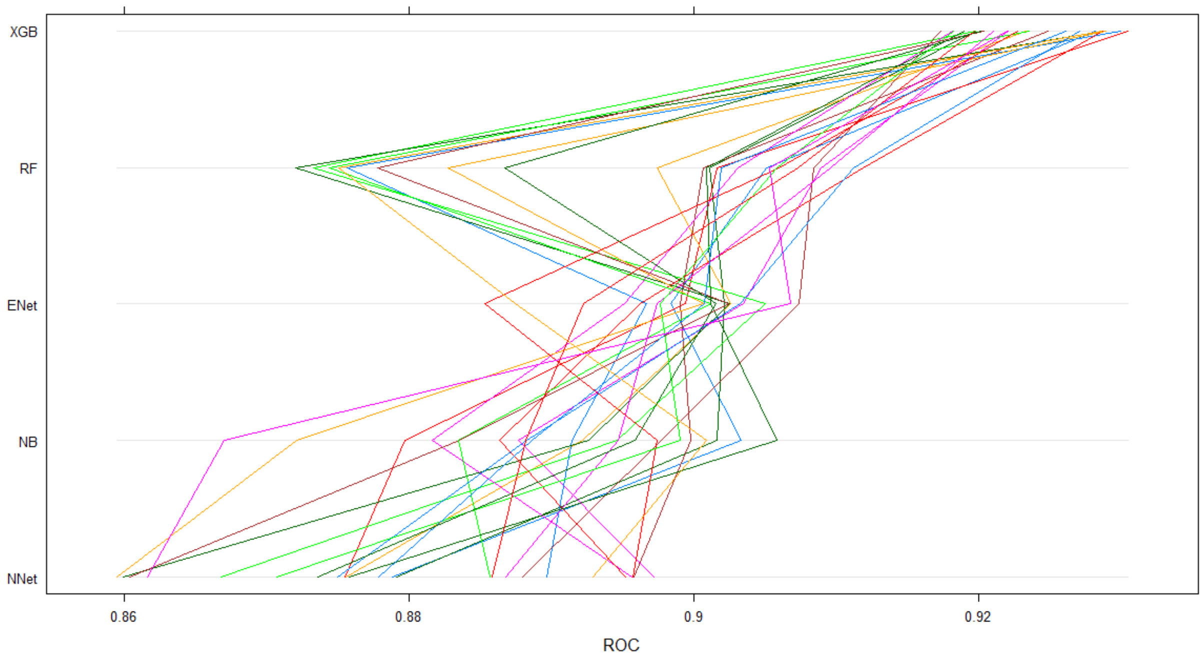


Figure 14:

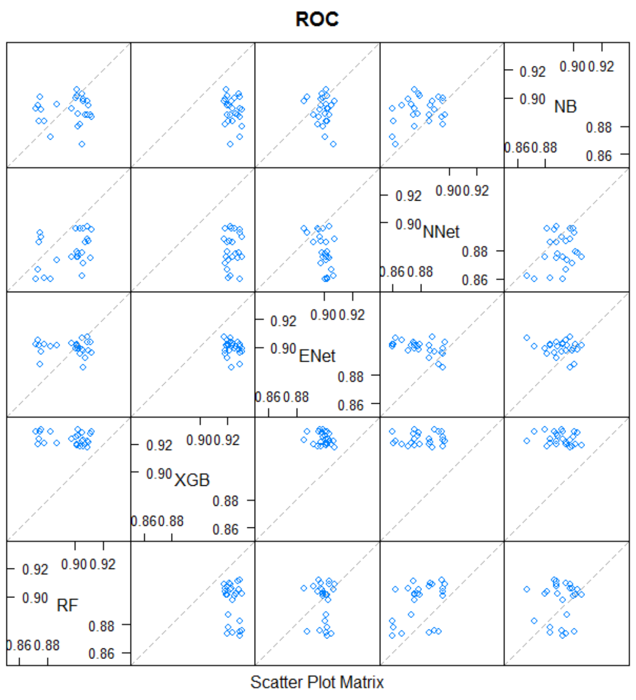


Figure 15:

```r
    FI_mod <- varImp(model)
    FI_dat <- FI_mod$importance
    FI_gini <- data.frame(Variables = row.names(FI_dat),
                          MeanDecreaseGini = FI_dat$Overall)

    FI_plot <- ggplot(FI_gini, aes(x=reorder(Variables, MeanDecreaseGini),
                                   y=MeanDecreaseGini, fill= "green")) +
              geom_bar(stat='identity') + coord_flip() +
              theme(legend.position="none") + labs(x="") +
              ggtitle(title) +
              theme(plot.title = element_text(hjust = 0.5))

    return(FI_plot)
}

# Run function, for example for GLM:
glm_imp_plot <- plot_imp(glm_plot, glm_model, "GLM Feature Importance")
```

Once we have the plots, they can be viewed like so:

```r
library(gridExtra)

# Normal
Normal_plots_imp <- grid.arrange(xgb_imp_plot, rf_imp_plot, nb_imp_plot,
             net_imp_plot, glm_imp_plot, ncol=2)
```

I will ignore the others since the resampling seems to have had very little effect on model accuracy. Based on these features, the ones which appear consistently are capital gains, marital status (married), education, age, occupation (professional specialty), Sex, hours per week, marital status (divorced), capital loss, relationship

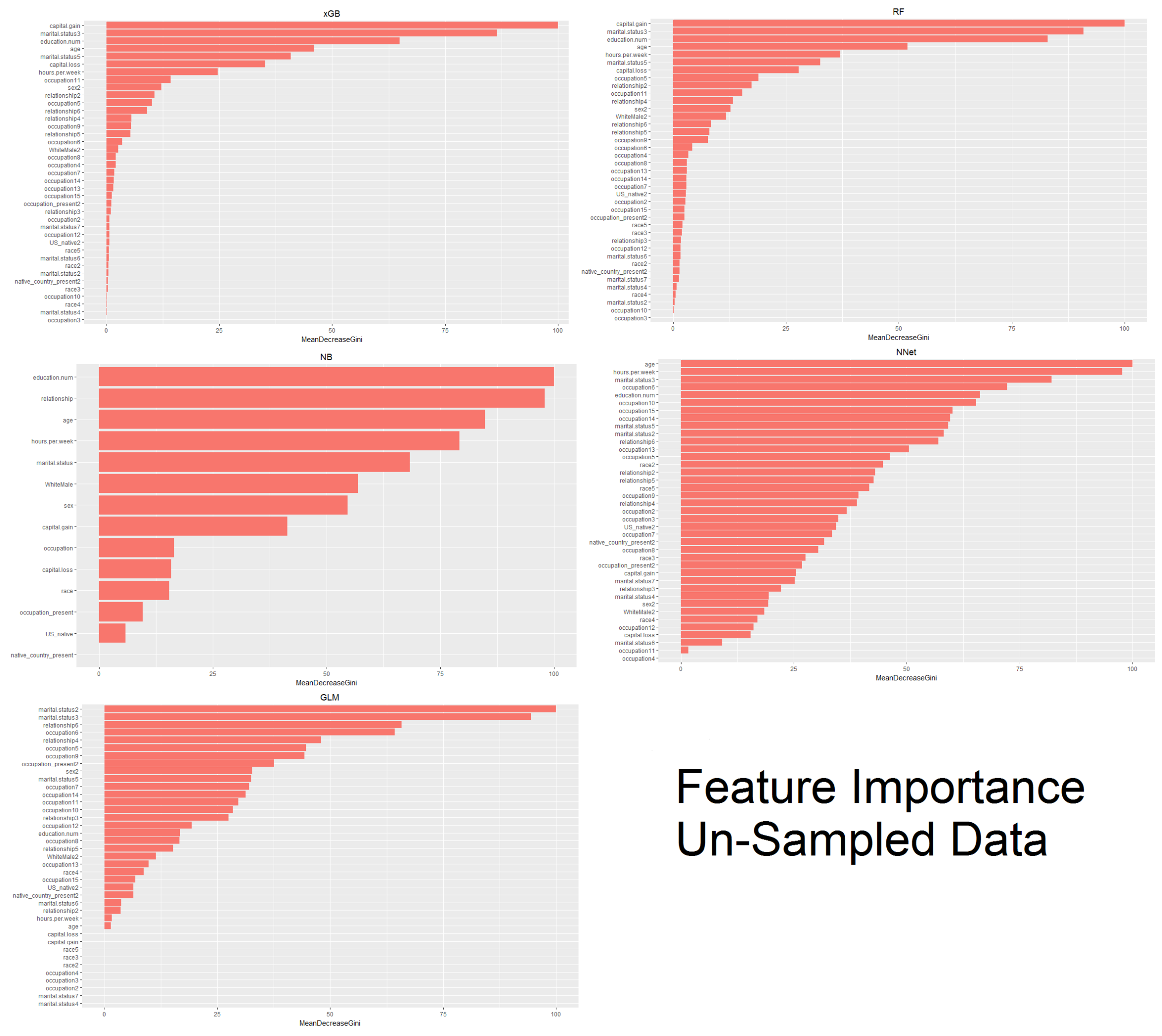

xGB
RF
NB
NNet
GLM
MeanDecreaseGini
0
25
50
75
100
Feature Importance
Un-Sampled Data

Figure 16:

(husband). This is broadly what we would expect.

We can also decide on the most important features by combining the importance and displaying the results using Tableau.

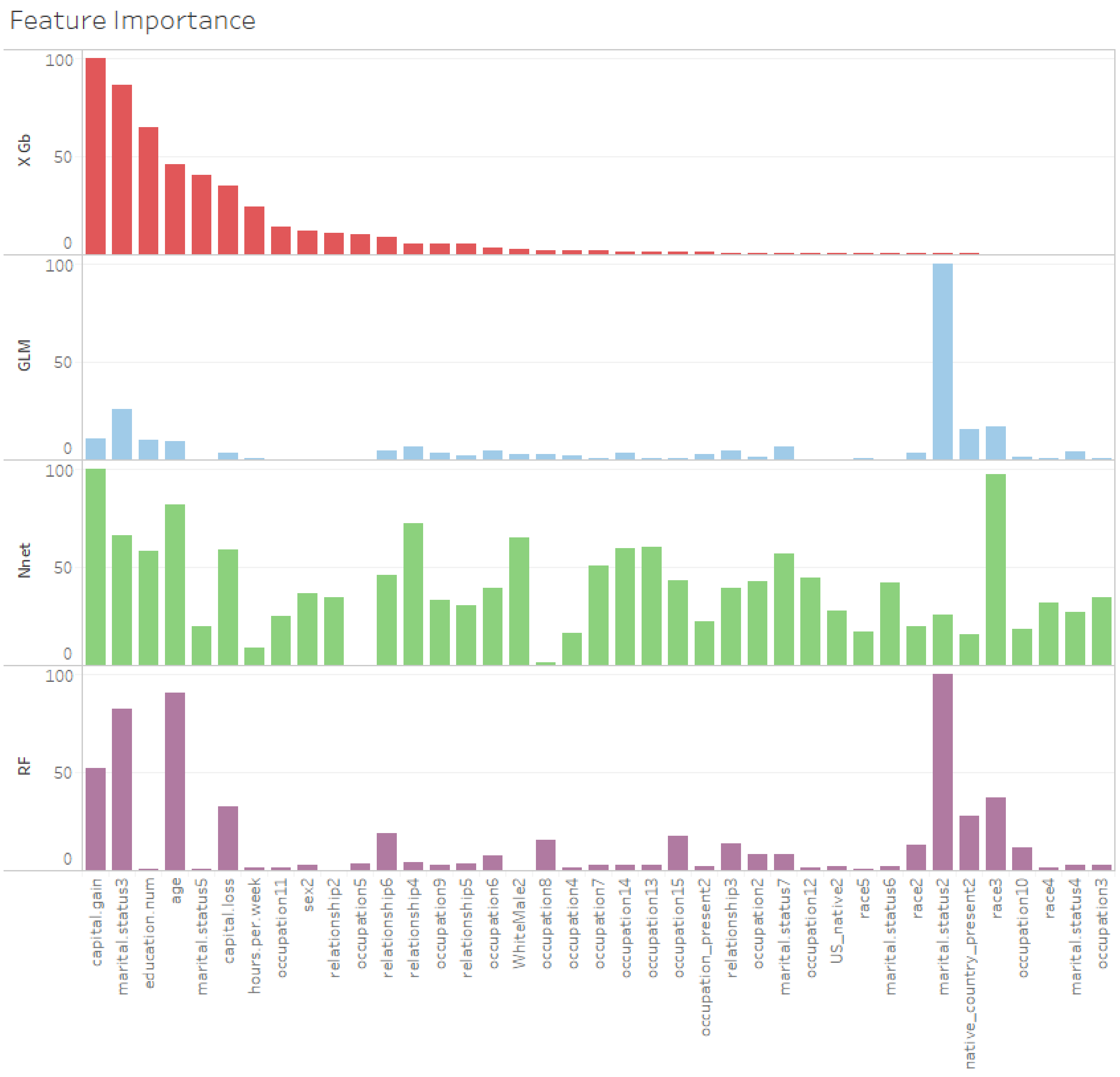
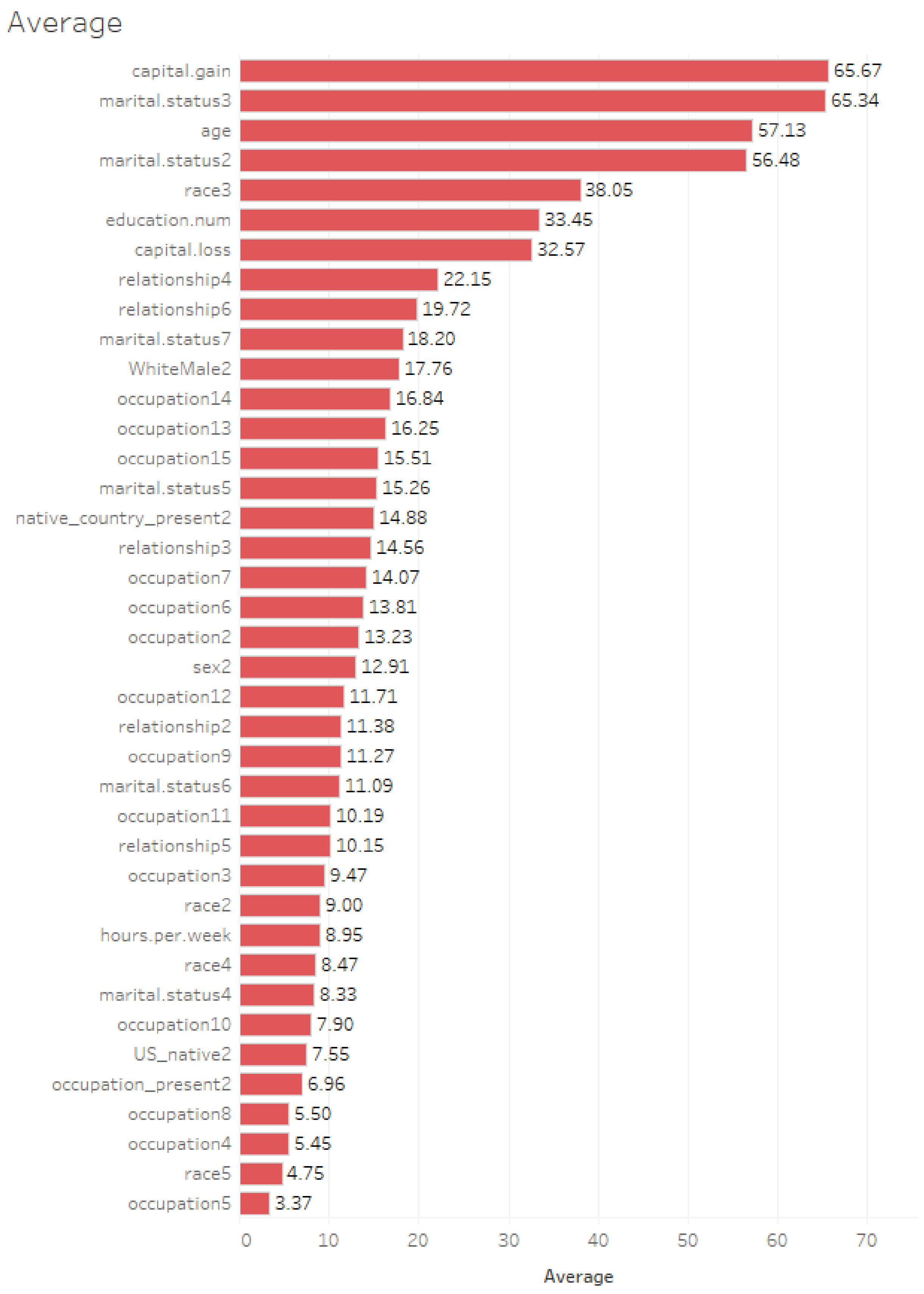


Figure 17:



Average
capital.gain 65.67
marital.status3 65.34
age 57.13
marital.status2 56.48
race3 38.05
education.num 33.45
capital.loss 32.57
relationship4 22.15
relationship6 19.72
marital.status7 18.20
WhiteMale2 17.76
occupation14 16.84
occupation13 16.25
occupation15 15.51
marital.status5 15.26
native_country_present2 14.88
relationship3 14.56
occupation7 14.07
occupation6 13.81
occupation2 13.23
sex2 12.91
occupation12 11.71
relationship2 11.38
occupation9 11.27
marital.status6 11.09
occupation11 10.19
relationship5 10.15
occupation3 9.47
race2 9.00
hours.per.week 8.95
race4 8.47
marital.status4 8.33
occupation10 7.90
US_native2 7.55
occupation_present2 6.96
occupation8 5.50
occupation4 5.45
race5 4.75
occupation5 3.37
0
10
20
30
40
50
60
70
Average

Figure 18: