SPM4450: Fundamentals of Data Analytics - Final Assignment Report

Performance of various Data Analytics Techniques on Kaggle's Problem Set 'Titanic: Machine Learning from Disaster'

Authors:

Nidhi Singh
4242246
n.singh-2@student.tudelft.nl
MSc. Computer Science

K. Chaitanya Akundi
4239008
k.c.akundi@student.tudelft.nl
MSc. Computer Science

List of Figures

1.1 Passenger class by Sex, Age and Fare 4
1.2 Frequency of Sibsp, Parch and Embarkment 4
1.3 Passengers Survived by Age, Fare and Place of Embarkment . . . 5
1.4 Passengers survived by Class and Gender 5

3.1 ROC curves comparing classification performance. 24
3.2 ROC curves comparing classification performance. 25

Chapter 1

Titanic Data Set

1.1 Problem Description

For our final assignment, we have taken up a challenge from Kaggle 'Predict survival on the Titanic'. The dataset includes details of people who travelled on RMS Titanic which sank in 1912 killing 1502 out of 2224 passengers. The aim of the Kaggle challenge is to complete the analysis of what sorts of people were likely to survive. In order to do so, we will apply different predictive models to the dataset and will finally evaluate their performance against each other. Kaggle also supports Leaderboards which evaluate the submitted results, but since this evaluation is based on only 50% of the test data, it makes sense to do performance evaluation of all the models.

Since we are given both training and test data set, this problem's predictive models will fall under the umbrella of Supervised Learning Algorithms. Also we have to decide whether a passenger survived or not, this makes it a classic Classification problem.

1.2 Data Exploration

Before diving deep into prediction making on test data, we will explore the dataset. We are given two sets of data, training (data containing attributes and known outcomes [survived or perished] for a subset of the passengers) and test (data containing attributes without outcomes for a subset of passengers).The given training data set has 891 observations of following 12 variables:

- PassengerId - Unique generated Id for each passenger
- Survived - Survival(0 = No; 1 = Yes)
- Pclass - Passenger Class (1 = 1st; 2 = 2nd; 3 = 3rd)
- Name - Name of the person
- sex - Sex
- Age - Age
- Sibsp - Number of Siblings/Spouses Aboard

- Parch - Number of Parents/Children Aboard
- Ticket - Ticket Number
- Fare - Passenger Fare
- Cabin - Cabin in the ship
- Embarked - Port of Embarkation (C = Cherbourg; Q = Queenstown; S = Southampton)

Let us start by looking at the type of these variables

```
## 'data.frame': 891 obs. of  12 variables:
##  $ PassengerId: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Survived   : int  0 1 1 1 0 0 0 0 1 1 ...
##  $ Pclass     : int  3 1 3 1 3 3 1 3 3 2 ...
##  $ Name       : Factor w/ 891 levels "Abbing, Mr. Anthony",..: 109 191 358 277 16 559 5
##  $ Sex        : Factor w/ 2 levels "female","male": 2 1 1 1 2 2 2 2 1 1 ...
##  $ Age        : num  22 38 26 35 35 NA 54 2 27 14 ...
##  $ SibSp      : int  1 1 0 1 0 0 0 3 0 1 ...
##  $ Parch      : int  0 0 0 0 0 0 0 1 2 0 ...
##  $ Ticket     : Factor w/ 681 levels "110152","110413",..: 525 596 662 50 473 276 86 39
##  $ Fare       : num  7.25 71.28 7.92 53.1 8.05 ...
##  $ Cabin      : Factor w/ 148 levels "","A10","A14",..: 1 83 1 57 1 1 131 1 1 1 ...
##  $ Embarked   : Factor w/ 4 levels "","C","Q","S": 4 2 4 4 4 3 4 4 4 2 ...
```

Here Factor refers to categorical data, since all the names are unique, we have 891 levels equal to number of observations.

```
prop.table(table(train_csv$Survived))

##
##      0      1
## 0.6162 0.3838
```

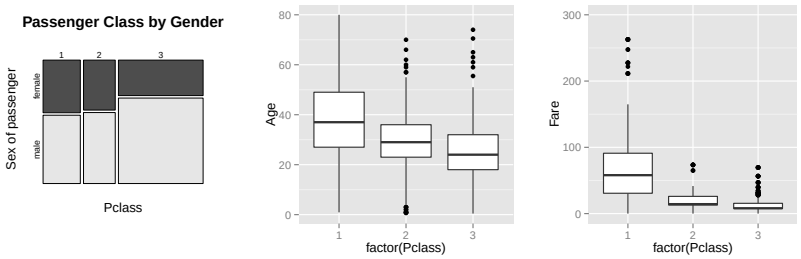
This shows that 61.6% of the passengers perished and only 38.3% survived. Running the same code for Sex, we find 35.2% females and 64.7% in the training data set.

```
summary(train_csv$Age)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##    0.42   20.10   28.00   29.70   38.00   80.00     177
```

Summary results on Age shows that this variable is missing for 177 passengers and the minimum age is 0.42 or 5 months and maximum is 80, while 90% of the passengers were below 50.

```
prop.table(table(train_csv$Pclass))

##
##      1      2      3
## 0.2424 0.2065 0.5511
```

More than 55% passengers were travelling in third class. It will be worthwhile to see the age and sex of people in each class.



(a) Passenger Class by Gender. (b) Passenger Class by Age. (c) Passenger Class by Fare.

Figure 1.1: Passenger class by Sex, Age and Fare

We see in Figure1.1 that third class has mostly males, since third class cabins were at the bottom of the ship this might be one of the reasons that most of the males could not survive. Also passengers in third class were younger with median below 25.With just one outlier above $500 for first class ticket fare, fare is below $100.

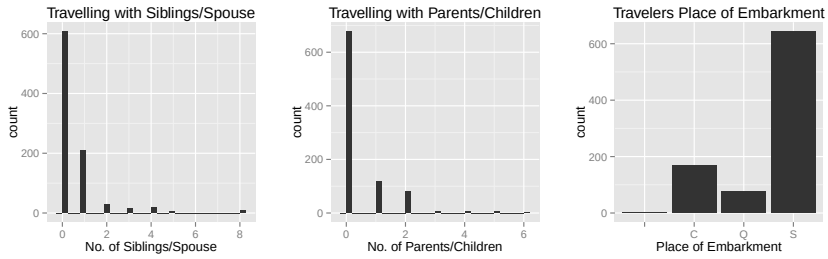


Figure 1.2: Frequency of Sibsp, Parch and Embarkment

We now look at other variables to see if they can have some influence on predictions. From Figure 1.2 we can see that most passengers travelled alone and started their journey from Southampton.

Other varibales 'Ticket' and 'Cabin' do not tell much as they have unique values, and are un-related to other variables.

1.2.1 Survived variable with other variables

Till now we looked at the variables and their values and frequencies and tried to get an initial understanding of the data. Since we have to predict the 'Survived' variable for the test set, in this section we will look at the relation between 'Survived' variable and other variables.

As we can see from Figure 1.3 age and fare doesnt seem to give much information about the survived variable, moreover most of the passengers were from Southampton so Place of Embarkment doesn't seem to play much role too.

But from Figure 1.4 we can find some interesting facts, people in 1st class outnumbered the people from 3rd class in survival rate.So there was a clear preference for elite poeple. From the second plot in Figure1.4 we can see another preference was for females, we would like to believe that there was preference for children but this is not yet evident from our data. The last plot in Figure 1.4 shows that surely there was a clear bias for females in 1st and 2nd class compared to males. This is a clear indicator that 'Sex' variable is hihgly important for our analysis with maybe 'Pclass' coming next.

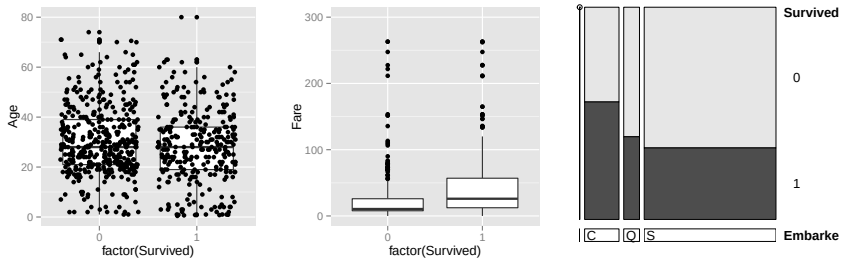


Figure 1.3: Passengers Survived by Age, Fare and Place of Embarkment

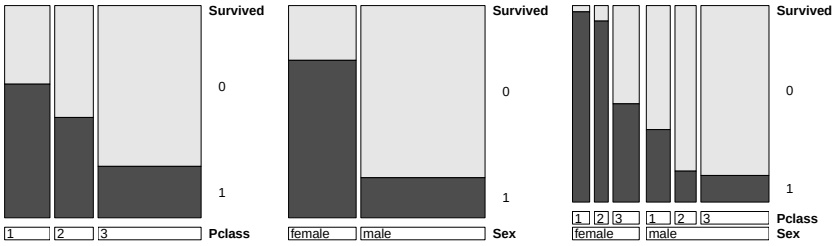


Figure 1.4: Passengers survived by Class and Gender

1.3 Feature Engineering

There are few variables which have 'NA' and missing values, before we run our prediction models we need to get rid of these. There are 177 records in the dataset where age value is NA. For these records, we assume the age to be the average age of the group.

1.3.1 Title

The title a passenger holds, such as 'Mister', 'Miss' and others, might be a useful bit of information for predicting the fate of the passenger. To get the title from the name field, part of the name string is extracted. Also, groups of similar titles are agglomerated into a single title.

1.3.2 FamilySize

Based on the assumption that larger families might have had trouble finding each other and getting to the lifeboats, we created a feature FamilySize which combines the SibSp and Parch features already available.

1.3.3 FamilyID

Continuing on the basis of the same assumption, we felt it would be useful to have a feature that combines family size and surname, so we can observe how large families are affected.

1.3.4 Training Data

The training dataset containing 891 records was further partitioned with 80% of it taken as the training set, and the remaining 20% as the test set, to evaluate the performance of the models. So there were 713 records in the training partition and 178 records in the test partition.

Chapter 2

Prediction Models

Since the 'Predict survival on the Titanic' challenge is a classification problem, we will start with Linear Classifiers and then go further with methods like Decision trees and Ensembles of classifiers. In the following sections we will explore each model in detail and will also report its evaluation on Kaggle.

2.1 Logistic Regression

Logistic Regression is a classical Classification algorithm. R's *glm* function is used to fit generalized linear models, With *family* variable set to "binomial", glm( ) produces a logistic regression.The output of a linear regression can be transformed to a logit functions as follows:

$$logitp = \log o = \log p/1 - p = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_k x_k$$

Exponentiating this we get,

$$\exp^{logitp} = \exp^{\beta_0} \exp^{\beta_1 x_1} \exp^{\beta_2 x_2} ... \exp^{\beta_k x_k}$$

rewriting we get,

$$o = p/1 - p = \exp^{\beta_0} \exp^{\beta_1 x_1} \exp^{\beta_2 x_2} ... \exp^{\beta_k x_k}$$

Here o represents the *odds*. From this we can say that if we know that a certain fact is true of a data point, then that will produce a constant change in the odds of the outcome.

We will run our first regression model with basic features provided within the dataset and can look at the results by calling summary on this model.Summary gives the value of *estimated coefficients alongwith their standard errors and p-value* of each input variable.

```
logit.m1 <- glm(Survived ~ Pclass + Sex + Age + SibSp +
                  Parch + Fare + Embarked,
                data = train.batch,
                family="binomial")
summary(logit.m1)
```

```
##
## Call:
## glm(formula = Survived ~ Pclass + Sex + Age + SibSp + Parch +
##     Fare + Embarked, family = "binomial", data = train.batch)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -2.325  -0.617  -0.430   0.631   2.426
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  5.17351    0.62955    8.22  < 2e-16 ***
## Pclass      -1.06431    0.16184   -6.58  4.8e-11 ***
## Sexmale     -2.65191    0.21986  -12.06  < 2e-16 ***
## Age         -0.04189    0.00881   -4.76  2.0e-06 ***
## SibSp       -0.34715    0.11884   -2.92   0.0035 **
## Parch       -0.06922    0.13855   -0.50   0.6174
## Fare         0.00301    0.00271    1.11   0.2658
## EmbarkedQ   -0.00725    0.41417   -0.02   0.9860
## EmbarkedS   -0.39828    0.26465   -1.50   0.1323
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 951.76  on 712  degrees of freedom
## Residual deviance: 634.98  on 704  degrees of freedom
## AIC: 653
##
## Number of Fisher Scoring iterations: 5
```

Logistic Regression uses the deviance test to estimate the goodness of the model. Its approach is based on estimating two models, it assumes that one excludes the hypothesized effects to be null, while to be included in the other. For each model, a deviance statistic, equal to -2 ln L for that model, is computed, which in this case is *947.99(Null model) and 641.48(estimated model)*. The deviance can be regarded as a measure of lack of fit between model and data. In general, the larger the deviance, the poorer the fit to the data. In our first iteration with basic features we can see reduction of *327.62* in deviance. The difference between the deviances has a large-sample chi-square distribution with degrees of freedom equal to the difference in the number of parameters estimated. Thus the difference in deviances can be tested against the chi-square distribution for significance, as done below.

```
pchisq(946.03-618.41, 9)
```

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

Also, we can see from the summary that only *Pclass, Sex, Age and SibSp(moderately)*

has effect on the model with significance less than .05 level. To see how these variables affect, we can also do *ANOVA* test, which tries adding the factors in the given order.

```
anova(logit.m1, test="Chisq")
```

```
## Analysis of Deviance Table
##
## Model: binomial, link: logit
##
## Response: Survived
##
## Terms added sequentially (first to last)
##
##
##          Df Deviance Resid. Df Resid. Dev Pr(>Chi)
## NULL                       712        952
## Pclass    1     80.1       711        872  < 2e-16 ***
## Sex       1    198.9       710        673  < 2e-16 ***
## Age       1     19.4       709        653  1.1e-05 ***
## SibSp     1     12.8       708        641  0.00035 ***
## Parch     1      0.1       707        640  0.79638
## Fare      1      2.4       706        638  0.12427
## Embarked  2      3.1       704        635  0.20912
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

From the ANOVA results too, we can see that the deviance is highly reduced by *Pclass, Sex* and weakly reduced by *Age and SibSp* and with more or less no affect on deviance by other variables(Parch, Fare and Embarked). To improve our model further we can make use of *engineered variables* from previous chapter, namely *Title,FamilySize and FamilyID2*

```
logit.m2 <- glm(Survived ~ Pclass + Sex + Age + SibSp + Parch +
                  Embarked + Title +
                  FamilyID2,
                data = train.batch,
                family="binomial")
```

```
pchisq(365.39, 30)
```

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

```
anova(logit.m2, test="Chisq")
```

```
## Analysis of Deviance Table
##
## Model: binomial, link: logit
##
```

```
## Response: Survived
##
## Terms added sequentially (first to last)
##
##
##           Df Deviance Resid. Df Resid. Dev Pr(>Chi)
## NULL                        712        952
## Pclass     1     80.1       711        872  < 2e-16 ***
## Sex        1    198.9       710        673  < 2e-16 ***
## Age        1     19.4       709        653  1.1e-05 ***
## SibSp      1     12.8       708        641  0.00035 ***
## Parch      1      0.1       707        640  0.79638
## Embarked   2      4.1       705        636  0.12871
## Title      8     45.5       697        591  3.0e-07 ***
## FamilyID2 21     45.4       676        546  0.00153 **
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The new model results in *504.44* deviance, reducing *442.59* points over null deviance of 946.03. The ANOVA test results show that the engineered variable *Title and FamilyID2* have a significant effect on deviance. Thus, we will be including it in our further modelling.

2.2 Naive Bayes

Naive Bayes (NB) and Logistic Regression are both linear classifiers but they estimate parameters differently. Logistic Regression estimates the best model fit by minimizing the error or deviances, whereas Naive Bayes estimates 'prior' probability from the training data and uses Bayes rule to predict new instances.

Since our training set has fewer instances, logistic regression might overfit, thus it would be interesting to see the performance of Naive Bayes. Also with Naive Bayes an assumption is made that our inputs are independent, thus we should not have collinear inputs in our model.

```
nb.classifier <- naiveBayes(as.factor(Survived) ~ Pclass + Sex +
                   Age + SibSp + Parch + Embarked +
                   Title + FamilyID2,
                  data = train.batch)
nb.predict <- predict(nb.classifier, test.batch)

## Warning:  NAs introduced by coercion
## Warning:  NAs introduced by coercion
```

To see the accuracy of the NB model on test data set, we can look at confusion matrix statistics

```
confusionMatrix(nb.predict,test.batch$Survived)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 98 17
##          1 14 49
##
##                Accuracy : 0.826
##                  95% CI : (0.762, 0.878)
##     No Information Rate : 0.629
##     P-Value [Acc > NIR] : 7.85e-09
##
##                   Kappa : 0.623
##  Mcnemar's Test P-Value : 0.719
##
##             Sensitivity : 0.875
##             Specificity : 0.742
##          Pos Pred Value : 0.852
##          Neg Pred Value : 0.778
##              Prevalence : 0.629
##          Detection Rate : 0.551
##    Detection Prevalence : 0.646
##       Balanced Accuracy : 0.809
##
##        'Positive' Class : 0
##
```

The NB model performs with 82.6% accuracy on test batch but performs poorly and same as Logistic Regression on Kaggle test batch with 77.03% accuracy.

2.3 Decision Trees

Decision tree algorithms work by repeatedly splitting the dataset into subsets based on a particular attribute value. This process is recursively carried out until further splitting does not add any value to the predictions. This is a greedy algorithm, which means that decisions with the highest immediate value are given preference.

We applied recursive partitioning on the Titanic dataset using the Decision Tree algorithm from R's *rpart* package . For this, the dependent variables used were Pclass, Sex, Age, Fare, Embarked, Title and FamilyID2.

The algorithm was run multiple times with different parameters and control values, and then analyzed the results through a confusion matrix.

2.3.1

```
dt.1 <- rpart(Survived ~ Pclass + Sex + Age + Fare + Embarked +
    Title + FamilyID2, data = train.batch, method = "class",
```

```
    control = rpart.control(minsplit = 2, cp = 0.001))
dt.1.pred <- predict(dt.1, test.batch, type = "class")
confusionMatrix(dt.1.pred, test.batch$Survived)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 91 19
##          1 21 47
##
##                Accuracy : 0.775
##                  95% CI : (0.707, 0.834)
##     No Information Rate : 0.629
##     P-Value [Acc > NIR] : 2.1e-05
##
##                   Kappa : 0.521
##  Mcnemar's Test P-Value : 0.874
##
##             Sensitivity : 0.812
##             Specificity : 0.712
##          Pos Pred Value : 0.827
##          Neg Pred Value : 0.691
##              Prevalence : 0.629
##          Detection Rate : 0.511
##    Detection Prevalence : 0.618
##       Balanced Accuracy : 0.762
##
##        'Positive' Class : 0
##
```

We first ran the rpart algorithm using only the variables from the given data, i.e Pclass, Sex, Age, SibSp, Parch, Fare and Embarked, and no control parameters. Here, the first parameter indicates the dependent variable being estimated with respect to the independent variables, the second indicates the training data set, the third indicates the method, which is set to 'class' as the variable 'Survived' has only two levels. The last argument specifies various parameters for the working of the recursive partitioning algorithm. In our case, we specify 'minsplit', which indicates the minimum number of observations required at a node to make a split. *cp* is the complexity parameter. Any split that does not reduce complexity by a factor of *cp* is not made.

2.3.2

In the second run, we added the engineered features Title and FamilyID2 to the estimation, while removing Age, SibSp and Parch which are already composed in Title and FamilyID2.

```
dt.2 <- rpart(Survived ~ Pclass + Fare + Embarked + Title + FamilyID2,
    data = train.batch, method = "class")
dt.2.pred <- predict(dt.2, test.batch, type = "class")
confusionMatrix(dt.2.pred, test.batch$Survived)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 105  20
##          1   7  46
##
##                Accuracy : 0.848
##                  95% CI : (0.787, 0.898)
##     No Information Rate : 0.629
##     P-Value [Acc > NIR] : 9.11e-11
##
##                   Kappa : 0.661
##  Mcnemar's Test P-Value : 0.0209
##
##             Sensitivity : 0.938
##             Specificity : 0.697
##          Pos Pred Value : 0.840
##          Neg Pred Value : 0.868
##              Prevalence : 0.629
##          Detection Rate : 0.590
##    Detection Prevalence : 0.702
##       Balanced Accuracy : 0.817
##
##        'Positive' Class : 0
##
```

This resulted in marginal gains in accuracy and sensitivity of the model, thus giving better results on unseen data.

2.3.3

In the third run, we added control parameters which specified *minsplit* as 2 and *cp* as 0. This allowed unbounded growth for the tree, and resulted in a complex structure with large a number of branches.

```
dt.3 <- rpart(Survived ~ Pclass + Age + Fare + Embarked + Title +
    FamilyID2, data = train.batch, method = "class", control = rpart.control(minsplit = 2,
    cp = 0))
dt.3.pred <- predict(dt.3, test.batch, type = "class")
confusionMatrix(dt.3.pred, test.batch$Survived)

## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction  0  1
##          0 91 19
##          1 21 47
##
##                Accuracy : 0.775
##                  95% CI : (0.707, 0.834)
##     No Information Rate : 0.629
##     P-Value [Acc > NIR] : 2.1e-05
##
##                   Kappa : 0.521
##  Mcnemar's Test P-Value : 0.874
##
##             Sensitivity : 0.812
##             Specificity : 0.712
##          Pos Pred Value : 0.827
##          Neg Pred Value : 0.691
##              Prevalence : 0.629
##          Detection Rate : 0.511
##    Detection Prevalence : 0.618
##       Balanced Accuracy : 0.762
##
##        'Positive' Class : 0
##
```

The results were very accurate on the training data. However, the model performed poorly with the test data. From this, we could conclude that the model is overfitting to a high degree.

2.4 Support Vector Machines

Support Vector Machine (SVM) algorithms perform classification by representing the given data as points in space, with a clear plane of separation between the different classes to which the points belong. SVMs use the 'kernel trick' to work with high-dimensional data without ever having to map the points in such spaces. In R, the `e1071` and `kernlab` packages offer methods for creating SVM models. We used the `kernlab` package to generate our SVM models with the `rbfdot` kernel, which uses a Gaussian Radial Basis Function.

2.4.1

Initially, the algorithm was run using all the features that were already present in the given dataset. The model performs well on the training dataset, with an accuracy of 84.27%. However, it did not do so well on the test set, being able to predict the correct outcome only around 62% of the time. More statistics from the tests are shown below.

```
svm.model.1 <- ksvm(Survived ~ Sex + Pclass + Age + Fare + SibSp +
    Parch + Embarked, data = train.batch, kernel = "rbfdot",
    type = "C-svc")
```

```
## Using automatic sigma estimation (sigest) for RBF or laplace kernel

svm.pred.1 <- predict(svm.model.1, type = "response", test.batch)
confusionMatrix(svm.pred.1, test.batch$Survived)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 106  28
##          1   6  38
##
##                Accuracy : 0.809
##                  95% CI : (0.743, 0.864)
##     No Information Rate : 0.629
##     P-Value [Acc > NIR] : 1.48e-07
##
##                   Kappa : 0.561
##  Mcnemar's Test P-Value : 0.000316
##
##             Sensitivity : 0.946
##             Specificity : 0.576
##          Pos Pred Value : 0.791
##          Neg Pred Value : 0.864
##              Prevalence : 0.629
##          Detection Rate : 0.596
##    Detection Prevalence : 0.753
##       Balanced Accuracy : 0.761
##
##        'Positive' Class : 0
##
```

Here, the type 'C-svc' indicates that the algorithm must run in the classification mode.

2.4.2

```
svm.model.2 <- ksvm(Survived ~ Sex + Pclass + Embarked, data = train.batch,
    kernel = "rbfdot", type = "C-svc")

## Using automatic sigma estimation (sigest) for RBF or laplace kernel

svm.pred.2 <- predict(svm.model.2, type = "response", test.batch)
confusionMatrix(svm.pred.2, test.batch$Survived)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
```

```
##          0 107  31
##          1   5  35
##
##                Accuracy : 0.798
##                  95% CI : (0.731, 0.854)
##     No Information Rate : 0.629
##     P-Value [Acc > NIR] : 8.83e-07
##
##                   Kappa : 0.528
##  Mcnemar's Test P-Value : 3.09e-05
##
##             Sensitivity : 0.955
##             Specificity : 0.530
##          Pos Pred Value : 0.775
##          Neg Pred Value : 0.875
##              Prevalence : 0.629
##          Detection Rate : 0.601
##    Detection Prevalence : 0.775
##       Balanced Accuracy : 0.743
##
##        'Positive' Class : 0
##
```

The algorithm was then run using a subset of the available features, containing Pclass, Sex and Embarked. This model proved to be slightly better performing with the training data, with an accuracy of 84.83%, while exhibiting higher sensitivity and lower specificity. The model performed significantly better on unseen data, correctly predicting outcomes 77.99% of the time.

2.4.3

```
svm.model.3 <- ksvm(Survived ~ Sex + Pclass + Embarked + Title +
    FamilyID2, data = train.batch, kernel = "rbfdot", type = "C-svc",
    prob.model = TRUE)

## Using automatic sigma estimation (sigest) for RBF or laplace kernel

svm.pred.3 <- predict(svm.model.3, type = "response", test.batch)
confusionMatrix(svm.pred.3, test.batch$Survived)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 107  29
##          1   5  37
##
##                Accuracy : 0.809
##                  95% CI : (0.743, 0.864)
```

```
##     No Information Rate : 0.629
##     P-Value [Acc > NIR] : 1.48e-07
##
##                   Kappa : 0.558
##  Mcnemar's Test P-Value : 8.00e-05
##
##             Sensitivity : 0.955
##             Specificity : 0.561
##          Pos Pred Value : 0.787
##          Neg Pred Value : 0.881
##              Prevalence : 0.629
##          Detection Rate : 0.601
##    Detection Prevalence : 0.764
##       Balanced Accuracy : 0.758
##
##        'Positive' Class : 0
##
```

In the next run, the engineered features Title and FamilyID2 were used along with Pclass and Sex to create the model. This model showed slightly better performance compared to the previous one, both with the training data and the test set from Kaggle.

2.4.4

In the final run, we constructed the SVM model using all the available features, and the features that were engineered. The performance of this model on the training set was comparable to that of the other models. On the unseen test data, the model performed poorly, predicting the right outcome only 61.2% of the time.

```
svm.model.4 <- ksvm(Survived ~ Sex + Pclass + Embarked + Fare +
    Age + FamilyID2 + SibSp + Parch + Title, data = train.batch,
    kernel = "rbfdot", type = "C-svc")

## Using automatic sigma estimation (sigest) for RBF or laplace kernel

svm.pred.4 <- predict(svm.model.4, type = "response", test.batch)
confusionMatrix(svm.pred.4, test.batch$Survived)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 106  25
##          1   6  41
##
##                Accuracy : 0.826
##                  95% CI : (0.762, 0.878)
##     No Information Rate : 0.629
```

```
##     P-Value [Acc > NIR] : 7.85e-09
##
##                   Kappa : 0.603
##  Mcnemar's Test P-Value : 0.00123
##
##             Sensitivity : 0.946
##             Specificity : 0.621
##          Pos Pred Value : 0.809
##          Neg Pred Value : 0.872
##              Prevalence : 0.629
##          Detection Rate : 0.596
##    Detection Prevalence : 0.736
##       Balanced Accuracy : 0.784
##
##        'Positive' Class : 0
##
```

2.5 Random Forests

Random Forests are an extension of tree methods. The inherent variability of the tree model, and susceptibility to data is a problem, in random forest this is considered a strength. The intuition is that by averaging across the high variance and low bias trees we will end up with low variance low bias estimated model. We will be using R's `randomForest` package which is based on Breiman and Cutler's original Fortran code.

Before modelling we need to decide on few input variables needed for the algorithm, the number of trees to grow, `ntree`, its value should be such that each input row should get predicted a few times. Also number of variables to be sampled as candidates at each split, `mtry`, should be decided. To find the optimal value of `mtry`, we can use `tuneRF` function of `randomForest` package.

```
bestmtry <- tuneRF(data.frame(train.batch$Pclass,train.batch$Sex,
                              train.batch$Age,train.batch$SibSp,
                              train.batch$Parch,train.batch$Embarked,
                              train.batch$Title,train.batch$FamilyID2),
                   train.batch$Survived, ntreeTry=100,
                   stepFactor=1.5,improve=0.01, trace=TRUE, plot=TRUE,
                   dobest=FALSE)
```

Here `ntreeTry` is the number of trees used at the tuning step, `stepFactor` value by which mtry is inflated at every iteration. This function gives the value of `mtry` as *3*. We now perform random forest algorithm, first with `ntree` set to 1000.

```
rf.m1 <-randomForest(as.factor(Survived) ~ Pclass + Sex + Age + SibSp +
                       Parch + Fare+ FamilySize +
                       Embarked + Title + FamilyID2,
```

```
                    data=train.batch, mtry=3, ntree=2000,
                    keep.forest=TRUE, importance=TRUE)
rf.predict.1 <- predict(rf.m1, test.batch, type = "response")
confusionMatrix(rf.predict.1, test.batch$Survived)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 104  21
##          1   8  45
##
##                Accuracy : 0.837
##                  95% CI : (0.774, 0.888)
##     No Information Rate : 0.629
##     P-Value [Acc > NIR] : 9.16e-10
##
##                   Kappa : 0.636
##  Mcnemar's Test P-Value : 0.0259
##
##             Sensitivity : 0.929
##             Specificity : 0.682
##          Pos Pred Value : 0.832
##          Neg Pred Value : 0.849
##              Prevalence : 0.629
##          Detection Rate : 0.584
##    Detection Prevalence : 0.702
##       Balanced Accuracy : 0.805
##
##        'Positive' Class : 0
##
```

and then again with `ntree` set to 2000

```
rf.m2 <-randomForest(as.factor(Survived) ~ Pclass + Sex + Age + SibSp +
                       Parch + Fare+ FamilySize +
                       Embarked + Title + FamilyID2,
                     data=train.batch, mtry=3, ntree=2000,
                     keep.forest=TRUE, importance=TRUE)
rf.predict.2 <- predict(rf.m2, test.batch, type = "response")
confusionMatrix(rf.predict.2, test.batch$Survived)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 103  21
##          1   9  45
##
##                Accuracy : 0.831
```

```
##                  95% CI : (0.768, 0.883)
##     No Information Rate : 0.629
##     P-Value [Acc > NIR] : 2.73e-09
##
##                   Kappa : 0.625
##  Mcnemar's Test P-Value : 0.0446
##
##             Sensitivity : 0.920
##             Specificity : 0.682
##          Pos Pred Value : 0.831
##          Neg Pred Value : 0.833
##              Prevalence : 0.629
##          Detection Rate : 0.579
##    Detection Prevalence : 0.697
##       Balanced Accuracy : 0.801
##
##        'Positive' Class : 0
##
```

With `ntree` set to 5000

```
rf.m3 <-randomForest(as.factor(Survived) ~ Pclass + Sex + Age + SibSp +
                       Parch + Fare+ FamilySize +
                       Embarked + Title + FamilyID2,
                     data=train.batch, mtry=3, ntree=5000,
                     keep.forest=TRUE, importance=TRUE)
rf.predict.3 <- predict(rf.m3, test.batch, type = "response")
confusionMatrix(rf.predict.3, test.batch$Survived)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 105  21
##          1   7  45
##
##                Accuracy : 0.843
##                  95% CI : (0.781, 0.893)
##     No Information Rate : 0.629
##     P-Value [Acc > NIR] : 2.95e-10
##
##                   Kappa : 0.648
##  Mcnemar's Test P-Value : 0.014
##
##             Sensitivity : 0.938
##             Specificity : 0.682
##          Pos Pred Value : 0.833
##          Neg Pred Value : 0.865
##              Prevalence : 0.629
##          Detection Rate : 0.590
```

```
##    Detection Prevalence : 0.708
##       Balanced Accuracy : 0.810
##
##        'Positive' Class : 0
##
```

From the statistics and confusion matrix, we can see that the accuracy of the model increases by approximately 1% when number of trees variable is increased from 1000 to 2000 but does not increase further if we set `ntree` to 5000, indicating the model has acheived its optimum(low bias and low variance) when there are 2000 trees in forest.

Chapter 3

Model Evaluation

For evaluation of models made in previous chapter we will use R's package `ROCR`, its major advantage is it eases the task of comparison by visualizing classifier performance. An ROC curve is a way of determining the performance of classifiers. It uses statistics such as True and False Positive Rates to evaluate the performance. The statistics are calculated using a confusion matrix. A confusion matrix is a tabulation of expected and observed values, along with the associated statistics. It is of the form shown below.

A basic ROC curve is as shown in Fig. 3.1 It shows the performance of five classifiers A, B, C, D and E. A classifier plotted at the point (0,0) in this graph never gives a positive classification, hence committing no false positive errors, but not gaining any true positives as well. On the other hand, the point (1,1) represents the opposite kind of strategy. A classifier plotted at (0,1) has perfect classification. The performance of classifiers can be measured relative to these points. A classifier is better than another if it either has higher true positive rate, lower false positive rate, or both. In this case, D is the best performing model.

3.0.1 Evaluation

From the obtained curves and the theory of ROC graphs, it can be observed that the classifier based on logistic regression has performed the best, while the SVM-based classifier has the worst performance for the given dataset. The accuracy of the classifier can be determined by computing the area under it's ROC curve. The area under the curve(auc) was calculated for all five classifiers. Logistic regression was the most accurate with an auc value of 0.909, followed by the Random Forest classifier at 0.895 and Naive Bayes-based classifier at 0.884. The SVM model performed the worst, with an auc of 0.786.

	Actual Value	
Prediction	0	1
0	A	B
1	C	D

Table 3.1: Confusion Matrix

Various statistics can be calculated from the confusion matrix.

- Accuracy: (A+D)/(A+B+C+D)
- 95% Confidence Interval: A measure of reliability, calculated using an exact binomial test
- No Information Rate: The largest class percentage in the data
- P-value: The probability of obtaining a test statistic result at least as extreme as the one that was actually observed, assuming that the null hypothesis is true
- True Positive Rate: D/(A+B+C+D)
- False Positive Rate: C/(A+B+C+D)
- Sensitivity: A/(A+C)
- Specificity: D/(B+D)

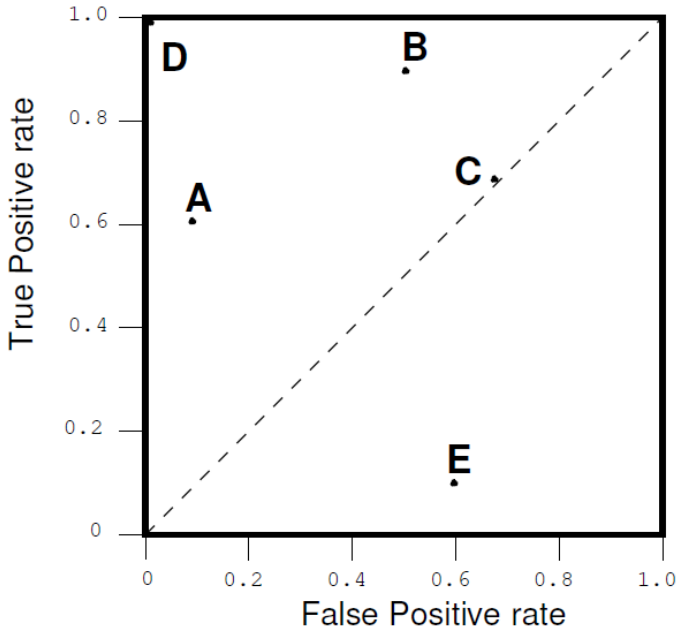


Figure 3.1: ROC curves comparing classification performance.

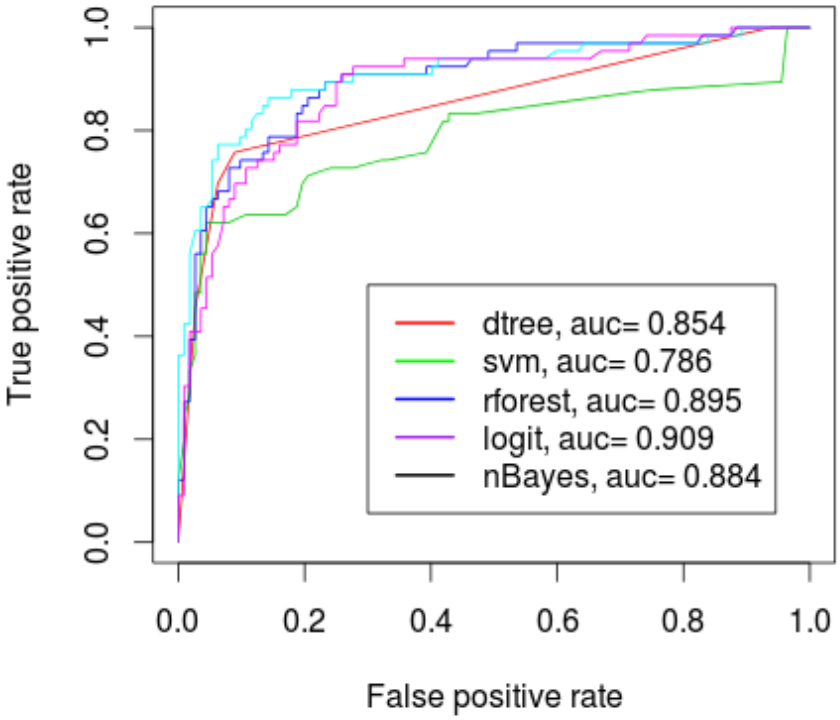


Figure 3.2: ROC curves comparing classification performance.