IS6481-Fall2018-Assignment-4: Binary Classifier Bake-Off

Bhuvananjali Challagalla

Load Libraries

```
library(tidyverse) # Beautiful Data Munging and visualizations
library(data.table) # Brlazingly fast Data Munging
library(harrypotter) # Pretty Colour Palette
library(mice) # Predictive Imputation of Missing Values

library(rpart)
library(e1071)
library(ROCR)
library(pROC)
```

Retrieve Data: The RMS Titanic Tragedy

We are going to check our predictive skills making use of the very popular dataset of the RMS Titanic Passangers.

With this data set we have the opportunity to test which characteristics of the passengers made them more prone to surviving the shipwreck. And it is a great opportunity to test simple binary classification models, in which we'll need to predict whether a passengers survives.

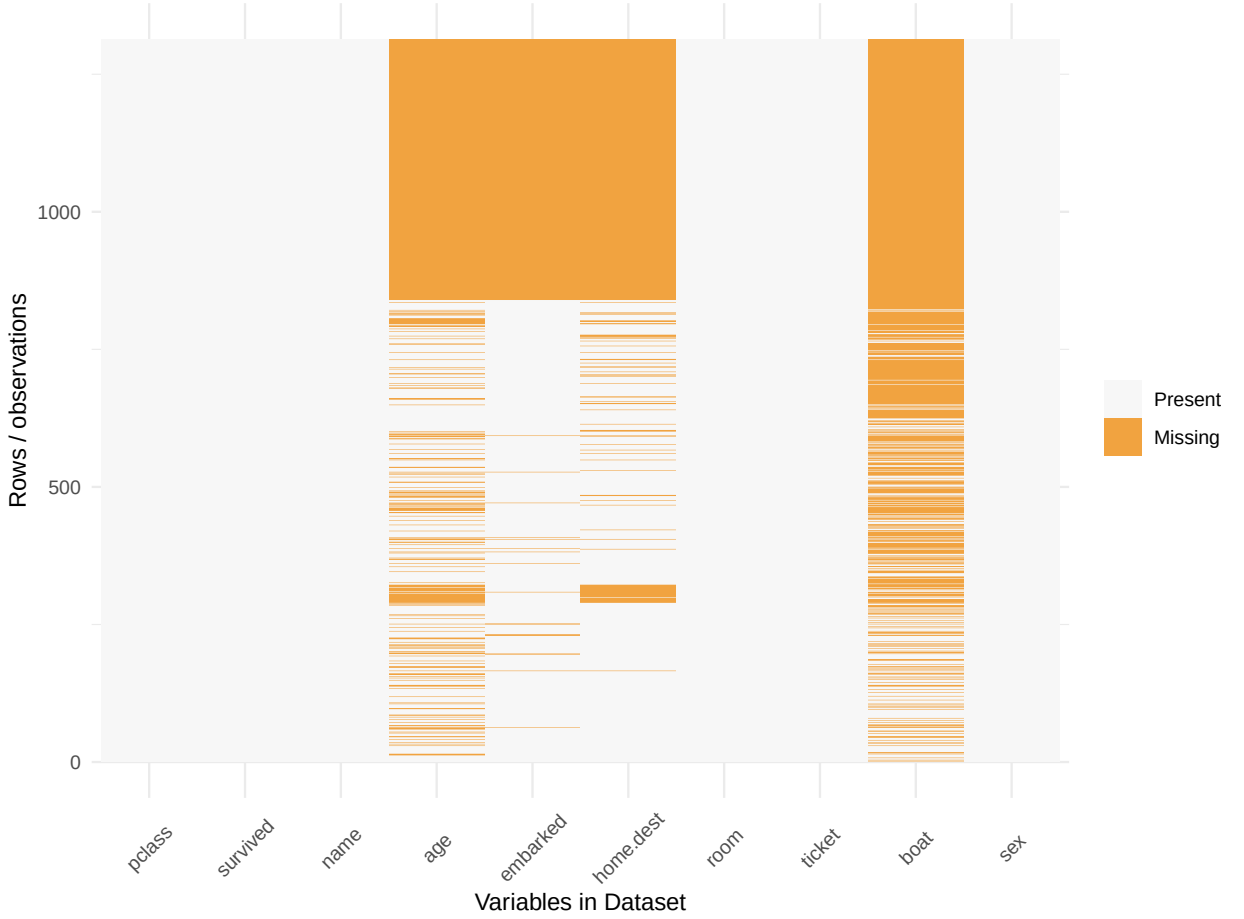
```
load(url("http://biostat.mc.vanderbilt.edu/wiki/pub/Main/DataSets/titanic.sav"))
titanic %>% glimpse()
```

```
## Observations: 1,313
## Variables: 10
## $ pclass    <fct> 1st, 1st, 1st, 1st, 1st, 1st, 1st, 1st, 1st, 1st, 1s...
## $ survived  <int> 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0...
## $ name      <chr> "Allen, Miss Elisabeth Walton", "Allison, Miss Helen...
## $ age       <dbl> 29.0000, 2.0000, 30.0000, 25.0000, 0.9167, 47.0000, ...
## $ embarked  <fct> Southampton, Southampton, Southampton, Southampton, ...
## $ home.dest <fct> St Louis, MO, Montreal, PQ / Chesterville, ON, Montr...
## $ room      <chr> "B-5", "C26", "C26", "C26", "C22", "E-12", "D-7", "A...
## $ ticket    <chr> "24160 L221", "", "", "", "", "", "13502 L77", "", "...
## $ boat      <fct> 2, NA, (135), NA, 11, 3, 10, NA, 2, (22), (124), 4, ...
## $ sex       <fct> female, female, male, female, male, male, female, ma...
```

Missing Data

Remove all the missing data

Firstly, we can take a look at where they are:



Looking at this map, what we can see is that we have some columns with a huge proportion of missing values.

Firstly, we should note that there are some `NA` values camouflaged as characteres that say `<NA>` or nothing at all `""`. Those are `NA` too.

```
hidden_na <- function(x) x %in% c("<NA>", "")

titanic <- titanic %>% mutate_all(funs(ifelse(hidden_na(.), NA, .)))
```

And then we do bootstrap in order to fill the gaps:

```
imputed <- na_replace(titanic)
```

```
## ... 10% ...
## ... 20% ...
## ... 30% ...
## ... 40% ...
## ... 50% ...
## ... 60% ...
## ... 70% ...
## ... 80% ...
## ... 90% ...
## ... 100% ...
```

```
titanic <- imputed
```

Feature Engineering

Deck

Looking at the values of the `room` variable, we can see some funny pattern. The first letter of the variable is telling us the `Deck`. So we can make a new variable out of this:

```
titanic$deck <- factor(sapply(titanic$room, function(x) strsplit(x,NULL)[[1]][[1]]))
```

Title

Where the Titanic sinked, society was much more classist than now, and so we can fairly assume that those people with a proper Title in their name would have had more chances to get help from the security forces of the ship, or maybe access to more robust survival resources. Anyway, we are going to craft a variable trying to stuff this information into a predictive feature.

```
titanic$title <- sub("[[:space:]].*", "", gsub('(.*, )|(\\..*)', '', titanic$name))

titles <- c("Mr", "Rev", "Miss", "Mrs", "Ms", "Dr")

titanic <-
    titanic %>%
    mutate(title = ifelse(title %in% titles, title, "none"))

titanic$name <- NULL
```

Training the Models

We firstly set character variables as factors

```
data <- titanic
features <- colnames(data)

for(f in features) {
    if ((class(data[[f]])=="factor") || (class(data[[f]])=="character")) {
        levels <- unique(data[[f]])
        data[[f]] <- (factor(data[[f]], levels=levels))
    }
}
titanic <- data
rm(data);gc()
```

```
##           used  (Mb) gc trigger  (Mb) max used  (Mb)
## Ncells 2161824 115.5    3774107 201.6  3774107 201.6
## Vcells 3821105  29.2    8388608  64.0  8388156  64.0
```

```
titanic %>% glimpse()
```

```
## Observations: 1,313
## Variables: 11
## $ pclass    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
## $ survived  <int> 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0...
## $ age       <dbl> 29.0000, 2.0000, 30.0000, 25.0000, 0.9167, 47.0000, ...
## $ embarked  <int> 3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 1, 1, 1, 3, 3, 1, 1, 1...
## $ home.dest <int> 311, 232, 232, 232, 232, 238, 163, 25, 23, 230, 238,...
## $ room      <fct> B-5, C26, C26, C26, C22, E-12, D-7, A-36, C-101, B-3...
## $ ticket    <fct> 24160 L221, 17608 L262 7s 6d, 17485 L56 18s 7d, 1113...
## $ boat      <int> 87, 79, 12, 87, 79, 88, 78, 15, 87, 34, 8, 89, 95, 9...
## $ sex       <int> 1, 1, 2, 1, 2, 2, 1, 2, 1, 2, 2, 1, 1, 2, 2, 1, 2, 2...
## $ deck      <fct> B, C, C, C, C, E, D, A, C, B, C, E, B, A, B, B, B, C...
## $ title     <fct> Miss, Miss, Mr, Mrs, none, Mr, Miss, none, Mrs, Mr, ...
```

Split into Training & Test sets

```
tr_id <- sample(1:nrow(titanic), nrow(titanic)*0.7)
train <- titanic[tr_id,]
test  <- titanic[-tr_id,]
```

Binary Logistic Regression

This model uses a logistic function to model the binary classification problem. Since the logistic function is bounded between 0 and 1, its output can be thought of as a probability. That way we can label as 1 any output greater than 0.5 and as 0 any output lower than 0.5.

```
blr_model <- glm(survived ~., family=binomial(link='logit'), data = train)

blr_preds_test <- ifelse(predict(blr_model,test, type = "response") > 0.5, 1, 0)
blr_preds_train <- ifelse(predict(blr_model,train, type = "response") > 0.5, 1, 0)

tr_acc <- round(1 - mean(blr_preds_train != train$survived), 4)
te_acc <- round(1 - mean(blr_preds_test  != test$survived), 4)
print(paste('Train Set Accuracy: ', tr_acc))
```

```
## [1] "Train Set Accuracy:  0.8487"
```

```
print(paste('Test Set Accuracy: ', te_acc))
```

```
## [1] "Test Set Accuracy:  0.7665"
```

In general, the *Train Set Accuracy* is always to be higher that the test set. We'll say that our model is overfitting if it is *much* higher. How much is that? There is no clear cut answer. But we can measure the *degree of overfitting* that our model is doing, by computing the ratio between both accuracies:

```
oc_blr <- round(100*(tr_acc - te_acc)/tr_acc,6)
print(paste0("Overfitting Coefficient: ", oc_blr))
```

```
## [1] "Overfitting Coefficient: 9.685401"
```

The greater this number, the worse is the overfitting.

Naive Bayes Classifier

This is a simple model that uses Bayes' theorem to assign probability to class labels based on the conditional probability of the features.

```
nbc_model <- naiveBayes(as.factor(survived)~., data = train)

nbc_preds_test <- ifelse(predict(nbc_model,test, type = "raw")[,2] > 0.5, 1, 0)
nbc_preds_train <- ifelse(predict(nbc_model,train, type = "raw")[,2] > 0.5, 1, 0)

tr_acc <- round(1 - mean(nbc_preds_train != train$survived), 4)
te_acc <- round(1 - mean(nbc_preds_test  != test$survived), 4)
print(paste('Train Set Accuracy: ', tr_acc))
```

```
## [1] "Train Set Accuracy:  0.7867"
```

```
print(paste('Test Set Accuracy: ', te_acc))
```

```
## [1] "Test Set Accuracy:  0.7792"
```

```
oc_nbc <- round(100*(tr_acc - te_acc)/tr_acc,6)
print(paste0("Overfitting Coefficient: ", oc_nbc))
```

```
## [1] "Overfitting Coefficient: 0.953349"
```

Recursive Partitioning Tree Model

Recursive partition is a method that creates recursive decision trees in order to split the population of variables into sub-populations, hence constructing probabilities in order to label the output.

```
rpart_model <- rpart(as.factor(survived)~., data = train)

rpart_preds_test <- ifelse(predict(rpart_model,test)[,2] > 0.5, 1, 0)
rpart_preds_train <- ifelse(predict(rpart_model,train)[,2] > 0.5, 1, 0)

tr_acc <- round(1 - mean(rpart_preds_train != train$survived), 4)
te_acc <- round(1 - mean(rpart_preds_test  != test$survived), 4)
print(paste('Train Set Accuracy: ', tr_acc))
```

```
## [1] "Train Set Accuracy:  0.8618"
```

```
print(paste('Test Set Accuracy: ', te_acc))
```

```
## [1] "Test Set Accuracy:  0.7716"
```

```
oc_rpart <- round(100*(tr_acc - te_acc)/tr_acc,6)
print(paste0("Overfitting Coefficient: ", oc_rpart))
```

```
## [1] "Overfitting Coefficient: 10.466466"
```

Comparing Results

So far we've seen the results of all three models and we can have a rough idea of what model we would prefer. However, I think measuring this once can be misleading, for the values of the overfitting coefficient may vary considerable just because of randomness. In order to smooth out this uncertainty, I decided to iterate this

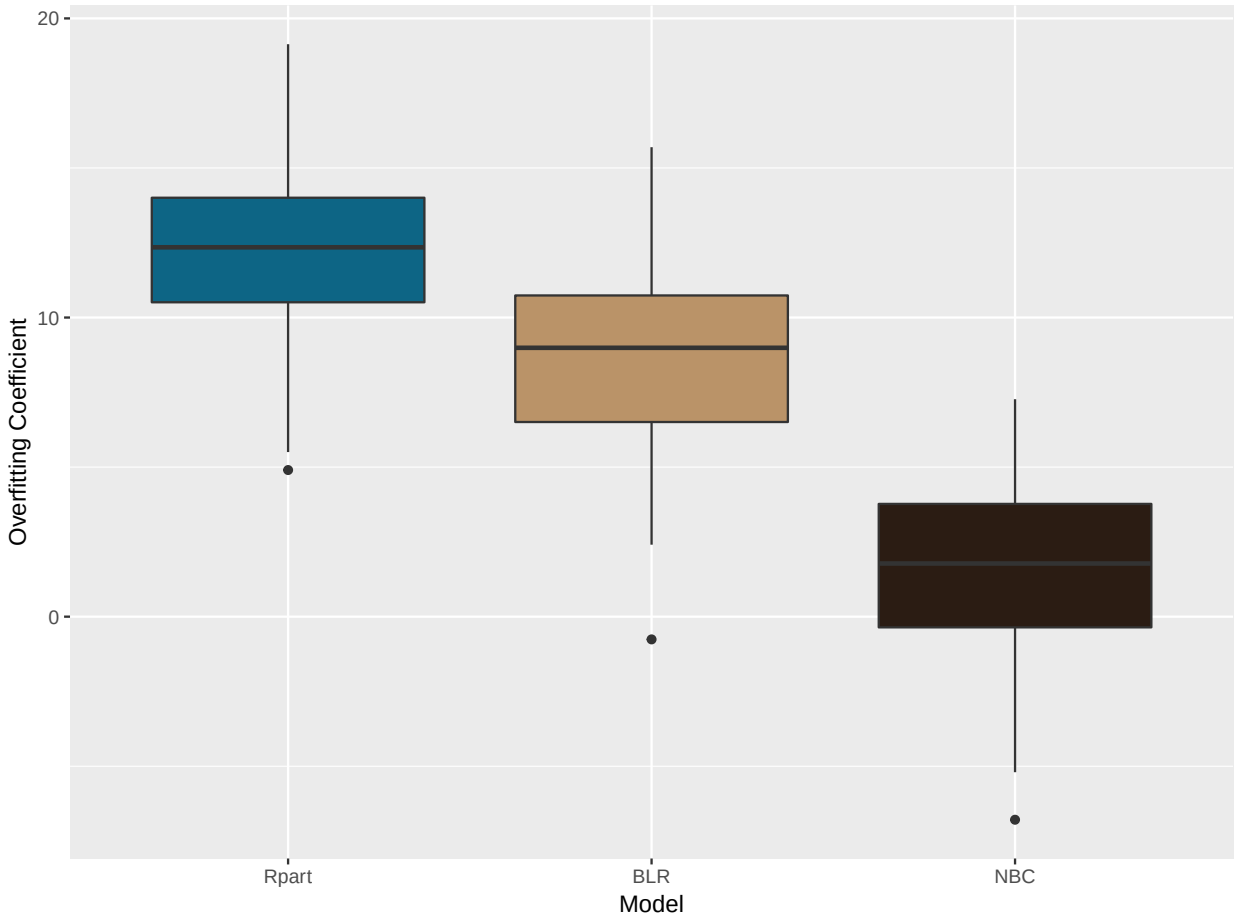
process 100 times and measure the obtained Overfitting Coefficient and Test Accuracy at each iteration. This way we can assess the performance of every model properly.

```
compute_accuracy <- function(model){
    if(model == "rpart"){
        rpart_model <- rpart(as.factor(survived)~., data = train)

rpart_preds_test <- ifelse(predict(rpart_model,test)[,2] > 0.5, 1, 0)
rpart_preds_train <- ifelse(predict(rpart_model,train)[,2] > 0.5, 1, 0)

tr_acc <- round(1 - mean(rpart_preds_train != train$survived), 4)
te_acc <- round(1 - mean(rpart_preds_test  != test$survived), 4)
    }else if(model == "blr"){
        blr_model <- glm(survived ~., family=binomial(link='logit'), data = train)

blr_preds_test <- ifelse(predict(blr_model,test, type = "response") > 0.5, 1, 0)
blr_preds_train <- ifelse(predict(blr_model,train, type = "response") > 0.5, 1, 0)

tr_acc <- round(1 - mean(blr_preds_train != train$survived), 4)
te_acc <- round(1 - mean(blr_preds_test  != test$survived), 4)
    }else{
nbc_model <- naiveBayes(as.factor(survived)~., data = train)

nbc_preds_test <- predict(nbc_model,test)
nbc_preds_train <- predict(nbc_model,train)

tr_acc <- round(1 - mean(nbc_preds_train != train$survived), 4)
te_acc <- round(1 - mean(nbc_preds_test  != test$survived), 4)
    }
oc <- round(100*(tr_acc - te_acc)/tr_acc,6)
return(list(oc = oc,tr_acc = tr_acc,te_acc = te_acc))
}
```

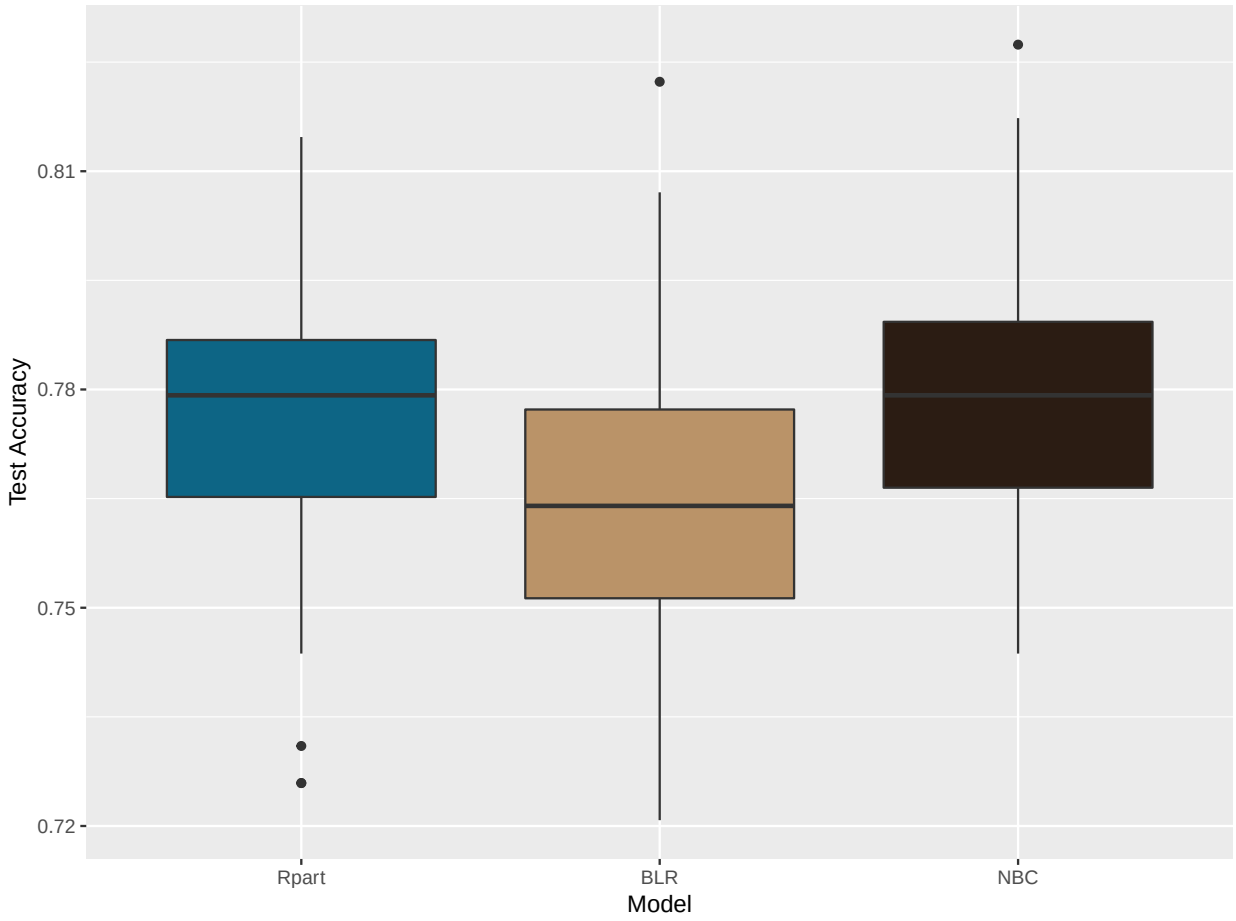
```
n_iters <- 100

rpart_oc <- c()
blr_oc <- c()
nbc_oc <- c()

rpart_te <- c()
blr_te <- c()
nbc_te <- c()
for(i in 1:n_iters){
    tr_id <- sample(1:nrow(titanic), nrow(titanic)*0.7)
    train <- titanic[tr_id,]
    test  <- titanic[-tr_id,]

    rpart_oc[[i]] <- compute_accuracy(model = "rpart")[["oc"]]
    blr_oc[[i]]   <- compute_accuracy(model = "blr")[["oc"]]
    nbc_oc[[i]]   <- compute_accuracy(model = "nbc")[["oc"]]

    rpart_te[[i]] <- compute_accuracy(model = "rpart")[["te_acc"]]
    blr_te[[i]]   <- compute_accuracy(model = "blr")[["te_acc"]]
    nbc_te[[i]]   <- compute_accuracy(model = "nbc")[["te_acc"]]
}
```

And visualize the results in handy boxplots

```
tibble(Rpart = rpart_oc, BLR = blr_oc, NBC = nbc_oc) %>%
    melt() %>%
    ggplot(aes(x = variable, y = value)) +
    geom_boxplot(aes(fill = variable)) +
    scale_fill_hp(discrete = TRUE, house = "Ravenclaw") +
    theme(legend.position="none") +
    xlab("Model") +
    ylab("Overfitting Coefficient")
```




```
tibble(Rpart = rpart_te, BLR = blr_te, NBC = nbc_te) %>%
    melt() %>%
    ggplot(aes(x = variable, y = value)) +
    geom_boxplot(aes(fill = variable)) +
    scale_fill_hp(discrete = TRUE, house = "Ravenclaw") +
    theme(legend.position="none") +
    xlab("Model") +
    ylab("Test Accuracy")
```



Now we can fairly say that the *Naive Bayes Classifier* tends to less overfitting than the others. And consequently its Test Accuracy tends to be higher.

ROC Analysis and AUC

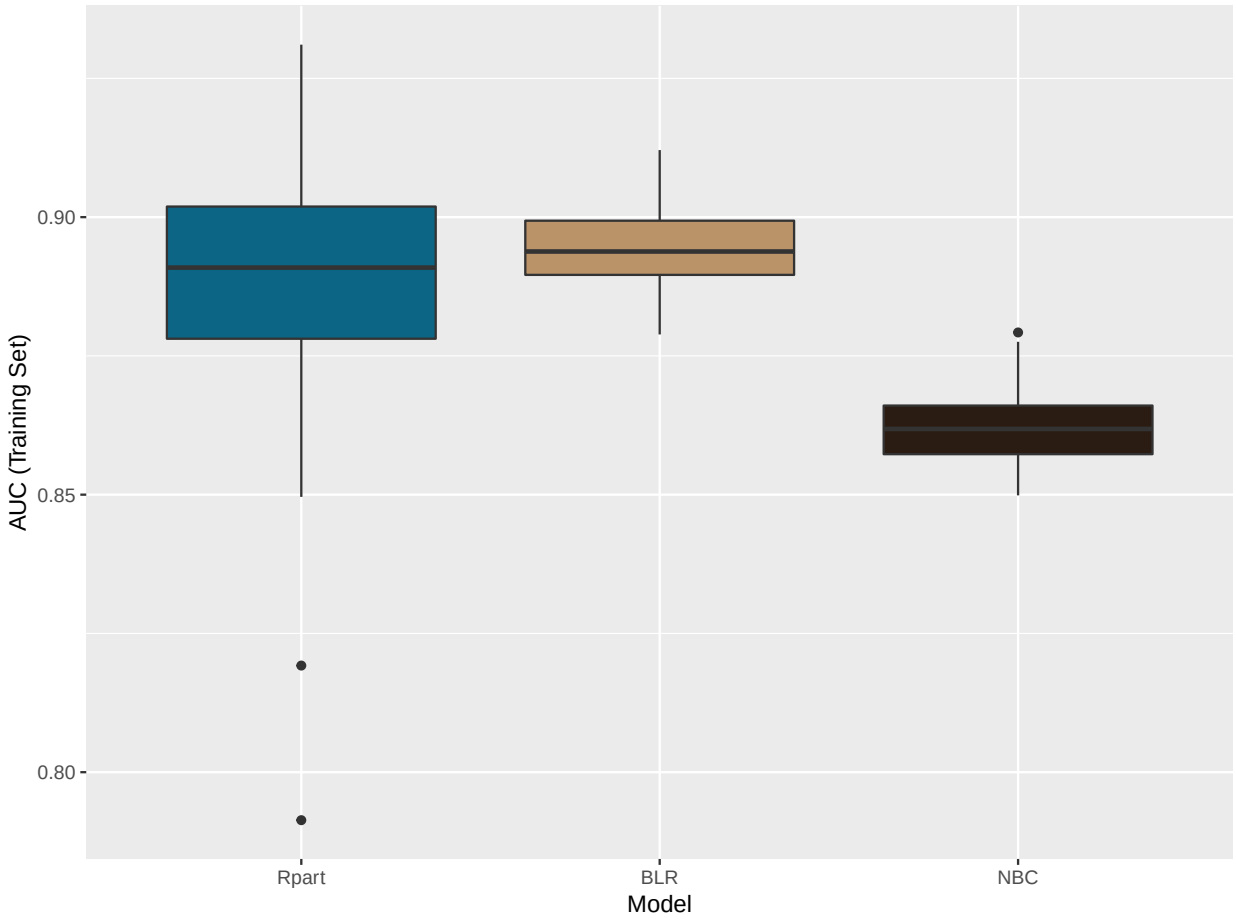
Now we are going to repeat the same analysis, but for the `AUC` The implicit goal of `AUC` is to deal with situations where you have a very skewed sample distribution, and don't want to overfit to a single class.

```
compute_auc <- function(model){
    if(model == "rpart"){
        rpart_model <- rpart(as.factor(survived)~., data = train)

        rpart_preds_test <-  predict(rpart_model,test)[,2]
        rpart_preds_train <- predict(rpart_model,train)[,2]

        te_auc <- auc(test$survived,rpart_preds_test)
        tr_auc <- auc(train$survived,rpart_preds_train)
    }else if(model == "blr"){
        blr_model <- glm(survived ~., family=binomial(link='logit'), data = train)

        blr_preds_test <-  predict(blr_model,test, type = "response")
        blr_preds_train <- predict(blr_model,train, type = "response")
```

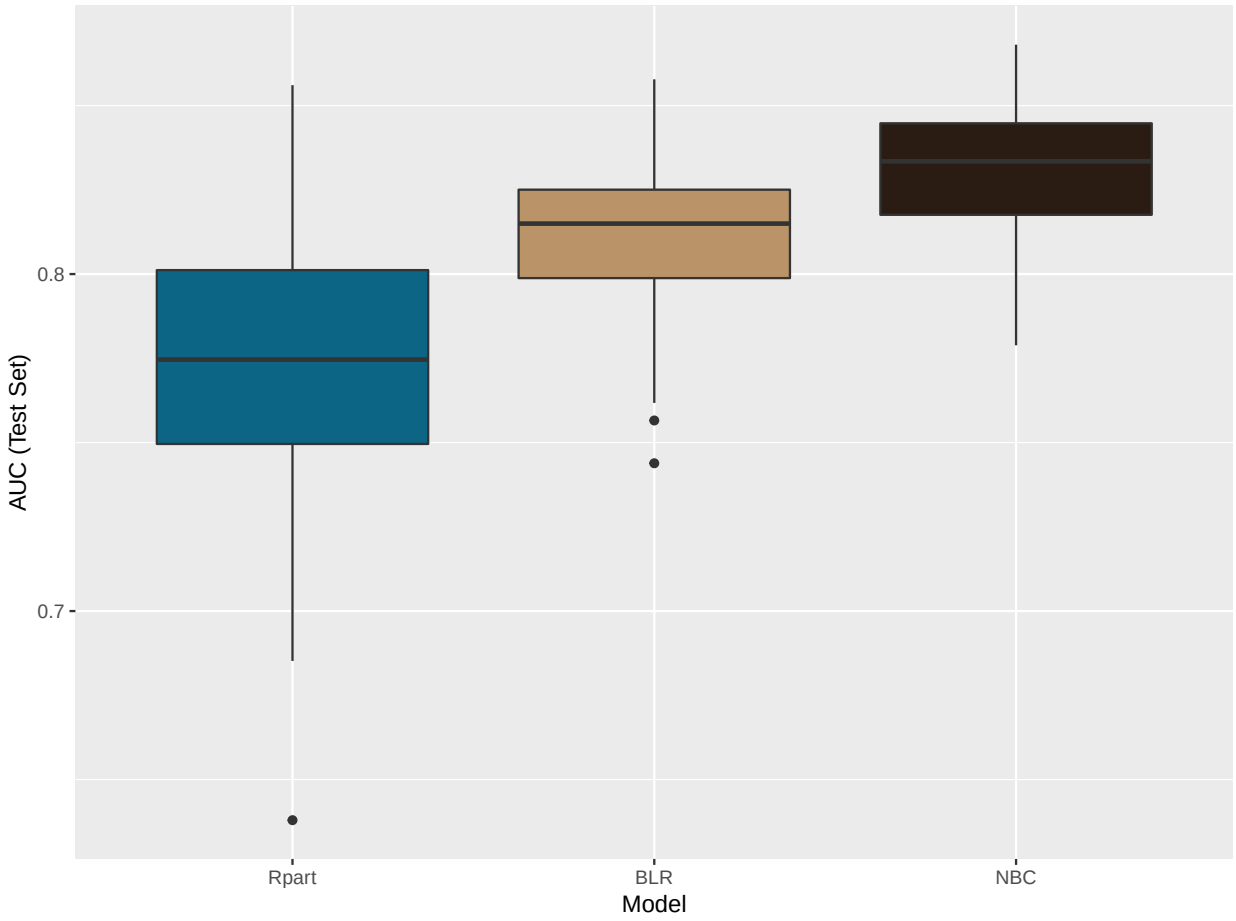
```
        te_auc <- auc(test$survived,blr_preds_test)
        tr_auc <- auc(train$survived,blr_preds_train)
    }else{
        nbc_model <- naiveBayes(as.factor(survived)~., data = train)

        nbc_preds_test <- predict(nbc_model,test, type = "raw")[,2]
        nbc_preds_train <- predict(nbc_model,train, type = "raw")[,2]

        te_auc <- auc(test$survived,nbc_preds_test)
        tr_auc <- auc(train$survived,nbc_preds_train)
    }
return(list(tr_auc = tr_auc,te_auc = te_auc))
}
```

```
rpart_auc_tr <- c()
blr_auc_tr   <- c()
nbc_auc_tr   <- c()

rpart_auc_te <- c()
blr_auc_te   <- c()
nbc_auc_te   <- c()

n_iters   <- 100

for(i in 1:n_iters){
    tr_id <- sample(1:nrow(titanic), nrow(titanic)*0.7)
    train <- titanic[tr_id,]
    test  <- titanic[-tr_id,]

    rpart_auc_tr[[i]] <- compute_auc(model = "rpart")[["tr_auc"]]
    blr_auc_tr[[i]]   <- compute_auc(model = "blr")[["tr_auc"]]
    nbc_auc_tr[[i]]   <- compute_auc(model = "nbc")[["tr_auc"]]

    rpart_auc_te[[i]] <- compute_auc(model = "rpart")[["te_auc"]]
    blr_auc_te[[i]]   <- compute_auc(model = "blr")[["te_auc"]]
    nbc_auc_te[[i]]   <- compute_auc(model = "nbc")[["te_auc"]]
}
```

```
tibble(Rpart = rpart_auc_tr, BLR = blr_auc_tr, NBC = nbc_auc_tr) %>%
    melt() %>%
    ggplot(aes(x = variable, y = value)) +
    geom_boxplot(aes(fill = variable)) +
    scale_fill_hp(discrete = TRUE, house = "Ravenclaw") +
    theme(legend.position="none") +
    xlab("Model") +
    ylab("AUC (Training Set)")
```



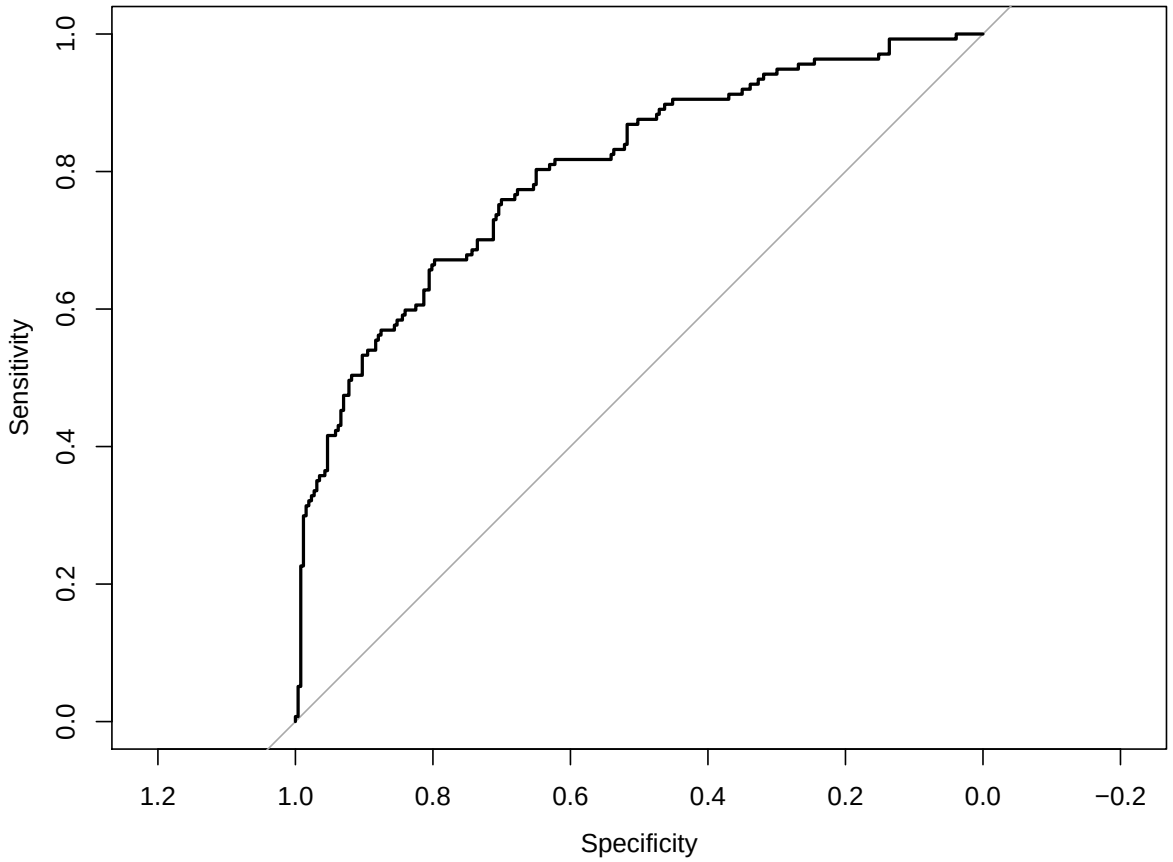

```
tibble(Rpart = rpart_auc_te, BLR = blr_auc_te, NBC = nbc_auc_te) %>%
    melt() %>%
    ggplot(aes(x = variable, y = value)) +
    geom_boxplot(aes(fill = variable)) +
    scale_fill_hp(discrete = TRUE, house = "Ravenclaw") +
    theme(legend.position="none") +
    xlab("Model") +
    ylab("AUC (Test Set)")
```



We would say that our best model is the `Naive Bayes Classificator`, because its `AUC` in the Test Set tends to be higher.

So its `ROC` curve would be:

```
tr_id <- sample(1:nrow(titanic), nrow(titanic)*0.7)
train <- titanic[tr_id,]
test  <- titanic[-tr_id,]

nbc_model <- naiveBayes(as.factor(survived)~., data = train)

nbc_preds_test <- predict(nbc_model,test, type = "raw")[,2]
nbc_preds_train <- predict(nbc_model,train, type = "raw")[,2]

plot.roc(test$survived,nbc_preds_test)
```


Sensitivity
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
Specificity