HarvardX-PH125.9x-Capstone-Project-Titanic-Report

Jollanda Shara

June, 2021

Contents

---

1 Introduction

RMS Titanic was a British passenger liner operated by the White Star Line that sank in the North Atlantic Ocean on 15 April 1912, after striking an iceberg during her maiden voyage from Southampton to New York City. From about 2,224 passengers and crew aboard, more than 1,500 died. Fewer than a third of those aboard Titanic survived the disaster. The ship carried a wide range of passengers of all ages and both genders, from luxury travelers in first-class to immigrants in the lower classes. However, not all passengers were equally likely to survive the accident.The figures show stark differences in the survival rates of the different classes aboard Titanic.We will use real data about a selection of 891 passengers to predict which passengers survived.

1.1 Objective

As we mentioned above, some groups of people were more likely to survive than others. In this project, I have built some machine learning models to predict the survival using passenger's data such as age, sex, fare etc.

```r
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: tidyverse

## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.4     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.0

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: caret

## Loading required package: lattice

##
## Attaching package: 'caret'

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

```r
if(!require(titanic)) install.packages("titanic", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: titanic
```

```r
if(!require(rpart)) install.packages("rpart", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: rpart
```

```r
if(!require(rpart.plot)) install.packages("rpart.plot", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: rpart.plot

## Warning: package 'rpart.plot' was built under R version 4.0.5
```

```r
if(!require(randomForest)) install.packages("randomForest", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: randomForest

## randomForest 4.6-14

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

##
## Attaching package: 'randomForest'

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

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

```r
if(!require(rattle)) install.packages("rattle", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: rattle
## Warning: package 'rattle' was built under R version 4.0.5

## Loading required package: bitops

## Rattle: A free graphical interface for data science with R.
## Version 5.4.0 Copyright (c) 2006-2020 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.

##
## Attaching package: 'rattle'

## The following object is masked from 'package:randomForest':
##
##     importance
```

```
if(!require(kernlab)) install.packages("kernlab", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: kernlab

##
## Attaching package: 'kernlab'

## The following object is masked from 'package:purrr':
##
##     cross

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

```
if(!require(gbm)) install.packages("gbm", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: gbm

## Warning: package 'gbm' was built under R version 4.0.5

## Loaded gbm 2.1.8
```

```
if(!require(nnet)) install.packages("nnet", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: nnet

## Warning: package 'nnet' was built under R version 4.0.5
```

```
library(titanic)    # loads titanic_train data frame
library(tidyverse)
library(rpart)
library(rpart.plot)
library(rattle)
library(caret)
library(randomForest)
library(kernlab)
library(gbm)
library(nnet)
```

```
options(digits = 3)

# clean the data - `titanic_train` is loaded with the titanic package
titanic_clean <- titanic_train %>%
  mutate(Survived = factor(Survived),
         Embarked = factor(Embarked),
         Age = ifelse(is.na(Age), median(Age, na.rm = TRUE), Age), # NA age to median age
```

```r
         FamilySize = SibSp + Parch + 1) %>%     # count family members
  select(Survived,  Sex, Pclass, Age, Fare, SibSp, Parch, FamilySize, Embarked)
titanic <- titanic_train %>%
  select(Survived, Pclass, Sex, Age, SibSp, Parch, Fare) %>%
  mutate(Survived = factor(Survived),
         Pclass = factor(Pclass),
         Sex = factor(Sex))
```

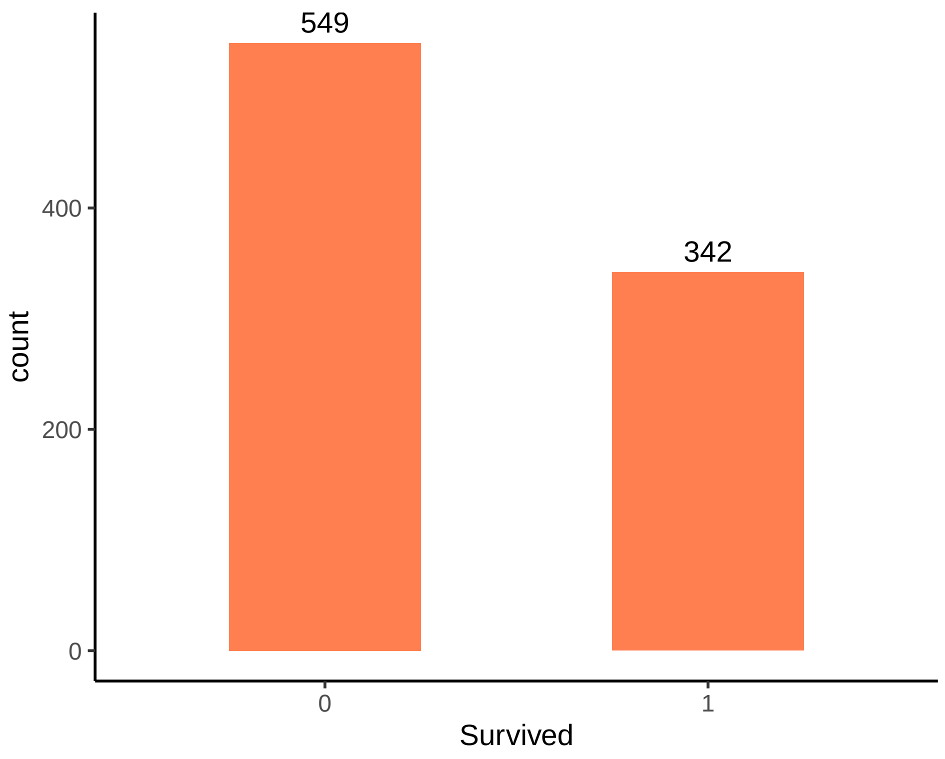
2 Data Analysis

2.1 Data Set.

We will use the titanic_train data frame from the titanic library as the starting point for this project

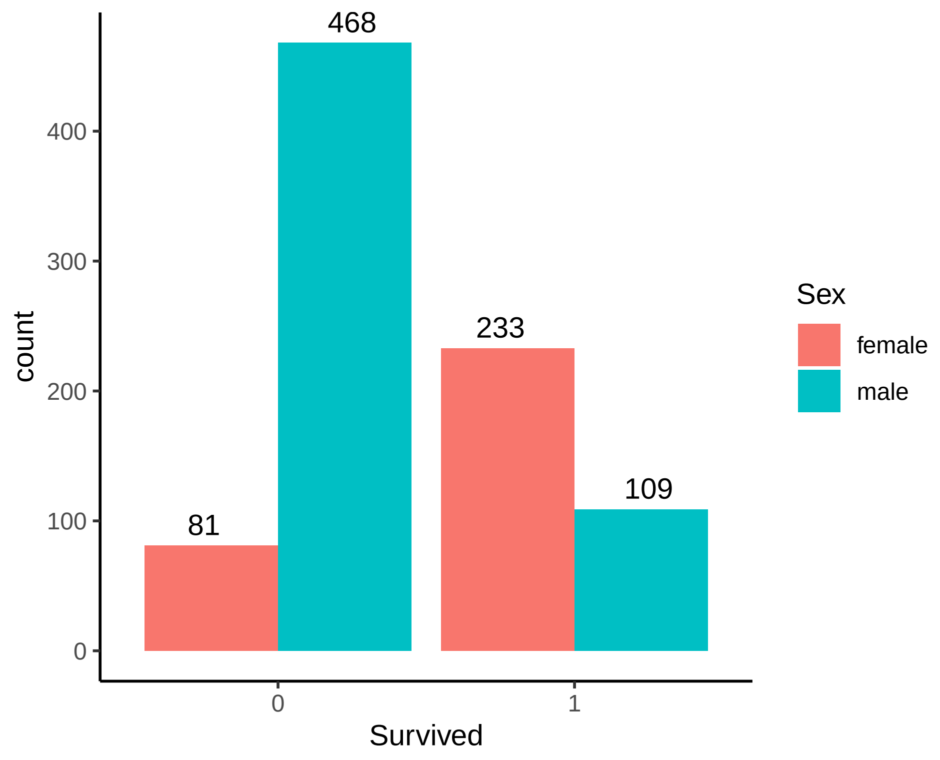
First, let us show how many passengers survived:

```r
titanic %>%
  ggplot(aes(x = Survived)) +
  geom_bar(width=0.5, fill = "coral") +
  geom_text(stat="count", aes(label=stat(count)), vjust=-0.5) +
  theme_classic()
```



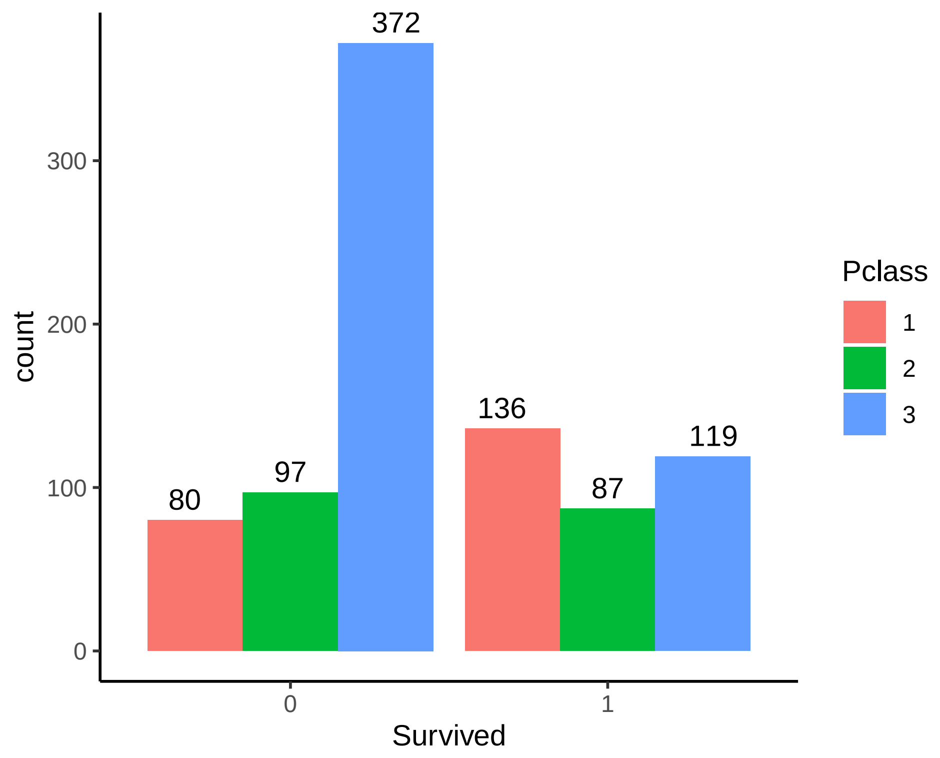
Survived count by Sex

```r
titanic %>%
  ggplot(aes(x = Survived, fill=Sex)) +
  geom_bar(position = position_dodge())+
  geom_text(stat="count",
            aes(label=stat(count)),
            position = position_dodge(width=1), vjust=-0.5)+
  theme_classic()
```



Survived count by Pclass

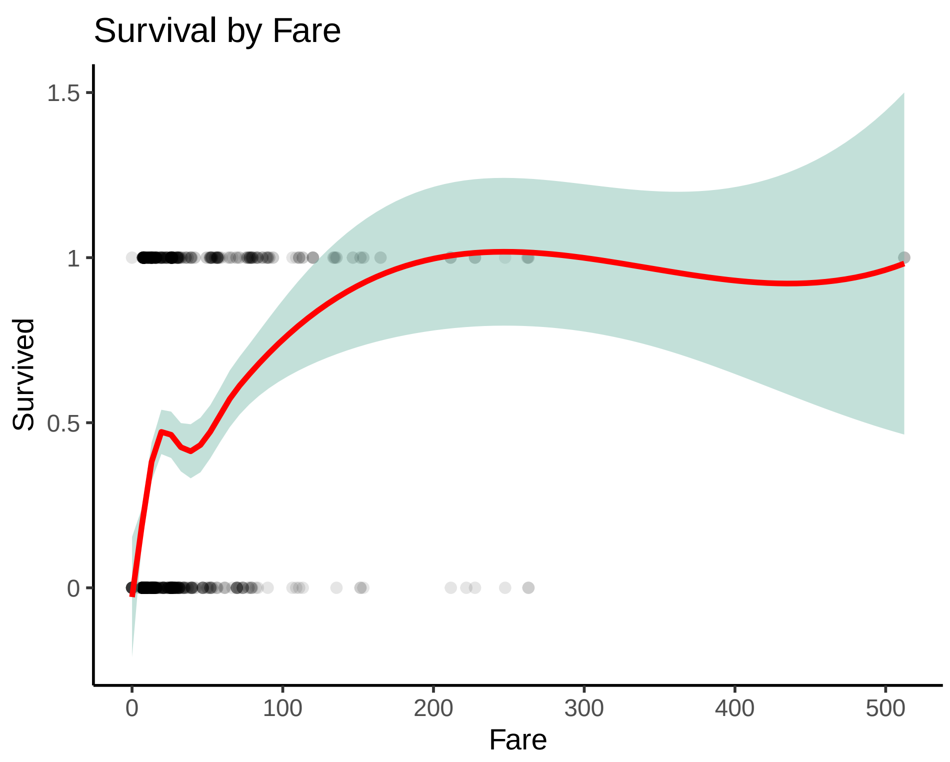
```
titanic %>%
  ggplot(aes(x = Survived, fill=Pclass)) +
  geom_bar(position = position_dodge()) +
  geom_text(stat="count",
            aes(label=stat(count)),
            position = position_dodge(width=1),
            vjust=-0.5) +
  theme_classic()
```



Survival Rate by Fare

```
titanic%>%ggplot(aes(x = Fare, y = as.numeric(Survived)))+
  geom_point(alpha = 0.1)+
  geom_smooth(color="red", fill="#69b3a2", se=TRUE)+
  scale_y_continuous(breaks = seq(1, 2.5, 0.5), labels = c (0, 0.5, 1, 1.5))+
  labs(title = "Survival by Fare",
       y = "Survived")+
  theme_classic()
```

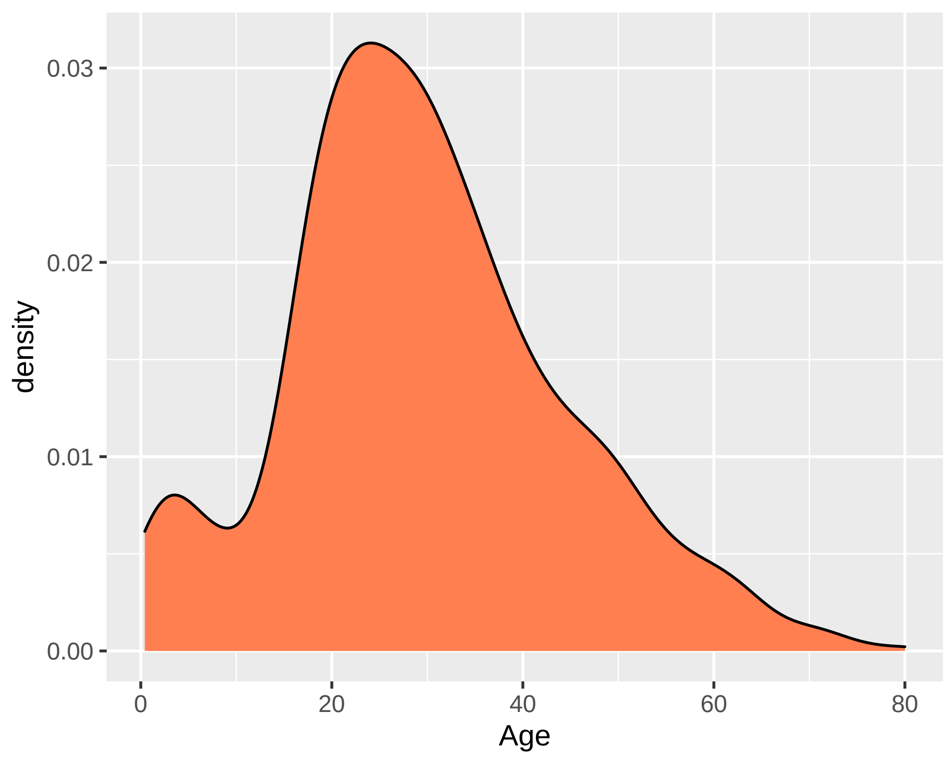
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Age Density

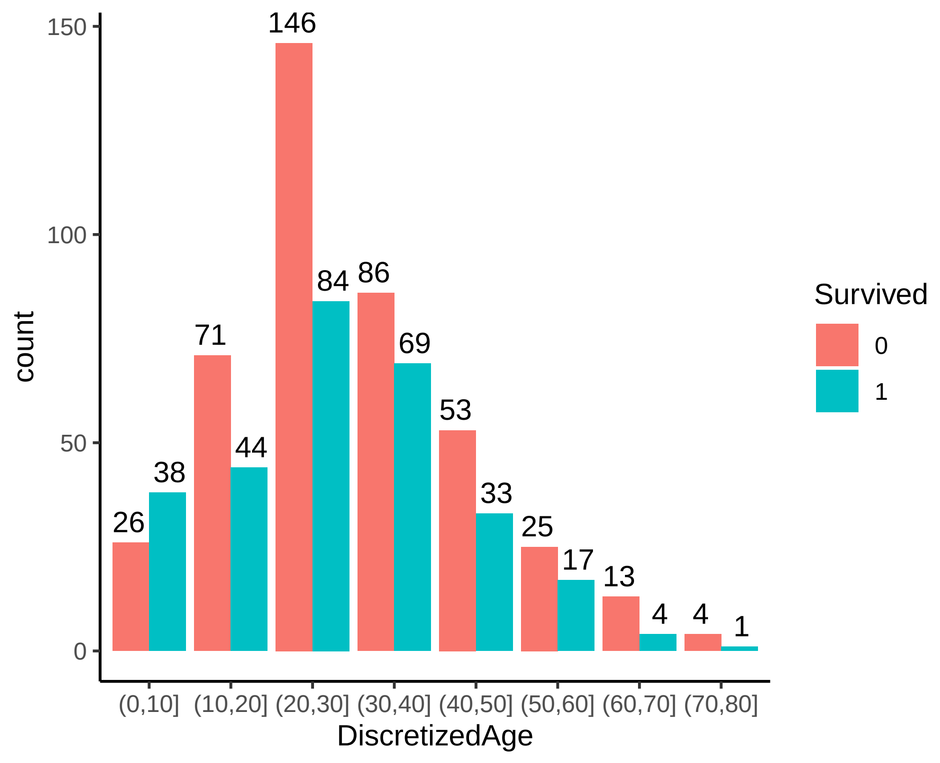
```r
ggplot(titanic, aes(x = Age)) +
  geom_density(fill="coral")
```

```
## Warning: Removed 177 rows containing non-finite values (stat_density).
```



Discretized Age Distributions

```
# Discretize age to plot survival

titanic$DiscretizedAge = cut(titanic$Age, c(0,10,20,30,40,50,60,70,80,100))

# Plot discretized age

titanic %>%
  filter(!is.na(titanic$DiscretizedAge))%>%
  ggplot(aes(x = DiscretizedAge, fill=Survived)) +
  geom_bar(position = position_dodge()) +
  geom_text(stat="count", aes(label=stat(count)), position = position_dodge(width=1), vjust=-0.5)+
  theme_classic()
```

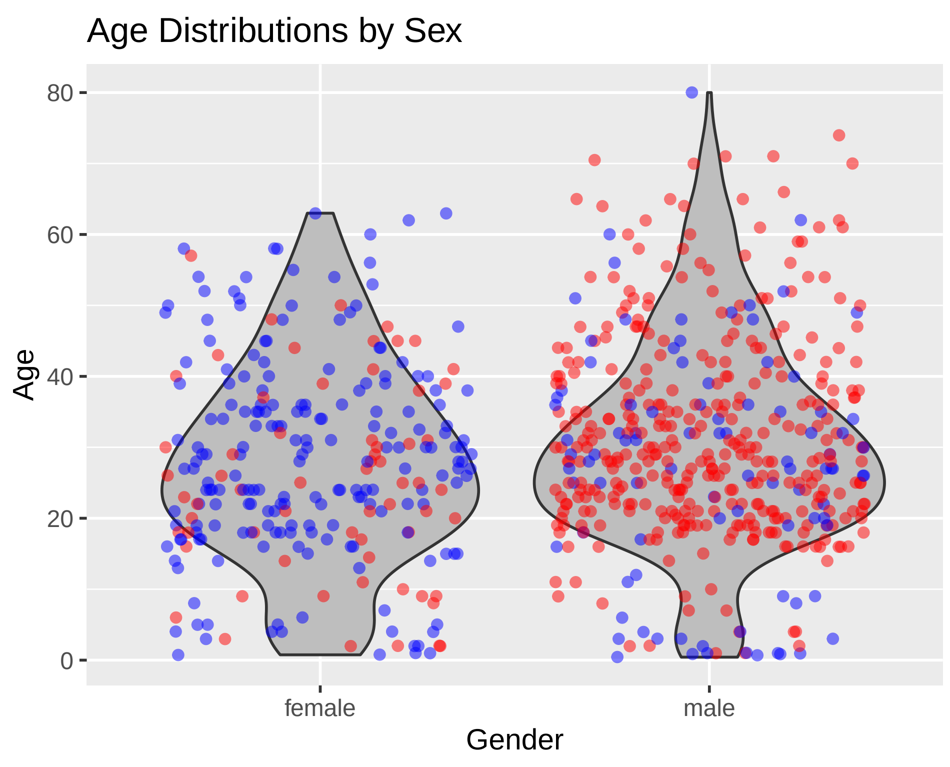


Age Distributions by Sex

```
titanic %>%  ggplot(aes(Age,Sex)) +
  geom_violin(fill="grey") +
  coord_flip() +
  geom_jitter(col=ifelse(titanic$Survived=="1","blue","red"),alpha=.5) +
  ggtitle("Age Distributions by Sex") +
  ylab("Gender") +
  xlab("Age")
```

```
## Warning: Removed 177 rows containing non-finite values (stat_ydensity).
```

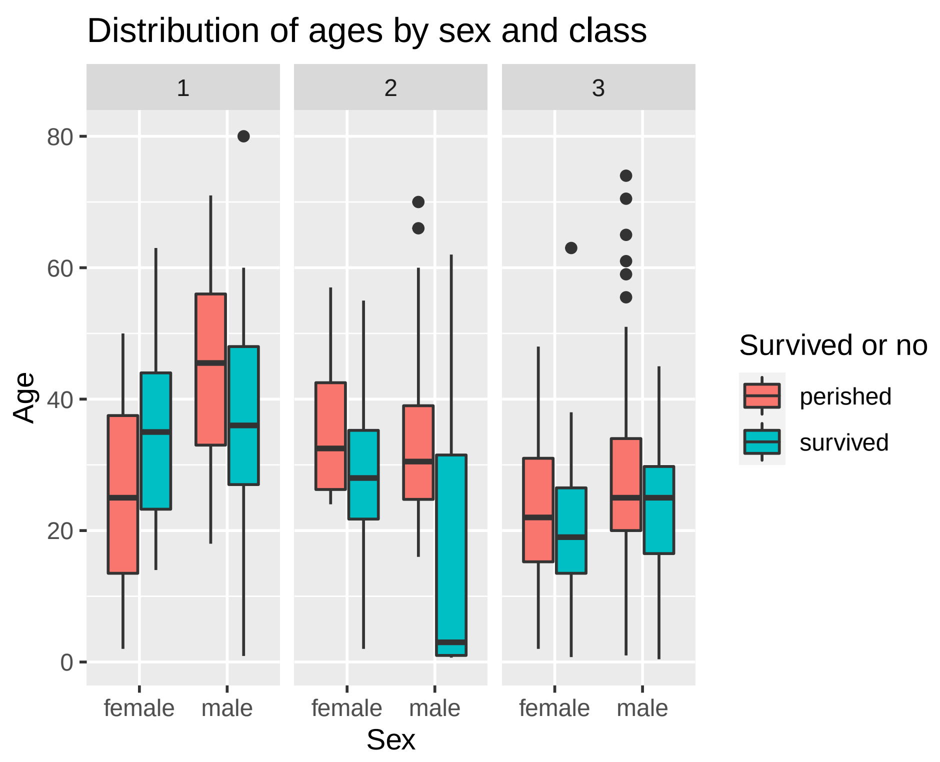
```
## Warning: Removed 177 rows containing missing values (geom_point).
```



Distributions of Ages by Sex and Pclass

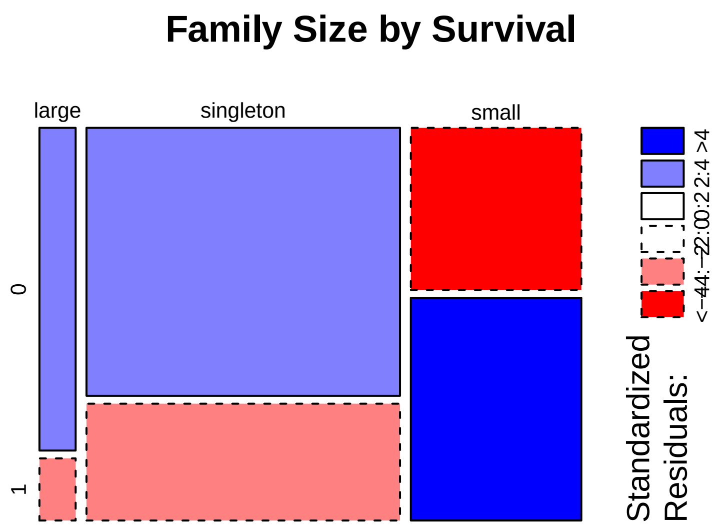
```
#Survival by Age, Sex and Pclass

ggplot(titanic, aes(y=Age,x=Sex, fill=Survived))+
  geom_boxplot()+
  ggtitle("Distribution of ages by sex and class") +
  xlab("Sex")+
  ylab("Age")+
  scale_fill_discrete("Survived or no",labels=c("perished","survived"))+
  facet_grid( ~ Pclass)
```

```
## Warning: Removed 177 rows containing non-finite values (stat_boxplot).
```



Family Size by Survival

```
titanic$Fsize <- titanic$SibSp + titanic$Parch + 1

# Create a family variable
# Discretize family size

titanic$FsizeD[titanic$Fsize == 1] <- "singleton"
titanic$FsizeD[titanic$Fsize < 5 & titanic$Fsize > 1] <- "small"
titanic$FsizeD[titanic$Fsize > 4] <- "large"

# Show family size by survival using a mosaic plot

mosaicplot(table(titanic$FsizeD, titanic$Survived), main="Family Size by Survival", shade=TRUE)
```



3 Machine Learning Models

First, we will split titanic_clean into test and training sets - after running the setup code, it should have 891 rows and 9 variables. Next, set the seed to 42, then use the caret package to create a 20% data partition based on the Survived column. Assign the 20% partition to test_set and the remaining 80% partition to train_set.

```
#Train and Test Sets

set.seed(42, sample.kind = "Rounding")     # simulate R 3.5
```

```
## Warning in set.seed(42, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
test_index <- createDataPartition(titanic_clean$Survived, times = 1, p = 0.2, list = FALSE)     # create
test_set <- titanic_clean[test_index,]
train_set <- titanic_clean[-test_index,]
```

Let us calculate the number of observations in the train and test sets.

```
nrow(train_set)
```

```
## [1] 712
```

```
nrow(test_set)
```

```
## [1] 179
```

Then the survival proportion

```
mean(train_set$Survived == 1)
```

```
## [1] 0.383
```

3.1 LDA

Survival by Fare+Age+Sex - LDA

```r
#set.seed(1) # R 3.5
set.seed(1, sample.kind = "Rounding") # R 3.6
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```r
train_lda <- train(Survived ~ Fare+Age+Sex,
                   method = "lda",
                   data = train_set)
lda_preds <- predict(train_lda, test_set)
confusionMatrix(lda_preds, test_set$Survived)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 96 18
##          1 14 51
##
##                Accuracy : 0.821
##                  95% CI : (0.757, 0.874)
##     No Information Rate : 0.615
##     P-Value [Acc > NIR] : 1.72e-09
##
##                   Kappa : 0.619
##
##  Mcnemar's Test P-Value : 0.596
##
##             Sensitivity : 0.873
##             Specificity : 0.739
##          Pos Pred Value : 0.842
##          Neg Pred Value : 0.785
##              Prevalence : 0.615
##          Detection Rate : 0.536
##    Detection Prevalence : 0.637
##       Balanced Accuracy : 0.806
##
##        'Positive' Class : 0
##
```

3.2 QDA

Survival by Fare+Age+Sex - QDA

```r
#set.seed(1) # R 3.5
set.seed(1, sample.kind = "Rounding") # R 3.6
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```r
train_qda <- train(Survived ~ Fare+Age+Sex,
                   method = "qda",
                   data = train_set)
```

```
qda_preds <- predict(train_qda, test_set)
confusionMatrix(qda_preds, test_set$Survived)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 93 17
##          1 17 52
##
##                Accuracy : 0.81
##                  95% CI : (0.745, 0.865)
##     No Information Rate : 0.615
##     P-Value [Acc > NIR] : 1.35e-08
##
##                   Kappa : 0.599
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.845
##             Specificity : 0.754
##          Pos Pred Value : 0.845
##          Neg Pred Value : 0.754
##              Prevalence : 0.615
##          Detection Rate : 0.520
##    Detection Prevalence : 0.615
##       Balanced Accuracy : 0.800
##
##        'Positive' Class : 0
##
```

3.3 kNN model

```
#set.seed(6)
set.seed(6, sample.kind = "Rounding")    # simulate R 3.5
```

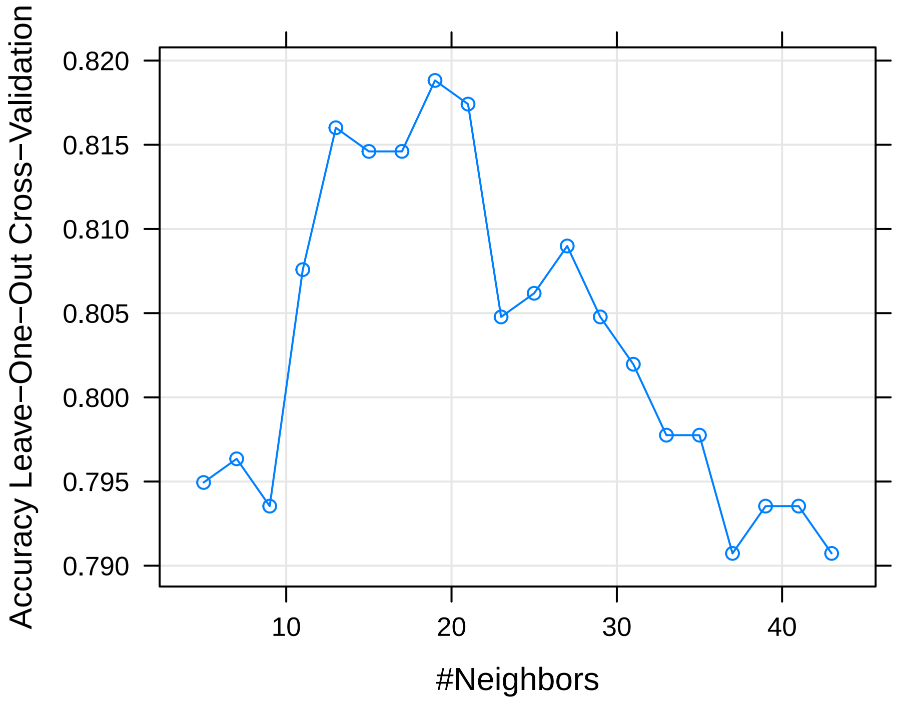
```
## Warning in set.seed(6, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
control = trainControl( method="LOOCV");
train_knn = train( Survived~Pclass+Sex+Age+Fare+Embarked+SibSp+Parch+FamilySize,
                   data=train_set,
                   method="knn",
                   trControl=control,
                   preProcess=c("center","scale"),
                   tuneLength=20 )
train_knn
```

```
## k-Nearest Neighbors
##
## 712 samples
##   8 predictor
##   2 classes: '0', '1'
##
```

```
## Pre-processing: centered (10), scaled (10)
## Resampling: Leave-One-Out Cross-Validation
## Summary of sample sizes: 711, 711, 711, 711, 711, 711, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy  Kappa
##    5  0.795     0.557
##    7  0.796     0.562
##    9  0.794     0.552
##   11  0.808     0.580
##   13  0.816     0.598
##   15  0.815     0.594
##   17  0.815     0.593
##   19  0.819     0.602
##   21  0.817     0.598
##   23  0.805     0.568
##   25  0.806     0.571
##   27  0.809     0.577
##   29  0.805     0.568
##   31  0.802     0.562
##   33  0.798     0.554
##   35  0.798     0.554
##   37  0.791     0.541
##   39  0.794     0.549
##   41  0.794     0.549
##   43  0.791     0.543
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 19.
```

```r
#plot the kNN model

plot( train_knn)
```



```r
#The accuracy of the kNN model on the test set is:

knn_preds <- predict(train_knn, test_set)
confusionMatrix(knn_preds, test_set$Survived)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 100  21
##          1  10  48
##
##                Accuracy : 0.827
##                  95% CI : (0.763, 0.879)
##     No Information Rate : 0.615
##     P-Value [Acc > NIR] : 5.83e-10
##
##                   Kappa : 0.623
##
##  Mcnemar's Test P-Value : 0.0725
##
##             Sensitivity : 0.909
##             Specificity : 0.696
##          Pos Pred Value : 0.826
##          Neg Pred Value : 0.828
##              Prevalence : 0.615
##          Detection Rate : 0.559
##    Detection Prevalence : 0.676
##       Balanced Accuracy : 0.802
```

```
##
##        'Positive' Class : 0
##
```

3.4 Cross-validation

We use 10-fold cross-validation where each partition consists of 10% of the total and tuning with k = seq(3, 51, 2). The optimal value of k using cross-validation is:

```
#Cross-validation
#We use 10-fold cross-validation where each partition
#consists of 10% of the total and tuning with k = seq(3, 51, 2).
#The optimal value of k using cross-validation is:

#set.seed(8)
set.seed(8, sample.kind = "Rounding")    # simulate R 3.5
```

```
## Warning in set.seed(8, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
train_knn_cv <- train(Survived ~ .,
                      method = "knn",
                      data = train_set,
                      tuneGrid = data.frame(k = seq(3, 51, 2)),
                      trControl = trainControl(method = "cv", number = 10, p = 0.9))
train_knn_cv$bestTune
```

```
##   k
## 2 5
```

```
#The accuracy on the test set using the cross-validated kNN model is:

knn_cv_preds <- predict(train_knn_cv, test_set)
confusionMatrix(knn_cv_preds, test_set$Survived)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 84 37
##          1 26 32
##
##                Accuracy : 0.648
##                  95% CI : (0.573, 0.718)
##     No Information Rate : 0.615
##     P-Value [Acc > NIR] : 0.200
##
##                   Kappa : 0.234
##
##  Mcnemar's Test P-Value : 0.208
##
##             Sensitivity : 0.764
##             Specificity : 0.464
##          Pos Pred Value : 0.694
##          Neg Pred Value : 0.552
##              Prevalence : 0.615
##          Detection Rate : 0.469
##    Detection Prevalence : 0.676
##       Balanced Accuracy : 0.614
##
##        'Positive' Class : 0
##
```

3.5 The Classification tree model

We use caret to train a decision tree with the rpart method tuning the complexity parameter with cp = seq(0, 0.05, 0.002). The optimal value of the complexity parameter (cp) is:

```
#set.seed(10)
set.seed(10, sample.kind = "Rounding")    # simulate R 3.5
```

```
## Warning in set.seed(10, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
train_rpart <- train(Survived ~ .,
                     method = "rpart",
                     tuneGrid = data.frame(cp = seq(0, 0.05, 0.002)),
                     data = train_set)
train_rpart$bestTune
```

```
##      cp
## 9 0.016
```

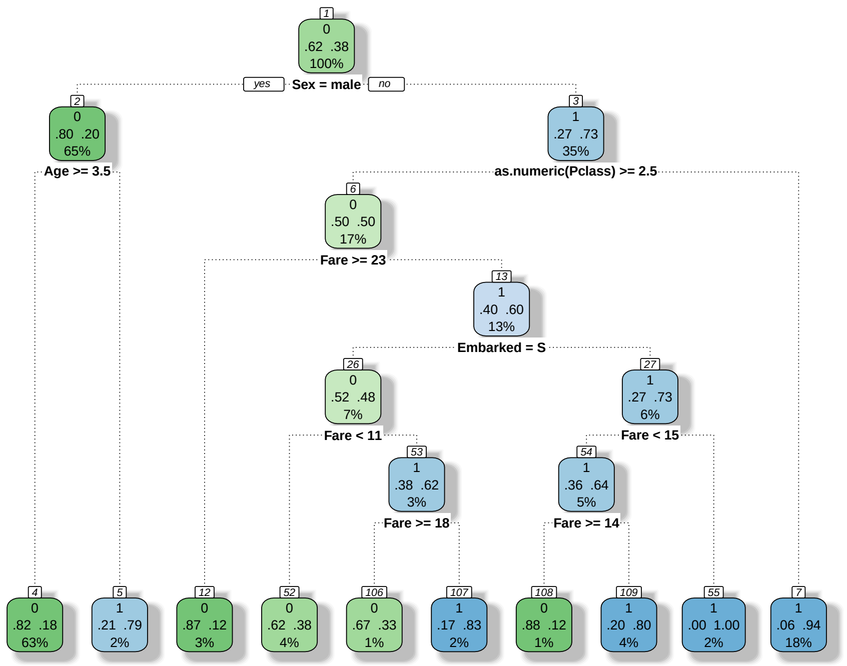
```
#The accuracy of the decision tree model on the test set is:

rpart_preds <- predict(train_rpart, test_set)
confusionMatrix(rpart_preds, test_set$Survived)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 97 16
##          1 13 53
##
##                Accuracy : 0.838
##                  95% CI : (0.776, 0.889)
##     No Information Rate : 0.615
##     P-Value [Acc > NIR] : 5.95e-11
##
##                   Kappa : 0.655
##
##  Mcnemar's Test P-Value : 0.71
##
##             Sensitivity : 0.882
##             Specificity : 0.768
##          Pos Pred Value : 0.858
##          Neg Pred Value : 0.803
##              Prevalence : 0.615
##          Detection Rate : 0.542
##    Detection Prevalence : 0.631
##       Balanced Accuracy : 0.825
```

```
##
##        'Positive' Class : 0
##
```

Using the rpart library we make a decision tree that takes into account the Class, Sex, Age, Siblings & Spouses, the Parch, Fare and Embarked and plot it

```
decision_tree <- rpart(Survived ~ as.numeric(Pclass) + Sex + Age + SibSp + Parch + Fare + Embarked,data
fancyRpartPlot(decision_tree)
```



Rattle 2021−Jun−07 17:02:33 i−techbology™

```
options(repr.plot.width=25, repr.plot.height=25)
```

3.6 Random Forest model

```
#set.seed(1234)
set.seed(1234, sample.kind = "Rounding")
```

```
## Warning in set.seed(1234, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
# Set control parameters

control <- trainControl(method="repeatedcv",
                        number=10,
                        repeats=5,
                        search = "random")

# Determine baseline mtry

mtry <- sqrt(ncol(train_set))
tune_grid = expand.grid(.mtry=mtry)
```

```r
#Train RF model
#Random generate 15 mtry values with tuneLength = 15

train_rf <- train(Survived ~ .,
                  data=train_set,
                  method="rf",
                  tuneLength=15,
                  trControl=control,
                  importance=TRUE,
                  localImp=TRUE,
                  tuneGrid=tune_grid)

# Explain final RF model

fit_rf <- train_rf$finalModel
fit_rf
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry, importance = TRUE,      localImp = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 18.1%
## Confusion matrix:
##     0   1 class.error
## 0 397  42      0.0957
## 1  87 186      0.3187
```

```r
# The Accuracy using the ConfusionMatrix

rf_preds <- predict(train_rf, test_set) %>%confusionMatrix(test_set$Survived)
rf_preds
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 103  20
##          1   7  49
##
##                Accuracy : 0.849
##                  95% CI : (0.788, 0.898)
##     No Information Rate : 0.615
##     P-Value [Acc > NIR] : 5.17e-12
##
##                   Kappa : 0.67
##
##  Mcnemar's Test P-Value : 0.0209
##
##             Sensitivity : 0.936
##             Specificity : 0.710
##          Pos Pred Value : 0.837
##          Neg Pred Value : 0.875
##              Prevalence : 0.615
##          Detection Rate : 0.575
##    Detection Prevalence : 0.687
##       Balanced Accuracy : 0.823
##
##        'Positive' Class : 0
##
```

Tuning is the process of maximizing a model's performance without overfitting or creating too high of a variance. In machine learning, this is accomplished by selecting appropriate hyperparameters. Mtry is the number of variables available for splitting at each tree node.

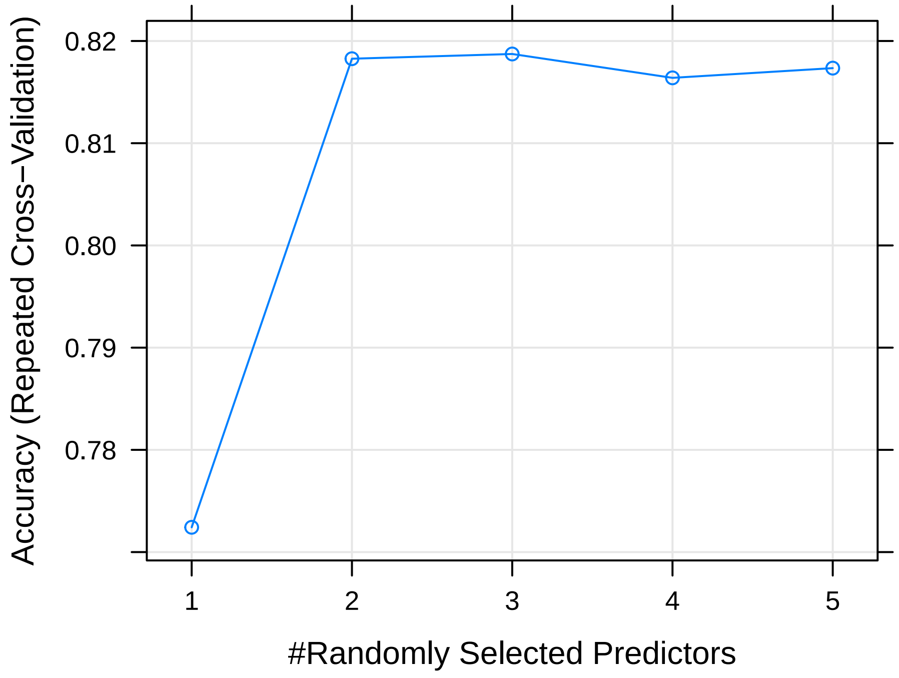
```
#set.seed(1234)
set.seed(1234, sample.kind = "Rounding")
```

```
## Warning in set.seed(1234, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
control <- trainControl(method="repeatedcv", number=10, repeats=3, search="grid")
tune_grid <- expand.grid(.mtry=c(1:5))
rf_gridsearch <- train(Survived~.,
                       data=train_set,
                       method="rf",
                       tuneGrid=tune_grid,
                       trControl=control)
print(rf_gridsearch)
```

```
## Random Forest
##
## 712 samples
##   8 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 641, 641, 641, 641, 641, 640, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy  Kappa
##   1     0.772     0.477
##   2     0.818     0.598
##   3     0.819     0.606
##   4     0.816     0.605
##   5     0.817     0.607
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 3.
```

```
plot(rf_gridsearch)
```



It seems that best value for mtry is 3. Finally, lets use this mtry to check best ntree (number of trees to grow) parameter:

```
# It seems that best value for mtry is 3.
#Finally, lets use this mtry to check best ntree (number of trees to grow) parameter:

# set.seed(1234)
set.seed(1234, sample.kind = "Rounding")    # simulate R 3.5
```

```
## Warning in set.seed(1234, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
control <- trainControl(method="repeatedcv", number=10, repeats=3, search="grid")
tune_grid <- expand.grid(.mtry=c(3))
modellist <- list()
for (ntree in c(100, 250, 500)) {
  rf_manual <- train(Survived~.,
                     data=train_set,
                     method="rf",
                     tuneGrid=tune_grid,
                     trControl=control,
                     ntree=ntree)
  key <- toString(ntree)
  modellist[[key]] <- rf_manual
}
results <- resamples(modellist)
summary(results)
```

```
##
## Call:
## summary.resamples(object = results)
```

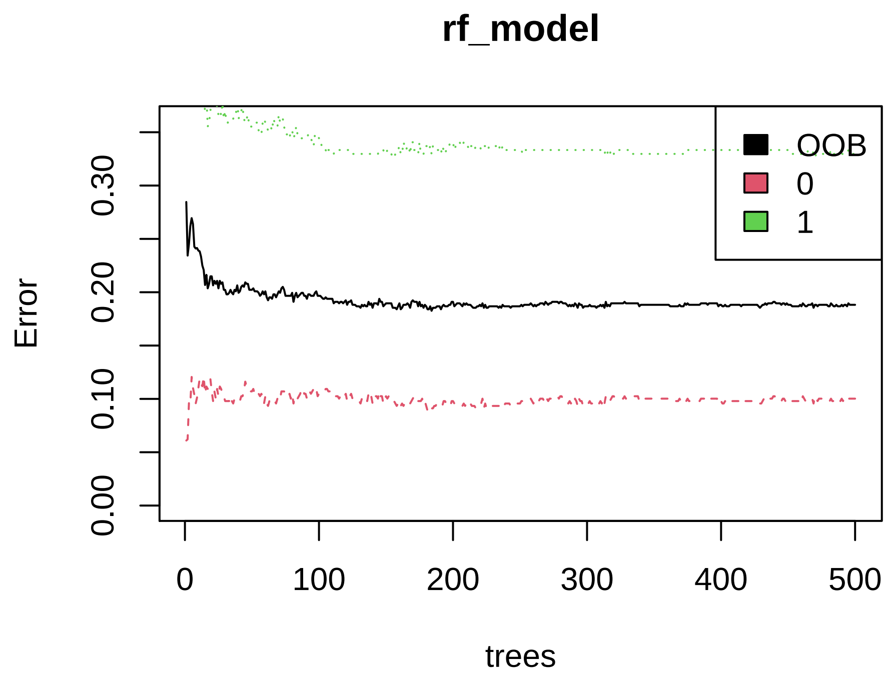
```
##
## Models: 100, 250, 500
## Number of resamples: 30
##
## Accuracy
##      Min. 1st Qu. Median  Mean 3rd Qu.  Max. NA's
## 100 0.732   0.794  0.817 0.819   0.845 0.889    0
## 250 0.750   0.789  0.811 0.818   0.845 0.943    0
## 500 0.736   0.792  0.825 0.824   0.845 0.930    0
##
## Kappa
##      Min. 1st Qu. Median  Mean 3rd Qu.  Max. NA's
## 100 0.393   0.563  0.603 0.605   0.666 0.760    0
## 250 0.445   0.527  0.591 0.603   0.663 0.878    0
## 500 0.418   0.547  0.611 0.615   0.664 0.850    0
```

Build a model by using randomForest on the train set for almost all variables, plot it and show model error.

```
#set.seed(14)
set.seed(14, sample.kind = "Rounding")     # simulate R 3.5
```

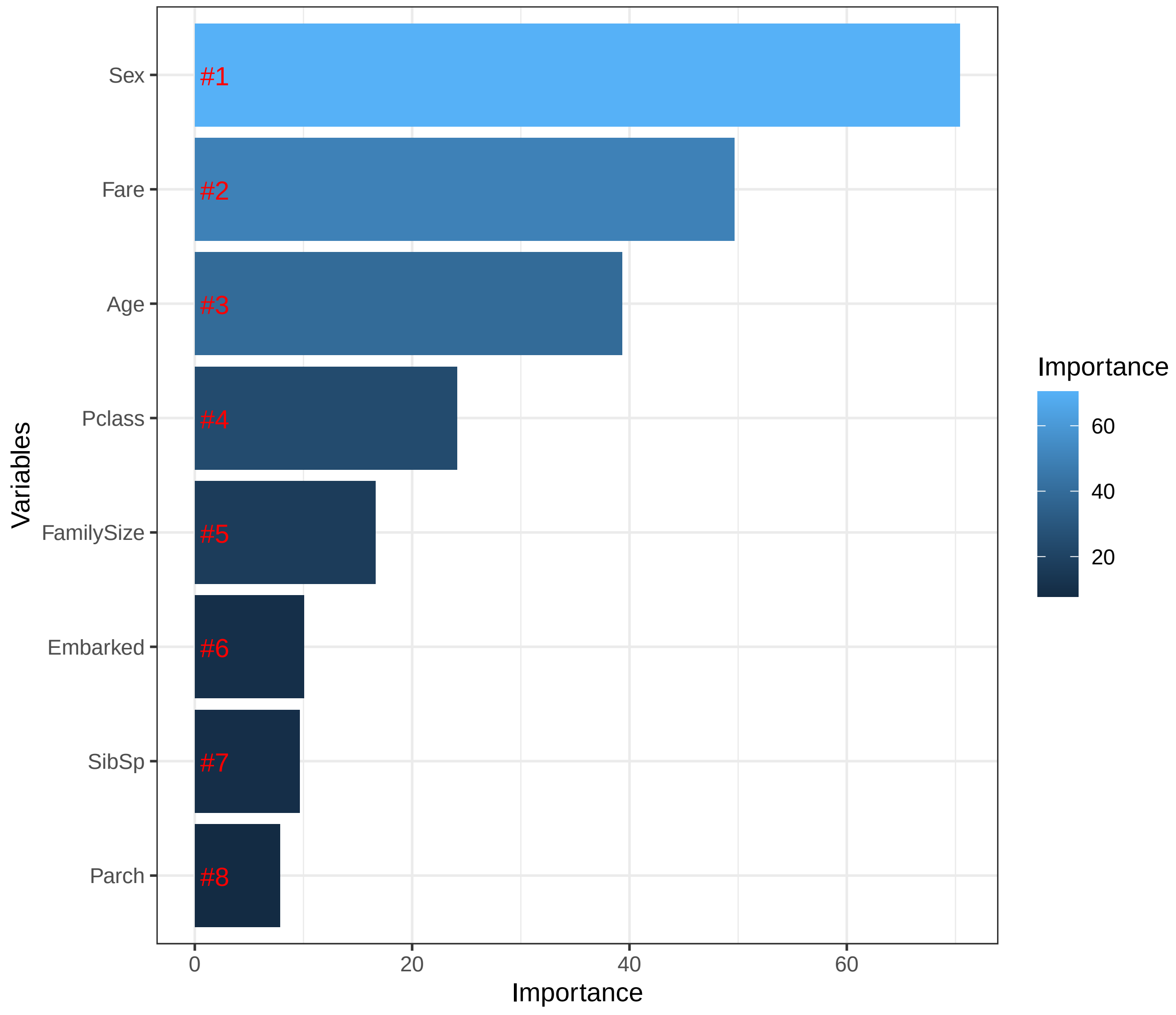
```
## Warning in set.seed(14, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
rf_model <- randomForest(Survived ~ Pclass + Sex + Age + SibSp + Parch + Fare + Embarked + FamilySize,

# Show model error

plot(rf_model, ylim = c(0, 0.36))
legend("topright", colnames(rf_model$err.rate), col = 1:3, fill = 1:3)
```



Let's look at relative variable importance by plotting the mean decrease in Gini calculated across all trees.

```r
# Get importance
importance    <- importance(rf_model)
varImportance <- data.frame(Variables = row.names(importance),
Importance = round(importance[ ,'MeanDecreaseGini'],2))

# Create a rank variable based on importance

rankImportance <- varImportance %>%
          mutate(Rank = paste0('#',dense_rank(desc(Importance))))

        # Use ggplot2 to visualize the relative importance of variables

        ggplot(rankImportance, aes(x = reorder(Variables, Importance),
        y = Importance, fill = Importance)) +
          geom_bar(stat='identity') +
          geom_text(aes(x = Variables, y = 0.5, label = Rank),
        hjust=0, vjust=0.55, size = 4, color = 'red') +
          labs(x = 'Variables') +
          coord_flip() +
          theme_bw()
```



3.7 Gradient Boosting Model

```
#set.seed(616)
set.seed(616, sample.kind = "Rounding")
```

```
## Warning in set.seed(616, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
control <- trainControl(method="repeatedcv",number=5, repeats=5)
gbmGrid <- expand.grid(interaction.depth = seq(1,7,by=2),
                       n.trees = seq(50, 250, by = 25),
                       shrinkage = c(0.01, 0.1),
                       n.minobsinnode = 5)

train_gbm <-train(Survived ~ .,
                  data = train_set,
                  method = "gbm",
                  verbose = FALSE,
                  trControl = control,
                  tuneGrid = gbmGrid)
train_gbm
```

```
## Stochastic Gradient Boosting
##
## 712 samples
##   8 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 5 times)
## Summary of sample sizes: 569, 571, 569, 570, 569, 570, ...
## Resampling results across tuning parameters:
##
##   shrinkage  interaction.depth  n.trees  Accuracy  Kappa
##   0.01       1                   50      0.772     0.495
##   0.01       1                   75      0.776     0.516
##   0.01       1                  100      0.777     0.519
##   0.01       1                  125      0.778     0.522
##   0.01       1                  150      0.779     0.524
##   0.01       1                  175      0.778     0.523
##   0.01       1                  200      0.779     0.524
##   0.01       1                  225      0.778     0.522
##   0.01       1                  250      0.780     0.525
##   0.01       3                   50      0.778     0.477
##   0.01       3                   75      0.782     0.497
##   0.01       3                  100      0.800     0.552
##   0.01       3                  125      0.807     0.573
##   0.01       3                  150      0.808     0.579
##   0.01       3                  175      0.811     0.585
##   0.01       3                  200      0.811     0.585
##   0.01       3                  225      0.815     0.593
##   0.01       3                  250      0.817     0.598
##   0.01       5                   50      0.785     0.500
##   0.01       5                   75      0.804     0.555
##   0.01       5                  100      0.809     0.570
##   0.01       5                  125      0.812     0.581
##   0.01       5                  150      0.814     0.586
##   0.01       5                  175      0.813     0.586
##   0.01       5                  200      0.814     0.588
##   0.01       5                  225      0.814     0.589
##   0.01       5                  250      0.812     0.585
##   0.01       7                   50      0.792     0.519
##   0.01       7                   75      0.811     0.571
##   0.01       7                  100      0.813     0.581
##   0.01       7                  125      0.813     0.582
##   0.01       7                  150      0.814     0.586
##   0.01       7                  175      0.815     0.589
##   0.01       7                  200      0.816     0.593
##   0.01       7                  225      0.818     0.598
##   0.01       7                  250      0.819     0.602
##   0.10       1                   50      0.797     0.560
##   0.10       1                   75      0.796     0.560
##   0.10       1                  100      0.796     0.561
##   0.10       1                  125      0.796     0.559
##   0.10       1                  150      0.796     0.560
##   0.10       1                  175      0.796     0.561
##   0.10       1                  200      0.795     0.561
##   0.10       1                  225      0.797     0.564
##   0.10       1                  250      0.797     0.565
##   0.10       3                   50      0.811     0.587
##   0.10       3                   75      0.815     0.598
##   0.10       3                  100      0.813     0.596
##   0.10       3                  125      0.816     0.603
##   0.10       3                  150      0.811     0.592
##   0.10       3                  175      0.813     0.597
##   0.10       3                  200      0.813     0.596
##   0.10       3                  225      0.814     0.600
##   0.10       3                  250      0.815     0.601
##   0.10       5                   50      0.822     0.613
##   0.10       5                   75      0.820     0.610
##   0.10       5                  100      0.821     0.612
##   0.10       5                  125      0.817     0.604
##   0.10       5                  150      0.812     0.595
##   0.10       5                  175      0.812     0.595
##   0.10       5                  200      0.809     0.589
##   0.10       5                  225      0.809     0.589
##   0.10       5                  250      0.807     0.585
##   0.10       7                   50      0.819     0.607
##   0.10       7                   75      0.820     0.612
##   0.10       7                  100      0.816     0.603
##   0.10       7                  125      0.815     0.600
##   0.10       7                  150      0.814     0.599
##   0.10       7                  175      0.811     0.594
##   0.10       7                  200      0.809     0.589
##   0.10       7                  225      0.807     0.585
##   0.10       7                  250      0.805     0.582
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 5
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 50, interaction.depth =
##  5, shrinkage = 0.1 and n.minobsinnode = 5.
```

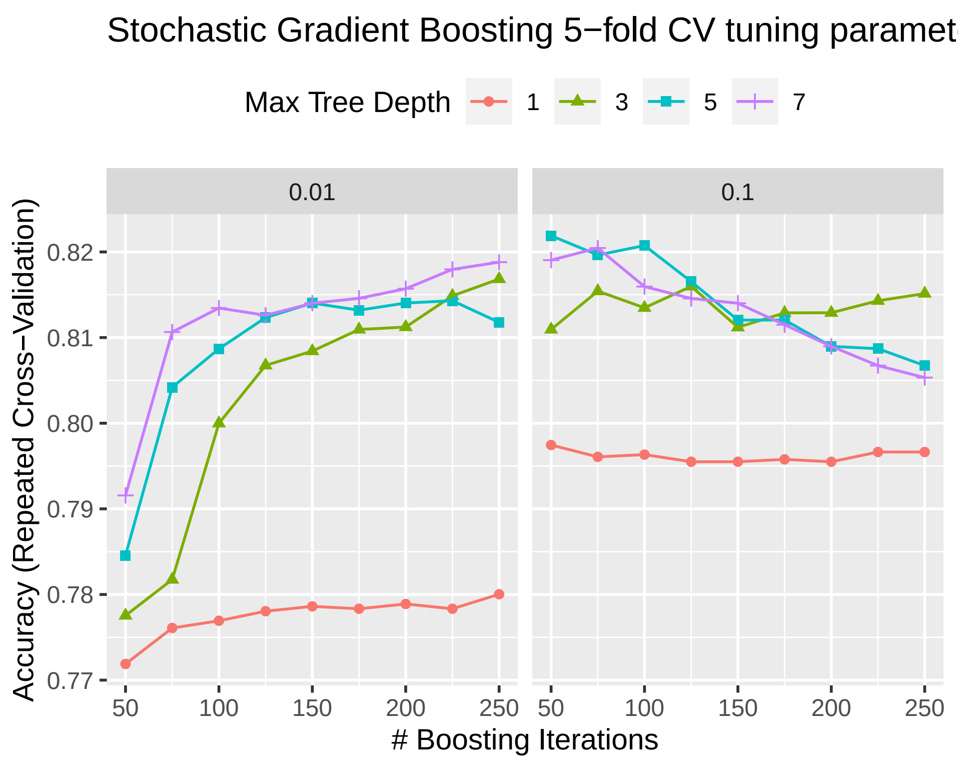
```r
train_gbm$bestTune
```

```
##    n.trees interaction.depth shrinkage n.minobsinnode
## 55      50                 5       0.1              5
```

```r
gbm_preds <- predict(train_gbm, test_set, type = "raw") %>%confusionMatrix(test_set$Survived)
gbm_preds
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 102  20
##          1   8  49
##
##                Accuracy : 0.844
##                    95% CI : (0.782, 0.893)
##       No Information Rate : 0.615
##       P-Value [Acc > NIR] : 1.79e-11
##
##                     Kappa : 0.659
##
##    Mcnemar's Test P-Value : 0.0376
##
##               Sensitivity : 0.927
##               Specificity : 0.710
##            Pos Pred Value : 0.836
##            Neg Pred Value : 0.860
##                Prevalence : 0.615
##            Detection Rate : 0.570
##      Detection Prevalence : 0.682
##         Balanced Accuracy : 0.819
##
##          'Positive' Class : 0
##
```

```
ggplot(train_gbm) +
  theme(legend.position = "top") +
  labs(title = "Stochastic Gradient Boosting 5-fold CV tuning parameter(s): tree depth, iterations, lea
```



3.8 Boosted Logistic Regression

```
# set.seed(616)
set.seed(616, sample.kind = "Rounding")
```

```
## Warning in set.seed(616, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

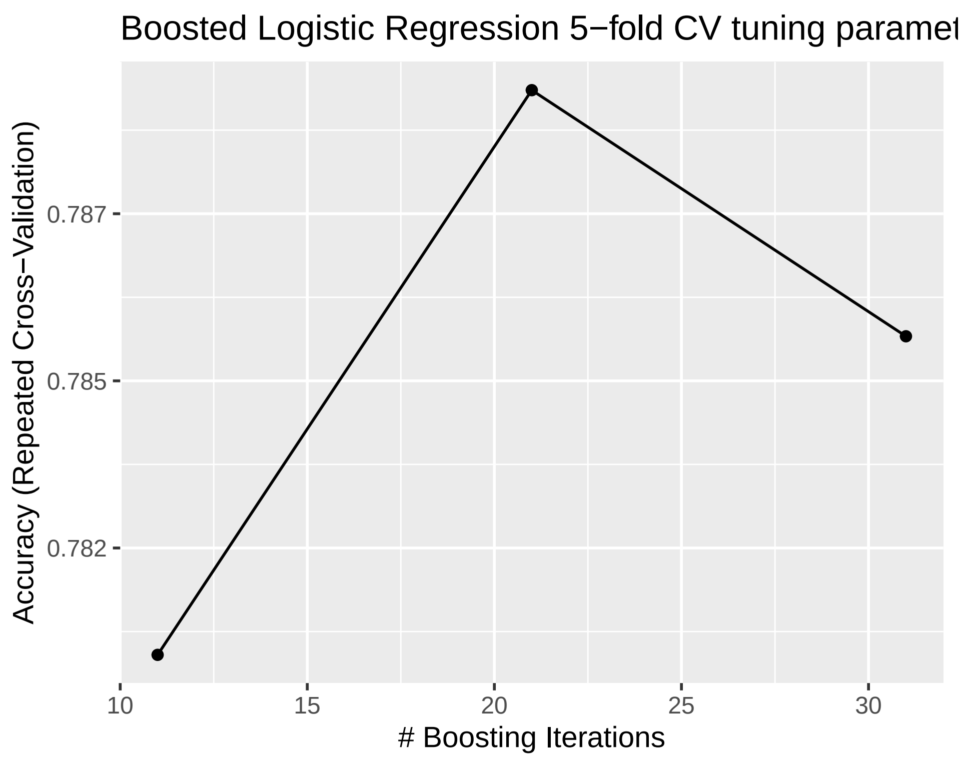
```
control = trainControl(method="repeatedcv",number=5, repeats=5)
train_blr = train(Survived~.,
                  data= train_set,
                  method="LogitBoost",
                  verbose=FALSE,
                  trControl=control)
train_blr
```

```
## Boosted Logistic Regression
##
## 712 samples
##   8 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 5 times)
## Summary of sample sizes: 569, 571, 569, 570, 569, 570, ...
## Resampling results across tuning parameters:
##
##   nIter  Accuracy  Kappa
##   11     0.781     0.527
##   21     0.789     0.547
##   31     0.786     0.534
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was nIter = 21.
```

```
preds_blr = predict(train_blr, test_set, type = "raw") %>%confusionMatrix(test_set$Survived)
preds_blr
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 96 20
##          1 14 49
##
##                Accuracy : 0.81
##                  95% CI : (0.745, 0.865)
##     No Information Rate : 0.615
##     P-Value [Acc > NIR] : 1.35e-08
##
##                   Kappa : 0.592
##
##  Mcnemar's Test P-Value : 0.391
##
##             Sensitivity : 0.873
##             Specificity : 0.710
##          Pos Pred Value : 0.828
##          Neg Pred Value : 0.778
##              Prevalence : 0.615
##          Detection Rate : 0.536
##    Detection Prevalence : 0.648
##       Balanced Accuracy : 0.791
##
##        'Positive' Class : 0
##
```

```
ggplot(train_blr) +
  theme(legend.position = "top")+
  labs(title = "Boosted Logistic Regression 5-fold CV tuning parameter(s): iterations")
```



3.9 Support Vector Machine Model

```
#set.seed(616)
set.seed(616, sample.kind = "Rounding")
```

```
## Warning in set.seed(616, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
control = trainControl(method="repeatedcv",number=5, repeats=5)
svmGrid = expand.grid(C = sapply(-3:3, exp), sigma = sapply(-3:1, exp))

train_svm = train(Survived ~.,
                  data=train_set,
                  method="svmRadial",
                  trControl=control,
                  tuneGrid=svmGrid)
train_svm
```

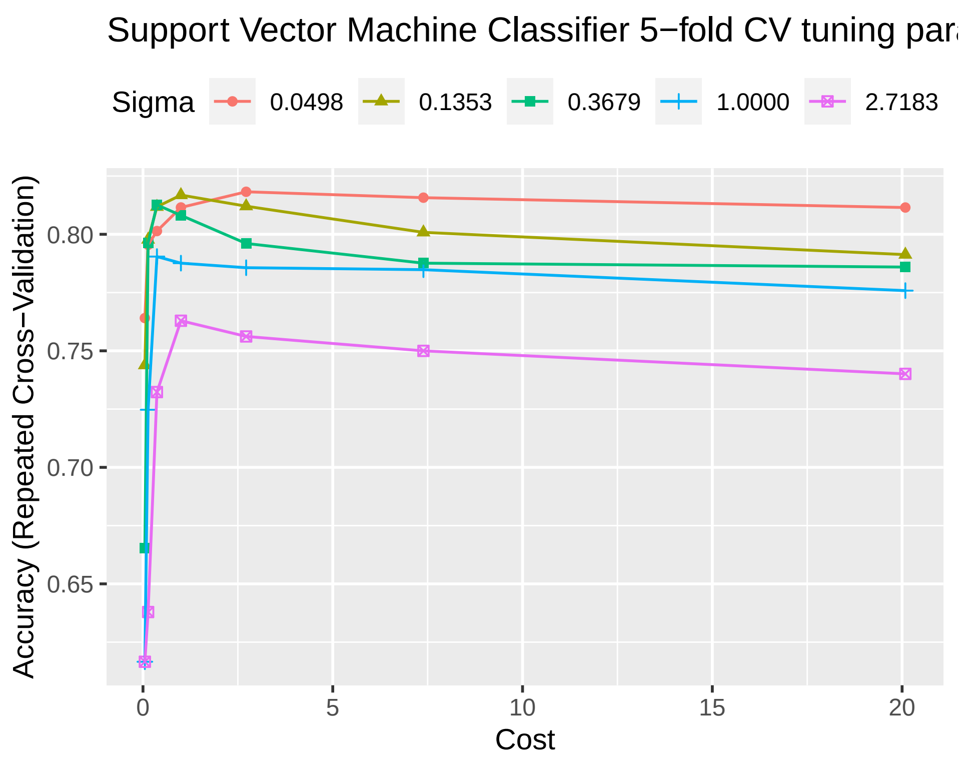
```
## Support Vector Machines with Radial Basis Function Kernel
##
```

```
## 712 samples
##   8 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 5 times)
## Summary of sample sizes: 569, 571, 569, 570, 569, 570, ...
## Resampling results across tuning parameters:
##
##   C         sigma   Accuracy  Kappa
##    0.0498  0.0498  0.764     0.4482
##    0.0498  0.1353  0.744     0.4015
##    0.0498  0.3679  0.665     0.1587
##    0.0498  1.0000  0.617     0.0000
##    0.0498  2.7183  0.617     0.0000
##    0.1353  0.0498  0.795     0.5527
##    0.1353  0.1353  0.798     0.5572
##    0.1353  0.3679  0.796     0.5498
##    0.1353  1.0000  0.725     0.3395
##    0.1353  2.7183  0.638     0.0752
##    0.3679  0.0498  0.801     0.5658
##    0.3679  0.1353  0.812     0.5905
##    0.3679  0.3679  0.813     0.5868
##    0.3679  1.0000  0.790     0.5375
##    0.3679  2.7183  0.732     0.3801
##    1.0000  0.0498  0.812     0.5896
##    1.0000  0.1353  0.817     0.5947
##    1.0000  0.3679  0.808     0.5769
##    1.0000  1.0000  0.788     0.5365
##    1.0000  2.7183  0.763     0.4991
##    2.7183  0.0498  0.818     0.6032
##    2.7183  0.1353  0.812     0.5834
##    2.7183  0.3679  0.796     0.5513
##    2.7183  1.0000  0.786     0.5340
##    2.7183  2.7183  0.756     0.4816
##    7.3891  0.0498  0.816     0.5916
##    7.3891  0.1353  0.801     0.5602
##    7.3891  0.3679  0.788     0.5345
##    7.3891  1.0000  0.785     0.5339
##    7.3891  2.7183  0.750     0.4668
##   20.0855  0.0498  0.811     0.5826
##   20.0855  0.1353  0.791     0.5419
##   20.0855  0.3679  0.786     0.5340
##   20.0855  1.0000  0.776     0.5147
##   20.0855  2.7183  0.740     0.4425
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.0498 and C = 2.72.
```

```r
svm_preds = predict(train_svm, test_set) %>%confusionMatrix(test_set$Survived)
svm_preds
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 99 17
##          1 11 52
##
##                Accuracy : 0.844
##                  95% CI : (0.782, 0.893)
##     No Information Rate : 0.615
##     P-Value [Acc > NIR] : 1.79e-11
##
##                   Kappa : 0.664
##
##  Mcnemar's Test P-Value : 0.345
##
##             Sensitivity : 0.900
##             Specificity : 0.754
##          Pos Pred Value : 0.853
##          Neg Pred Value : 0.825
##              Prevalence : 0.615
##          Detection Rate : 0.553
##    Detection Prevalence : 0.648
##       Balanced Accuracy : 0.827
##
##        'Positive' Class : 0
##
```

```
ggplot(train_svm) +
  theme(legend.position = "top") +
  labs(title = "Support Vector Machine Classifier 5-fold CV tuning parameter(s): cost, sigma")
```



3.10 Neural Network Model

```
# We select parameters for neural network by using 10-fold cross-validation with caret
# This make take several minutes


tune_grid <-expand.grid(size=1:10,decay=c(0,0.0001,0.05,0.1))

# set.seed(1)

set.seed(1, sample.kind = "Rounding")
```

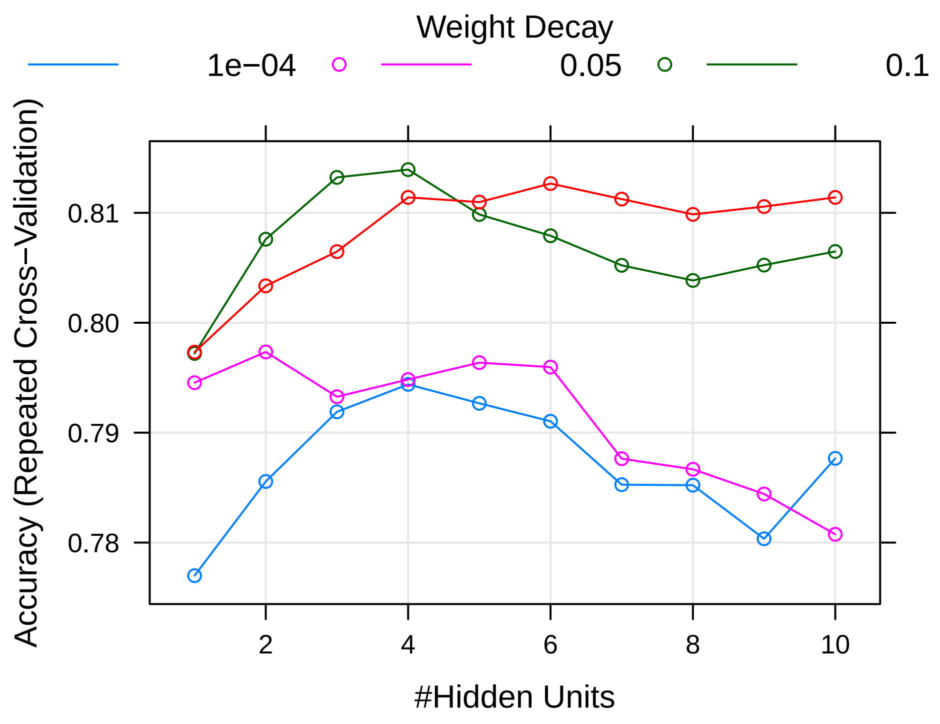
```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
control <- trainControl(method="repeatedcv", number=10, repeats=10)

train_nnet <-train(Survived~.,
                   data=train_set,
                   method="nnet",
                   trControl=control,
                   preProcess="range",
                   tuneLength=10,
                   tuneGrid=tune_grid,
                   trace=FALSE,
                   verbose=FALSE,
                   maxit=500)
nnet_preds <- predict(train_nnet,test_set)%>%confusionMatrix(test_set$Survived)
nnet_preds
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction    0   1
##            0 103  20
##            1   7  49
##
##                  Accuracy : 0.849
##                    95% CI : (0.788, 0.898)
##       No Information Rate : 0.615
##       P-Value [Acc > NIR] : 5.17e-12
##
##                     Kappa : 0.67
##
##    Mcnemar's Test P-Value : 0.0209
##
##               Sensitivity : 0.936
##               Specificity : 0.710
##            Pos Pred Value : 0.837
##            Neg Pred Value : 0.875
##                Prevalence : 0.615
##            Detection Rate : 0.575
##      Detection Prevalence : 0.687
##         Balanced Accuracy : 0.823
##
```

```
##        'Positive' Class : 0
##
```

```
plot(train_nnet)
```



```
best_size <- train_nnet$bestTune[1,"size"]
best_size
```

```
## [1] 4
```

```
best_decay <- train_nnet$bestTune[1,"decay"]
best_decay
```

```
## [1] 0.05
```

```
best_nn <- nnet( Survived ~ .,
                 method="nnet",
                 data = train_set,
                 size=best_size,
                 decay=best_decay,
                 maxit=500,
                 trace=FALSE)

best_nn_pred <- predict( best_nn, newdata=test_set, type="class" )  # yields "0", "1"
confusionMatrix(factor(best_nn_pred), test_set$Survived)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 105  22
##          1   5  47
```

```
##
##                Accuracy : 0.849
##                  95% CI : (0.788, 0.898)
##     No Information Rate : 0.615
##     P-Value [Acc > NIR] : 5.17e-12
##
##                   Kappa : 0.666
##
##  Mcnemar's Test P-Value : 0.00208
##
##             Sensitivity : 0.955
##             Specificity : 0.681
##          Pos Pred Value : 0.827
##          Neg Pred Value : 0.904
##              Prevalence : 0.615
##          Detection Rate : 0.587
##    Detection Prevalence : 0.709
##       Balanced Accuracy : 0.818
##
##        'Positive' Class : 0
##
```

4 Results

Most of the passengers were in age group of 20 to 40. We can see that a very less number of people survived and in those more number of females survived than males. Looking at the age<10 years, we see that the survival rate is high. And the survival rate is low and drops beyond the age of 45. Passengers who survived generally payed higher fares than those who did not survive. Only one individual paid a fare around $500. That individual survived. Most individuals who paid a fare around $8 did not survive. The largest group of passengers was third-class males. Most first-class and second-class females survived. Almost all second-class males did not survive, with the exception of children. It is evident that the survival rate of children, across 1st and 2nd class was the highest. Except for 1 girl child all children travelling 1st and 2nd class survived. The survival rates were lowest for men travelling 3rd class.

From our analysis we see that the accuracies for our models are as follows: LDA gives an accuracy 0.821, QDA 0.81, i.e. lower than that of LDA, kNN 0.827, the cross validation model gives the lowest result for the accuracy 0.648 only, the classification tree model 0.838 which is better than the above models, the random forest 0.849, the gradient boosting model 0.844, the boosted logistic regression 0.81, the support vector machine 0.844 and the neural network model gives the accuracy 0.849.

5 Conclusion.

From the above results, it follows that the best predicting models are those of the random forest and the neural networks. They, both, give the best accuracy which is 0.849.

6 References

1. Rafael A. Irizarry, "Introduction to Data Science Data Analysis and Prediction Algorithms with R", 2019.
2. https://www.kaggle.com
3. https://www.en.wikipedia.org
4. https://www.britannica.com/topic/Titanic