

Chapter 1

INTRODUCTION

1.1 Overview

In today's era, data has been increasing in volume, velocity and variety. Due to large and complex collection of datasets, it is difficult to process on traditional data processing application. So, this leads to emerging of new technology called data analysis. Data analysis is a science of exploring raw data and elicitation the useful information and hidden pattern. The main aim of data analysis is to use advance analytics techniques for huge and different datasets. Datasets may vary in sizes from terabytes to zettabytes and can be structured or unstructured. [1] The sinking of the RMS Titanic is one of the most infamous shipwrecks in history. In this project, we are performing the data analysis of Titanic Dataset using R.

On April 15, 1912, during her maiden voyage, the Titanic sank after colliding with an iceberg, killing 1502 out of 2224 passengers and crew. This sensational tragedy shocked the international community and led to better safety regulations for ships.

One of the reasons that the shipwreck led to such loss of life was that there were not enough lifeboats for the passengers and crew. Although there was some element of luck involved in surviving the sinking, some groups of people were more likely to survive than others, such as women, children, and the upper-class.

The dataset that is being used in this project is downloaded from Kaggle, which is a platform for predictive modeling and analytics competitions on which companies and researchers post their data and statisticians and data miners from all over the world compete to produce the best models. The dataset we are using comprises of two datasets. First, the training dataset, with the outcome (or target variable) for a group of passengers as well as a collection of other parameters such as their age, gender, etc. This is the dataset on which we are training for generating predictive model. Second, the test dataset, for which we must predict the now unknown target variable based on the other passenger attributes that are provided for both datasets

1.2 Objectives

In this project, we are analyzing the rate of survival of people sailing in RMS Titanic. We are using different statistical modeling approaches and machine learning algorithms for finding different patterns in the given dataset, which will help us in generating the predictions. These machine learning algorithms will help us in building a model, which will further help us in getting the rates of survivability.

Our main objective here is to first test this data against different algorithms, in order to see that which algorithm gives us the best result or the most accurate model, and then use that particular algorithm to make a Shiny web-app which, when supplied with the details of an individual as input, will give us the final result regarding whether the individual survived or not, based on the training data. Apart from that, we are also using our test data to make the predictions of survivability of people whose details are given in the test data.

Another major application of this project is to show the use of machine learning algorithms using R for a particular scenario. Thus, we can use the same algorithms for different scenarios also, like the infamous Bhopal Gas Tragedy.

1.3 Motivation

Data analysis has been an interesting field for very long time. In the past 5 years, the industry of data sciences and data analytics is booming because large amounts of data are being generated nowadays, mainly through the internet on which you can gain important insights and perform predictions.

The main motivation for us for making this project was to show that how data analysis is performed for any problem, and how different machine learning algorithms are used for making the prediction. Another reason for us for making this project is that the sinking of Titanic ship has been the one of the biggest tragedies of the world where almost 1500 people on the crew died. The data for this problem was easily available and finding different patterns in one of the biggest tragedies of mankind has a lot of significance.

Chapter 2

LITERATURE SURVEY

Data analysis has been a booming industry of recent times. Due to the increase in availability of data, data analysis has become an important part sector of all major companies. Analytics is important for your business to the extent that making good decisions is. The practice of analytics is all about supporting decision making by providing the relevant facts that will allow you to make a better decision. Below are some of major uses of data analysis. Data analysis helps in structuring the findings from different sources of data. First, data analysis is very helpful in breaking a macro problem into micro parts. Second, data analysis acts like a filter when it comes to acquiring meaningful insights out of huge data set. Third data analysis helps in keeping human bias away from the research conclusion with the help of proper statistical treatment.

2.1 Related work

Zahra Nematzadeh et al., in paper name [2] have worked on Comparative Studies on Breast Cancer Classifications with K-Fold Cross Validations Using Machine Learning Techniques. In the past years there is several machine learning techniques have been proposed to design precise classification systems for several medical issues. The paper mentioned above compares and analyses breast cancer classifications with different machine learning algorithms using k-Fold Cross Validation (KCV) technique. Decision Tree, Naïve Bayes, Neural Network and Support Vector Machine algorithm with three different kernel functions are used as classifier to classify original and prognostic Wisconsin breast cancer. The comparative analysis of the studies are focusing on the impact of k in k-fold cross validation and achieve higher accuracy. Benchmarking dataset is used in UCI in the experiments. In theory the common choice is to select $k=10$ for KCV. However, this comes at an increased computational cost whereby the more the folds the more models you need to train. The overall results showed important conclusion; we cannot always expect to have more accurate result by having greater value of k in k-fold cross validation and the results of this problem vary from the value of k.

Shruti Kohli et al., in paper name [3] have worked on the description of Data Analysis with R. Analytics companies develop the ability to support their decisions through analytic reasoning using a variety of statistical and mathematical techniques. Thomas Devonport in his book titled, “Competing on analytics: The new science of winning”, claims that a significant proportion of high-performance companies have high analytical skills among their personnel. On the other hand, a recent study has also revealed that more than 59% of the organizations do not have information required for decision-making. Learning “Data Analysis with R” not only adds to existing analytics knowledge and methodology, but also equips with exposure into latest analytics techniques including forecasting, social media analytics, text mining & so on. It gives an opportunity to work on real time data from Twitter, Facebook & other social networking sites.

Gong Shang-fu et al., in paper name [4] of Intrusion Detection System Based on Classification as shared that with the network security issues being more prominent, the safety of system and network resources become more and more important problem. Intrude detecting (ID) has become a top research topic nowadays. Considering the strong generalization ability, high sorting precision and such advantages the support vector machine (SVM) shows in practices involves small sample, high dimension, we will mainly focus on studying and consummating the SVM methods in intrude detecting. ID always generates huge data sets; such raw data sets are incapable of being training due to its large scale and high dimension and redundancy. Intrusion detection system always has the disadvantages such as overloaded, occupying too much resource, an extension of training and forecasting time... therefore, the simplification of practical information becomes such a necessity. Recursive support vector machine (R-SVM) and Rough set were used for exacting main features of raw data, and many kinds of classification algorithms were used here and it has been tested by KDDCUP1999 date set. The result shows that, the SVM classification based on R-SVM runs excellent, its accuracy is as good as the SVM classification based on the whole features and considerably reduces the training and testing time.

Shaoning Pang et al., in paper name [5] has worked on r-SVMT or Discovering the Knowledge of Association Rule over SVM Classification Trees. In this paper, a

novel method for of extraction by encoding the knowledge of the data into an SVM classification tree (SVMT), and decoding the trained SVMT into asset of linguistic association rules is presented. The method of rule extraction over the SVMT(r-SVMT), in the spirit of decision-tree rule extraction, achieves rule extraction not only from SVM, but also over the obtained decision-tree structure. The benefits of r-SVMT are that the decision-tree rule provides better comprehensibility, and the support-vector rule retains the good classification accuracy of SVM. Furthermore, r-SVMT is capable of performing a very robust classification on such datasets that have seriously, even overwhelmingly, class-imbalanced data distribution, which profits from the super generalization ability of SVMT owing to the aggregation of a group of SVMs.

Nicola Pezzotti et al., in paper name [6] has worked on Approximated and User Steerable tSNE for Progressive Visual Analytics. Progressive Visual Analytics aims at improving the interactivity in existing analytics techniques by means of visualization as well as interaction with intermediate results. One key method for data analysis is dimensionality reduction, for example, to produce 2D embeddings that can be visualized and analyzed efficiently. t-Distributed Stochastic Neighbor Embedding(tSNE) is a well-suited technique for the visualization of high-dimensional data. tSNE can create meaningful intermediate results but suffers from as low initialization that constrains its application in Progressive Visual Analytics. We introduce a controllable tSNE approximation (A-tSNE), which trades off speed and accuracy, to enable interactive data exploration.

Mohammad M. Ghassemi et al., in paper name [7], which talks about A Visualization of Evolving Clinical Sentiment Using Vector Representations of Clinical Notes, has performed an analysis on utilization of seven years of unstructured free text notes from the Multi parameter Intelligent Monitoring in Intensive Care (MIMIC) database. He found different results for different class of patients using tSNE algorithms.

Manish Varma Datla, in his paper name [8] regarding Bench Marking of Classification Algorithms: Decision Trees and Random Forests – A Case Study using R, says that Decision Trees and Random Forests are leading Machine Learning Algorithms, which are used for Classification purposes. Through the course of the

paper, a comparison is made of classification results of these two algorithms, for classifying data sets obtained from Kaggle's Bike Sharing System and Titanic problems. The solution methodology deployed is primarily broken into two segments. First, being Feature Engineering where the given instance variables are made noise free and two or more variables are used together to give rise to a valuable third. Secondly, the classification parameters are worked out, consisting of correctly classified instances, incorrectly classified instances, Precision and Accuracy. This process ensured that the instance variables and classification parameters were best treated before they were deployed with the two algorithms i.e. Decision Trees and Random Forests. The developed model has been validated by using Systems data and the Classification results. From the model it can safely be concluded that for all classification problems Decision Trees is handy with small data sets i.e. less number of instances and Random Forests gives better results for the same number of attributes and large data sets i.e. with greater number of instances. R language has been used to solve the problem and to present the results.

Ratna Astuti Nugrahaeni et al., in his paper name [9] regarding Comparative Analysis of Machine Learning KNN, SVM, and Random Forests Algorithm for Facial Expression Classification, have discussed the following. Human depicts their emotions through facial expression or their way of speech. In order to make this process possible for a machine, a training mechanism is needed to give machine the ability to recognize human expression. In the paper, he compares and analyses the performance of three machine learning algorithm to do the task of classifying human facial expression. The total of 23 variables calculated from the distance of facial features is used as the input for the classification process, with the output of seven categories, such as: angry, disgust, fear, happy, neutral, sad, and surprise. Some test cases were made to test the system, in which each test case has different amount of data, ranging from 165-520 training data. The result for each algorithm is quite satisfying with the accuracy of 75.15% for K-Nearest Neighbor (KNN), 80% for Support Vector Machine (SVM), and 76.97% for Random Forests algorithm, tested using test case with the smallest amount of data. As for the result using the largest amount of data, the accuracy is 98.85% for KNN, 90% for SVM, and 98.85% for Random Forests algorithm. The training data for each test case was also classified using Discriminant Analysis with the result 97.7% accuracy.

2.2 Recent work

Arun Jalanila et al., in his paper name [10] titled Comparing SAS® Text Miner, Python, R Analysis on Random Forest and SVM Models for Text Mining has mentioned the following. In order to answer some of the questions around performance and ease of tool usage and visualization, a comparison between SAS® Text Miner, Python and R Programming tools was conducted. We incorporated the data that were provided as part of ICHI's data analytic challenge, which addresses categorizing user questions in a healthcare forum into predefined categories. The purpose of this study was to evaluate some of the tools available to perform these tasks based on our user experience.

Juan Huo et al., in his paper [11], titled Comparison of Random Forest and SVM for Electrical Short-term Load Forecast with Different Data Sources, says that there is always argument about which machine learning algorithm is the best one for electrical load forecast. In this paper, they compare two open-source machine learning algorithm SVM (Support Vector Machine) and RFR (Random Forest Regression). They evaluate their difference by implementing them into different data sets. They clarify that both SVM and RFR are excellent choices for electrical load forecast, however, their good performance is parameter and data dependent. Parameter setting is more important for SVM.

Chapter 3

PROBLEM STATEMENT

3.1 Problem description

In this project, we are trying to bring out some insights from the infamous RMS Titanic tragedy, where we are analyzing the data of Titanic dataset. The primary step in the lifecycle of data science projects is to first identify the person who knows what data to acquire and when to acquire based on the question to be answered. The dataset that is being used in this project is downloaded from Kaggle, which is a platform for predictive modeling and analytics competitions on which companies and researchers post their data and statisticians and data miners from all over the world compete to produce the best models. The dataset we are using comprises of two datasets. The next step is the data cleaning or data wrangling phase. Having acquired the data, we have to clean and reformat the data by manually editing it in the spreadsheet or by writing code, so that the data is in usable and easy-to-understand format. In our case, the data which we are downloading from Kaggle is already cleaned, and we don't need to clean it again.

So, we are starting the project with a tidy dataset, with the main goal to find the patterns within the given dataset. We need to perform exploratory analysis, predictive analysis and at the same time develop different machine learning models which will predict the survivability occurrence for the inputs provided in the test dataset. We are also trying to develop a Web-App which, when supplied with the details of an individual as input, will give us the final result regarding whether the individual survived or not, based on the training data. Apart from that, we are using this project as a means to show that how data analysis is performed in different project, with the application of different machine learning algorithms.

3.3 Proposed Work

The next step after cleaning the data is exploratory modeling. We are starting the problem from this phase. Exploratory data analysis forms an integral part at this stage as summarization of the clean data can help identify outliers, anomalies and patterns that can

Be usable in the subsequent steps. This is the step that helps us to answer the question on as to what do they actually want to do with this data. Then we are modeling that data. Diverse machine learning techniques are applied to the data to identify the machine learning model that best fits the business needs. All the contending machine learning models are trained with the training data sets. After creating different models, we are evaluating these models, and testing their performance. For instance, if the machine learning model aims to predict the daily stock then the RMSE (root mean squared error) will have to be considered for evaluation. If the model aims to classify spam emails then performance metrics like average accuracy, AUC and log loss have to be considered. Machine learning model performances is measured and compared using cross-validation and test sets to identify the best model based on model accuracy and over-fitting. After this, the machine learning models are first deployed in a pre-production or test environment before actually deploying them into production.

3.4 Advantages of Proposed Work

- *Feature learning:* One of the interesting advantages of data analysis and machine learning is that a system randomly initialized and trained on some datasets will eventually learn good feature representations for a given task. Classical approaches involved handcrafting features by an expert human. This took several years of painstakingly fine tuning several parameters to get it right. Nowadays machine learning is used to discover relevant features in otherwise disordered datasets. Such features can be useful for things such as face detection, face recognition, speech recognition or image classification. Deep learning in particular aims to build higher-level abstract feature representation of data layer by layer. These features can be very powerful in speech and image recognition.
- *Parameter optimization:* This is similar to feature learning as a group of tunable parameters can be visualized as a feature. Data analysis and Machine learning mostly employs a gradient based method of optimizing a large array of parameters. Again such parameters may be large in number for example; a deep neural architecture can have billions of tunable parameters. These parameters when well set can result in a system working properly. It is not feasible for a human or humans to find such an optimal setting for large number of parameters by hand, thus large

- scale machine learning algorithms such as stochastic gradient descent are used to find an optimal setting.
- *Sophisticated pattern recognition:* Along with noting relationships, you can determine the type and quantify as well. This is not just happening with key or even secondary variables, but on every relationship that takes part in the pattern. This feature delineates irrelevant data as well, which provides the benefits of mitigating pre-processing requirements and accelerating processing. Since the solution has an ordering or ranking capability, the key variables self-identify as a part of the processing.
- *Intelligent decisions:* Along with the capability to note irrelevant data, and rank the relative importance of variables, data analysis will make decisions either aided by the user or not. This becomes apparent when modeling to predict a rare event. The solution can distinguish subclasses and make determinations on what data should be included and which shouldn't with very little instruction.
- *Self modifying:* Clearly, being able to tweak, add, or drop different aspects of an algorithm to better typify the data is a time saver. However, as this is also taking place to accommodate minor variables and sub-classes, so time demands are being held in check while the accuracy of the algorithm and its ability to predict are significantly improved.

Chapter 4

REQUIREMENT ANALYSIS AND FEASIBILITY STUDY

4.1 Functional Requirements

Functional requirement describes what a software system should do. We have the following functional requirements for our system. First and foremost, after cleaning the data, our system must have some exploratory models that answer some basic questions of Titanic dataset, like which sex had more survival chances, or which class of passengers are likely to survive. All these answers are based on the graphs obtained from the data. After completing the exploratory models, we must try different machine learning algorithms in order to predict the survivability of the test dataset. This includes trying different parameters or variables from the train dataset which have the highest amount of influence on while training these models, and then using these variable combinations in building the final model. After building the final model, we are deploying a Web-UI, which will give us different graphs for various input variables and output of survivability for a set of input variables.

4.2 Non-Functional Requirements

Non-functional requirements place constraints on how the system will achieve it's the functional requirements. We are having the following non-functional requirements. First, our data that is being used should be thoroughly cleaned and checked for errors or null values. Other that, the exploratory models should give good description of various patterns of data through different graphs. Also, we need to see that the machine learning algorithms that we are using are having an accuracy of 60% and they are giving us clear predictions for the test dataset after utilizing the training dataset.

4.3 SYSTEM REQUIREMENTS:

Bellow are the system hardware and software requirements of our project. All these are recommended system requirements.

4.3.1 Hardware Requirement:

- PROCESSOR : Intel Core i5 3.2 GHz
- RAM : 4 GB DDR3 RAM recommended
- MONITOR : 15” COLOR
- HARD DISK : 20 GB minimum
- FLOPPY DRIVE : 1.44 MB
- CDDRIVE : LG 52X
- KEYBOARD : STANDARD 102 KEYS
- MOUSE : 3 BUTTONS

4.3.2 Software Requirements:

- Front end : Shiny Web-App
- Tools Used : RStudio (IDE), R (programminglanguage)
- Operating System : Ubuntu

Chapter 5

TECHNOLOGY USED

5.1 R (Programming Language)

R is an open source programming language and software environment for statistical computing and graphics that is supported by the R Foundation for Statistical Computing. The R language is widely used among statisticians and data miners for developing statistical software and data analysis.

5.2 Design

R is a GNU package. The source code for the R software environment is written primarily in C, FORTRAN, and R.

5.3 History

R is an implementation of the S programming language combined with lexical scoping semantics inspired by Scheme. S was created by John Chambers while at Bell Labs. There are some important differences, but much of the code written for S runs unaltered.

R was created by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand, and is currently developed by the R Development Core Team, of which Chambers is a member. R is named partly after the first names of the first two R authors and partly as a play on the name of S. The project was conceived in 1992, with an initial version released in 1995 and a stable beta version in 2000.

5.4 Features of R

- R is an interpreted language; users typically access it through a command-line interpreter. If a user types `2+2` at the R command prompt and presses enter, the computer replies with 4.
- This calculation is interpreted as the sum of two single-element vectors, resulting in a single-element vector. The prefix indicates that the list of elements

- following it on the same line starts with the first element of the vector (a feature that is useful when the output extends over multiple lines).
- Like other similar languages such as APL and MATLAB, R supports matrix arithmetic. R's data structures include vectors, matrices, arrays, data frames (similar to tables in a relational database) and lists. R's extensible object system includes objects for (among others): regression models, time-series and geo-spatial coordinates. The scalar data type was never a data structure of R. Instead, a scalar is represented as a vector with length one.
- R supports procedural programming with functions and, for some functions, object-oriented programming with generic functions. A generic function acts differently depending on the classes of arguments passed to it. In other words, the generic function dispatches the function (method) specific to that class of object. For example, R has a generic print function that can print almost every class of object in R with a simple print (objectname) syntax.
- Although used mainly by statisticians and other practitioners requiring an environment for statistical computation and software development, R can also operate as a general matrix calculation toolbox – with performance benchmarks comparable to GNU Octave or MATLAB. Arrays are stored in column-major order.

5.5 List of Libraries Used

- ggplot2
- stringr
- randomForest

- caret
- doSNOW
- rpart
- infotheo
- e1071
- rattle
- Shiny

5.6 RStudio (IDE)

RStudio is a free and open-source integrated development environment (IDE) for R, a programming language for statistical computing and graphics. RStudio was founded by JJ Allaire, creator of the programming language ColdFusion. Hadley Wickham is the Chief Scientist at RStudio. RStudio is available in two editions: RStudio Desktop, where the program is run locally as a regular desktop application; and RStudio Server, which allows accessing RStudio using a web browser while it is running on a remote Linux server. For this project, we are using RStudio Desktop, with different libraries of R programming language.

Prepackaged distributions of RStudio Desktop are available for Windows, macOS, and Linux. RStudio is available in open source and commercial editions and runs on the desktop (Windows, macOS, and Linux) or in a browser connected to RStudio Server or RStudio Server Pro (Debian, Ubuntu, Red Hat Linux, CentOS, openSUSE and SLES).

5.7 Design

RStudio is written in the C++ programming language and uses the Qt framework for its graphical user interface, which is build upon Javascript and JQuery.

Chapter 6

SYSTEM DESIGN

Design and Testing is the actual process of producing a solution according to the specification derived from the analysis stage.

6.1 System Preliminary Design:

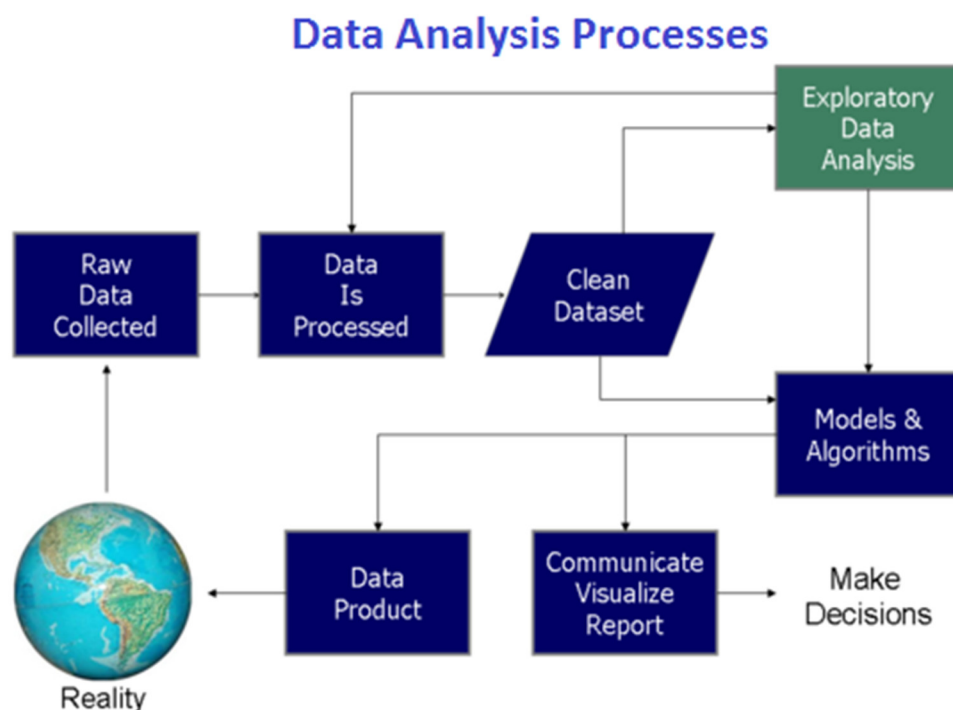


Fig. 6.1.1 Data Analysis Processes

The primary step in the lifecycle of data analysis projects is to first identify the person who knows what data to acquire and when to acquire based on the question to be answered. In our case, the data we are using is freely available on website called Kaggle.

a. Raw Data Collected -

Data is downloaded from Kaggle has two parts – test dataset and training dataset.

It is real world data, hence has the odd missing (in passenger age) and a number of columns with messy data, which might be employed to create additional variables. For the purpose of validation about 90% of the data gets flagged to be training set.

The training set is used to build the machine learning models. For the training set, we provide the outcome (also known as the “ground truth”) for each passenger. Our model will be based on “features” like passengers’ gender and class. We are also using feature engineering to create new features.

The test set is used to see how well your model performs on unseen data. For the test set, the ground truth for each passenger (whether he survived or not) is not provided. It is our job to predict these outcomes. The model we have trained to predict whether or not they survived the sinking of the Titanic is used for each passenger in the test set. Our goal is to predict Survived variable for the test dataset.

b. Data Preparation/ Data Wrangling (data is processed and cleaned) -

The second phase consists of Data Preparation, often referred as data wrangling phase. Data acquired in the first step of a data analysis project is usually not in a usable format to run the required analysis and might contain missing entries, inconsistencies and semantic errors. We have to clean and reformat the data by manually editing it in the spreadsheet or by writing code. This step of the data science project lifecycle does not produce any meaningful insights. But in our case, the data that is being downloaded is present in easy-to-use CSV format and the number of errors / missing entries is negligible. So, we can directly move to the third stage known as Exploratory Modeling.

c. Exploratory Modeling -

Before you train your models, you need to develop a sound understanding of the data. This phase is known as Exploratory Modeling. Real-world datasets are often noisy or are missing values or have a host of other discrepancies. Data summarization and visualization can be used to audit the quality of your data and provide the information needed to process the data before it is ready for modeling. The objective of this step is to understand the main characteristics of the data. This analysis is generally done using visualizing tools. Performing an Exploratory analysis helps us:

- to understand causes of an observed event
- to understand the nature of the data we are dealing with
- assess assumptions on which our analysis will be based
- to identify the key features in the data needed for the analysis.

d. Model Generation & Validation -

The fourth phase is known as Model Generation & Validation. This step involves extracting features from the data and feeding them into the machine learning algorithms to build a model. Model is the solution proposed for the problem statement. This step involves: Feature Engineering, Model selection, model training and model evaluation.

- *Model selection:* Based on the type of problem we are dealing; a model will be built. For example, if the objective of the analysis is to predict a future event, we need to build a Regression model for prediction.
- *Model Training:* After selecting the Model for the analysis, the entire dataset is divided into 2 parts – Training data & Test Data. 3/4th of the entire data will be fed as input to the Model Algorithms.
- *Feature engineering:* It involves inclusion, aggregation and transformation of raw variables to create the features used in the analysis. If you want insight into what is driving a model, then you need to understand how features are related to each other and how the machine learning algorithms are to use those features. This step requires a creative combination of domain expertise and insights obtained from the data exploration step. This is a balancing act of finding and including informative variables while avoiding too many unrelated variables. Informative variables improve our result; unrelated variables introduce unnecessary noise into the model. You also need to generate these features for any new data obtained during scoring. So the generation of these features can only depend on data that is available at the time of scoring.
- *Model Evaluation:* Once the model is built. The next step is to test the model & validate it. The data used for testing the model is the remaining 1/3rd of the dataset in the previous step.
- *Visualize Results:* This is the final step of Data analysis where the results of the model & problem solved will be presented generally in visual plots/graphs. Few visualizing tools: d3.js, ggplot2, tableau. For this project, we are using following machine learning algorithms – SVM, Random Forests, and K-Means Cross Validation.

- d. Deployment* – The last and final phase of this project is to interpret results and deployment. The machine learning models are first deployed in a pre-production or test environment before actually deploying them into production. In our case, we have deployed the most efficient random forest model in the form of a Shiny-Web UI.

6.2 Functionality:

The analysis of the dataset offers the following functionality. The exploratory models give us the general pattern of the different variables of data and tell us that which variables influence the survivability factor the most. The two main machine learning algorithms – SVM and RandomForest helps us in the prediction of the survivability rate of test dataset. We also use the k-means cross validation method for checking the scenario of over-fitting of our model.

Lastly, we build a shiny web-app. Shiny is a package from RStudio that can be used to build interactive web pages with R. While that may sound scary because of the words “web pages”, it’s geared to R users who have 0 experiences with web development, and you do not need to know any HTML/CSS/JavaScript.

You can do quite a lot with Shiny: think of it as an easy way to make an interactive web page, and that web page can seamlessly interact with R and display R objects (plots, tables, of anything else you do in R). To get a sense of the wide range of things you can do with Shiny, you can visit my Shiny server, which hosts some of my own Shiny apps. This web-app gives us graphs for changing input variable in real time, and also mentions whether a passenger survived or not for a given combination of input variables.

6.3 Use Case Diagram:

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases.

The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted. Use Case diagrams are formally included in two modeling languages defined by the OMG: The Unified Modeling Language (UML) and the Systems Modeling Language (SysML).

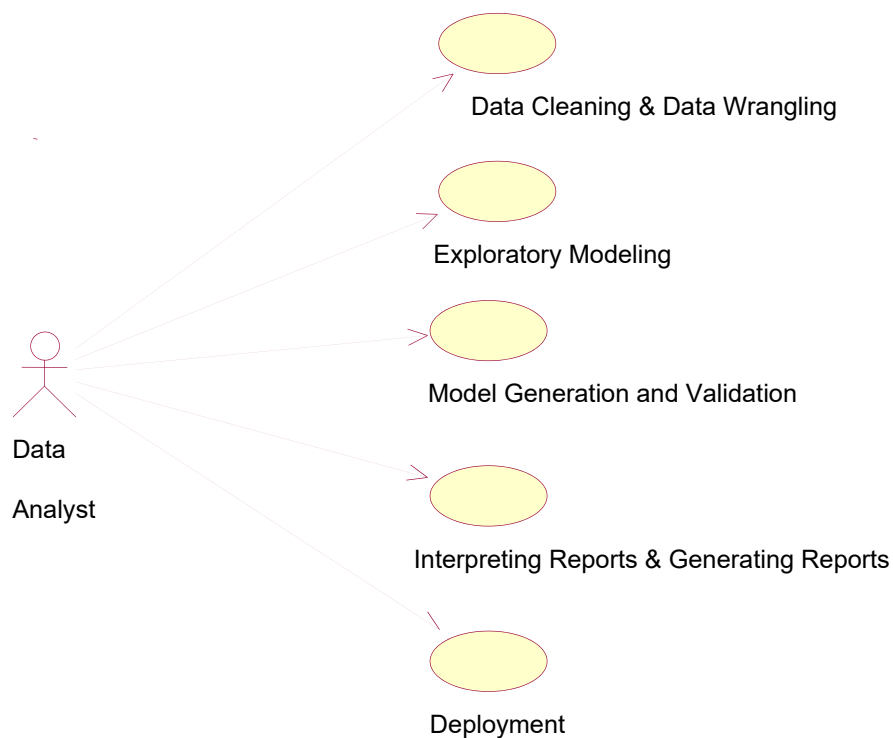


Fig: 6.3.1 Use case diagram

6.4 Description of different variables in the dataset

From a sample of the RMS Titanic data, we can see the various features present for each passenger on the ship:

- Survived: Outcome of survival (0 = No; 1 = Yes)
- Pclass: Socio-economic class (1 = Upper class; 2 = Middle class; 3 = Lower class)
- Name: Name of passenger
- Sex: Sex of the passenger
- Age: Age of the passenger (Some entries contain NaN)
- SibSp: Number of siblings and spouses of the passenger aboard
- Parch: Number of parents and children of the passenger aboard
- Ticket: Ticket number of the passenger
- Fare: Fare paid by the passenger
- Cabin: Cabin number of the passenger (Some entries contain NaN)
- Embarked: Port of embarkation of the passenger at the place from where the passenger has started the journey. (C = Cherbourg; Q = Queenstown; S = Southampton).

Chapter 7

SYSTEM IMPLEMENTATION

7.1 Modular implementation:

The project mainly has been divided in to seven modules. They are:

- Data Exploration and Exploratory Modeling
- Predicting values of training dataset using SVM Algorithm
- Exploratory modeling for Random Forests Algorithm
- K-Means Cross validation of Random Forests Algorithm
- Single Decision Trees to perform final feature engineering and developing final Random Forest Model
- Shiny web-app which gives graphs and survivability factor for real time input

7.1 Data Exploration and Exploratory Modeling

The objective of this module is to understand the main characteristics of the data. This analysis is generally done using visualizing tools. We have several questions, which we are answering through different graphs.

7.1.1 Are there any duplicate names in the dataset?

We are having 2 pairs of people with the same name, but since they differ in ages, hence they are not same. Therefore, we are not having any duplicate names in the dataset. Here are the details of those people -

| Survived | name | age | ticket |
|----------|----------------------|------|--------|
| 1 | Connolly, Miss. Kate | 22.0 | 370373 |
| None | Connolly, Miss. Kate | 30.0 | 330972 |
| 0 | Kelly, Mr. James | 44.0 | 363592 |
| None | Kelly, Mr. James | 34.5 | 330911 |

7.1.2 Do rich folks have higher survivability than poor folks?

Here, we are verifying the hypothesis, checking that the guys who had Upper class (rich folks) easily survived because upper class was closer to the deck and lower class was closer to the base of the ship. From the graph shown bellow, we can see that the people travelling in Class-1 have higher survivability ratio as compared to people of Class-2 and Class-3. Hence, by looking at the graphs, our hypothesis is proved right. We can also conclude that most of the casualties are coming from Class-3 and more than 50% people of Class-1 have survived.

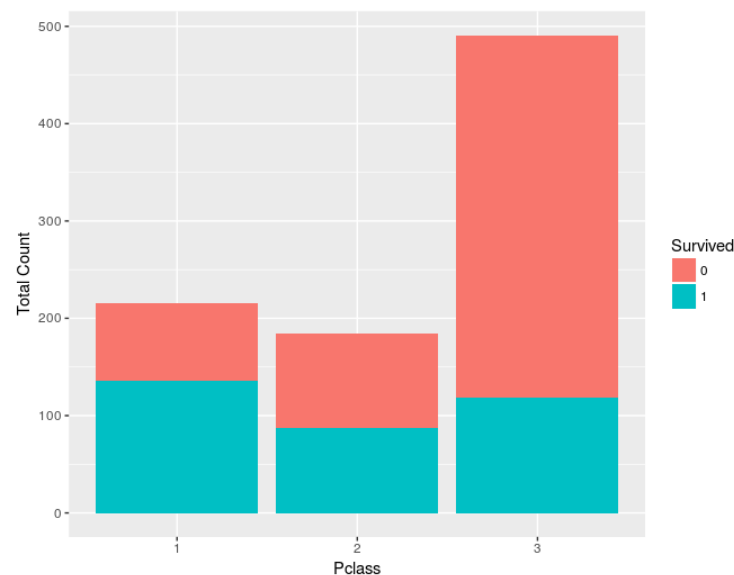


Fig. 7.1.2.1 Relationship between pclass and Survived

7.1.3 Are women more likely to survive than men?

When we look at the graph, the patterns are clearly visible. For all the three classes, women have more number of females have survived as compared to men. On the top of that, the survivability rate of females travelling in first and second class is more than 90%, which is quite astonishing when compared to the males of the same class.

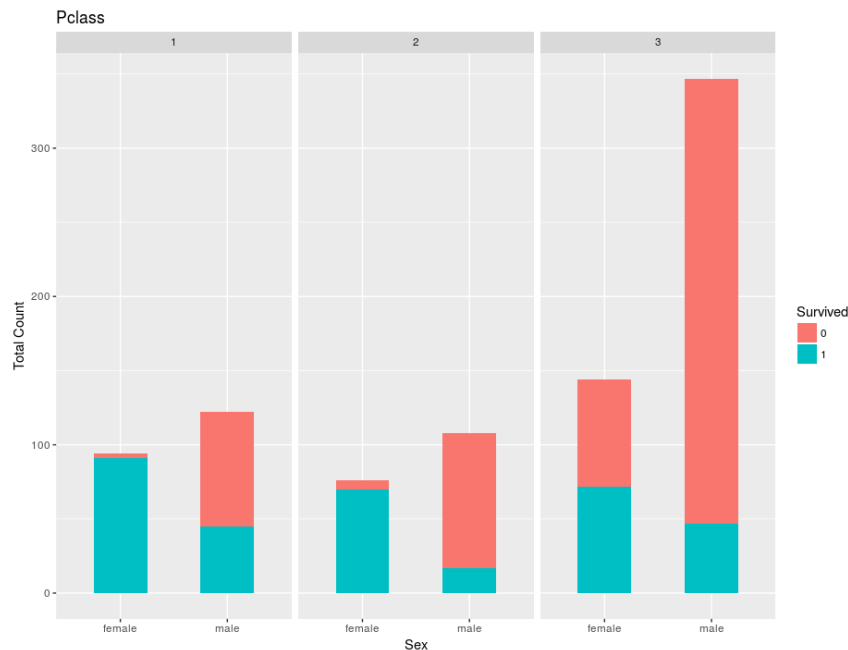


Fig. 7.1.3.1 Relationship between pclass and Survived

7.1.4 Is it true that young boys with the title of “Master” have higher chances of survival?

From the graph shown bellow, we can see that individual with “Master” title range from age group of 0 to 15 years, and they are having survival rate of 100% for class 1 and class 2. For class 3, we cannot draw any meaningful conclusion. We can also see that the title ‘Master’ applies only for people in the range of 0 to 15 years, and we do not have any unmarried guy more than the age of 15 years. This is a very peculiar case.

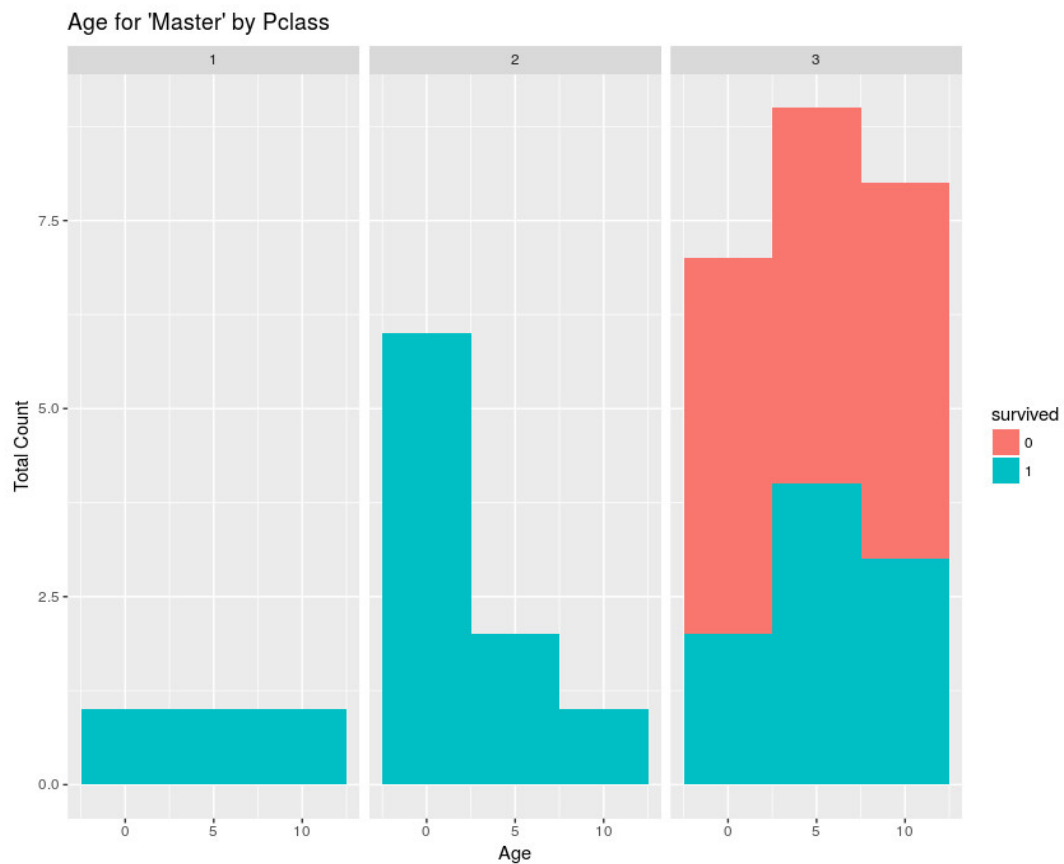


Fig. 7.1.4.1 Relationship between pclass and Survived and age

7.1.5 Do we have any pattern between survived and family size?

We are not having any specific pattern between survived variable and family size variable. But, we can deduce that all the people of class-3 having family size greater than 8 failed to survive. We can also see that people with family size 5 in class 2 have 100% survival rate. Other than this, we cannot draw any specific conclusions.

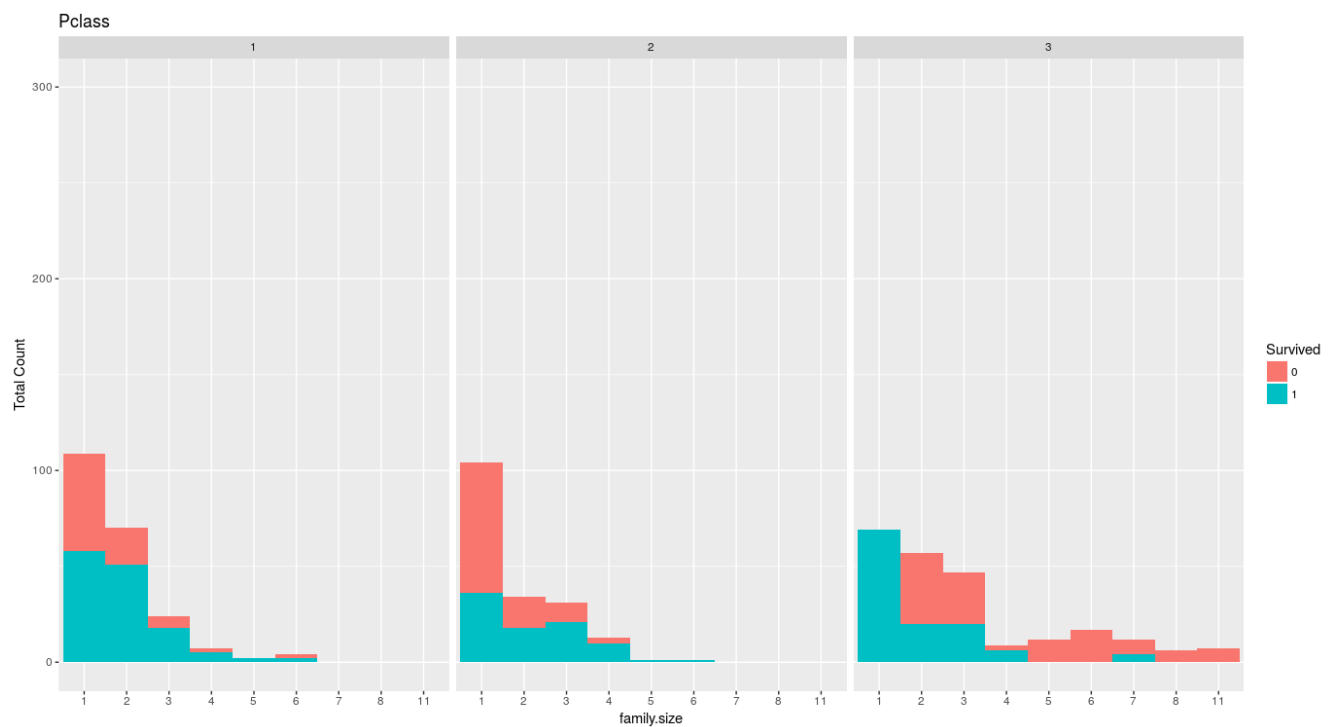


Fig. 7.1.5.1 Relationship between family size, pclass and Survived

7.1.6 What is the fare distribution for this dataset?

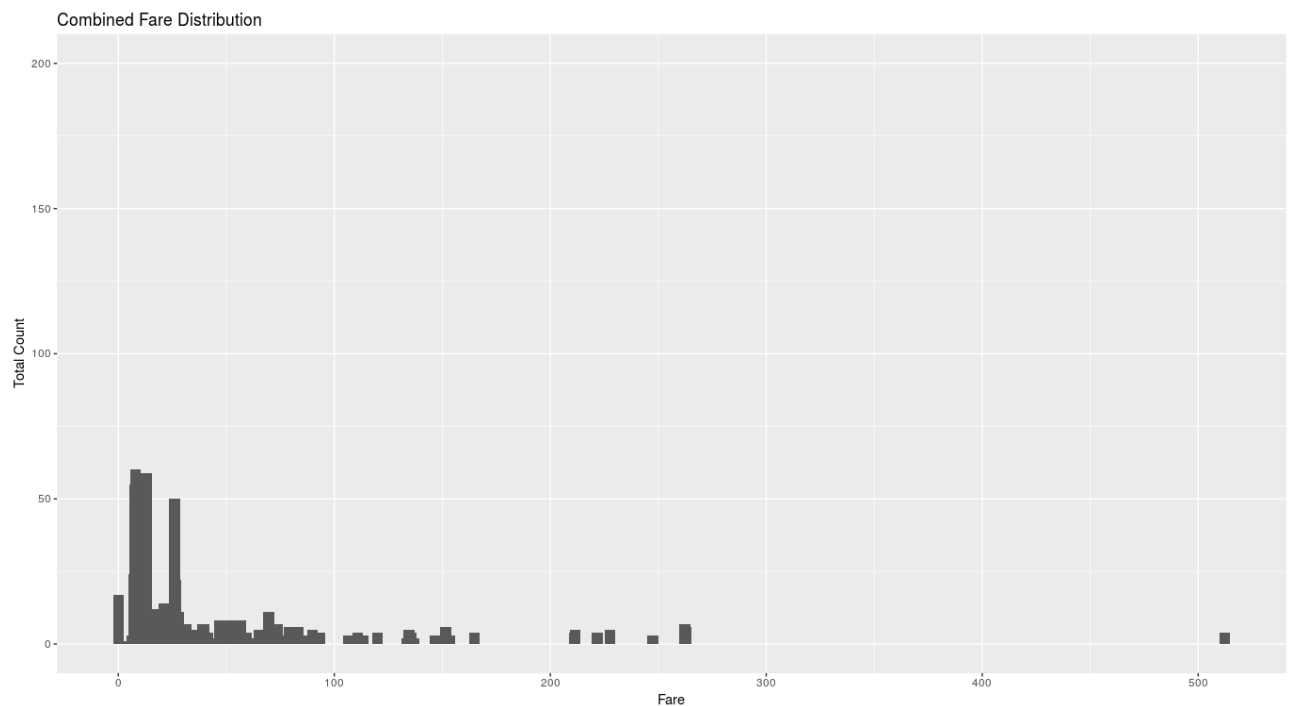


Fig. 7.1.6.1 Fare distribution of dataset

The fair distribution is shown in the above graph. For most people, the fare is about 14 dollars. (Since 14.454 is the median of fare)

7.1.7 Can we find any patterns on the basis of destination of the Titanic ship?

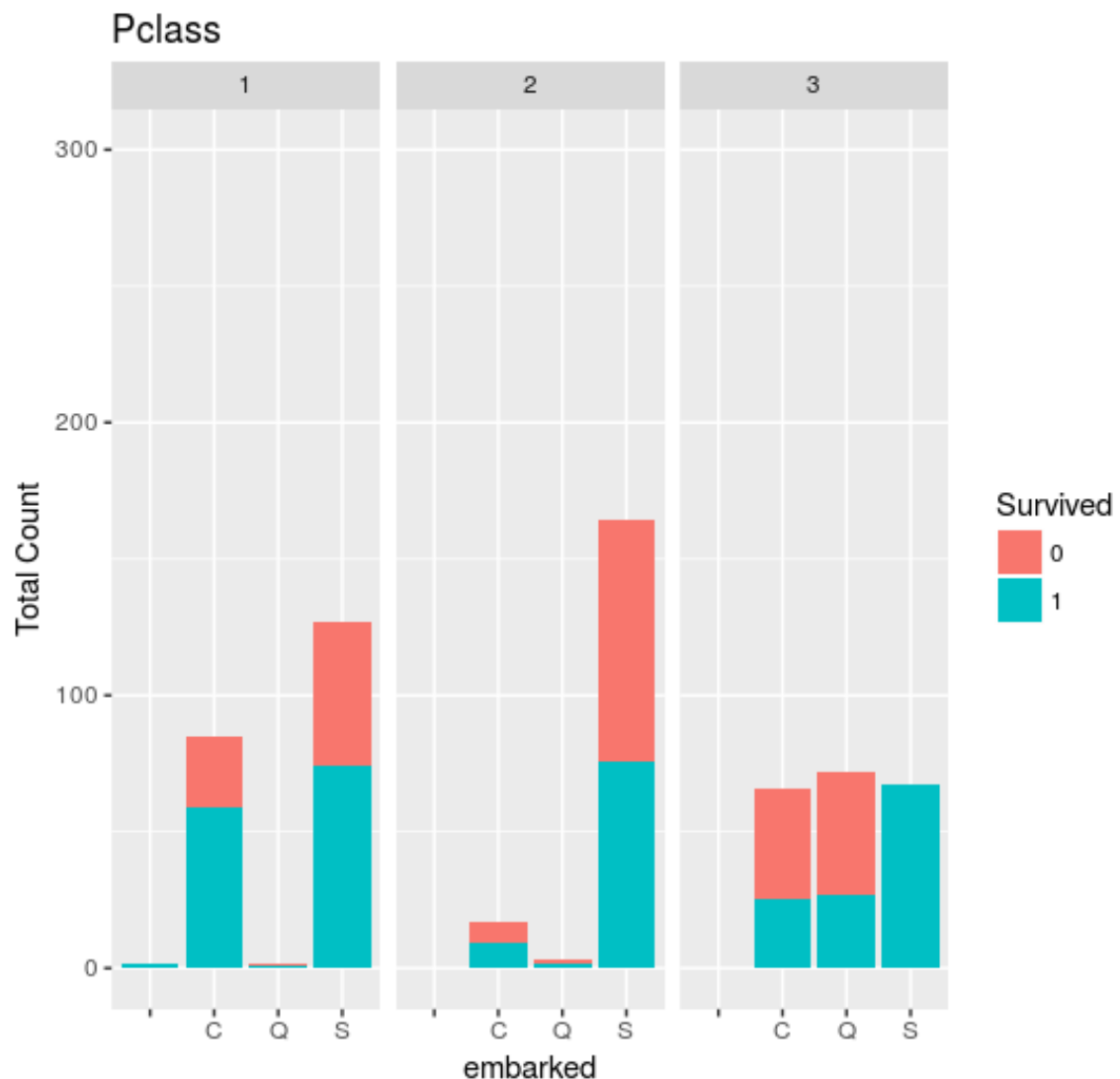


Fig. 7.1.7.1 Graph between embarked, survived and pclass

Here, C denotes Cherbourg, Q denotes Queenstown and S denotes Southampton. From the graph, we can clearly see that all the people going to Southampton from class 3 survived, which is quite astonishing. Further, most of the people travelling in Titanic that belong to class 2 and 3 are travelling to Cherbourg and Southampton.

7.2 Predicting values of training dataset using SVM Algorithm

7.2.1 ALGORITHM

“Support Vector Machine” (SVM) is a supervised machine learning algorithm which can be used for both classification and regression challenges. However, it is mostly used in classification problems. In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiate the two classes very well (look at the below snapshot).

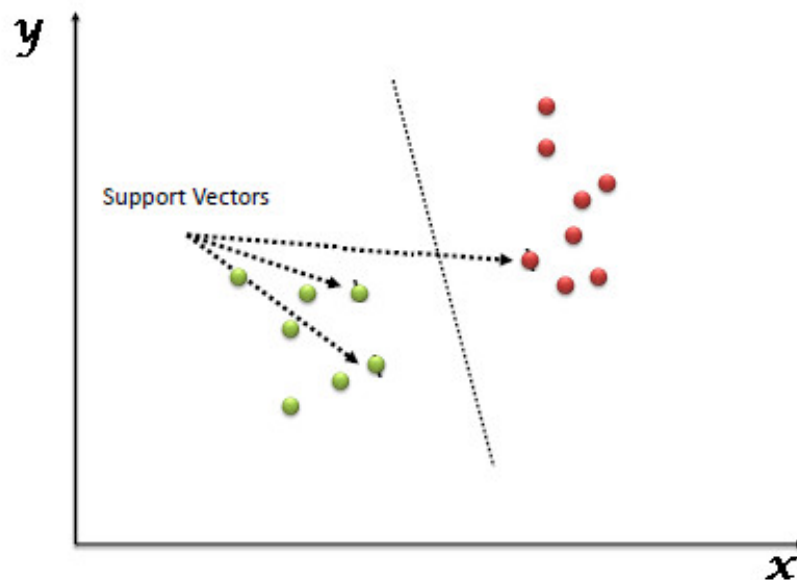


Fig. 7.2.1.1 SVM Graph-1

Support Vectors are simply the co-ordinates of individual observation. Support Vector Machine is a frontier which best segregates the two classes (hyper-plane/ line). Support Vectors are simply the co-ordinates of individual observation.

Above, we got accustomed to the process of segregating the two classes with a hyper-plane. Now the burning question is “How can we identify the right hyper-plane?”

Let’s understand:

- **Identify the right hyper-plane (Scenario-1):** Here, we have three hyper-planes (A, B and C). Now, identify the right hyper-plane to classify star and circle.

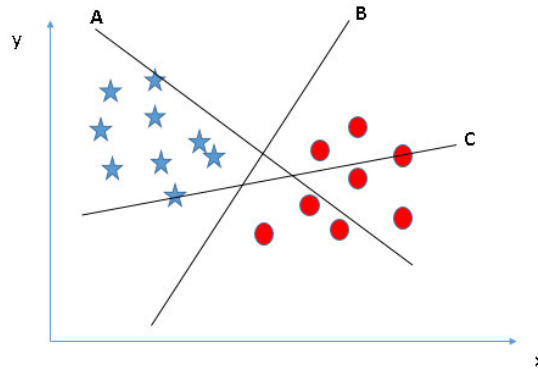


Fig. 7.1.1.2 SVM Graph-2

- You need to remember a thumb rule to identify the right hyper-plane: “Select the hyper-plane which segregates the two classes better”. In this scenario, hyper-plane “B” has excellently performed this job.
- **Identify the right hyper-plane (Scenario-2):** Here, we have three hyper-planes (A, B and C) and all are segregating the classes well. Now, how can we identify the right hyper-plane?

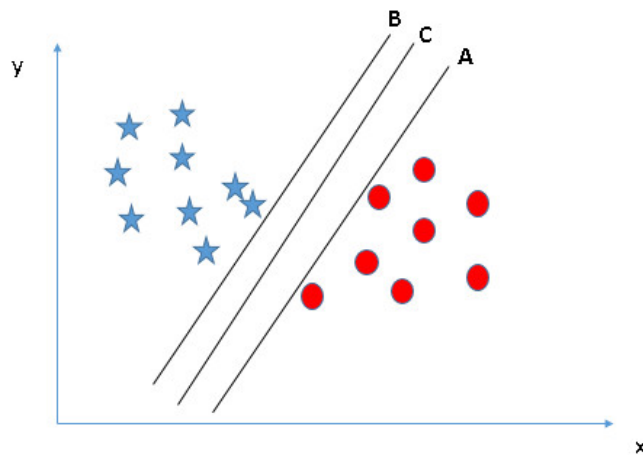


Fig. 7.1.1.3 SVM Graph-3

Here, maximizing the distances between nearest data points (either class) and hyper-plane will help us to decide the right hyper-plane. This distance is called as **Margin**. Let's look at the below snapshot:

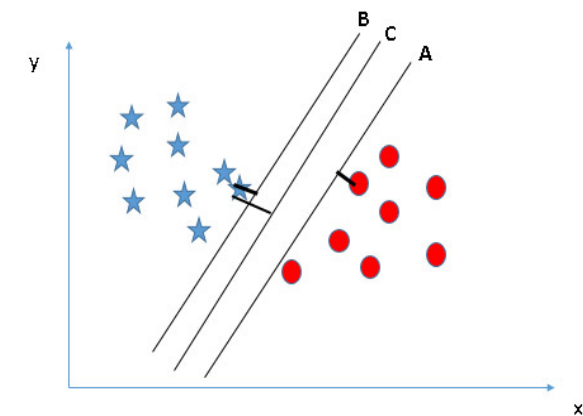


Fig. 7.1.1.4 SVM Graph-4

Above, you can see that the margin for hyper-plane C is high as compared to both A and B. Hence, we name the right hyper-plane as C. Another lightning reason for selecting the hyper-plane with higher margin is robustness. If we select a hyper-plane having low margin, then there is high chance of miss-classification.

- **Identify the right hyper-plane (Scenario-3):** Hint: Use the rules as discussed in previous section to identify the right hyper-plane

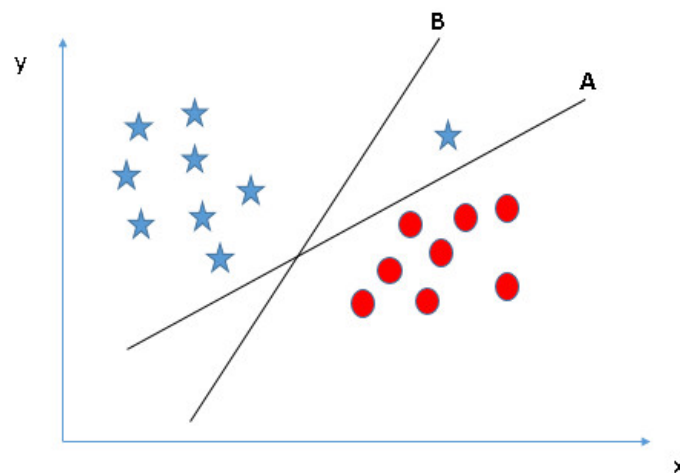


Fig. 7.1.1.5 SVM Graph-5

Some of you may have selected the hyper-plane B as it has higher margin compared to A. But, here is the catch, SVM selects the hyper-plane which classifies the classes accurately prior to maximizing margin. Here, hyper-plane B has a classification error and A has classified all correctly. Therefore, the right hyper-plane is A

- **Can we classify two classes (Scenario-4) ?:** Below, I am unable to segregate the two classes using a straight line, as one of star lies in the territory of other(circle) class as an outlier.

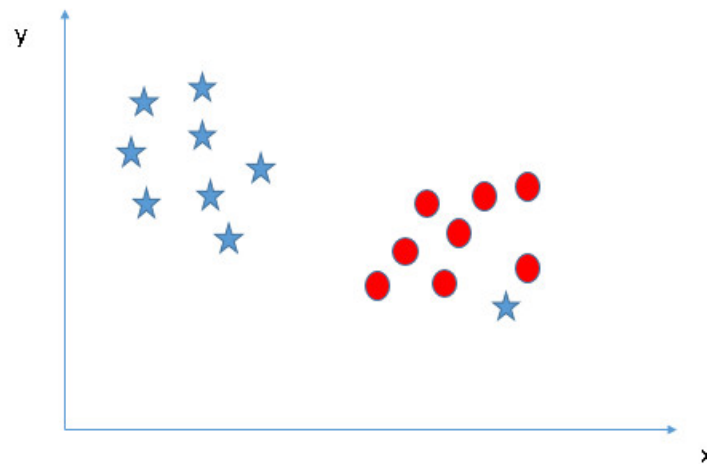


Fig. 7.1.1.6 SVM Graph-6

- As I have already mentioned, one star at other end is like an outlier for star class. SVM has a feature to ignore outliers and find the hyper-plane that has maximum margin. Hence, we can say, SVM is robust to outliers.

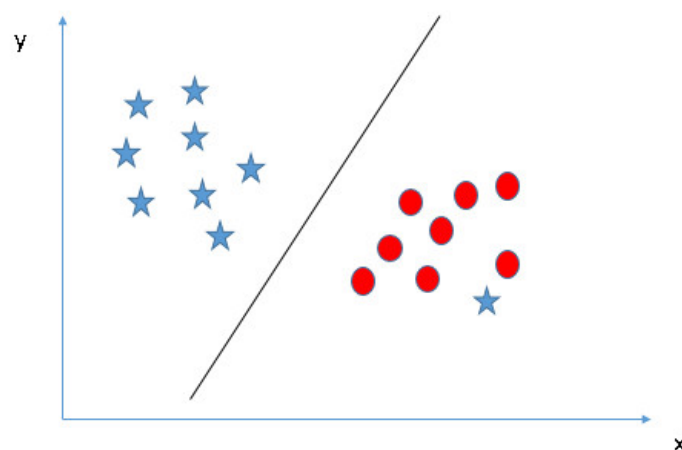


Fig. 7.1.1.7 SVM Graph-7

- **Find the hyper-plane to segregate to classes (Scenario-5):** In the scenario below, we can't have linear hyper-plane between the two classes, so we have only looked at the linear hyperplane till now.

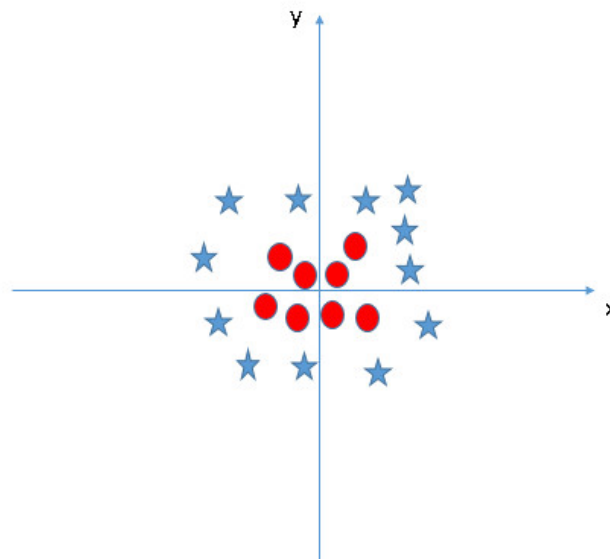


Fig. 7.1.1.8 SVM Graph-8

- SVM can solve this problem. Easily! It solves this problem by introducing additional feature. Here, we will add a new feature $z = x^2 + y^2$. Now, let's plot the data points on axis x and z:

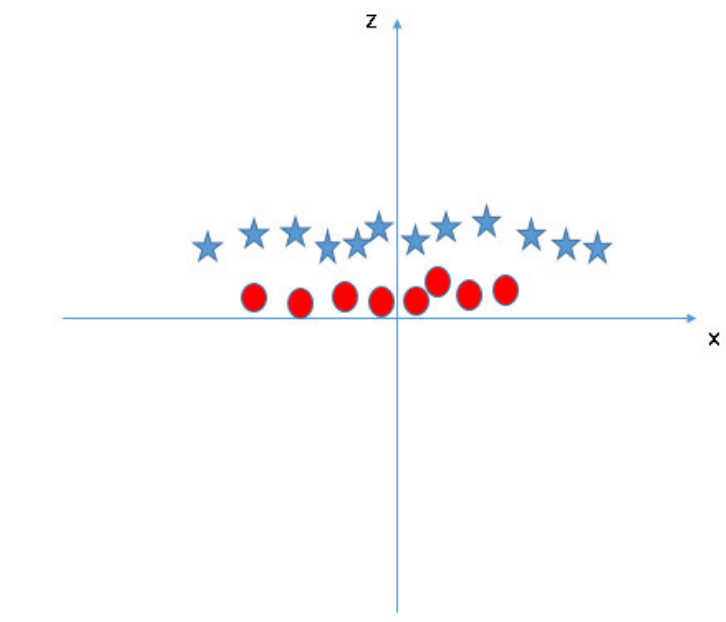


Fig. 7.1.1.9 SVM Graph-9

In above plot, points to consider are

- All values for z would be positive always because z is the squared sum of both x and y
- In the original plot, red circles appear close to the origin of x and y axes, leading to lower value of z and star relatively away from the origin result to higher value of z .

In SVM, it is easy to have a linear hyper-plane between these two classes. But, another burning question which arises is, should we need to add this feature manually to have a hyper-plane. No, SVM has a technique called the **kernel trick**. These are functions which takes low dimensional input space and transform it to a higher dimensional space i.e. it converts not separable problem to separable problem, these functions are called kernels. It is mostly useful in non-linear separation problem. Simply put, it does some extremely complex data transformations, then find out the process to separate the data based on the labels or outputs you've defined.

When we look at the hyper-plane in original input space it looks like a circle:

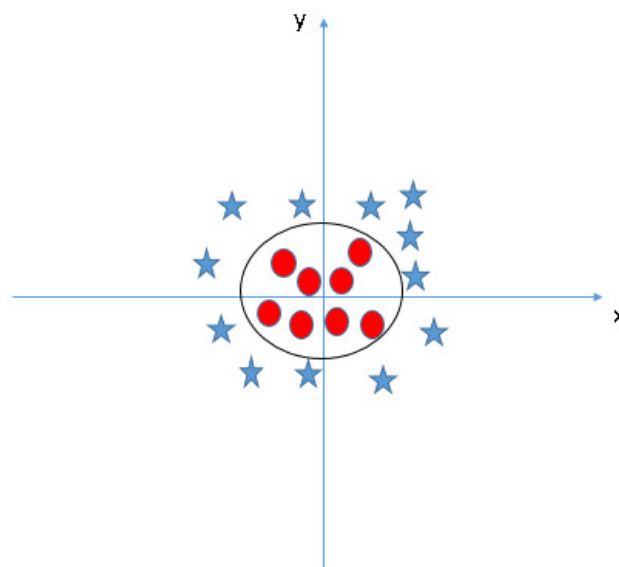


Fig. 7.1.1.10 SVM Graph-10

7.2.2 In Layman's Terms

A vanilla SVM is a type of linear separator. Suppose we want to split black circles from the white one's below by drawing a line. Notice that there are an infinite number of lines that will accomplish this task. SVMs, in particular, find the "maximum-margin" line this

is the line "in the middle". Intuitively, this works well because it allows for noise and is most tolerant to mistakes on either side.

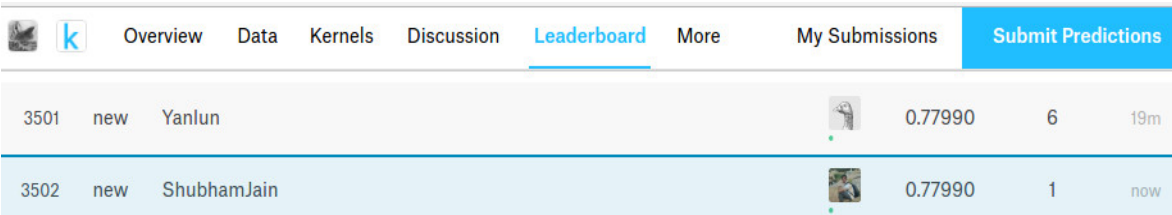
That's what (binary) SVM really is. We draw a straight line through our data down the middle to separate it into two classes. But what if we don't want a straight line but a curved one? We achieve this not by drawing curves, but by "lifting" the features we observe into higher dimensions. For example, if we can't draw a line in the space (x_1, x_2) then we may try adding a third dimension, $(x_1, x_2, x_1 * x_2)$. If we project the "line" (actually called a "hyperplane") in this higher dimension down to our original dimension, it looks like a curve.

7.2.3 APPLICATION

First, we are performing some feature engineering, which

Here, we are using the SVM algorithms to predict the survival variable of test dataset using the training dataset. When we train the training dataset and check its accuracy, it comes out to be 80.9%.

However, when we use this model and train the test dataset for submitting it to Kaggle, the results are different. In this scenario, the result comes out to be 77.99%, which is less accurate as compared to the random forests algorithm which we are using in the next module. Here is a screenshot of SVM submission.



| Rank | Status | Username | Score | Votes | Time |
|------|--------|-------------|---------|-------|------|
| 3501 | new | Yanlun | 0.77990 | 6 | 19m |
| 3502 | new | ShubhamJain | 0.77990 | 1 | now |

Fig. 7.2.3.1 Kaggle LeaderBoard-1

We can see that the SVM submission gives us accuracy of 77.990%, which is good but not as good as random forests.

We have performed further improvements, and this has resulted in an increase in accuracy to 78.947%. These improvements show up after feature engineering, where we have increased the number of variables that we are feeding to SVM, by modifying the initial variables. For eg., we have created variables named Title, FamilySize, FamilyID, Child and

Mother using the original data, and when we feed this to our model, the accuracy improves and becomes 78.947%.

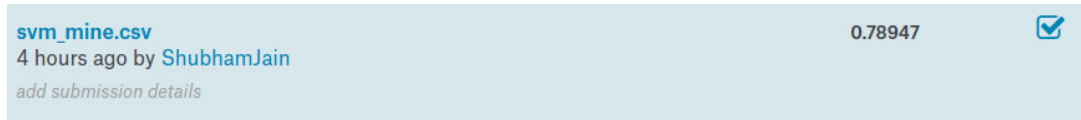


Fig. 7.2.3.2 Kaggle LeaderBoard-2

7.3 Exploratory modeling for Random Forests Algorithm

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

In this scenario, we are testing the Random Forests algorithm against the set of different variables of training dataset in order to find that which set of variables give us the highest of accuracy. In order to understand random forests, we need to understand a lot of different concepts like decision trees, bagging, etc.

7.3.1 Algorithm

➤ Decision Tree

Decision tree is a type of supervised learning algorithm (having a pre-defined target variable) that is mostly used in classification problems. It works for both categorical and continuous input and output variables. In this technique, we split the population or sample into two or more homogeneous sets (or sub-populations) based on most significant splitter / differentiator in input variables.

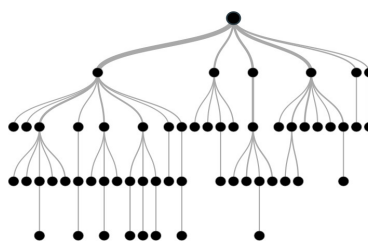


Fig. 7.3.1.1 Decision Tree-1

Example: -

Let's say we have a sample of 30 students with three variables Gender (Boy/ Girl), Class (IX/ X) and Height (5 to 6 ft). 15 out of these 30 play cricket in leisure time. Now, I want to create a model to predict who will play cricket during leisure period? In this problem, we need to segregate students who play cricket in their leisure time based on highly significant input variable among all three.

This is where decision tree helps; it will segregate the students based on all values of three variables and identify the variable, which creates the best homogeneous sets of students (which are heterogeneous to each other). In the snapshot below, you can see that variable Gender is able to identify best homogeneous sets compared to the other two variables.

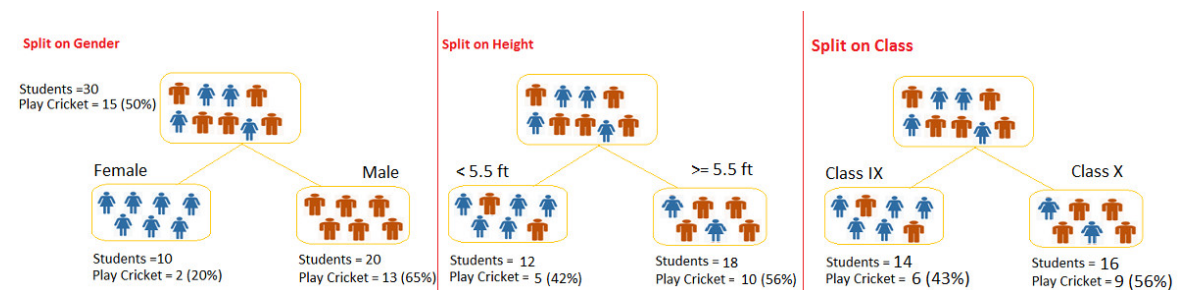


Fig. 7.3.1.2 Decision Tree-2

As mentioned above, decision tree identifies the most significant variable and its value that gives best homogeneous sets of population. Now the question which arises is, how does it identify the variable and the split? To do this, decision tree uses various algorithms, which we shall discuss in the following section.

- **Types of Decision Trees**

Types of decision tree are based on the type of target variable we have. It can be of two types:

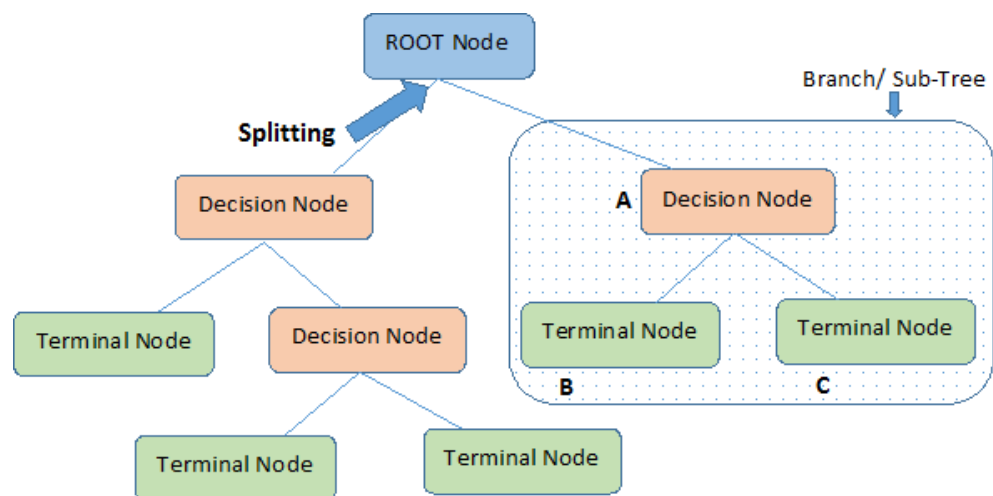
1. **Categorical Variable Decision Tree:** Decision Tree which has categorical target variable then it called as categorical variable decision tree. Example: - In above scenario of student problem, where the target variable was “Student will play cricket or not” i.e. YES or NO.
2. **Continuous Variable Decision Tree:** Decision Tree has continuous target variable then it is called as Continuous Variable Decision Tree

Example: - Let's say we have a problem to predict whether a customer will pay his renewal premium with an insurance company (yes/ no). Here we know that income of customer is a significant variable but insurance company does not have income details for all customers. Now, as we know this is an important variable, then we can build a decision tree to predict customer income based on occupation, product and various other variables. In this case, we are predicting values for continuous variable.

Important Terminology related to Decision Trees

Let's look at the basic terminology used with Decision trees:

1. **Root Node:** It represents entire population or sample and this further gets divided into two or more homogeneous sets.
2. **Splitting:** It is a process of dividing a node into two or more sub-nodes.
3. **Decision Node:** When a sub-node splits into further sub-nodes, then it is called decision node.
4. **Leaf/ Terminal Node:** Nodes do not split is called Leaf or Terminal node.



Note:- A is parent node of B and C.

Fig. 7.3.1.3 Decision Tree-3

5. **Pruning:** When we remove sub-nodes of a decision node, this process is called pruning. You can say opposite process of splitting.
6. **Branch / Sub-Tree:** A sub section of entire tree is called branch or sub-tree.
7. **Parent and Child Node:** A node, which is divided into sub-nodes is called parent node of sub-nodes where as sub-nodes are the child of parent node.

8. These are the terms commonly used for decision trees. As we know that every algorithm has advantages and disadvantages, below are the important factors which one should know.

Advantages -

1. **Easy to Understand:** Decision tree output is very easy to understand even for people from non-analytical background. It does not require any statistical knowledge to read and interpret them. Its graphical representation is very intuitive and users can easily relate their hypothesis.
2. **Useful in Data exploration:** Decision tree is one of the fastest way to identify most significant variables and relation between two or more variables. With the help of decision trees, we can create new variables / features that has better power to predict target variable. It can also be used in data exploration stage. For example, we are working on a problem where we have information available in hundreds of variables, there decision tree will help to identify most significant variable.
3. **Less data cleaning required:** It requires less data cleaning compared to some other modeling techniques. It is not influenced by outliers and missing values to a fair degree.
4. **Data type is not a constraint:** It can handle both numerical and categorical variables.
5. **Non Parametric Method:** Decision tree is considered to be a non-parametric method. This means that decision trees have no assumptions about the space distribution and the classifier structure.

Disadvantages -

1. **Over fitting:** Over fitting is one of the most practical difficulty for decision tree models. This problem gets solved by setting constraints on model parameters and pruning (discussed in detailed below).
2. **Not fit for continuous variables:** While working with continuous numerical variables, decision tree loses information when it categorizes variables in different categories

➤ Regression Trees vs. Classification Trees

We all know that the terminal nodes (or leaves) lie at the bottom of the decision tree. This means that decision trees are typically drawn upside down such that leaves are the the bottom & roots are the tops (shown below).



Fig. 7.3.1.4 Classification Tree

Both the trees work almost similar to each other; let's look at the primary differences & similarity between classification and regression trees:

1. Regression trees are used when dependent variable is continuous. Classification trees are used when dependent variable is categorical.
2. In case of regression tree, the value obtained by terminal nodes in the training data is the mean response of observation falling in that region. Thus, if an unseen data observation falls in that region, we'll make its prediction with mean value.
3. In case of classification tree, the value (class) obtained by terminal node in the training data is the mode of observations falling in that region. Thus, if an unseen data observation falls in that region, we'll make its prediction with mode value.
4. Both the trees divide the predictor space (independent variables) into distinct and non-overlapping regions. For the sake of simplicity, you can think of these regions as high dimensional boxes or boxes.
5. Both the trees follow a top-down greedy approach known as recursive binary splitting. We call it as 'top-down' because it begins from the top of tree when all the observations are available in a single region and successively splits the

6. predictor space into two new branches down the tree. It is known as ‘greedy’ because, the algorithm cares (looks for best variable available) about only the current split, and not about future splits which will lead to a better tree.
7. This splitting process is continued until a user defined stopping criteria is reached. For example: we can tell the the algorithm to stop once the number of observations per node becomes less than 50.
8. In both the cases, the splitting process results in fully grown trees until the stopping criteria is reached. But, the fully grown tree is likely to overfit data, leading to poor accuracy on unseen data. This bring ‘pruning’. Pruning is one of the technique used tackle overfitting. We’ll learn more about it in following section.

➤ Decision of making the tree split

The decision of making strategic splits heavily affects a tree’s accuracy. The decision criteria is different for classification and regression trees.

Decision trees use multiple algorithms to decide to split a node in two or more sub-nodes. The creation of sub-nodes increases the homogeneity of resultant sub-nodes. In other words, we can say that purity of the node increases with respect to the target variable. Decision tree splits the nodes on all available variables and then selects the split which results in most homogeneous sub-nodes.

The algorithm selection is also based on type of target variables. Let’s look at the some most commonly used algorithms in decision tree:

➤ Gini Index

Gini index says, if we select two items from a population at random then they must be of same class and probability for this is 1 if population is pure.

1. It works with categorical target variable “Success” or “Failure”.
2. It performs only Binary splits
3. Higher the value of Gini higher the homogeneity.
4. CART (Classification and Regression Tree) uses Gini method to create binary splits.

Bellow are the steps for calculating Gini Index for a split -

1. Calculate Gini for sub-nodes, using formula sum of square of probability for success and failure (p^2+q^2).
2. Calculate Gini for split using weighted Gini score of each node of that split

Example: – Referring to example used above, where we want to segregate the students based on target variable (playing cricket or not). In the snapshot below, we split the population using two input variables Gender and Class. Now, I want to identify which split is producing more homogeneous sub-nodes using Gini index.

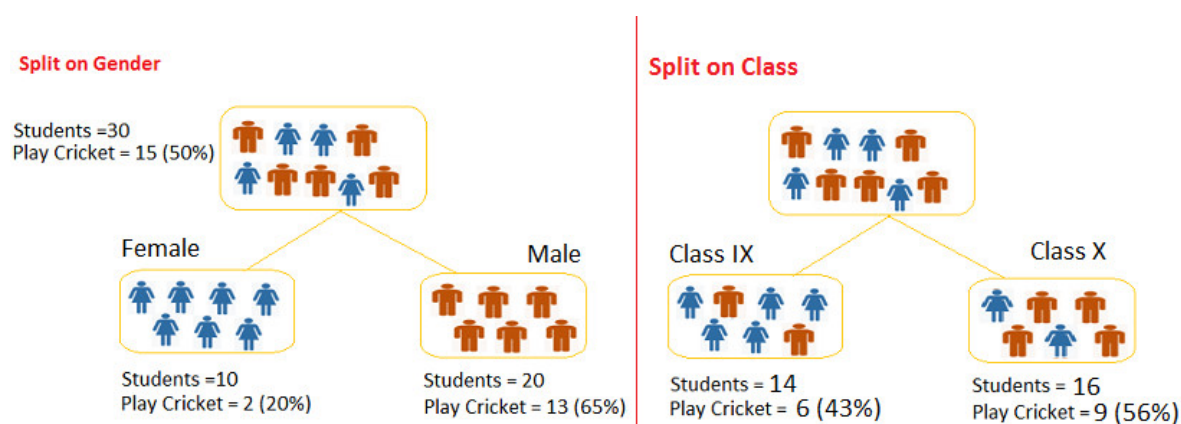


Fig. 7.3.1.5 Decision Tree-4

Split on Gender:

1. Calculate, Gini for sub-node Female = $(0.2)*(0.2)+(0.8)*(0.8)=0.68$
2. Gini for sub-node Male = $(0.65)*(0.65)+(0.35)*(0.35)=0.55$
3. Calculate weighted Gini for Split Gender = $(10/30)*0.68+(20/30)*0.55 = \mathbf{0.59}$

Similar for Split on Class:

1. Gini for sub-node Class IX = $(0.43)*(0.43)+(0.57)*(0.57)=0.51$
2. Gini for sub-node Class X = $(0.56)*(0.56)+(0.44)*(0.44)=0.51$
3. Calculate weighted Gini for Split Class = $(14/30)*0.51+(16/30)*0.51 = \mathbf{0.51}$

Above, you can see that Gini score for *Split on Gender* is higher than *Split on Class*, hence, the node split will take place on Gender.

➤ Reduction in Variance

Till now, we have discussed the algorithms for categorical target variable. Reduction in variance is an algorithm used for continuous target variables (regression problems). This algorithm uses the standard formula of variance to choose the best split. The split with lower variance is selected as the criteria to split the population:

$$\text{Variance} = \frac{\sum (X - \bar{X})^2}{n}$$

Fig. 7.3.1.6.1 Formula for Variance

Above X-bar is mean of the values, X is actual and n is number of values.

Steps to calculate Variance:

1. Calculate variance for each node.
2. Calculate variance for each split as weighted average of each node variance.

Until here, we learnt about the basics of decision trees and the decision making process involved to choose the best splits in building a tree model. As I said, decision tree can be applied both on regression and classification problems. Let's understand these aspects in detail.

➤ Key parameters of tree modeling to avoid over-fitting in decision trees

Over-fitting is one of the key challenges faced while modeling decision trees. If there is no limit set of a decision tree, it will give you 100% accuracy on training set because in the worse case it will end up making 1 leaf for each observation. Thus, preventing overfitting is pivotal while modeling a decision tree and it can be done in 2 ways:

1. Setting constraints on tree size
2. Tree pruning

Let's discuss both of these briefly.

Setting Constraints on Tree Size

This can be done by using various parameters which are used to define a tree. First, let's look at the general structure of a decision tree:

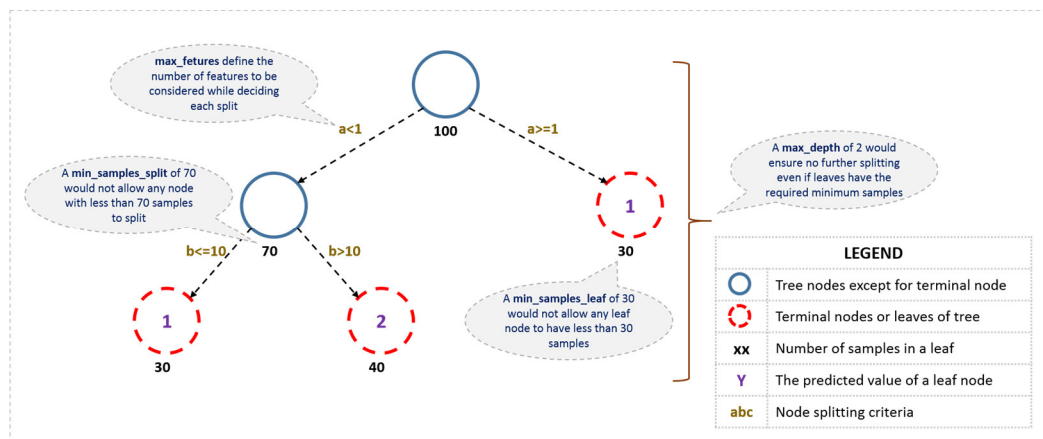


Fig. 7.3.1.6.2 Decision Tree-5

The parameters used for defining a tree are further explained below. The parameters described below are irrespective of tool. It is important to understand the role of parameters used in tree modeling. These parameters are available in R & Python.

1. Minimum samples for a node split

- Defines the minimum number of samples (or observations) which are required in a node to be considered for splitting.
- Used to control over-fitting. Higher values prevent a model from learning relations which might be highly specific to the particular sample selected for a tree.
- Too high values can lead to under-fitting hence, it should be tuned using CV.

2. Minimum samples for a terminal node (leaf)

- Defines the minimum samples (or observations) required in a terminal node or leaf.
- Used to control over-fitting similar to `min_samples_split`.
- Generally lower values should be chosen for imbalanced class problems because the regions in which the minority class will be in majority will be very small.

3. Maximum depth of tree (vertical depth)

- Defines the maximum depth of a tree.
- Used to control over-fitting as higher depth will allow model to learn relations very specific to a particular sample.
- Should be tuned using CV.

4. Maximum number of terminal nodes

- The maximum number of terminal nodes or leaves in a tree.
- Can be defined in place of `max_depth`. Since binary trees are created, a depth of 'n' would produce a maximum of 2^n leaves.

5. Maximum features to consider for split

- The number of features to consider while searching for a best split. These will be randomly selected.

- Higher values can lead to over-fitting but depends on case to case.

➤ Tree Pruning

As discussed earlier, the technique of setting constraint is a greedy-approach. In other words, it will check for the best split instantaneously and move forward until one of the specified stopping condition is reached. Let's consider the following case when you're driving:

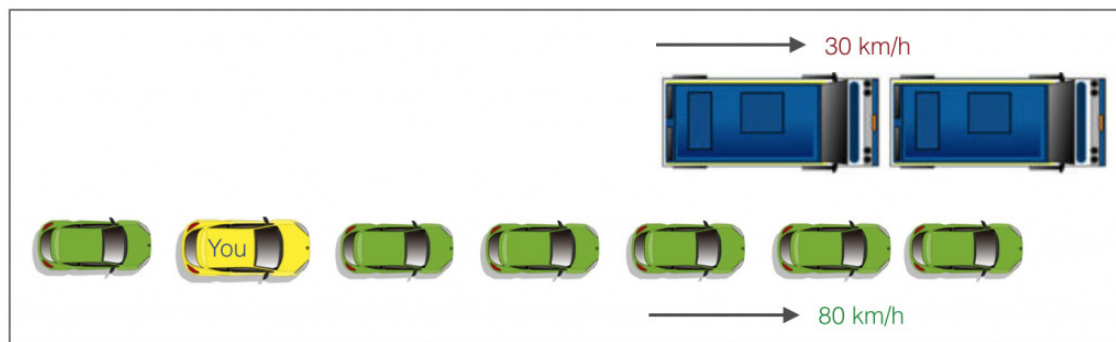


Fig 7.3.1.7 Tree Pruning

There are 2 lanes:

1. A lane with cars moving at 80km/h
2. A lane with trucks moving at 30km/h

At this instant, you are the yellow car and you have 2 choices

1. Take a left and overtake the other 2 cars quickly
2. Keep moving in the present lane

Let's analyze these choice. In the former choice, you'll immediately overtake the car ahead and reach behind the truck and start moving at 30 km/h, looking for an opportunity to move back right. All cars originally behind you move ahead in the meanwhile. This would be the optimum choice if your objective is to maximize the distance covered in next say 10 seconds. In the later choice, your sale through at same speed, cross trucks and then overtake maybe depending on situation ahead.

This is exactly the difference between normal decision tree & pruning. A decision tree with constraints won't see the truck ahead and adopt a greedy approach by taking a left. On the other hand, if we use pruning, we in effect look at a few steps ahead and make a choice. So we know pruning is better. But how to implement it in decision tree? The idea is simple.

1. We first make the decision tree to a large depth.

2. Then we start at the bottom and start removing leaves which are giving us negative returns when compared from the top.
3. Suppose a split is giving us a gain of say -10 (loss of 10) and then the next split on that gives us a gain of 20. A simple decision tree will stop at step 1 but in pruning, we will see that the overall gain is +10 and keep both leaves.

Note that sklearn's decision tree classifier does not currently support pruning. Advanced packages like xgboost have adopted tree pruning in their implementation. But the library *rpart* in R, provides a function to prune.

➤ Tree based models vs linear models

Actually, you can use any algorithm. It is dependent on the type of problem you are solving.

Let's look at some key factors which will help you to decide which algorithm to use:

1. If the relationship between dependent & independent variable is well approximated by a linear model, linear regression will outperform tree based model.
2. If there is a high non-linearity & complex relationship between dependent & independent variables, a tree model will outperform a classical regression method.
3. If you need to build a model which is easy to explain to people, a decision tree model will always do better than a linear model. Decision tree models are even simpler to interpret than linear regression!

➤ Ensemble methods in tree based modeling -

The literary meaning of word 'ensemble' is *group*. Ensemble methods involve group of predictive models to achieve a better accuracy and model stability. Ensemble methods are known to impart supreme boost to tree based models.

Like every other model, a tree based model also suffers from the plague of bias and variance. Bias means, 'how much on an average are the predicted values different from the actual value.' Variance means, 'how different will the predictions of the model be at the same point if different samples are taken from the same population'.

You build a small tree and you will get a model with low variance and high bias. How do you manage to balance the tradeoff between bias and variance?

Normally, as you increase the complexity of your model, you will see a reduction in prediction error due to lower bias in the model. As you continue to make your model more

complex, you end up over-fitting your model and your model will start suffering from high variance.

A champion model should maintain a balance between these two types of errors. This is known as the **trade-off management** of bias-variance errors. Ensemble learning is one way to execute this trade off analysis.

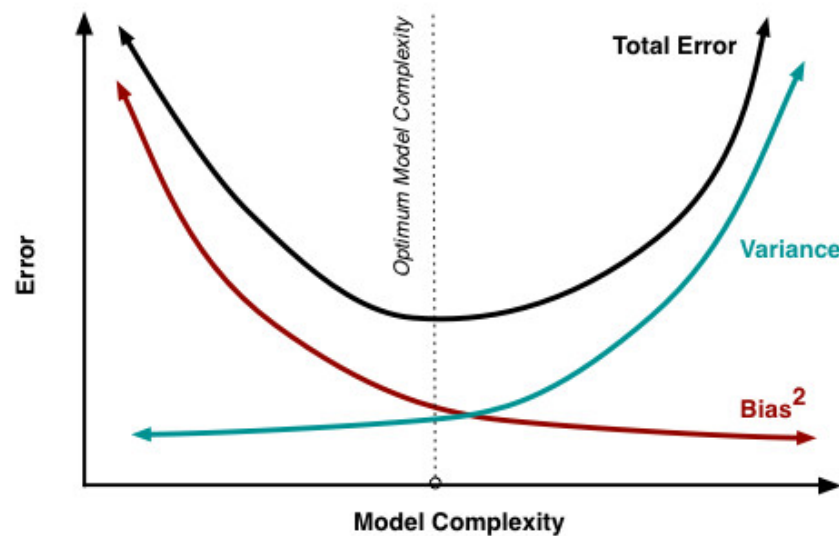


Fig. 7.3.1.8 Trade Off Management

Some of the commonly used ensemble methods include: Bagging, Boosting and Stacking. In this tutorial, we'll focus on Bagging and Boosting in detail.

➤ Bagging

Bagging is a technique used to reduce the variance of our predictions by combining the result of multiple classifiers modeled on different sub-samples of the same data set. The following figure will make it clearer:

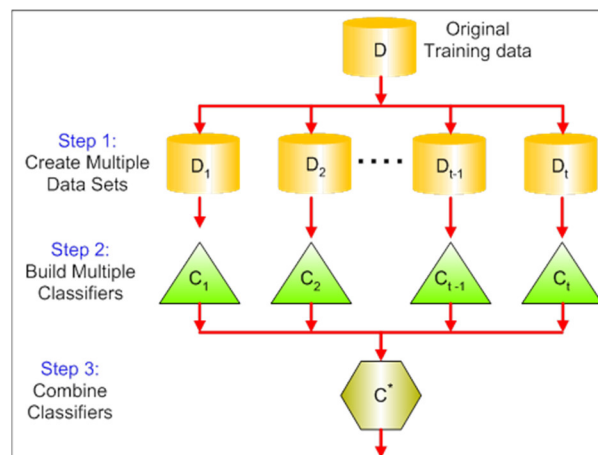


Fig 7.3.1.9 Bagging

The steps followed in bagging are as follows -

1. Create Multiple DataSets:

- Sampling is done *with replacement* on the original data and new datasets are formed.
- The new data sets can have a fraction of the columns as well as rows, which are generally hyper-parameters in a bagging model
- Taking row and column fractions less than 1 helps in making robust models, less prone to overfitting

2. Build Multiple Classifiers:

- Classifiers are built on each data set.
- Generally, the same classifier is modeled on each data set and predictions are made.

3. Combine Classifiers:

- The predictions of all the classifiers are combined using a mean, median or mode value depending on the problem at hand.
- The combined values are generally more robust than a single model.

Note that, here the number of models built is not a hyper-parameter. Higher number of models is always better or may give similar performance than lower numbers. It can be theoretically shown that the variance of the combined predictions is reduced to $1/n$ (n : number of classifiers) of the original variance, under some assumptions.

There are various implementations of bagging models. Random forest is one of them and we'll discuss it next.

➤ **Random Forests**

Random Forest is considered to be a *panacea* of all data science problems. On a funny note, when you can't think of any algorithm (irrespective of situation), use random forest!

Random Forest is a versatile machine learning method capable of performing both regression and classification tasks. It also undertakes dimensional reduction methods, treats missing values, outlier values and other essential steps of data exploration and does a fairly good job. It is a type of ensemble learning method, where a group of weak models combine to form a powerful model. In Random Forest, we grow multiple trees as

opposed to a single tree in CART model. To classify a new object based on attributes, each tree gives a classification and we say the tree “votes” for that class. The forest chooses the classification having the most votes (over all the trees in the forest) and in case of regression, it takes the average of outputs by different trees.

It works in the following manner. Each tree is planted & grown as follows:

1. Assume number of cases in the training set is N . Then, sample of these N cases is taken at random but *with replacement*. This sample will be the training set for growing the tree.
2. If there are M input variables, a number $m < M$ is specified such that at each node, m variables are selected at random out of the M . The best split on this M is used to split the node. The value of m is held constant while we grow the forest.
3. Each tree is grown to the largest extent possible and there is no pruning.
4. Predict new data by aggregating the predictions of the n tree trees (i.e., majority votes for classification, average for regression).

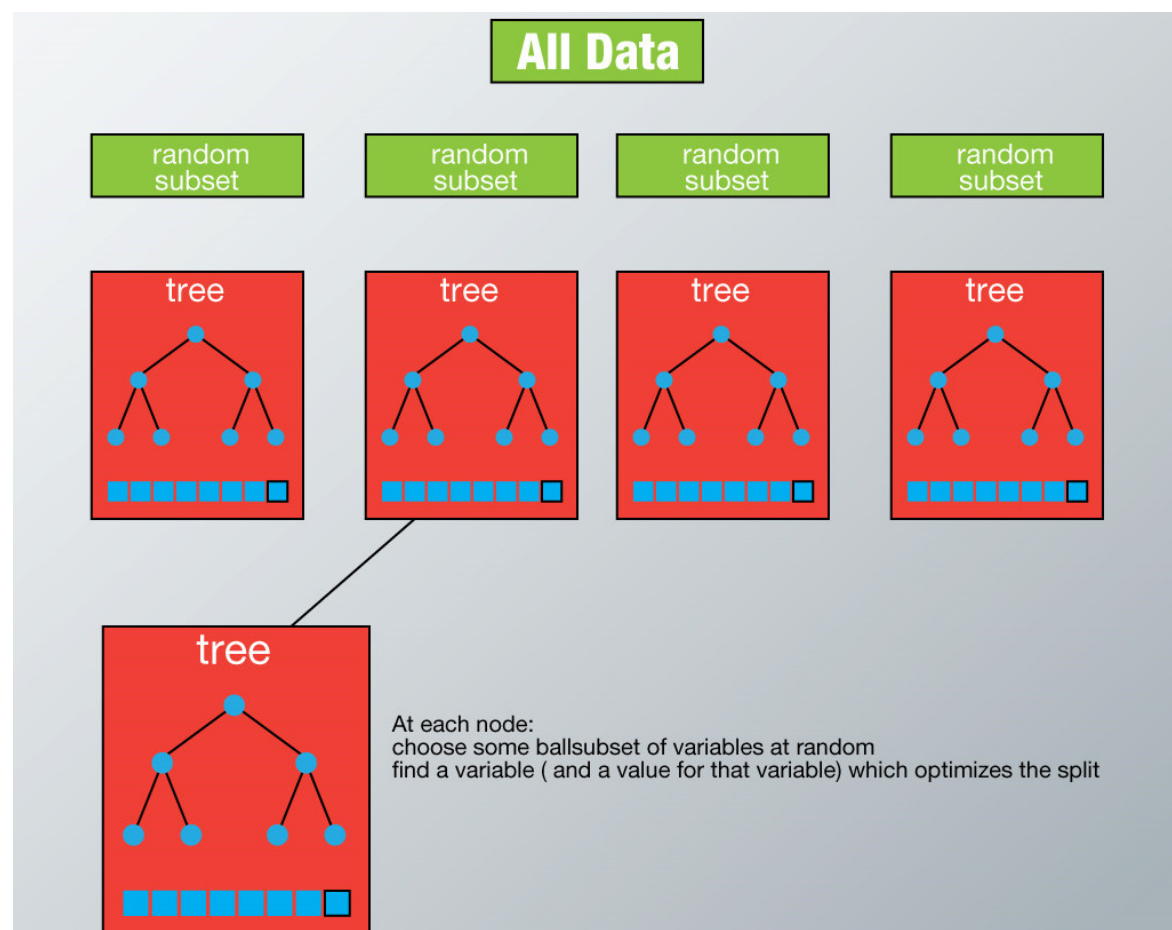


Fig 7.3.1.10 Random Forests Trees

Advantages of Random Forest

- This algorithm can solve both type of problems i.e. classification and regression and does a decent estimation at both fronts.
- One of benefits of Random forest which excites me most is, the power of handle large data set with higher dimensionality. It can handle thousands of input variables and identify most significant variables so it is considered as one of the dimensionality reduction methods. Further, the model outputs **Importance of variable**, which can be a very handy feature (on some random data set).
- It has an effective method for estimating missing data and maintains accuracy when a large proportion of the data are missing.
- It has methods for balancing errors in data sets where classes are imbalanced.
- The capabilities of the above can be extended to unlabeled data, leading to unsupervised clustering, data views and outlier detection.
- Random Forest involves sampling of the input data with replacement called as bootstrap sampling. Here one third of the data is not used for training and can be used to testing. These are called the **out of bag** samples. Error estimated on these out of bag samples is known as *out of bag error*. Study of error estimates by Out of bag, gives evidence to show that the out-of-bag estimate is as accurate as using a test set of the same size as the training set. Therefore, using the out-of-bag error estimate removes the need for a set aside test set.

Disadvantages of Random Forest

- It surely does a good job at classification but not as good as for regression problem as it does not give precise continuous nature predictions. In case of regression, it doesn't predict beyond the range in the training data, and that they may over-fit data sets that are particularly noisy.
- Random Forest can feel like a black box approach for statistical modelers – you have very little control on what the model does. You can at best – try different parameters and random seeds!

Hence, we can summarize random forests as follows -

- Random forests algorithm is an algorithm for classification, regression and other tasks, that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes or means prediction of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.
- We assume that the user knows about the construction of single classification trees. Random Forests grows many classification trees. To classify a new object from an input vector, put the input vector down each of the trees in the forest. Each tree gives a classification, and we say the tree "votes" for that class. The forest chooses the classification having the most votes (over all the trees in the forest). Each tree is grown.
- If the number of cases in the training set is N , sample N cases at random - but with replacement, from the original data. This sample will be the training set for growing the tree. If there are M input variables, a number $m \ll M$ is specified such that at each node, m variables are selected at random out of the M and the best split on these m is used to split the node. The value of m is held constant during the forest growing.
- Each tree is grown to the largest extent possible. In the original paper on random forests, it was shown that the forest error rate depends on two things: The correlation between any two trees in the forest. Increasing the correlation increases the forest error rate. The strength of each individual tree in the forest. A tree with a low error rate is a strong classifier. Increasing the strength of the individual trees decreases the forest error rate.

7.3.2 Random Forests in Layman's Terms

- Let's say you are a resort owner considering your next advertising campaign. You will be sending out 1,000 vouchers that give two *free* days at the resort, with the hope that the customer will decide to extend it to a full week (who want's a two-day vacation?).
- You have a rather extensive database of customer data, including where they are from, income range, duration of stay(s) and money spent at each stay, number of

- children, marital status, and employment status (working, unemployed, retired). You would like to create a decision tree to help select the customers to send the vouchers to (specifically, those who will spend at least \$200 per day and stay at least one week.
- To create the tree, a computer will perform the following steps:
 - From the above list of customer data, find the property that best separates the customers into two groups.
 - Repeat step one for each of the above groups using the remaining properties.
 - In the end, you will have a tree where at each point, you can make one of two decisions. Following a path leads to a decision. The split points will be chosen to maximize the probability of a correct classification.
 - Each of the trees is a decision tree.

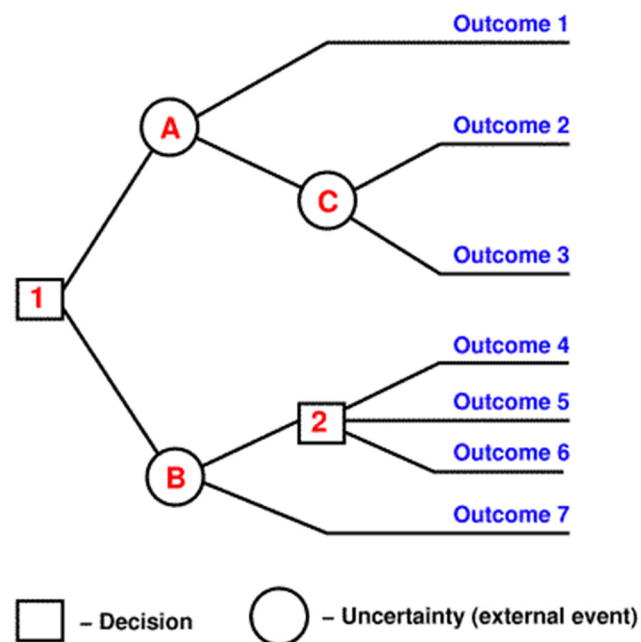


Fig 7.3.2.1 Decision Tree Example

Now, let's understand Random Forest....

- Let's say you want to know if tomorrow will be sunny, rainy or cloudy based on several measurements from the previous day like temperature, dew point, cloud cover, humidity etc. Imagine you have a lot of measurements, say 50.

- You want to build a decision tree to decide the weather forecast, for example if temperature on previous day > 30 and cloud cover is $< 20\%$ then the weather is Sunny. But you don't know which features to use, you have a lot.
- So you take a random set of measures and a random sample of your training set (days with measurements and known weather outlook) and you build a decision tree. Then you do the same many times using a different random set of measurements and a random sample of data each time. At the end you have many decision trees, you use each of them to forecast the weather and then decide the final forecast based on a simple majority.

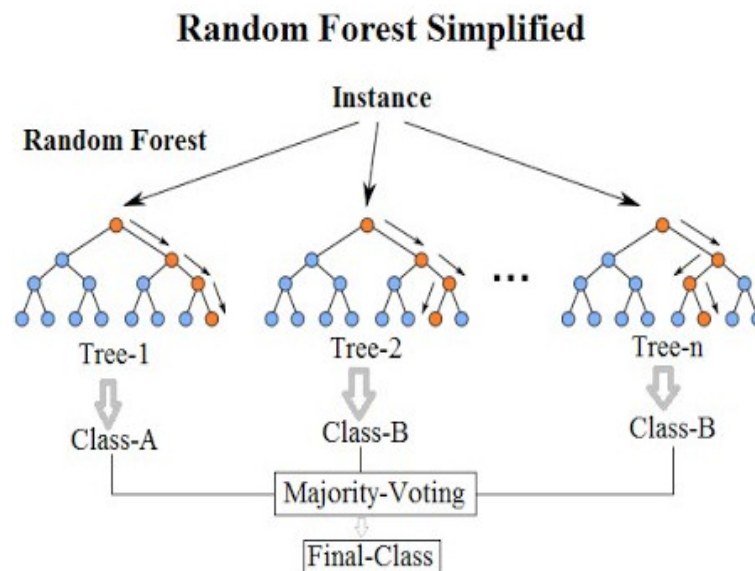


Fig 7.3.2.2 Random Forest Example

7.3.3 Application-

In this module, we are testing random forests against different training dataset variables. First, we are plotting the importance factor of different variables of our random forests algorithm. The plot is shown bellow.

- ✓ “title” is the most important variable. But still, we will build different random forest models in order to explore the best combination of variables.

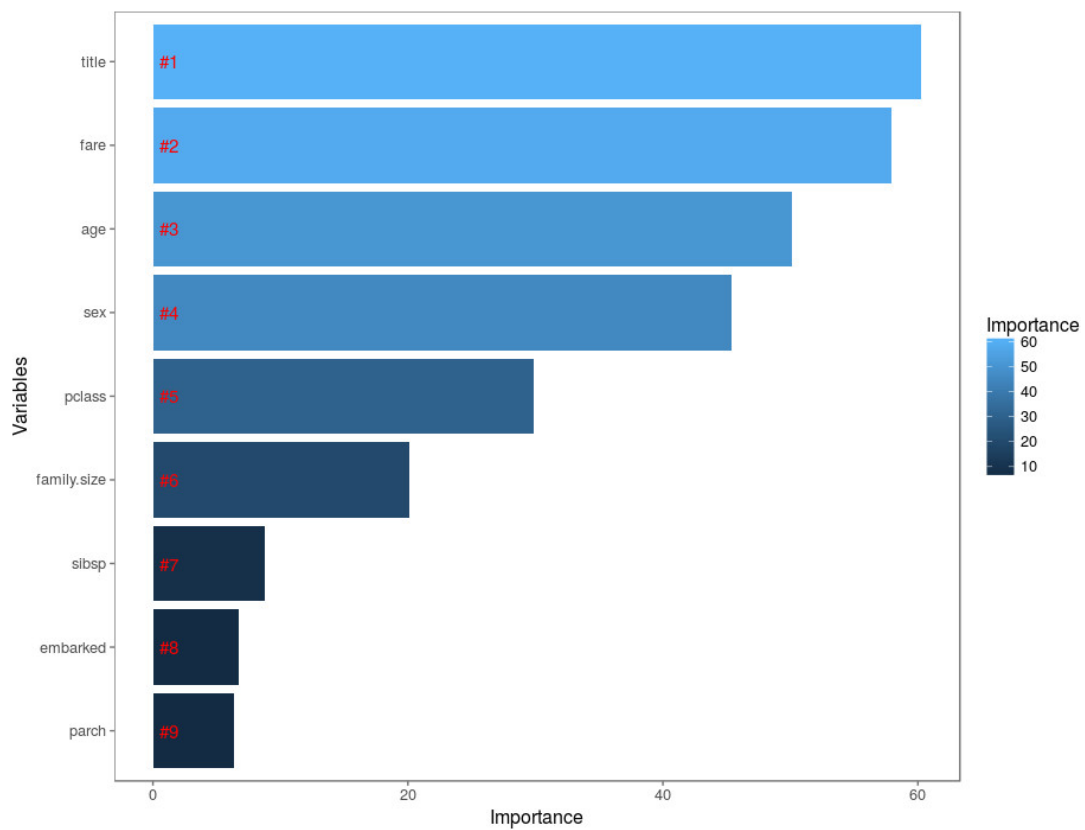


Fig. 7.3.3.1 Random Forest Variable Importance Factor

- ✓ Our first model is trained with variable pclass and title, which is having an OOB estimate of error rate being equal to 20.76%. This means that the accuracy of our model is 79.24% accurate.

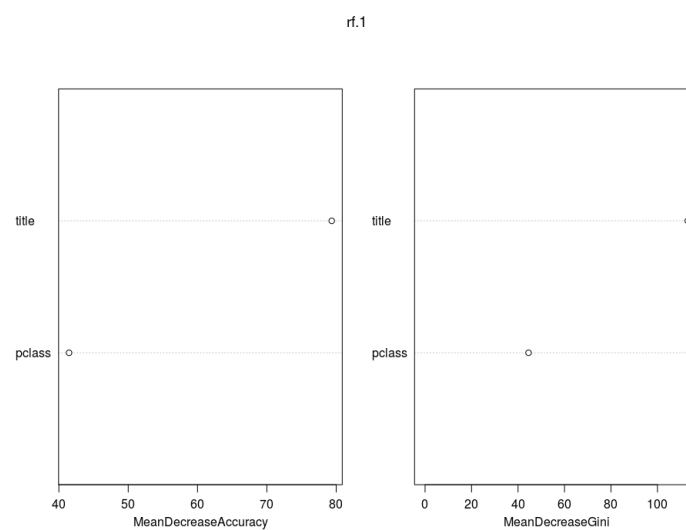


Fig 7.3.3.2 Random Forest Graph-1

- ✓ Our second model is trained with variable pclass, sibsp and title, which is having an OOB estimate of error rate equal to 19.75%. This means that the accuracy of our model is 80.24% accurate, which is more accurate as compared to the previous model.

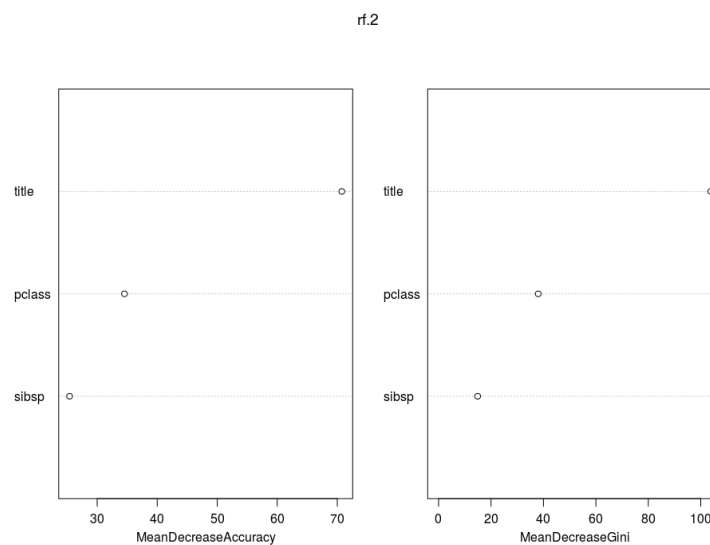


Fig 7.3.3.3 Random Forest Graph-2

- ✓ Our third model is trained with variable pclass, parch and title, which is having an OOB estimate of error rate equal to 19.98%, which is more accurate as compared to the previous model.

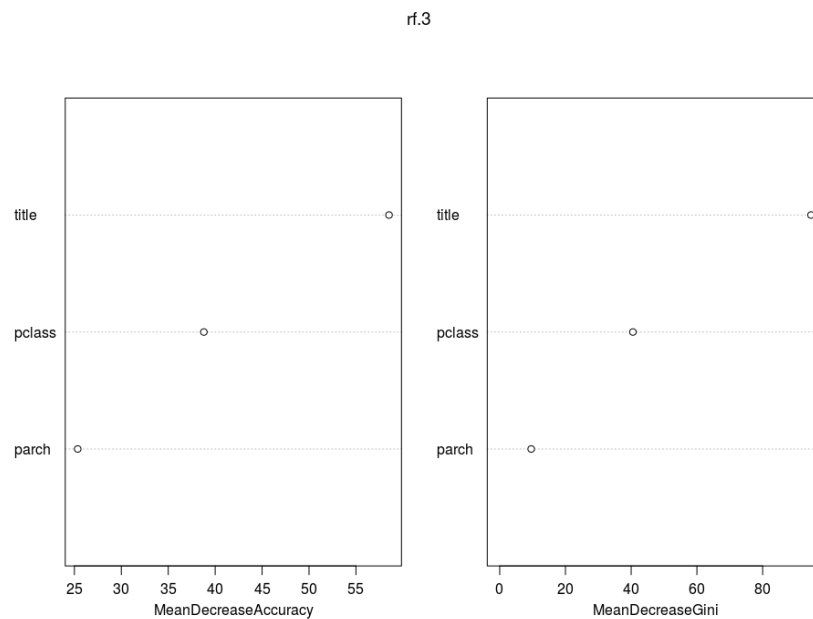


Fig 7.3.3.4 Random Forest Graph-3

- ✓ Our fourth model is trained with variable pclass, parch, sibsp and title, which is having an OOB estimate of error rate equal to 18.52%. This means that the accuracy of our model has increased by 2%.

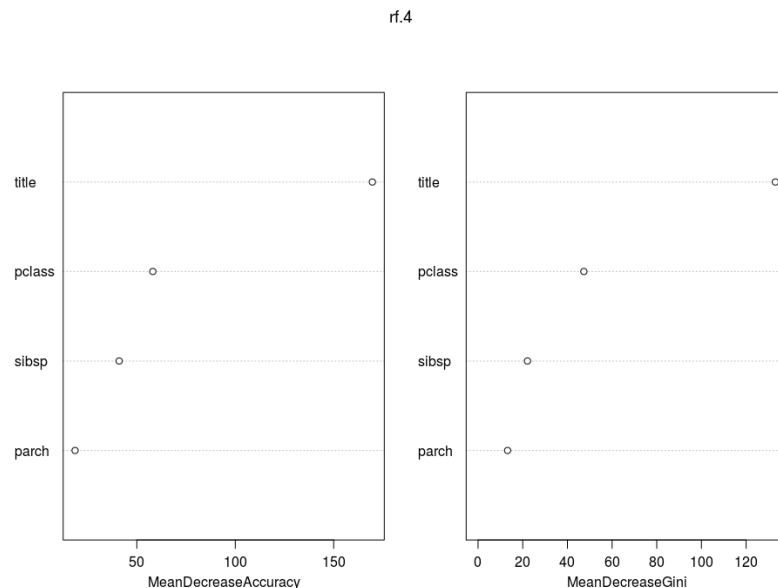


Fig 7.3.3.5 Random Forest Graph-4

- ✓ Our fifth model is trained with variable pclass, family.size, and title, which is having an OOB estimate of error rate equal to 18.41%. This is our most accurate model. We have further constructed two more models, but they are not accurate as the fifth

- ✓ model. Since the size of project report is restricted, it's shown in demo and the code section only.

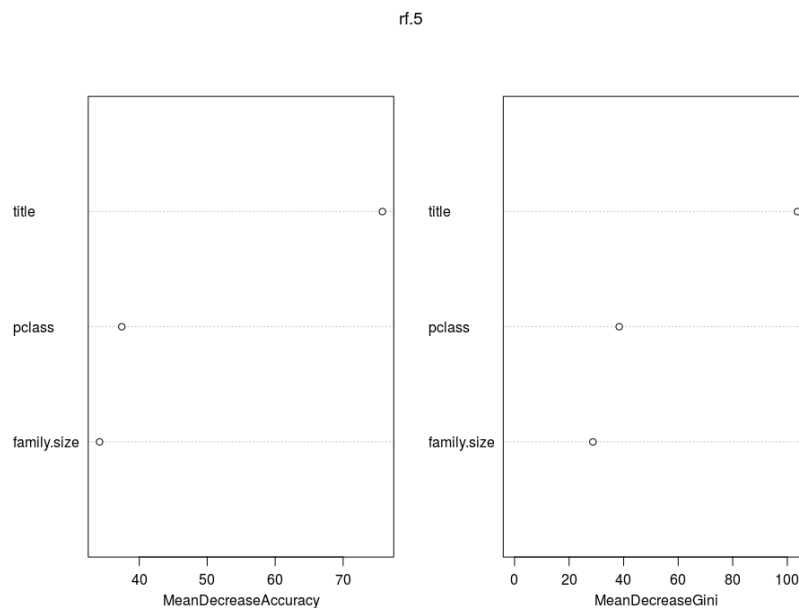


Fig 7.3.3.6 Random Forest Graph-5

When we submit the test dataset trained on rf.5 or the fifth model, our accuracy is 79.426%, which is slightly better than SVM.

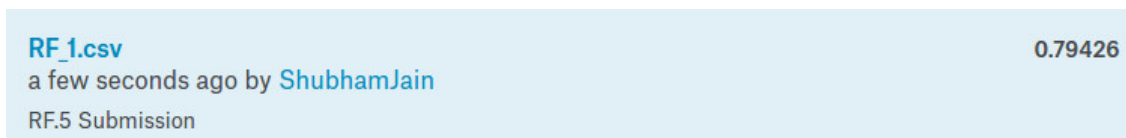


Fig 7.3.3.6 Final Result of Random Forest Algorithm

7.4 Cross validation of Random Forests Algorithm

In this phase, look into cross-validation using the caret package to see if we can get more accurate estimates. Research has shown that 10-fold Cross Validation repeated 10 times is the best place to start; hence we create total 100 folds, but ensure that the ratio of those that survived and perished in each fold matches the overall training set. This is known as stratified cross validation and generally provides better results.

7.4.1 Cross Validation and its needs

Let's understand this using the snapshot illustrating fit of various models below:

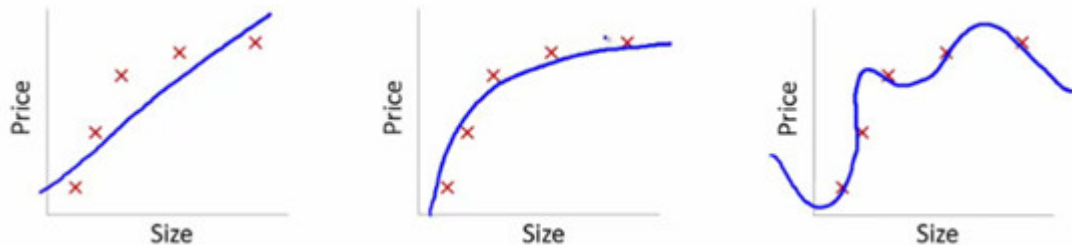


Fig 7.4.1.1 Fitting of different models

Here, we are trying to find the relationship between size and price. For which, we've taken the following steps:

1. We've established the relationship using a linear equation for which the plots have been shown. First plot has high error from training data points. Therefore, this will not perform well at both public and private leader board. This is an example of “**Under fitting**”. In this case, our model fails to capture the underlying trend of the data.
2. In second plot, we just found the right relationship between price and size i.e. low training error and generalization of relationship
3. In third plot, we found a relationship which has almost zero training error. This is because, the relationship is developed by considering each deviation in the data point (including noise) i.e. model is too sensitive and captures random patterns which are present only in the current data set. This is an example of “**Over fitting**”.

In this relationship, there could be high deviation in public and private leader board. A common practice in data science competitions is to iterate over various models to find a better performing model. However, it becomes difficult to distinguish whether this improvement in score is coming because we are capturing the relationship better or we are just over-fitting the data. To find the right answer of this question, we use cross validation technique. This method helps us to achieve more generalized relationships.

➤ Cross Validations

Cross Validation is a technique which involves reserving a particular sample of a data set on which you do not train the model. Later, you test the model on this sample before

finalizing the model. Here are the steps involved in cross validation: You *reserve* a sample data set.

1. Train the model using the remaining part of the data set.
2. Use the reserve sample of the data set test (validation) set. This will help you to know the effectiveness of model performance. If your model delivers a positive result on validation data, go ahead with current model.

7.4.2 K-Fold Cross Validation (or K-Means Cross Validation) Algorithm

A good cross validation model should have the following properties –

- We should train model on large portion of data set. Else, we'd fail every time to read the underlying trend of data sets. Eventually, resulting in higher bias.
- We also need a good ratio testing data points. As, we have seen that lower data points can lead to variance error while testing the effectiveness of model.
- We should iterate on training and testing process multiple times. We should change the train and test data set distribution. This helps to validate the model effectiveness well.

Below is the visualization of how does a k-fold validation work for k=10.

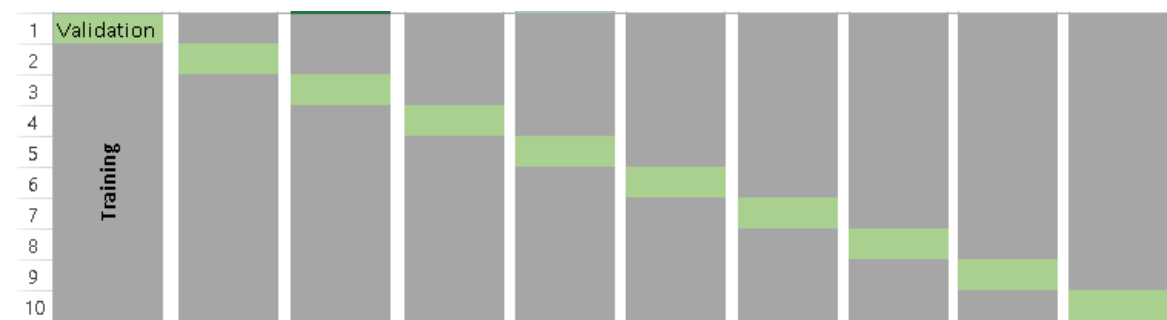


Fig 7.4.2.1 10-Fold CV Example

K- Fold cross validation has all these properties, is easy to follow and implement. Here are the quick steps:

1. Randomly split your entire dataset into k “folds”.
2. For each k folds in your dataset, build your model on $k - 1$ folds of the data set. Then, test the model to check the effectiveness for k^{th} fold.
3. Record the error you see on each of the predictions.
4. Repeat this until each of the k folds has served as the test set.
5. The average of your k recorded errors is called the cross-validation error and will serve as your performance metric for the model.

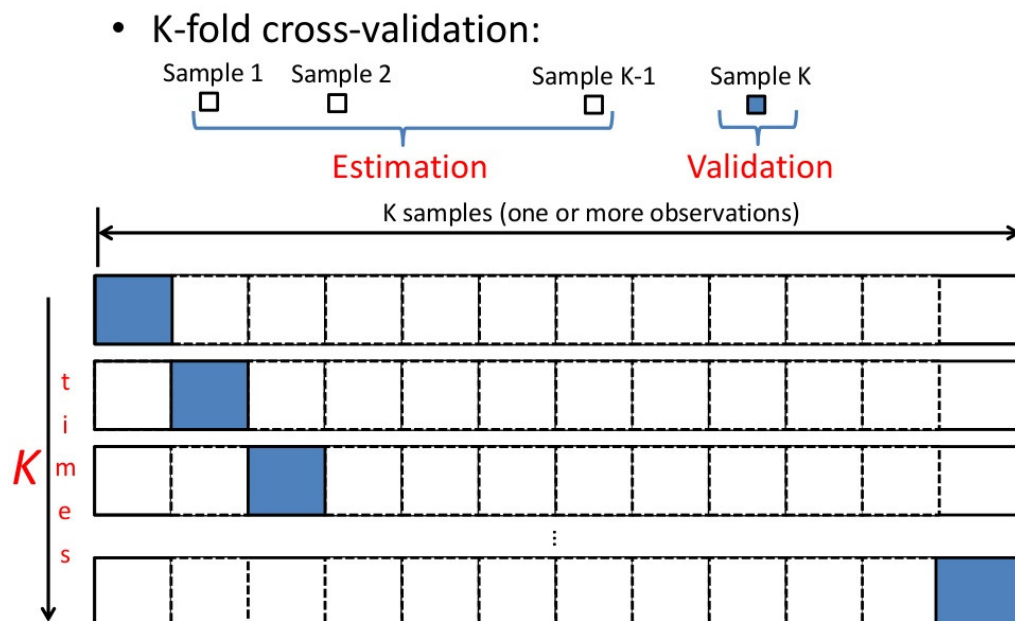


Fig 7.4.2.2 K-Fold CV Example

But we need to remember that the lower value of K is more biased and hence undesirable. On the other hand, higher value of K is less biased, but can suffer from large variability. We need to check the value of K again and again and see which value of K gives us the best match. In this project, we have checked three values of K , that is, $K = 3$, $K = 5$ and $K = 10$. We found that $K = 3$ gives us the best result.

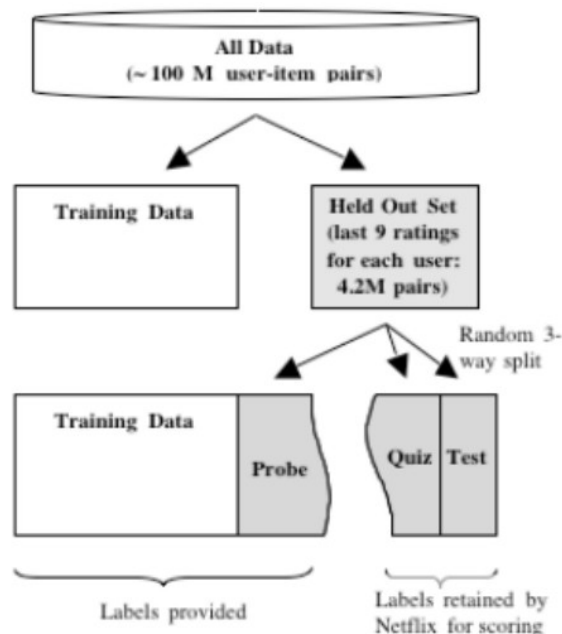


Fig 7.4.2.3 Splitting of Data for CV

- In other words, while performing k-fold cross validation we are doing the following things.
- The data set is divided into k subsets, and the holdout method is repeated k times.
- Each time, one of the k subsets is used as the test set and the other k-1 subsets are put together to form a training set.
- Then the average error across all k trials is computed. The advantage of this method is that it matters less how the data gets divided.
- Every data point gets to be in a test set exactly once, and gets to be in a training set k-1 times.
- The variance of the resulting estimate is reduced as k is increased.
- The disadvantage of this method is that the training algorithm has to be rerun from scratch k times, which means it takes k times as much computation to make an evaluation.
- A variant of this method is to randomly divide the data into a test and training set k different times. The advantage of doing this is that you can independently choose how large each test set is and how many trials you average over.

There are two other methods for cross validation, which are out of scope of this project, but keeping the reader's perspective in mind, we are giving a brief overview about them,

- The holdout method is the simplest kind of cross validation. The data set is separated into two sets, called the training set and the testing set. The function approximator fits a function using the training set only. Then the function approximator is asked to predict the output values for the data in the testing set (it has never seen these output values before). The errors it makes are accumulated as before to give the mean absolute test set error, which is used to evaluate the model. The advantage of this method is that it is usually preferable to the residual method and takes no longer to compute. However, its evaluation can have a high variance. The evaluation may depend heavily on which data points end up in the training set and which end up in the test set, and thus the evaluation may be significantly different depending on how the division is made.
- Leave-one-out cross validation is K-fold cross validation taken to its logical extreme, with K equal to N, the number of data points in the set. That means that N separate times, the function approximator is trained on all the data except for one point and a prediction is made for that point. As before the average error is computed and used to evaluate the model. The evaluation given by leave-one-out cross validation error (LOO-XVE) is good, but at first pass it seems very expensive to compute. Fortunately, locally weighted learners can make LOO predictions just as easily as they make regular predictions. That means computing the LOO-XVE takes no more time than computing the residual error and it is a much better way to evaluate models.
- The method that we have used is K-Fold cross validation, and this is the best fit for out use case.

7.4.3 Cross Validation in Layman's Terms

I want to catch the subway to go to my office. My plan is to take my car, park at the subway and then take the train to go to my office. My goal is to catch the train at 8.15 am every day

so that I can reach my office on time. I need to decide the following: (a) the time at which I need to leave from my home and (b) the route I will take to drive to the station.

In the above example, I have two parameters (i.e., time of departure from home and route to take to the station) and I need to choose these parameters such that I reach the station by 8.15 am.

In order to solve the above problem, I may try out different sets of 'parameters' (i.e., different combination of times of departure and route) on Mondays, Wednesdays, and Fridays, to see which combination is the 'best' one. The idea is that once I have identified the best combination I can use it every day so that I achieve my objective.

- *Problem of Overfitting*

The problem with the above approach is that I may overfit which essentially means that the best combination I identify may in some sense may be unique to Mon, Wed and Fridays and that combination may not work for Tue and Thu. Overfitting may happen if in my search for the best combination of times and routes I exploit some aspect of the traffic situation on Mon/Wed/Fri which does not occur on Tue and Thu.

- *One Solution to Overfitting: Cross-Validation*

Cross-validation is one solution to overfitting. The idea is that once we have identified our best combination of parameters (in our case time and route) we test the performance of that set of parameters in a different context. Therefore, we may want to test on Tue and Thu as well to ensure that our choices work for those days as well.

Of course, cross validation is not perfect. Going back to our example of the subway, it can happen that even after cross-validation, our best choice of parameters may not work one month down the line because of various issues (e.g., construction, traffic volume changes over time etc).

7.4.4 Application

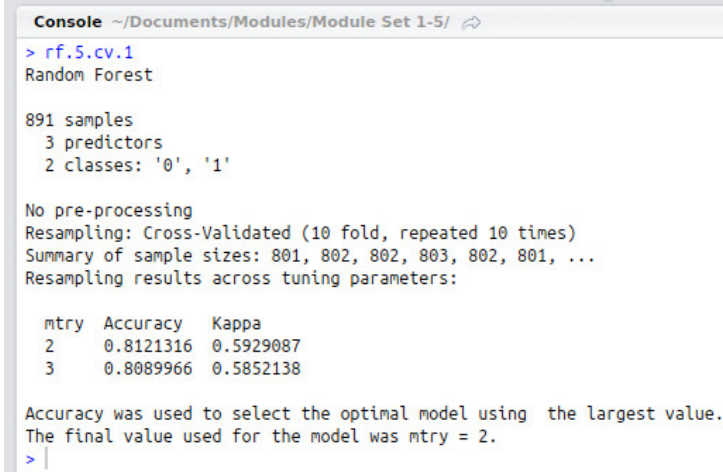
In k-fold cross validation model, we are using our 5th Random Forest model and training it against 10 fold, 5 fold and 3 fold cross validation, where each fold is being repeated 10 times.

- The result of 10 fold cross validation (repeated 10 times) gives us an accuracy of 81.21316%, which is slightly less than the accuracy of 5th random forest algorithm

(being equal to 81.59%)

- The result of 5 fold cross validation (repeated 10 times) gives us an accuracy of 81.28009%. Therefore 10 fold Cross Validation is better as compared to 5 fold cross validation.
- The result of 3 fold cross validation (repeated 10 times) gives us an accuracy of 80.98765 %. Therefore 3 fold Cross Validation is better as compared to 10 fold cross validation. For future cross validation, we'll use 3-fold since it's the best.

From the above three cross validation models, we can conclude that the difference of accuracy of cross validation model and random forest model is very less (equal to 0.60235%). Hence, our model is not overfitting, and we can use this model to train the test dataset. In case if this difference was more (say greater than 10%), then we would go back to feature engineering of random forest model in order to find a different set of variables for Random forests algorithm.



```
Console ~/Documents/Modules/Module Set 1-5/ ↗
> rf.5.cv.1
Random Forest

891 samples
 3 predictors
 2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 10 times)
Summary of sample sizes: 801, 802, 802, 803, 802, 801, ...
Resampling results across tuning parameters:

  mtry  Accuracy  Kappa
    2    0.8121316 0.5929087
    3    0.8089966 0.5852138

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was mtry = 2.
> |
```

Fig 7.4.4.1 Sample output for 10-fold cross validation with accuracy of 0.8121316

7.5 Single Decision Trees

We will use single decision trees for better understanding of what's going on with our features or variables. Obviously Random Forests are far more powerful than single trees, but single trees have the advantage of being easier to understand.

The technique applied in this project is a manual implementation of a simple machine learning model, the decision tree. A decision tree splits a set of data into smaller and smaller groups (called nodes), by one feature at a time. Each time a subset of the data is split, our predictions become more accurate if each of the resulting subgroups are more homogeneous (contain similar labels) than before. The advantage of having a computer do things for us is that it will be more exhaustive and more precise than our manual exploration

above. A decision tree is just one of many models that come from supervised learning. In supervised learning, we attempt to use features of the data to predict or model things with objective outcome labels. That is to say, each of our data points has a known outcome value, such as a categorical, discrete label like 'Survived', or a numerical, continuous value like predicting the price of a house.

7.5.1 Algorithm

We have already discussed the Single Decision Tree Algorithm in Random Forests. Here, we will summarize this algorithm.

A decision tree is a simple representation for classifying examples. For this section, assume that all of the input features have finite discrete domains, and there is a single target feature called the classification. Each element of the domain of the classification is called a class. A decision tree or a classification tree is a tree in which each internal (non-leaf) node is labeled with an input feature. The arcs coming from a node labeled with an input feature are labeled with each of the possible values of the target or output feature or the arc leads to a subordinate decision node on a different input feature. Each leaf of the tree is labeled with a class or a probability distribution over the classes.

A tree can be "learned" by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called recursive partitioning. See the examples illustrated in the figure for spaces that have and have not been partitioned using recursive partitioning, or recursive binary splitting. The recursion is completed when the subset at a node has all the same value of the target variable, or when splitting no longer adds value to the predictions. This process of top-down induction of decision trees (TDIDT) is an example of a greedy algorithm, and it is by far the most common strategy for learning decision trees from data.

In data mining, decision trees can be described also as the combination of mathematical and computational techniques to aid the description, categorization and generalization of a given set of data.

Data comes in records of the form:

$$(x, Y) = (x_1, x_2, x_3, \dots, x_k, Y)$$

The dependent variable, Y , is the target variable that we are trying to understand, classify or generalize. The vector x is composed of the input variables, x_1, x_2, x_3 etc., that are used for that task. You can see the figure in the example given bellow, which is giving an apt description.



Fig 7.5.1.1 Construction of DecisionTree

7.5.2 Application

Our first decision tree is based upon the variables pclass, title, family.size.

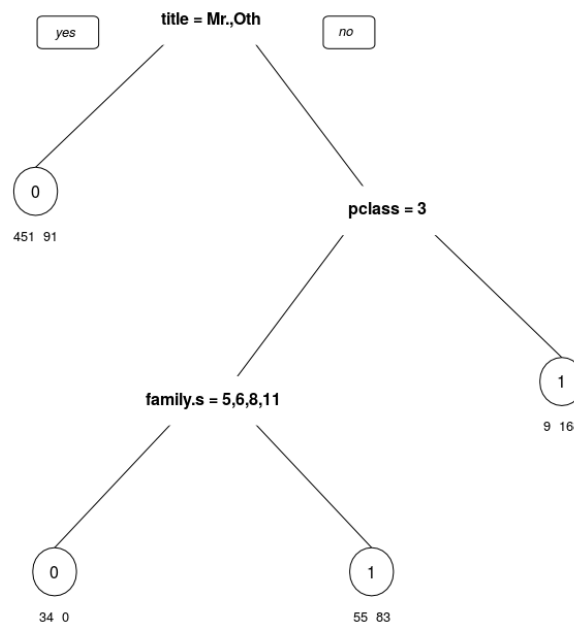


Fig 7.5.2.1 Decision Tree with Variables pclass, title, family.size

The plot brings out some interesting lines of investigation. Namely:

1. Titles of "Mr." and "Other" are predicted to perish at an overall accuracy rate of 83.2 %. (Error = $91/(451+91)$ & Accuracy = $1-\text{Error} = 0.832013$)
2. Titles of "Master.", "Miss.", & "Mrs." in 1st & 2nd class are predicted to survive at an overall accuracy rate of 94.9%.

3. Titles of "Master.", "Miss.", & "Mrs." in 3rd class with family sizes equal to 5, 6, 8, & 11 are predicted to perish with 100% accuracy.
4. Titles of "Master.", "Miss.", & "Mrs." in 3rd class with family sizes not equal to 5, 6, 8, or 11 are predicted to survive with 59.6% accuracy.

Our second decision tree is based upon the feature engineering of title variable. We have re-modified the title, and after that we are getting the following decision tree.

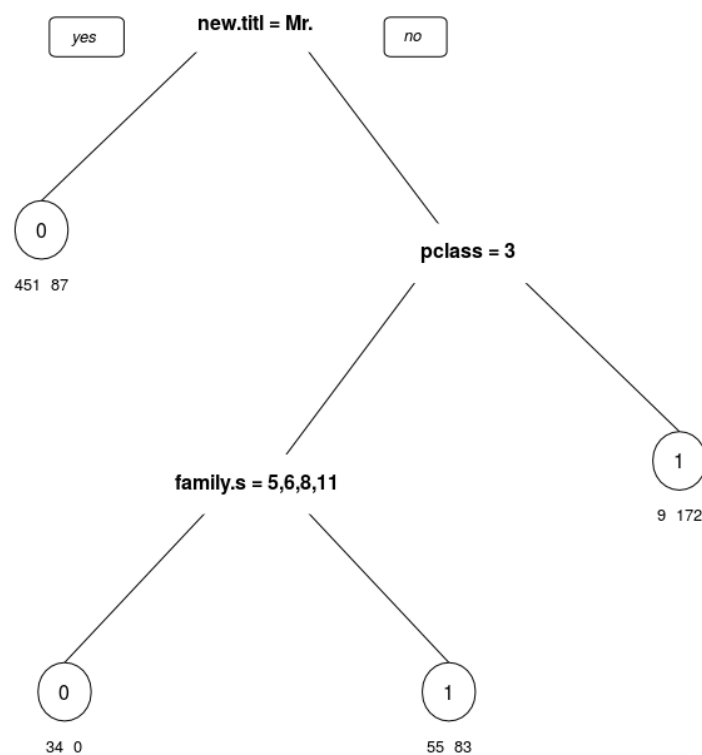


Fig 7.5.2.2 Decision Tree with Variables pclass, new.title, family.size

In the above graph, we can observe that the above decision tree is more accurate is compared to the first decision tree. Titles of "Mr." and "Other" are predicted to perish at an overall accuracy rate of 83.82%, hence our accuracy slightly increased. Same thing is happened with the node where pclass = 3 and family.size is not equal to 5, 6, 8 or 11, where accuracy has increased from 94.9% to 95.027%

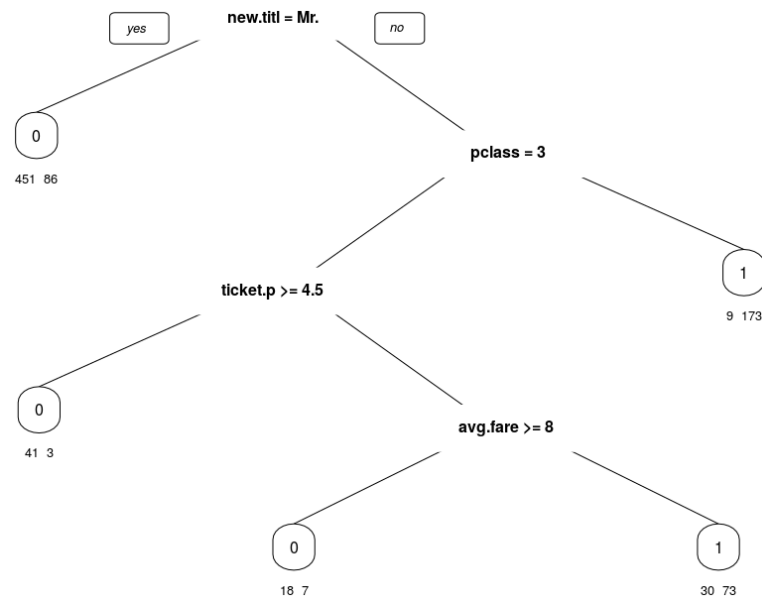


Fig 7.5.2.3 Decision Trees with Variables- pclass, new.title, family.size, ticket.party.size, avg.fare

Our third decision tree is based upon the feature engineering of ticket and fare variable. In this model, we have excluded family size, and have used the variables named pclass, new.title, family.size, ticket.party.size, avg.fare. We have used the re-modified title of previous decision tree, and luckily our accuracy has slightly increased. The accuracy of title has increased to 83.985%, with various other variables, that were not available in previous decision trees. We will use this model as our final model for making the predictions. We are using the variables of this decision tree as the input variables of our random forests. And the OOB estimate of error rate for this model is 16.16%. In other words, our model has an accuracy of 83.84%, which is higher than all our previous cases. After kaggle submission, our training data has an accuracy of 80.861%.

7.6 Final Shiny Web-App

Shiny is a new package from RStudio that makes it incredibly easy to build interactive web applications with R. Shiny enables you to write powerful interactive web applications entirely in R. Using R you create a user interface and server and Shiny compiles your code into the HTML, CSS and JavaScript needed to display your

application on the web. What makes a Shiny app particularly powerful is that it can execute R code on the backend so your app can perform any R calculation you can run on your desktop. Perhaps you want your app to slice and dice a dataset based on user inputs. Or maybe you want your web app to run linear models, GAMs or machine learning methods on user-selected data. In either case, Shiny can help. Creating and running simple web applications is relatively easy and there are great resources for doing this. But when you want more control of the application functionality understanding the key concepts is challenging. Shiny apps have two components:

- a user-interface script
- a server script
- The user-interface (ui) script controls the layout and appearance of your app. It is defined in a source script named ui.R. The server.R script contains the instructions that your computer needs to build your app.



```

ui.R
library(shiny)

# Define UI for application that draws a histogram
shinyUI(fluidPage(

  # Application title
  titlePanel("Hello Shiny!"),

  # Sidebar with a slider input for the number of bins
  sidebarLayout(
    sidebarPanel(
      sliderInput("bins",
        "Number of bins:",
        min = 1,
        max = 50,
        value = 30)
    ),

    # Show a plot of the generated distribution
    mainPanel(
      plotOutput("distPlot")
    )
  )
))

server.R
library(shiny)

# Define server logic required to draw a histogram
shinyServer(function(input, output) {

  # Expression that generates a histogram. The expression is
  # wrapped in a call to renderPlot to indicate that:
  #
  # 1) It is "reactive" and therefore should re-execute automatically
  #    when inputs change
  # 2) Its output type is a plot

  output$distPlot <- renderPlot({
    x <- faithful[, 2] # Old Faithful Geyser data
    bins <- seq(min(x), max(x), length.out = input$bins + 1)

    # draw the histogram with the specified number of bins
    hist(x, breaks = bins, col = 'darkgray', border = 'white')
  })
})

```

Fig 7.6.1 ui.R & server.R (sample script)

At one level, the Hello Shiny server.R script is very simple. The script does some calculations and then plots a histogram with the requested number of bins.

However, you'll also notice that most of the script is wrapped in a call to `renderPlot`. Your R session will be busy while the Hello Shiny app is active, so you will not be able to run any R commands. R is monitoring the app and executing the app's reactions. To get your R session back, hit escape or click the stop sign icon (found in the upper right corner of the RStudio console panel). In our Shiny app, we have four components, described below. Our Shiny App has 5 tabs.

7.6.1 Context Tab

The context tab is our home page, which is giving a brief overview of what happened in the RMS Titanic tragedy.

Context The data Age repartition Decision tree Did he survive ?

The story

The sinking of the RMS Titanic is one of the most infamous shipwrecks in history. On April 15, 1912, during her maiden voyage, the Titanic sank after colliding with an iceberg, killing 1502 out of 2224 passengers and crew. This sensational tragedy shocked the international community and led to better safety regulations for ships.



Fig 7.6.1 First Tab

7.6.2 The Data Tab



Fig 7.6.2 Second Tab

This tab is plotting real-time graphs for different data exploration according to the variables/features selected by the user. For e.g., in the above case, we are plotting graphs

of where x-axis denotes age, y-axis denotes fare and the variables that distinguish our plot is sex. Also, we are subdividing the graph into two facets, namely Survive and Class.

7.6.3 Age Repartition Tab

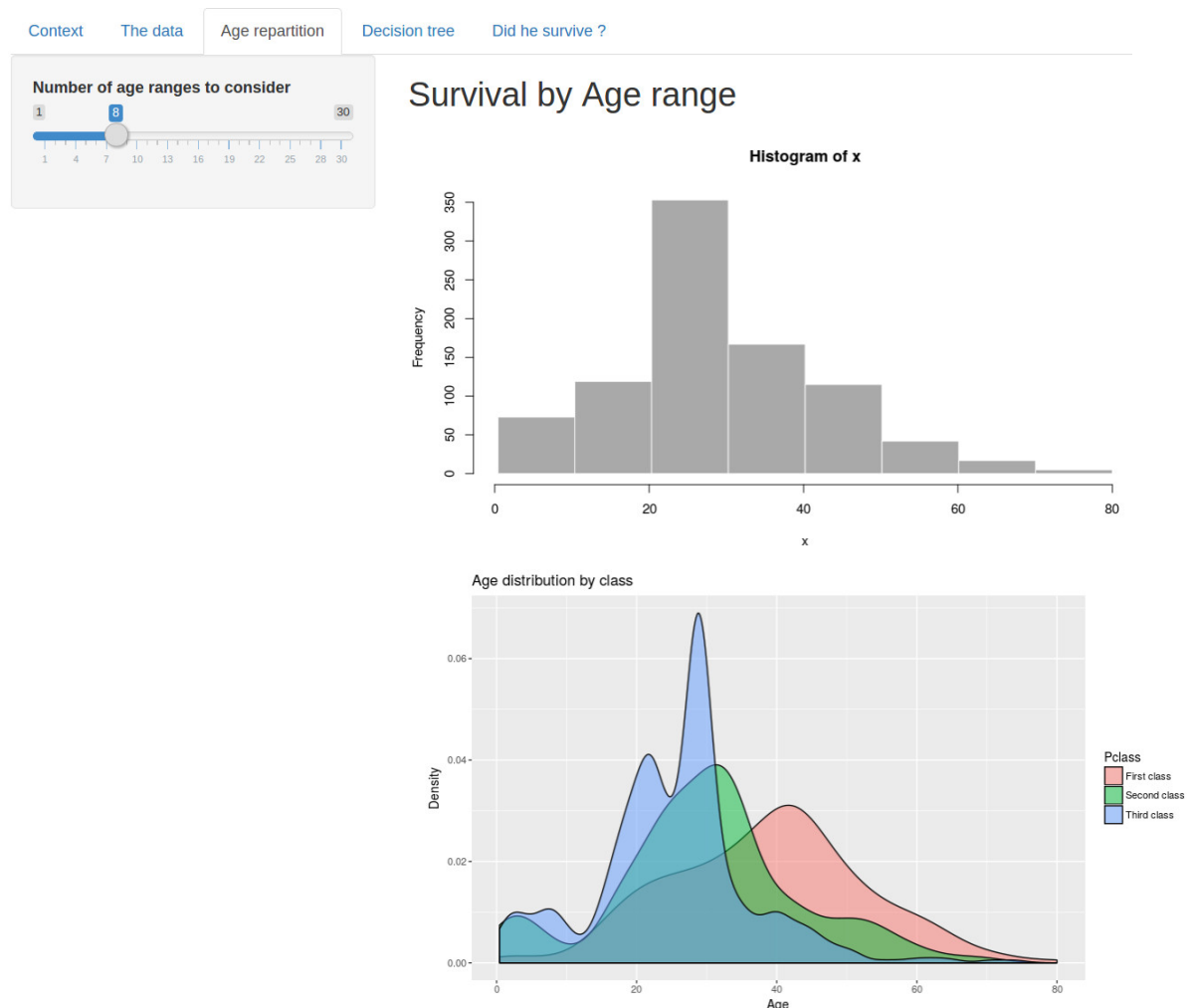


Fig 7.6.3 Third Tab

In this tab, we are plotting a dynamic histogram based upon the user input of age variable, and survival. The range of age depends upon the user input. The second graph is a static graph, which shows the age distribution by class vs. Survival density.

7.6.4 Decision Tree Tab

In this tab, we are plotting the decision trees on the basis of the variables defined by user. For the figure bellow, we are plotting a decision tree on the basis of Sex, Age and Class. We can give custom user inputs and see the changes that occur in decision tree. Also, we

can use these decision tree for checking weather a person with a particular set of inputs will survive in this tragedy or not.

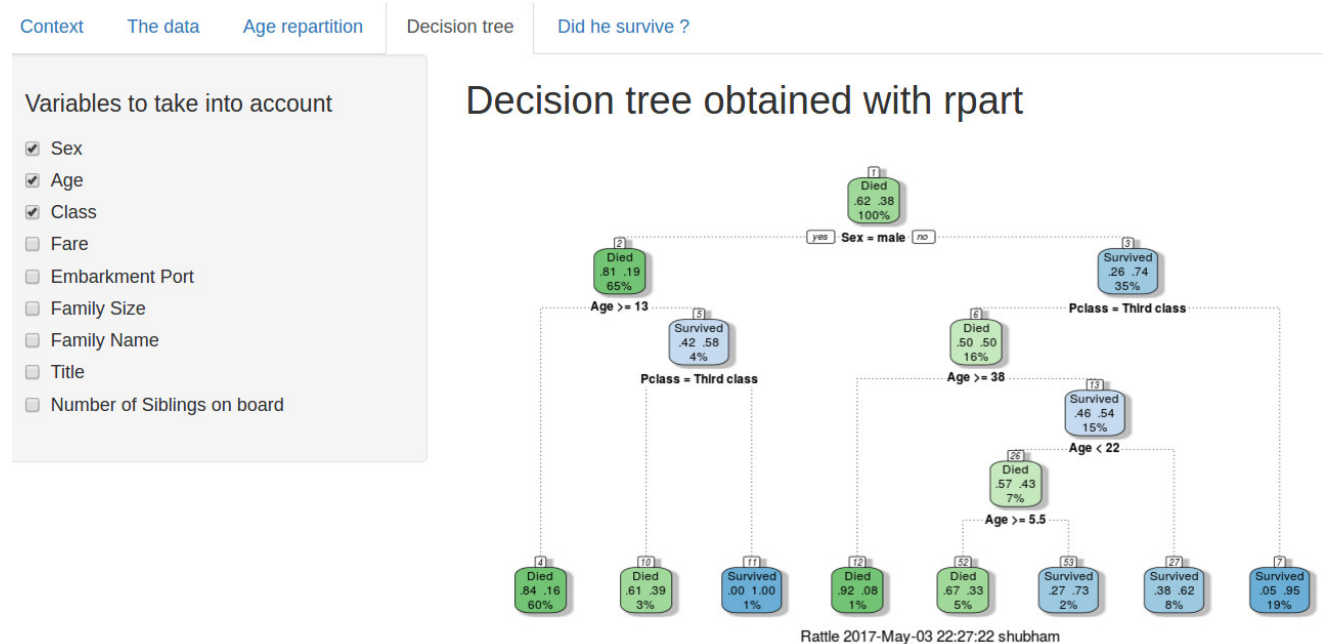


Fig 7.6.4 Fourth Tab

7.6.5 Did he Survive Tab

In this tab, we are showing that whether a person survives or not, on the basis of input variables given by the user. Only one set of input variables can be given at a time.

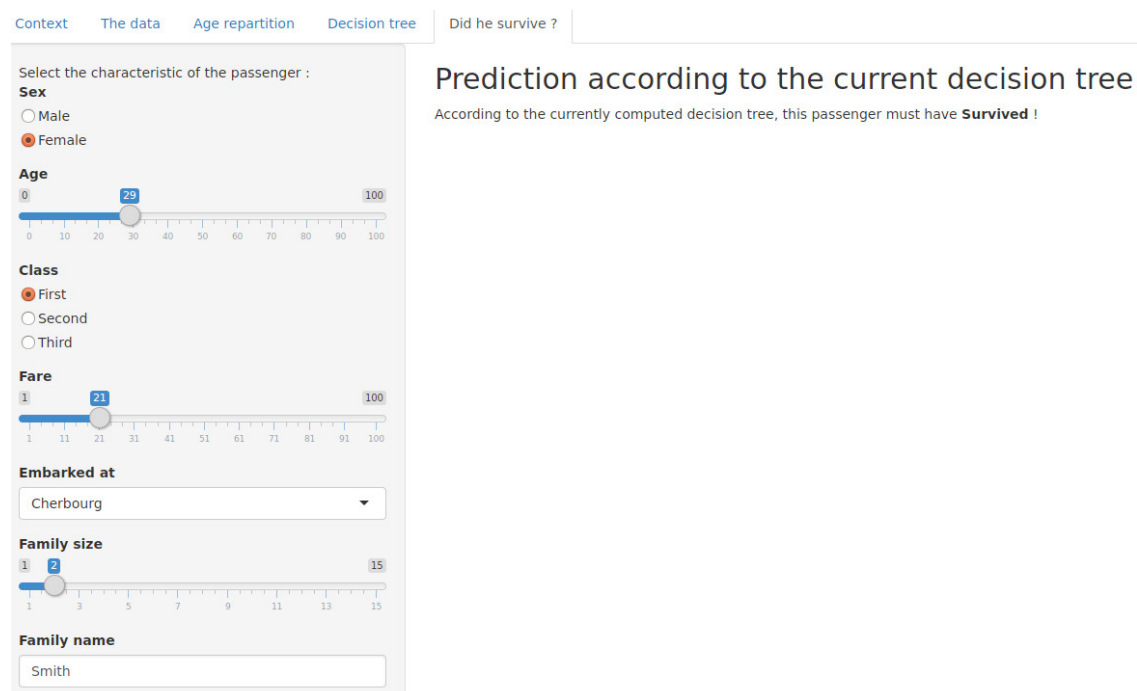


Fig 7.6.5 Fifth Tab

7.7 CODE

7.7.1 Module1

```
train <- read.csv("train.csv", header = TRUE)
test <- read.csv("test.csv", header = TRUE)
test.survived <- data.frame(survived = rep("None", nrow(test)), test[,])
data.combined <- rbind(train, test.survived)
str(data.combined)
data.combined$survived <- as.factor(data.combined$survived)
data.combined$pclass <- as.factor(data.combined$pclass)
table(data.combined$survived)
table(data.combined$pclass)
library(ggplot2)
train$pclass <- as.factor(train$pclass)
ggplot(train, aes(x = pclass, fill = factor(survived))) +
  geom_bar() +
  xlab("Pclass") +
  ylab("Total Count") +
  labs(fill = "Survived")
head(as.character(train$name)) # Data is in the format- Last Name, Title_First Name
length(unique(as.character(data.combined$name)))
dup.names <-
as.character(data.combined[which(duplicated(as.character(data.combined$name))),
"name"])
data.combined[which(data.combined$name %in% dup.names),]
library(stringr)
misses <- data.combined[which(str_detect(data.combined$name, "Miss.")),]
misses[1:5,]
mrses <- data.combined[which(str_detect(data.combined$name, "Mrs.")), ]
mrses[1:5,
```



```
males <- data.combined[which(train$sex == "male"), ]
males[1:5,]
extractTitle <- function(name) {
  name <- as.character(name)

  if (length(grep("Miss.", name)) > 0) {
    return ("Miss.")
  } else if (length(grep("Master.", name)) > 0) {
    return ("Master.")
  } else if (length(grep("Mrs.", name)) > 0) {
    return ("Mrs.")
  } else if (length(grep("Mr.", name)) > 0) {
    return ("Mr.")
  } else {
    return ("Other")
  }
}
titles <- NULL
for (i in 1:nrow(data.combined)) {
  titles <- c(titles, extractTitle(data.combined[i,"name"]))
}
data.combined$title <- as.factor(titles)
ggplot(data.combined[1:891,], aes(x = title, fill = survived)) + # fill means colour code it
on the basis of value of survived
  stat_count(width = 0.5) +
  facet_wrap(~pclass) +
  ggtitle("Pclass") +
  xlab("Title") +
  ylab("Total Count") +
  labs(fill = "Survived")
table(data.combined$sex)
ggplot(data.combined[1:891,], aes(x = sex, fill = survived)) +
```

```
stat_count(width = 0.5) +
facet_wrap(~pclass) +
ggtitle("Pclass") +
xlab("Sex") +
ylab("Total Count") +
labs(fill = "Survived")
summary(data.combined$age)
summary(data.combined[1:891,"age"])
ggplot(data.combined[1:891,], aes(x = age, fill = survived)) +
  facet_wrap(~sex + pclass) +
  geom_histogram(binwidth = 10) +
  xlab("Age") +
  ylab("Total Count")
boys <- data.combined[which(data.combined$title == "Master."),]
summary(boys$age)
ggplot(boys[boys$survived != "None",], aes(x = age, fill = survived)) +
  facet_wrap(~pclass) +
  geom_histogram(binwidth = 5) +
  ggtitle("Age for 'Master' by Pclass") +
  xlab("Age") +
  ylab("Total Count")
misses <- data.combined[which(data.combined$title == "Miss."),]
summary(misses$age)
ggplot(misses[misses$survived != "None",], aes(x = age, fill = survived)) +
  facet_wrap(~pclass) +
  geom_histogram(binwidth = 5) +
  ggtitle("Age for 'Miss.' by Pclass") +
  xlab("Age") +
  ylab("Total Count")
misses.alone <- misses[which(misses$sibsp == 0 & misses$parch == 0),]
summary(misses.alone$age)
```

length(which(misses.alone\$age <= 14.5)) # Which is = to 4, hence we have very less female children.

```
summary(data.combined$sibsp)
```

```
length(unique(data.combined$sibsp))
```

```
data.combined$sibsp <- as.factor(data.combined$sibsp)
```

```
ggplot(data.combined[1:891,], aes(x = sibsp, fill = survived)) +
```

```
  stat_count(width = 1) +
```

```
  facet_wrap(~pclass + title) +
```

```
  ggtitle("Pclass, Title") +
```

```
  xlab("SibSp") +
```

```
  ylab("Total Count") +
```

```
  ylim(0,300) +
```

```
  labs(fill = "Survived")
```

```
data.combined$parch <- as.factor(data.combined$parch)
```

```
ggplot(data.combined[1:891,], aes(x = parch, fill = survived)) +
```

```
  stat_count(width = 1) +
```

```
  facet_wrap(~pclass + title) +
```

```
  ggtitle("Pclass, Title") +
```

```
  xlab("ParCh") +
```

```
  ylab("Total Count") +
```

```
  ylim(0,300) +
```

```
  labs(fill = "Survived")
```

```
temp.sibsp <- c(train$sibsp, test$sibsp)
```

```
temp.parch <- c(train$parch, test$parch)
```

```
data.combined$family.size <- as.factor(temp.sibsp + temp.parch + 1)
```

```
ggplot(data.combined[1:891,], aes(x = family.size, fill = survived)) +
```

```
  stat_count(width = 1) +
```

```
  facet_wrap(~pclass + title) +
```

```
  ggtitle("Pclass, Title") +
```

```
  xlab("family.size") +
```

```
  ylab("Total Count") +
```

```
  ylim(0,300) +
```

```
labs(fill = "Survived")
ggplot(data.combined[1:891,], aes(x = family.size, fill = survived)) +
  stat_count(width = 1) +
  facet_wrap(~pclass) +
  ggtitle("Pclass") +
  xlab("family.size") +
  ylab("Total Count") +
  ylim(0,300) +
  labs(fill = "Survived")
str(data.combined$ticket)
data.combined$ticket <- as.character(data.combined$ticket)
data.combined$ticket[1:20]
ticket.first.char <- ifelse(data.combined$ticket == "", " ", substr(data.combined$ticket, 1,
1)) # substr(x, start, stop)
unique(ticket.first.char) # Find the unique values in ticket.first.char variable
data.combined$ticket.first.char <- as.factor(ticket.first.char)
ggplot(data.combined[1:891,], aes(x = ticket.first.char, fill = survived)) +
  geom_bar() +
  ggtitle("Survivability by ticket.first.char") +
  xlab("ticket.first.char") +
  ylab("Total Count") +
  ylim(0,350) +
  labs(fill = "Survived")
ggplot(data.combined[1:891,], aes(x = ticket.first.char, fill = survived)) +
  geom_bar() +
  facet_wrap(~pclass) +
  ggtitle("Pclass") +
  xlab("ticket.first.char") +
  ylab("Total Count") +
  ylim(0,150) +
  labs(fill = "Survived")
ggplot(data.combined[1:891,], aes(x = ticket.first.char, fill = survived)) +
```

```
geom_bar() +  
facet_wrap(~pclass + title) +  
ggtitle("Pclass, Title") +  
xlab("ticket.first.char") +  
ylab("Total Count") +  
ylim(0,200) +  
labs(fill = "Survived")  
summary(data.combined$fare)  
length(unique(data.combined$fare))  
ggplot(data.combined, aes(x = fare)) +  
  stat_count(width = 5) +  
  ggtitle("Combined Fare Distribution") +  
  xlab("Fare") +  
  ylab("Total Count") +  
  ylim(0,200)  
ggplot(data.combined[1:891,], aes(x = fare, fill = survived)) +  
  stat_count(width = 5) +  
  facet_wrap(~pclass + title) +  
  ggtitle("Pclass, Title") +  
  xlab("fare") +  
  ylab("Total Count") +  
  ylim(0,50) +  
  labs(fill = "Survived")  
str(data.combined$cabin)  
data.combined$cabin <- as.character(data.combined$cabin)  
data.combined$cabin[1:100] # we use 100 because lost of cabin entries are empty  
data.combined[which(data.combined$cabin == ""), "cabin"] <- "U"  
data.combined$cabin[1:100]  
cabin.first.char <- as.factor(substr(data.combined$cabin, 1, 1))  
str(cabin.first.char)  
levels(cabin.first.char)  
data.combined$cabin.first.char <- cabin.first.char
```

```
ggplot(data.combined[1:891,], aes(x = cabin.first.char, fill = survived)) +  
  geom_bar() +  
  ggtitle("Survivability by cabin.first.char") +  
  xlab("cabin.first.char") +  
  ylab("Total Count") +  
  ylim(0,750) +  
  labs(fill = "Survived")  
ggplot(data.combined[1:891,], aes(x = cabin.first.char, fill = survived)) +  
  geom_bar() +  
  facet_wrap(~pclass) +  
  ggtitle("Survivability by cabin.first.char") +  
  xlab("Pclass") +  
  ylab("Total Count") +  
  ylim(0,500) +  
  labs(fill = "Survived")  
ggplot(data.combined[1:891,], aes(x = cabin.first.char, fill = survived)) +  
  geom_bar() +  
  facet_wrap(~pclass + title) +  
  ggtitle("Pclass, Title") +  
  xlab("cabin.first.char") +  
  ylab("Total Count") +  
  ylim(0,500) +  
  labs(fill = "Survived")  
data.combined$cabin.multiple <- as.factor(ifelse(str_detect(data.combined$cabin, " "),  
"Y", "N"))  
ggplot(data.combined[1:891,], aes(x = cabin.multiple, fill = survived)) +  
  geom_bar() +  
  facet_wrap(~pclass + title) +  
  ggtitle("Pclass, Title") +  
  xlab("cabin.multiple") +  
  ylab("Total Count") +  
  ylim(0,350) +
```

```
labs(fill = "Survived")
ggplot(data.combined[1:891,], aes(x = cabin.multiple, fill = survived)) +
  geom_bar() +
  facet_wrap(~pclass) +
  ggtitle("Pclass") +
  xlab("cabin.multiple") +
  ylab("Total Count") +
  ylim(0,350) +
  labs(fill = "Survived")
str(data.combined$embarked)
levels(data.combined$embarked)
ggplot(data.combined[1:891,], aes(x = embarked, fill = survived)) +
  geom_bar() +
  facet_wrap(~pclass + title) +
  ggtitle("Pclass, Title") +
  xlab("embarked") +
  ylab("Total Count") +
  ylim(0,300) +
  labs(fill = "Survived")
ggplot(data.combined[1:891,], aes(x = embarked, fill = survived)) +
  geom_bar() +
  facet_wrap(~pclass) +
  ggtitle("Pclass") +
  xlab("embarked") +
  ylab("Total Count") +
  ylim(0,300) +
  labs(fill = "Survived")
```

7.7.2 Module2.1

```
library(e1071)
set.seed(1)
extractFeatures <- function(data) {
```

```
features <- c("Pclass",
             "Age",
             "Sex",
             "Parch",
             "SibSp",
             "Fare",
             "Embarked")

fea <- data[,features]
fea$Age[is.na(fea$Age)] <- -1
fea$Fare[is.na(fea$Fare)] <- median(fea$Fare, na.rm=TRUE)
fea$Embarked[fea$Embarked==""] = "S"
fea$Sex <- as.factor(fea$Sex)
fea$Embarked <- as.factor(fea$Embarked)
return(fea)
}

train<-read.csv("train.csv",header=TRUE,sep="," ,stringsAsFactors=FALSE)
train<-within(train,{
  Survived<-as.factor(Survived)})
test<-read.csv("test.csv",header=TRUE,sep="," ,stringsAsFactors=FALSE)
result<-
read.csv("gender_submission.csv",header=TRUE,sep="," ,stringsAsFactors=FALSE)
result<-within(result,{
  Survived<-as.factor(Survived)})
fit.svm<-svm(train$Survived~.,data=extractFeatures(train))
svm.pred<-predict(fit.svm,extractFeatures(test))
table(svm.pred,result$Survived,dnn=c("Pred","Actul"))
mean(svm.pred == result$Survived) #0.9617225
submission.svm <- data.frame(PassengerId = test$PassengerId)
submission.svm$Survived <- svm.pred
write.csv(submission.svm, file = "2_svm_submission.csv", row.names=FALSE)
```


7.7.3 Module 2.2

```
library(e1071)
train <- read.csv("train.csv")
test <- read.csv("test.csv")
test$Survived <- NA
combi <- rbind(train, test)
combi$Name <- as.character(combi$Name)
combi$Title <- sapply(combi$Name, FUN=function(x) {strsplit(x, split='[,.]')[[1]][2]})
combi$Surname <- sapply(combi$Name, FUN=function(x) {strsplit(x,
split='[,.]')[[1]][1]})
combi$Title <- sub(' ', "", combi$Title)
combi$Title[combi$Title %in% c('Mme', 'Mlle', 'Ms', 'Miss')] <- 'Miss'
combi$Title[combi$Title %in% c('Rev', 'Dr')] <- 'Rev'
combi$Title[combi$Title %in% c('Capt', 'Don', 'Major', 'Sir')] <- 'Sir'
combi$Title[combi$Title %in% c('Dona', 'Lady', 'the Countess', 'Jonkheer')] <- 'Lady'
combi$Title <- factor(combi$Title)
combi$FamilySize <- combi$SibSp + combi$Parch + 1
combi$FamilyID <- paste(as.character(combi$FamilySize), combi$Surname, sep="")
combi$FamilyID[combi$FamilySize == 1] <- 'Singleton'
combi$FamilyID[1 < combi$FamilySize & combi$FamilySize <= 3] <- 'Small'
combi$FamilyID[3 < combi$FamilySize & combi$FamilySize <= 5] <- 'Normal'
combi$FamilyID[combi$FamilySize > 5] <- 'Large'
famIDs <- data.frame(table(combi$FamilyID))
famIDs <- famIDs[famIDs$Freq <= 2,]
combi$FamilyID[combi$FamilyID %in% famIDs$Var1] <- 'Small'
combi$FamilyID <- factor(combi$FamilyID)
Embark_Fare <- combi %>%
  filter( PassengerId != 62 & PassengerId != 830)
ggplot(Embark_Fare, aes(x = Embarked, y = Fare, fill = factor(Pclass))) +
  geom_boxplot() +
  geom_hline(aes(yintercept=80),
```

```
colour='red', linetype='dashed', lwd=2) +
scale_y_continuous(labels=dollar_format()) +
theme_few()
combi$Embarked[c(62,830)] = "C"
combi$Embarked <- factor(combi$Embarked)
which(is.na(combi$Fare))
combi$Fare[1044] <- median(combi[combi$Pclass == '3' & combi$Embarked == 'S',
]$Fare, na.rm = TRUE)
combi$Fare[1044] <- median(combi[combi$Pclass == '3' & combi$Embarked == 'S',
]$Fare, na.rm = TRUE)
summary(combi$Age)
combi$Age2 <- combi$Age
Agefit <- rpart(Age2 ~ Pclass + Sex + SibSp + Parch + Fare + Embarked + Title +
FamilySize,
               data=combi[!is.na(combi$Age2),], method="anova")
combi$Age2[is.na(combi$Age2)] <- predict(Agefit, combi[is.na(combi$Age2),])
par(mfrow=c(1,2))
hist(combi$Age, freq=F, main='Age: Original Data',
     col='darkgreen', ylim=c(0,0.04))
hist(combi$Age2, freq=F, main='Age: Original Data',
     col='darkgreen', ylim=c(0,0.04))
combi$Age <- combi$Age2
combi$Age2 <- NULL
sum(is.na(combi$Age))
combi$Child[combi$Age < 18] <- 'Child'
combi$Child[combi$Age >= 18] <- 'Adult'
combi$Mother <- 'Not Mother'
combi$Mother[combi$Sex == 'female' & combi$Parch > 0 & combi$Age > 18 &
combi$Title != 'Miss'] <- 'Mother'
combi$Child <- factor(combi$Child)
combi$Mother <- factor(combi$Mother)
```

```
ticket.count <- aggregate(combi$Ticket, by=list(combi$Ticket), function(x) sum(
!is.na(x) ))
combi$Price<-apply(combi, 1, function(x) as.numeric(x["Fare"]) /
ticket.count[which(ticket.count[, 1] == x["Ticket"]), 2])
pclass.price<-aggregate(combi$Price, by = list(combi$Pclass), FUN = function(x)
median(x, na.rm = T))
combi[which(combi$Price==0), "Price"] <- apply(combi[which(combi$Price==0), ], 1,
function(x) pclass.price[pclass.price[, 1]==x["Pclass"], 2])
combi$TicketCount<-apply(combi, 1, function(x) ticket.count[which(ticket.count[, 1] ==
x["Ticket"]), 2])
train <- combi[1:891,]
test <- combi[892:1309,]
set.seed(415)
inTrain<-createDataPartition(train$Survived, p = 0.8)[[1]]
svm.new <- svm(as.factor(Survived) ~ Pclass + Sex + Age + SibSp + Parch + Fare
+ Embarked + Title + FamilySize + FamilyID + Price + TicketCount
+ Child + Mother, data = train)
summary(svm.new)
confusionMatrix(train[-inTrain,"Survived"], predict(svm.new, train[-inTrain,], OOB =
TRUE, type = "response"))$overall[1]
test$Survived = 0
Prediction <- predict(svm.new, test, OOB=TRUE, type = "response")
submit <- data.frame(PassengerId = test$PassengerId, Survived = Prediction)
write.csv(submit, file = "svm.csv", row.names = FALSE)
```

7.7.4 Module 3

```
train <- read.csv("train.csv", header = TRUE)
test <- read.csv("test.csv", header = TRUE)
test.survived <- data.frame(survived = rep("None", nrow(test)), test[,])
data.combined <- rbind(train, test.survived)
str(data.combined)
data.combined$survived <- as.factor(data.combined$survived)
```

```
data.combined$pclass <- as.factor(data.combined$pclass)
train$pclass <- as.factor(train$pclass)
head(as.character(train$name))
length(unique(as.character(data.combined$name)))
dup.names <-
as.character(data.combined[which(duplicated(as.character(data.combined$name))),
"name"])
data.combined[which(data.combined$name %in% dup.names),]
library(stringr)
misses <- data.combined[which(str_detect(data.combined$name, "Miss.")),]
misses[1:5,]
mrses <- data.combined[which(str_detect(data.combined$name, "Mrs.")), ]
mrses[1:5,]
males <- data.combined[which(train$sex == "male"), ]
males[1:5,]
extractTitle <- function(name) {
  name <- as.character(name)
  if (length(grep("Miss.", name)) > 0) {
    return ("Miss.")
  } else if (length(grep("Master.", name)) > 0) {
    return ("Master.")
  } else if (length(grep("Mrs.", name)) > 0) {
    return ("Mrs.")
  } else if (length(grep("Mr.", name)) > 0) {
    return ("Mr.")
  } else {
    return ("Other")
  }
}
titles <- NULL
for (i in 1:nrow(data.combined)) {
  titles <- c(titles, extractTitle(data.combined[i,"name"]))
}
```

```
}  
data.combined$title <- as.factor(titles)  
boys <- data.combined[which(data.combined$title == "Master."),]  
summary(boys$age)  
misses <- data.combined[which(data.combined$title == "Miss."),]  
summary(misses$age)  
misses.alone <- misses[which(misses$sibsp == 0 & misses$parch == 0),]  
summary(misses.alone$age)  
length(which(misses.alone$age <= 14.5))  
summary(data.combined$sibsp)  
length(unique(data.combined$sibsp))  
data.combined$sibsp <- as.factor(data.combined$sibsp)  
temp.sibsp <- c(train$sibsp, test$sibsp)  
temp.parch <- c(train$parch, test$parch)  
data.combined$family.size <- as.factor(temp.sibsp + temp.parch + 1)  
data.combined$ticket <- as.character(data.combined$ticket)  
data.combined$ticket[1:20]  
ticket.first.char <- ifelse(data.combined$ticket == "", " ", substr(data.combined$ticket, 1,  
1))  
unique(ticket.first.char)  
data.combined$ticket.first.char <- as.factor(ticket.first.char)  
str(data.combined$cabin)  
data.combined$cabin <- as.character(data.combined$cabin)  
data.combined$cabin[1:100]  
data.combined[which(data.combined$cabin == ""), "cabin"] <- "U"  
data.combined$cabin[1:100]  
cabin.first.char <- as.factor(substr(data.combined$cabin, 1, 1))  
str(cabin.first.char)  
levels(cabin.first.char)  
data.combined$cabin.first.char <- cabin.first.char  
library(randomForest)  
train_rf <- data.combined[1:891,]
```

```
test_rf <- data.combined[892:1309,]
rf_model <- randomForest(factor(survived) ~ pclass + sex + age + sibsp + parch + fare +
embarked + title + family.size, data = train_rf, na.action = na.exclude)
importance <- importance(rf_model)
varImportance <- data.frame(Variables = row.names(importance),
                             Importance = round(importance[, 'MeanDecreaseGini'], 2))
rankImportance <- varImportance %>%
  mutate(Rank = paste0('#', dense_rank(desc(Importance))))
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, colour = 'red') +
  labs(x = 'Variables') +
  coord_flip() +
  theme_few()

rf.train.1 <- data.combined[1:891, c("pclass", "title")] # Create a new rf.train.1
dataframe and put the pclass and title values
rf.label <- as.factor(train$survived)
set.seed(1234)
rf.1 <- randomForest(x = rf.train.1, y = rf.label, importance = TRUE, ntree = 1000)
rf.1
varImpPlot(rf.1)

rf.train.2 <- data.combined[1:891, c("pclass", "title", "sibsp")]
set.seed(1234)
rf.2 <- randomForest(x = rf.train.2, y = rf.label, importance = TRUE, ntree = 1000)
rf.2
varImpPlot(rf.2)

rf.train.3 <- data.combined[1:891, c("pclass", "title", "parch")]
set.seed(1234)
rf.3 <- randomForest(x = rf.train.3, y = rf.label, importance = TRUE, ntree = 1000)
rf.3
```

```
varImpPlot(rf.3)
rf.train.4 <- data.combined[1:891, c("pclass", "title", "sibsp", "parch")]
set.seed(1234)
rf.4 <- randomForest(x = rf.train.4, y = rf.label, importance = TRUE, ntree = 1000)
rf.4
varImpPlot(rf.4)
rf.train.5 <- data.combined[1:891, c("pclass", "title", "family.size")]
set.seed(1234)
rf.5 <- randomForest(x = rf.train.5, y = rf.label, importance = TRUE, ntree = 1000)
rf.5
varImpPlot(rf.5)
rf.train.6 <- data.combined[1:891, c("pclass", "title", "sibsp", "family.size")]
set.seed(1234)
rf.6 <- randomForest(x = rf.train.6, y = rf.label, importance = TRUE, ntree = 1000)
rf.6
varImpPlot(rf.6)
rf.train.7 <- data.combined[1:891, c("pclass", "title", "parch", "family.size")]
set.seed(1234)
rf.7 <- randomForest(x = rf.train.7, y = rf.label, importance = TRUE, ntree = 1000)
rf.7
varImpPlot(rf.7)
```

7.7.5 Module 4

```
train <- read.csv("train.csv", header = TRUE)
test <- read.csv("test.csv", header = TRUE)
test.survived <- data.frame(survived = rep("None", nrow(test)), test[,])
data.combined <- rbind(train, test.survived)
str(data.combined)
data.combined$survived <- as.factor(data.combined$survived)
data.combined$pclass <- as.factor(data.combined$pclass)
library(stringr)
misses <- data.combined[which(str_detect(data.combined$name, "Miss.")),]
```

```
misses[1:5,]
mrses <- data.combined[which(str_detect(data.combined$name, "Mrs.")), ]
mrses[1:5,]
males <- data.combined[which(train$sex == "male"), ]
males[1:5,]
extractTitle <- function(name) {
  name <- as.character(name)
  if (length(grep("Miss.", name)) > 0) {
    return ("Miss.")
  } else if (length(grep("Master.", name)) > 0) {
    return ("Master.")
  } else if (length(grep("Mrs.", name)) > 0) {
    return ("Mrs.")
  } else if (length(grep("Mr.", name)) > 0) {
    return ("Mr.")
  } else {
    return ("Other")
  }
}
titles <- NULL
for (i in 1:nrow(data.combined)) {
  titles <- c(titles, extractTitle(data.combined[i,"name"]))
}
data.combined$title <- as.factor(titles)
boys <- data.combined[which(data.combined$title == "Master."),]
summary(boys$age)
misses <- data.combined[which(data.combined$title == "Miss."),]
summary(misses$age)
misses.alone <- misses[which(misses$sibsp == 0 & misses$parch == 0),]
summary(misses.alone$age)
length(which(misses.alone$age <= 14.5))
summary(data.combined)
```



```
length(unique(data.combined$sibsp))
data.combined$sibsp <- as.factor(data.combined$sibsp)
data.combined$parch <- as.factor(data.combined$parch)
temp.sibsp <- c(train$sibsp, test$sibsp)
temp.parch <- c(train$parch, test$parch)
data.combined$family.size <- as.factor(temp.sibsp + temp.parch + 1)
data.combined$ticket <- as.character(data.combined$ticket)
data.combined$ticket[1:20]
ticket.first.char <- ifelse(data.combined$ticket == "", " ", substr(data.combined$ticket, 1,
1))
unique(ticket.first.char)
data.combined$ticket.first.char <- as.factor(ticket.first.char)
str(data.combined$cabin)
data.combined$cabin <- as.character(data.combined$cabin)
data.combined$cabin[1:100]
data.combined[which(data.combined$cabin == ""), "cabin"] <- "U"
data.combined$cabin[1:100]
cabin.first.char <- as.factor(substr(data.combined$cabin, 1, 1))
str(cabin.first.char)
levels(cabin.first.char)
data.combined$cabin.first.char <- cabin.first.char
data.combined$cabin.multiple <- as.factor(ifelse(str_detect(data.combined$cabin, "Y"),
"Y", "N"))
str(data.combined$embarked)
levels(data.combined$embarked)
library(randomForest)
rf.train.1 <- data.combined[1:891, c("pclass", "title")]
rf.label <- as.factor(train$survived)
set.seed(1234)
rf.1 <- randomForest(x = rf.train.1, y = rf.label, importance = TRUE, ntree = 1000)
rf.1
varImpPlot(rf.1)
```

```
rf.train.2 <- data.combined[1:891, c("pclass", "title", "sibsp")]
set.seed(1234)
rf.2 <- randomForest(x = rf.train.2, y = rf.label, importance = TRUE, ntree = 1000)
rf.2
varImpPlot(rf.2)
rf.train.3 <- data.combined[1:891, c("pclass", "title", "parch")]
set.seed(1234)
rf.3 <- randomForest(x = rf.train.3, y = rf.label, importance = TRUE, ntree = 1000)
rf.3
varImpPlot(rf.3)
rf.train.4 <- data.combined[1:891, c("pclass", "title", "sibsp", "parch")]
set.seed(1234)
rf.4 <- randomForest(x = rf.train.4, y = rf.label, importance = TRUE, ntree = 1000)
rf.4
varImpPlot(rf.4)
rf.train.5 <- data.combined[1:891, c("pclass", "title", "family.size")]
set.seed(1234)
rf.5 <- randomForest(x = rf.train.5, y = rf.label, importance = TRUE, ntree = 1000)
rf.5
varImpPlot(rf.5)
rf.train.6 <- data.combined[1:891, c("pclass", "title", "sibsp", "family.size")]
set.seed(1234)
rf.6 <- randomForest(x = rf.train.6, y = rf.label, importance = TRUE, ntree = 1000)
rf.6
varImpPlot(rf.6)
rf.train.7 <- data.combined[1:891, c("pclass", "title", "parch", "family.size")]
set.seed(1234)
rf.7 <- randomForest(x = rf.train.7, y = rf.label, importance = TRUE, ntree = 1000)
rf.7
varImpPlot(rf.7)
test.submit.df <- data.combined[892:1309, c("pclass", "title", "family.size")]
rf.5.preds <- predict(rf.5, test.submit.df)
```

```
table(rf.5.preds)
submit.df <- data.frame(PassengerId = rep(892:1309), Survived = rf.5.preds)
write.csv(submit.df, file = "RF_1.csv", row.names = FALSE)
library(caret)
library(doSNOW) # For using multi-cores
set.seed(2348)
cv.10.folds <- createMultiFolds(rf.label, k = 10, times = 10) # k is no of folds, times
means no of times. rf.label is a vector outcome
table(rf.label)
342 / 549 # 62.29508% people perished
table(rf.label[cv.10.folds[[33]]]) #Chosing 33rd fold
308 / 494 # 62.34818% <- it is nearly same
ctrl.1 <- trainControl(method = "repeatedcv", number = 10, repeats = 10,
                        index = cv.10.folds)
cl <- makeCluster(6, type = "SOCK") # 6 child processess and
registerDoSNOW(cl)
set.seed(34324)
rf.5.cv.1 <- train(x = rf.train.5, y = rf.label, method = "rf", tuneLength = 3,
                  ntree = 1000, trControl = ctrl.1)
stopCluster(cl)
rf.5.cv.1
set.seed(5983)
cv.5.folds <- createMultiFolds(rf.label, k = 5, times = 10)
ctrl.2 <- trainControl(method = "repeatedcv", number = 5, repeats = 10,
                        index = cv.5.folds)
cl <- makeCluster(6, type = "SOCK")
registerDoSNOW(cl)
set.seed(89472)
rf.5.cv.2 <- train(x = rf.train.5, y = rf.label, method = "rf", tuneLength = 3,
                  ntree = 1000, trControl = ctrl.2)
stopCluster(cl)
rf.5.cv.2
```

```
set.seed(37596)
cv.3.folds <- createMultiFolds(rf.label, k = 3, times = 10)
ctrl.3 <- trainControl(method = "repeatedcv", number = 3, repeats = 10,
                        index = cv.3.folds)
cl <- makeCluster(6, type = "SOCK")
registerDoSNOW(cl)
set.seed(94622)
rf.5.cv.3 <- train(x = rf.train.5, y = rf.label, method = "rf", tuneLength = 3,
                  ntree = 64, trControl = ctrl.3)
stopCluster(cl)
rf.5.cv.3
```

7.7.6 Module 5

```
train <- read.csv("train.csv", header = TRUE)
test <- read.csv("test.csv", header = TRUE)
test.survived <- data.frame(survived = rep("None", nrow(test)), test[,])
data.combined <- rbind(train, test.survived)
str(data.combined)
data.combined$survived <- as.factor(data.combined$survived)
data.combined$pclass <- as.factor(data.combined$pclass)
table(data.combined$survived)
table(data.combined$pclass)
library(ggplot2)
train$pclass <- as.factor(train$pclass)
ggplot(train, aes(x = pclass, fill = factor(survived))) +
  geom_bar() +
  xlab("Pclass") +
  ylab("Total Count") +
  labs(fill = "Survived")
head(as.character(train$name))
length(unique(as.character(data.combined$name)))
```

```
dup.names <-  
as.character(data.combined[which(duplicated(as.character(data.combined$name))),  
"name"])  
data.combined[which(data.combined$name %in% dup.names),]  
library(stringr)  
misses <- data.combined[which(str_detect(data.combined$name, "Miss.")),]  
misses[1:5,]  
mrses <- data.combined[which(str_detect(data.combined$name, "Mrs.")), ]  
mrses[1:5,]  
males <- data.combined[which(train$sex == "male"), ]  
males[1:5,]  
extractTitle <- function(name) {  
  name <- as.character(name)  
  if (length(grep("Miss.", name)) > 0) {  
    return ("Miss.")  
  } else if (length(grep("Master.", name)) > 0) {  
    return ("Master.")  
  } else if (length(grep("Mrs.", name)) > 0) {  
    return ("Mrs.")  
  } else if (length(grep("Mr.", name)) > 0) {  
    return ("Mr.")  
  } else {  
    return ("Other")  
  }  
}  
titles <- NULL  
for (i in 1:nrow(data.combined)) {  
  titles <- c(titles, extractTitle(data.combined[i,"name"]))  
}  
data.combined$title <- as.factor(titles)  
ggplot(data.combined[1:891,], aes(x = title, fill = survived)) +  
  geom_bar() +
```

```
facet_wrap(~pclass) +
ggtitle("Pclass") +
xlab("Title") +
ylab("Total Count") +
labs(fill = "Survived")
table(data.combined$sex)
ggplot(data.combined[1:891,], aes(x = sex, fill = survived)) +
  geom_bar() +
  facet_wrap(~pclass) +
  ggtitle("Pclass") +
  xlab("Sex") +
  ylab("Total Count") +
  labs(fill = "Survived")
summary(data.combined$age)
summary(data.combined[1:891,"age"])
ggplot(data.combined[1:891,], aes(x = age, fill = survived)) +
  facet_wrap(~sex + pclass) +
  geom_histogram(binwidth = 10) +
  xlab("Age") +
  ylab("Total Count")
boys <- data.combined[which(data.combined$title == "Master."),]
summary(boys$age)
misses <- data.combined[which(data.combined$title == "Miss."),]
summary(misses$age)
ggplot(misses[misses$survived != "None" & !is.na(misses$age),], aes(x = age, fill =
survived)) +
  facet_wrap(~pclass) +
  geom_histogram(binwidth = 5) +
  ggtitle("Age for 'Miss.' by Pclass") +
  xlab("Age") +
  ylab("Total Count")
misses.alone <- misses[which(misses$sibsp == 0 & misses$parch == 0),]
```

```
summary(misses.alone$age)
length(which(misses.alone$age <= 14.5))
summary(data.combined$sibsp)
length(unique(data.combined$sibsp))
data.combined$sibsp <- as.factor(data.combined$sibsp)
ggplot(data.combined[1:891,], aes(x = sibsp, fill = survived)) +
  geom_bar() +
  facet_wrap(~pclass + title) +
  ggtitle("Pclass, Title") +
  xlab("SibSp") +
  ylab("Total Count") +
  ylim(0,300) +
  labs(fill = "Survived")
data.combined$parch <- as.factor(data.combined$parch)
ggplot(data.combined[1:891,], aes(x = parch, fill = survived)) +
  geom_bar() +
  facet_wrap(~pclass + title) +
  ggtitle("Pclass, Title") +
  xlab("ParCh") +
  ylab("Total Count") +
  ylim(0,300) +
  labs(fill = "Survived")
temp.sibsp <- c(train$sibsp, test$sibsp)
temp.parch <- c(train$parch, test$parch)
data.combined$family.size <- as.factor(temp.sibsp + temp.parch + 1)
ggplot(data.combined[1:891,], aes(x = family.size, fill = survived)) +
  geom_bar() +
  facet_wrap(~pclass + title) +
  ggtitle("Pclass, Title") +
  xlab("family.size") +
  ylab("Total Count") +
  ylim(0,300) +
```

```
labs(fill = "Survived")
str(data.combined$ticket)
data.combined$ticket <- as.character(data.combined$ticket)
data.combined$ticket[1:20]
ticket.first.char <- ifelse(data.combined$ticket == "", " ", substr(data.combined$ticket, 1,
1))
unique(ticket.first.char)
data.combined$ticket.first.char <- as.factor(ticket.first.char)
ggplot(data.combined[1:891,], aes(x = ticket.first.char, fill = survived)) +
  geom_bar() +
  ggtitle("Survivability by ticket.first.char") +
  xlab("ticket.first.char") +
  ylab("Total Count") +
  ylim(0,350) +
  labs(fill = "Survived")
ggplot(data.combined[1:891,], aes(x = ticket.first.char, fill = survived)) +
  geom_bar() +
  facet_wrap(~pclass) +
  ggtitle("Pclass") +
  xlab("ticket.first.char") +
  ylab("Total Count") +
  ylim(0,150) +
  labs(fill = "Survived")
ggplot(data.combined[1:891,], aes(x = ticket.first.char, fill = survived)) +
  geom_bar() +
  facet_wrap(~pclass + title) +
  ggtitle("Pclass, Title") +
  xlab("ticket.first.char") +
  ylab("Total Count") +
  ylim(0,200) +
  labs(fill = "Survived")
summary(data.combined$fare)
```



```
length(unique(data.combined$fare))
ggplot(data.combined, aes(x = fare)) +
  geom_histogram(binwidth = 5) +
  ggtitle("Combined Fare Distribution") +
  xlab("Fare") +
  ylab("Total Count") +
  ylim(0,200)

ggplot(data.combined[1:891,], aes(x = fare, fill = survived)) +
  geom_histogram(binwidth = 5) +
  facet_wrap(~pclass + title) +
  ggtitle("Pclass, Title") +
  xlab("fare") +
  ylab("Total Count") +
  ylim(0,50) +
  labs(fill = "Survived")

str(data.combined$cabin)

data.combined$cabin <- as.character(data.combined$cabin)
data.combined$cabin[1:100]
data.combined[which(data.combined$cabin == ""), "cabin"] <- "U"
data.combined$cabin[1:100]

cabin.first.char <- as.factor(substr(data.combined$cabin, 1, 1))
str(cabin.first.char)
levels(cabin.first.char)

data.combined$cabin.first.char <- cabin.first.char

ggplot(data.combined[1:891,], aes(x = cabin.first.char, fill = survived)) +
  geom_bar() +
  ggtitle("Survivability by cabin.first.char") +
  xlab("cabin.first.char") +
  ylab("Total Count") +
  ylim(0,750) +
  labs(fill = "Survived")

ggplot(data.combined[1:891,], aes(x = cabin.first.char, fill = survived)) +
```

```
geom_bar() +  
facet_wrap(~pclass) +  
ggtitle("Survivability by cabin.first.char") +  
xlab("Pclass") +  
ylab("Total Count") +  
ylim(0,500) +  
labs(fill = "Survived")  
ggplot(data.combined[1:891,], aes(x = cabin.first.char, fill = survived)) +  
geom_bar() +  
facet_wrap(~pclass + title) +  
ggtitle("Pclass, Title") +  
xlab("cabin.first.char") +  
ylab("Total Count") +  
ylim(0,500) +  
labs(fill = "Survived")  
data.combined$cabin.multiple <- as.factor(ifelse(str_detect(data.combined$cabin, " "),  
"Y", "N"))  
ggplot(data.combined[1:891,], aes(x = cabin.multiple, fill = survived)) +  
geom_bar() +  
facet_wrap(~pclass + title) +  
ggtitle("Pclass, Title") +  
xlab("cabin.multiple") +  
ylab("Total Count") +  
ylim(0,350) +  
labs(fill = "Survived")  
str(data.combined$embarked)  
levels(data.combined$embarked)  
ggplot(data.combined[1:891,], aes(x = embarked, fill = survived)) +  
geom_bar() +  
facet_wrap(~pclass + title) +  
ggtitle("Pclass, Title") +  
xlab("embarked") +
```

```
ylab("Total Count") +
ylim(0,300) +
labs(fill = "Survived")
library(randomForest)
rf.train.1 <- data.combined[1:891, c("pclass", "title")]
rf.label <- as.factor(train$survived)
set.seed(1234)
rf.1 <- randomForest(x = rf.train.1, y = rf.label, importance = TRUE, ntree = 1000)
rf.1
varImpPlot(rf.1)
plot(rf.1, ylim=c(0,0.36))
legend('topright', colnames(rf.1$err.rate), col=1:3, fill=1:3)
rf.train.2 <- data.combined[1:891, c("pclass", "title", "sibsp")]
set.seed(1234)
rf.2 <- randomForest(x = rf.train.2, y = rf.label, importance = TRUE, ntree = 1000)
rf.2
varImpPlot(rf.2)
plot(rf.2, ylim=c(0,0.36))
legend('topright', colnames(rf.2$err.rate), col=1:3, fill=1:3)
rf.train.3 <- data.combined[1:891, c("pclass", "title", "parch")]
set.seed(1234)
rf.3 <- randomForest(x = rf.train.3, y = rf.label, importance = TRUE, ntree = 1000)
rf.3
varImpPlot(rf.3)
plot(rf.3, ylim=c(0,0.36))
legend('topright', colnames(rf.3$err.rate), col=1:3, fill=1:3)
rf.train.4 <- data.combined[1:891, c("pclass", "title", "sibsp", "parch")]
set.seed(1234)
rf.4 <- randomForest(x = rf.train.4, y = rf.label, importance = TRUE, ntree = 1000)
rf.4
varImpPlot(rf.4)
plot(rf.4, ylim=c(0,0.36))
```

```
legend('topright', colnames(rf.4$serr.rate), col=1:3, fill=1:3)
rf.train.5 <- data.combined[1:891, c("pclass", "title", "family.size")]
set.seed(1234)
rf.5 <- randomForest(x = rf.train.5, y = rf.label, importance = TRUE, ntree = 1000)
rf.5
varImpPlot(rf.5)
plot(rf.5, ylim=c(0,0.36))
legend('topright', colnames(rf.5$serr.rate), col=1:3, fill=1:3)
rf.train.6 <- data.combined[1:891, c("pclass", "title", "sibsp", "family.size")]
set.seed(1234)
rf.6 <- randomForest(x = rf.train.6, y = rf.label, importance = TRUE, ntree = 1000)
rf.6
varImpPlot(rf.6)
plot(rf.6, ylim=c(0,0.36))
legend('topright', colnames(rf.6$serr.rate), col=1:3, fill=1:3)
rf.train.7 <- data.combined[1:891, c("pclass", "title", "parch", "family.size")]
set.seed(1234)
rf.7 <- randomForest(x = rf.train.7, y = rf.label, importance = TRUE, ntree = 1000)
rf.7
varImpPlot(rf.7)
plot(rf.7, ylim=c(0,0.36))
legend('topright', colnames(rf.7$serr.rate), col=1:3, fill=1:3)
test.submit.df <- data.combined[892:1309, c("pclass", "title", "family.size")]
rf.5.preds <- predict(rf.5, test.submit.df)
table(rf.5.preds)
submit.df <- data.frame(PassengerId = rep(892:1309), Survived = rf.5.preds)
write.csv(submit.df, file = "RF_SUB_20160215_1.csv", row.names = FALSE)
library(caret)
library(doSNOW)
set.seed(2348)
cv.10.folds <- createMultiFolds(rf.label, k = 10, times = 10)
table(rf.label)
```

342 / 549

```
table(rf.label[cv.10.folds[[33]]])
```

308 / 494

```
ctrl.1 <- trainControl(method = "repeatedcv", number = 10, repeats = 10,  
                      index = cv.10.folds)  
cl <- makeCluster(6, type = "SOCK")  
registerDoSNOW(cl)  
set.seed(34324)  
rf.5.cv.1 <- train(x = rf.train.5, y = rf.label, method = "rf", tuneLength = 3,  
                 ntree = 1000, trControl = ctrl.1)  
stopCluster(cl)  
rf.5.cv.1  
set.seed(5983)  
cv.5.folds <- createMultiFolds(rf.label, k = 5, times = 10)  
ctrl.2 <- trainControl(method = "repeatedcv", number = 5, repeats = 10,  
                      index = cv.5.folds)  
cl <- makeCluster(6, type = "SOCK")  
registerDoSNOW(cl)  
set.seed(89472)  
rf.5.cv.2 <- train(x = rf.train.5, y = rf.label, method = "rf", tuneLength = 3,  
                 ntree = 1000, trControl = ctrl.2)  
stopCluster(cl)  
rf.5.cv.2  
set.seed(37596)  
cv.3.folds <- createMultiFolds(rf.label, k = 3, times = 10)  
ctrl.3 <- trainControl(method = "repeatedcv", number = 3, repeats = 10,  
                      index = cv.3.folds)  
cl <- makeCluster(6, type = "SOCK")  
registerDoSNOW(cl)  
set.seed(94622)  
rf.5.cv.3 <- train(x = rf.train.5, y = rf.label, method = "rf", tuneLength = 3,  
                 ntree = 64, trControl = ctrl.3)
```

```
stopCluster(cl)
rf.5.cv.3
library(rpart)
library(rpart.plot)
rpart.cv <- function(seed, training, labels, ctrl) {
  cl <- makeCluster(6, type = "SOCK")
  registerDoSNOW(cl)

  set.seed(seed)
  # Leverage formula interface for training
  rpart.cv <- train(x = training, y = labels, method = "rpart", tuneLength = 30,
                    trControl = ctrl)

  #Shutdown cluster
  stopCluster(cl)
  return (rpart.cv)
}
features <- c("pclass", "title", "family.size")
rpart.train.1 <- data.combined[1:891, features]
rpart.1.cv.1 <- rpart.cv(94622, rpart.train.1, rf.label, ctrl.3)
rpart.1.cv.1
rpart.1.cv.1$finalModel
prp(rpart.1.cv.1$finalModel, type = 0, extra = 1, under = TRUE)
table(data.combined$title)
data.combined[1:25, "name"]
name.splits <- str_split(data.combined$name, ",")
name.splits[1]
last.names <- sapply(name.splits, "[", 1)
last.names[1:10]
data.combined$last.name <- last.names
name.splits <- str_split(sapply(name.splits, "[", 2), " ")
titles <- sapply(name.splits, "[", 2)
```

```
unique(titles)
data.combined[which(titles == "the"),]
titles[titles %in% c("Dona.", "the")] <- "Lady."
titles[titles %in% c("Ms.", "Mlle.")] <- "Miss."
titles[titles == "Mme."] <- "Mrs."
titles[titles %in% c("Jonkheer.", "Don.")] <- "Sir."
titles[titles %in% c("Col.", "Capt.", "Major.")] <- "Officer"
table(titles)
data.combined$new.title <- as.factor(titles)
ggplot(data.combined[1:891,], aes(x = new.title, fill = survived)) +
  geom_bar() +
  facet_wrap(~ pclass) +
  ggtitle("Survival Rates for new.title by pclass")
indexes <- which(data.combined$new.title == "Lady.")
data.combined$new.title[indexes] <- "Mrs."
indexes <- which(data.combined$new.title == "Dr." |
  data.combined$new.title == "Rev." |
  data.combined$new.title == "Sir." |
  data.combined$new.title == "Officer")
data.combined$new.title[indexes] <- "Mr."
ggplot(data.combined[1:891,], aes(x = new.title, fill = survived)) +
  geom_bar() +
  facet_wrap(~ pclass) +
  ggtitle("Survival Rates for Collapsed new.title by pclass")
features <- c("pclass", "new.title", "family.size")
rpart.train.2 <- data.combined[1:891, features]
rpart.2.cv.1 <- rpart.cv(94622, rpart.train.2, rf.label, ctrl.3)
rpart.2.cv.1
rpart.2.cv.1$finalModel
prp(rpart.2.cv.1$finalModel, type = 0, extra = 1, under = TRUE)
indexes.first.mr <- which(data.combined$new.title == "Mr." & data.combined$pclass ==
"1")
```

```
first.mr.df <- data.combined[indexes.first.mr, ]
summary(first.mr.df)
first.mr.df[first.mr.df$sex == "female",]
indexes <- which(data.combined$new.title == "Mr." &
                 data.combined$sex == "female")
data.combined$new.title[indexes] <- "Mrs."
length(which(data.combined$sex == "female" &
             (data.combined$new.title == "Master." |
              data.combined$new.title == "Mr.")))
indexes.first.mr <- which(data.combined$new.title == "Mr." & data.combined$pclass ==
                          "1")
first.mr.df <- data.combined[indexes.first.mr, ]
summary(first.mr.df[first.mr.df$survived == "1",])
View(first.mr.df[first.mr.df$survived == "1",])
indexes <- which(data.combined$ticket == "PC 17755" |
                 data.combined$ticket == "PC 17611" |
                 data.combined$ticket == "113760")
View(data.combined[indexes,])
ggplot(first.mr.df, aes(x = fare, fill = survived)) +
  geom_density(alpha = 0.5) +
  ggtitle("1st Class 'Mr.' Survival Rates by fare")
ticket.party.size <- rep(0, nrow(data.combined))
avg.fare <- rep(0.0, nrow(data.combined))
tickets <- unique(data.combined$ticket)
for (i in 1:length(tickets)) {
  current.ticket <- tickets[i]
  party.indexes <- which(data.combined$ticket == current.ticket)
  current.avg.fare <- data.combined[party.indexes[1], "fare"] / length(party.indexes)
  for (k in 1:length(party.indexes)) {
    ticket.party.size[party.indexes[k]] <- length(party.indexes)
    avg.fare[party.indexes[k]] <- current.avg.fare
  }
}
```



```

}
data.combined$ticket.party.size <- ticket.party.size
data.combined$avg.fare <- avg.fare
first.mr.df <- data.combined[indexes.first.mr, ]
summary(first.mr.df)
ggplot(first.mr.df[first.mr.df$survived != "None",], aes(x = ticket.party.size, fill =
survived)) +
  geom_density(alpha = 0.5) +
  ggtitle("Survival Rates 1st Class 'Mr.' by ticket.party.size")
ggplot(first.mr.df[first.mr.df$survived != "None",], aes(x = avg.fare, fill = survived)) +
  geom_density(alpha = 0.5) +
  ggtitle("Survival Rates 1st Class 'Mr.' by avg.fare")
summary(data.combined$avg.fare)
data.combined[is.na(data.combined$avg.fare), ]
indexes <- with(data.combined, which(pclass == "3" & title == "Mr." & family.size == 1
& ticket != "3701"))
similar.na.passengers <- data.combined[indexes,]
summary(similar.na.passengers$avg.fare)
data.combined[is.na(avg.fare), "avg.fare"] <- 7.840
preproc.data.combined <- data.combined[, c("ticket.party.size", "avg.fare")]
preProc <- preProcess(preproc.data.combined, method = c("center", "scale"))
postproc.data.combined <- predict(preProc, preproc.data.combined)
cor(postproc.data.combined$ticket.party.size, postproc.data.combined$avg.fare)
indexes <- which(data.combined$pclass == "1")
cor(postproc.data.combined$ticket.party.size[indexes],
  postproc.data.combined$avg.fare[indexes])
features <- c("pclass", "new.title", "family.size", "ticket.party.size", "avg.fare")
rpart.train.3 <- data.combined[1:891, features]
rpart.3.cv.1 <- rpart.cv(94622, rpart.train.3, rf.label, ctrl.3)
rpart.3.cv.1
rpart.3.cv.1$finalModel
prp(rpart.3.cv.1$finalModel, type = 0, extra = 1, under = TRUE)

```

7.7.7 Module 6

```
train <- read.csv("train.csv", header = TRUE)
test <- read.csv("test.csv", header = TRUE)
test.survived <- data.frame(survived = rep("None", nrow(test)), test[,])
data.combined <- rbind(train, test.survived)
str(data.combined)
data.combined$survived <- as.factor(data.combined$survived)
data.combined$pclass <- as.factor(data.combined$pclass)
table(data.combined$survived)
table(data.combined$pclass)
library(ggplot2)
train$pclass <- as.factor(train$pclass)
ggplot(train, aes(x = pclass, fill = factor(survived))) +
  geom_bar() +
  xlab("Pclass") +
  ylab("Total Count") +
  labs(fill = "Survived")
head(as.character(train$name))
length(unique(as.character(data.combined$name)))
dup.names <-
as.character(data.combined[which(duplicated(as.character(data.combined$name))),
"name"])
data.combined[which(data.combined$name %in% dup.names),]
library(stringr)
misses <- data.combined[which(str_detect(data.combined$name, "Miss.")),]
misses[1:5,]
mrses <- data.combined[which(str_detect(data.combined$name, "Mrs.")), ]
mrses[1:5,]
males <- data.combined[which(train$sex == "male"), ]
males[1:5,]
extractTitle <- function(name) {
```

```
name <- as.character(name)
if (length(grep("Miss.", name)) > 0) {
  return ("Miss.")
} else if (length(grep("Master.", name)) > 0) {
  return ("Master.")
} else if (length(grep("Mrs.", name)) > 0) {
  return ("Mrs.")
} else if (length(grep("Mr.", name)) > 0) {
  return ("Mr.")
} else {
  return ("Other")
}
}
titles <- NULL
for (i in 1:nrow(data.combined)) {
  titles <- c(titles, extractTitle(data.combined[i,"name"]))
}
data.combined$title <- as.factor(titles)
ggplot(data.combined[1:891,], aes(x = title, fill = survived)) +
  geom_bar() +
  facet_wrap(~pclass) +
  ggtitle("Pclass") +
  xlab("Title") +
  ylab("Total Count") +
  labs(fill = "Survived")
table(data.combined$sex)
ggplot(data.combined[1:891,], aes(x = sex, fill = survived)) +
  geom_bar() +
  facet_wrap(~pclass) +
  ggtitle("Pclass") +
  xlab("Sex") +
  ylab("Total Count") +
```

```
labs(fill = "Survived")
summary(data.combined$age)
summary(data.combined[1:891,"age"])
ggplot(data.combined[1:891,], aes(x = age, fill = survived)) +
  facet_wrap(~sex + pclass) +
  geom_histogram(binwidth = 10) +
  xlab("Age") +
  ylab("Total Count")
boys <- data.combined[which(data.combined$title == "Master."),]
summary(boys$age)
misses <- data.combined[which(data.combined$title == "Miss."),]
summary(misses$age)
ggplot(misses[misses$survived != "None" & !is.na(misses$age),], aes(x = age, fill =
survived)) +
  facet_wrap(~pclass) +
  geom_histogram(binwidth = 5) +
  ggtitle("Age for 'Miss.' by Pclass") +
  xlab("Age") +
  ylab("Total Count")
misses.alone <- misses[which(misses$sibsp == 0 & misses$parch == 0),]
summary(misses.alone$age)
length(which(misses.alone$age <= 14.5))
summary(data.combined$sibsp)
length(unique(data.combined$sibsp))
data.combined$sibsp <- as.factor(data.combined$sibsp)
ggplot(data.combined[1:891,], aes(x = sibsp, fill = survived)) +
  geom_bar() +
  facet_wrap(~pclass + title) +
  ggtitle("Pclass, Title") +
  xlab("SibSp") +
  ylab("Total Count") +
  ylim(0,300) +
```

```
labs(fill = "Survived")
data.combined$parch <- as.factor(data.combined$parch)
ggplot(data.combined[1:891,], aes(x = parch, fill = survived)) +
  geom_bar() +
  facet_wrap(~pclass + title) +
  ggtitle("Pclass, Title") +
  xlab("ParCh") +
  ylab("Total Count") +
  ylim(0,300) +
  labs(fill = "Survived")
temp.sibsp <- c(train$sibsp, test$sibsp)
temp.parch <- c(train$parch, test$parch)
data.combined$family.size <- as.factor(temp.sibsp + temp.parch + 1)
ggplot(data.combined[1:891,], aes(x = family.size, fill = survived)) +
  geom_bar() +
  facet_wrap(~pclass + title) +
  ggtitle("Pclass, Title") +
  xlab("family.size") +
  ylab("Total Count") +
  ylim(0,300) +
  labs(fill = "Survived")
str(data.combined$ticket)
data.combined$ticket <- as.character(data.combined$ticket)
data.combined$ticket[1:20]
ticket.first.char <- ifelse(data.combined$ticket == "", " ", substr(data.combined$ticket, 1,
1))
unique(ticket.first.char)
data.combined$ticket.first.char <- as.factor(ticket.first.char)
ggplot(data.combined[1:891,], aes(x = ticket.first.char, fill = survived)) +
  geom_bar() +
  ggtitle("Survivability by ticket.first.char") +
  xlab("ticket.first.char") +
```

```
ylab("Total Count") +  
ylim(0,350) +  
labs(fill = "Survived")  
ggplot(data.combined[1:891,], aes(x = ticket.first.char, fill = survived)) +  
geom_bar() +  
facet_wrap(~pclass) +  
ggtitle("Pclass") +  
xlab("ticket.first.char") +  
ylab("Total Count") +  
ylim(0,150) +  
labs(fill = "Survived")  
ggplot(data.combined[1:891,], aes(x = ticket.first.char, fill = survived)) +  
geom_bar() +  
facet_wrap(~pclass + title) +  
ggtitle("Pclass, Title") +  
xlab("ticket.first.char") +  
ylab("Total Count") +  
ylim(0,200) +  
labs(fill = "Survived")  
summary(data.combined$fare)  
length(unique(data.combined$fare))  
ggplot(data.combined, aes(x = fare)) +  
geom_histogram(binwidth = 5) +  
ggtitle("Combined Fare Distribution") +  
xlab("Fare") +  
ylab("Total Count") +  
ylim(0,200)  
ggplot(data.combined[1:891,], aes(x = fare, fill = survived)) +  
geom_histogram(binwidth = 5) +  
facet_wrap(~pclass + title) +  
ggtitle("Pclass, Title") +  
xlab("fare") +
```

```
ylab("Total Count") +
ylim(0,50) +
labs(fill = "Survived")
str(data.combined$cabin)
data.combined$cabin <- as.character(data.combined$cabin)
data.combined$cabin[1:100]
data.combined[which(data.combined$cabin == ""), "cabin"] <- "U"
data.combined$cabin[1:100]
cabin.first.char <- as.factor(substr(data.combined$cabin, 1, 1))
str(cabin.first.char)
levels(cabin.first.char)
data.combined$cabin.first.char <- cabin.first.char
ggplot(data.combined[1:891,], aes(x = cabin.first.char, fill = survived)) +
  geom_bar() +
  ggtitle("Survivability by cabin.first.char") +
  xlab("cabin.first.char") +
  ylab("Total Count") +
  ylim(0,750) +
  labs(fill = "Survived")
ggplot(data.combined[1:891,], aes(x = cabin.first.char, fill = survived)) +
  geom_bar() +
  facet_wrap(~pclass) +
  ggtitle("Survivability by cabin.first.char") +
  xlab("Pclass") +
  ylab("Total Count") +
  ylim(0,500) +
  labs(fill = "Survived")
ggplot(data.combined[1:891,], aes(x = cabin.first.char, fill = survived)) +
  geom_bar() +
  facet_wrap(~pclass + title) +
  ggtitle("Pclass, Title") +
  xlab("cabin.first.char") +
```

```
ylab("Total Count") +
ylim(0,500) +
labs(fill = "Survived")
data.combined$cabin.multiple <- as.factor(ifelse(str_detect(data.combined$cabin, " "),
"Y", "N"))
ggplot(data.combined[1:891,], aes(x = cabin.multiple, fill = survived))
  geom_bar() +
  facet_wrap(~pclass + title) +
  ggtitle("Pclass, Title") +
  xlab("cabin.multiple") +
  ylab("Total Count") +
  ylim(0,350) +
  labs(fill = "Survived")
str(data.combined$embarked)
levels(data.combined$embarked)
ggplot(data.combined[1:891,], aes(x = embarked, fill = survived)) +
  geom_bar() +
  facet_wrap(~pclass + title) +
  ggtitle("Pclass, Title") +
  xlab("embarked") +
  ylab("Total Count") +
  ylim(0,300) +
  labs(fill = "Survived")
library(randomForest)
rf.train.1 <- data.combined[1:891, c("pclass", "title")]
rf.label <- as.factor(train$survived)
set.seed(1234)
rf.1 <- randomForest(x = rf.train.1, y = rf.label, importance = TRUE, ntree = 1000)
rf.1
varImpPlot(rf.1)
rf.train.2 <- data.combined[1:891, c("pclass", "title", "sibsp")]
set.seed(1234)
```



```
rf.2 <- randomForest(x = rf.train.2, y = rf.label, importance = TRUE, ntree = 1000)
rf.2
varImpPlot(rf.2)
rf.train.3 <- data.combined[1:891, c("pclass", "title", "parch")]
set.seed(1234)
rf.3 <- randomForest(x = rf.train.3, y = rf.label, importance = TRUE, ntree = 1000)
rf.3
varImpPlot(rf.3)
rf.train.4 <- data.combined[1:891, c("pclass", "title", "sibsp", "parch")]
set.seed(1234)
rf.4 <- randomForest(x = rf.train.4, y = rf.label, importance = TRUE, ntree = 1000)
rf.4
varImpPlot(rf.4)
rf.train.5 <- data.combined[1:891, c("pclass", "title", "family.size")]
set.seed(1234)
rf.5 <- randomForest(x = rf.train.5, y = rf.label, importance = TRUE, ntree = 1000)
rf.5
varImpPlot(rf.5)
rf.train.6 <- data.combined[1:891, c("pclass", "title", "sibsp", "family.size")]
set.seed(1234)
rf.6 <- randomForest(x = rf.train.6, y = rf.label, importance = TRUE, ntree = 1000)
rf.6
varImpPlot(rf.6)
rf.train.7 <- data.combined[1:891, c("pclass", "title", "parch", "family.size")]
set.seed(1234)
rf.7 <- randomForest(x = rf.train.7, y = rf.label, importance = TRUE, ntree = 1000)
rf.7
varImpPlot(rf.7)
test.submit.df <- data.combined[892:1309, c("pclass", "title", "family.size")]
rf.5.preds <- predict(rf.5, test.submit.df)
table(rf.5.preds)
submit.df <- data.frame(PassengerId = rep(892:1309), Survived = rf.5.preds)
```

```
write.csv(submit.df, file = "RF_SUB_20160215_1.csv", row.names = FALSE)
library(caret)
library(doSNOW)
set.seed(2348)
cv.10.folds <- createMultiFolds(rf.label, k = 10, times = 10)
table(rf.label)
342 / 549
table(rf.label[cv.10.folds[[33]]])
308 / 494
ctrl.1 <- trainControl(method = "repeatedcv", number = 10, repeats = 10,
                        index = cv.10.folds)
cl <- makeCluster(6, type = "SOCK")
registerDoSNOW(cl)
set.seed(34324)
rf.5.cv.1 <- train(x = rf.train.5, y = rf.label, method = "rf", tuneLength = 3,
                  ntree = 1000, trControl = ctrl.1)
stopCluster(cl)
rf.5.cv.1
set.seed(5983)
cv.5.folds <- createMultiFolds(rf.label, k = 5, times = 10)
ctrl.2 <- trainControl(method = "repeatedcv", number = 5, repeats = 10,
                        index = cv.5.folds)
cl <- makeCluster(6, type = "SOCK")
registerDoSNOW(cl)
set.seed(89472)
rf.5.cv.2 <- train(x = rf.train.5, y = rf.label, method = "rf", tuneLength = 3,
                  ntree = 1000, trControl = ctrl.2)
stopCluster(cl)
rf.5.cv.2
set.seed(37596)
cv.3.folds <- createMultiFolds(rf.label, k = 3, times = 10)
ctrl.3 <- trainControl(method = "repeatedcv", number = 3, repeats = 10)
```

```
        index = cv.3.folds)
cl <- makeCluster(6, type = "SOCK")
registerDoSNOW(cl)
set.seed(94622)
rf.5.cv.3 <- train(x = rf.train.5, y = rf.label, method = "rf", tuneLength = 3,
                  ntree = 64, trControl = ctrl.3)
stopCluster(cl)
rf.5.cv.3
library(rpart)
library(rpart.plot)
rpart.cv <- function(seed, training, labels, ctrl) {
  cl <- makeCluster(6, type = "SOCK")
  registerDoSNOW(cl)

  set.seed(seed)
  # Leverage formula interface for training
  rpart.cv <- train(x = training, y = labels, method = "rpart", tuneLength = 30,
                   trControl = ctrl)

  #Shutdown cluster
  stopCluster(cl)

  return (rpart.cv)
}
features <- c("pclass", "title", "family.size")
rpart.train.1 <- data.combined[1:891, features]
rpart.1.cv.1 <- rpart.cv(94622, rpart.train.1, rf.label, ctrl.3)
rpart.1.cv.1
prp(rpart.1.cv.1$finalModel, type = 0, extra = 1, under = TRUE)
table(data.combined$title)
data.combined[1:25, "name"]
name.splits <- str_split(data.combined$name, ",")
```

```
name.splits[1]
last.names <- sapply(name.splits, "[", 1)
last.names[1:10]
data.combined$last.name <- last.names
name.splits <- str_split(sapply(name.splits, "[", 2), " ")
titles <- sapply(name.splits, "[", 2)
unique(titles)
data.combined[which(titles == "the"),]
titles[titles %in% c("Dona.", "the")] <- "Lady."
titles[titles %in% c("Ms.", "Mlle.")] <- "Miss."
titles[titles == "Mme."] <- "Mrs."
titles[titles %in% c("Jonkheer.", "Don.")] <- "Sir."
titles[titles %in% c("Col.", "Capt.", "Major.")] <- "Officer"
table(titles)
data.combined$new.title <- as.factor(titles)
ggplot(data.combined[1:891,], aes(x = new.title, fill = survived)) +
  geom_bar() +
  facet_wrap(~ pclass) +
  ggtitle("Survival Rates for new.title by pclass")
indexes <- which(data.combined$new.title == "Lady.")
data.combined$new.title[indexes] <- "Mrs."
indexes <- which(data.combined$new.title == "Dr." |
  data.combined$new.title == "Rev." |
  data.combined$new.title == "Sir." |
  data.combined$new.title == "Officer")
data.combined$new.title[indexes] <- "Mr."
ggplot(data.combined[1:891,], aes(x = new.title, fill = survived)) +
  geom_bar() +
  facet_wrap(~ pclass) +
  ggtitle("Survival Rates for Collapsed new.title by pclass")
features <- c("pclass", "new.title", "family.size")
rpart.train.2 <- data.combined[1:891, features]
```

```
rpart.2.cv.1 <- rpart.cv(94622, rpart.train.2, rf.label, ctrl.3)
rpart.2.cv.1
prp(rpart.2.cv.1$finalModel, type = 0, extra = 1, under = TRUE)
indexes.first.mr <- which(data.combined$new.title == "Mr." & data.combined$pclass ==
"1")
first.mr.df <- data.combined[indexes.first.mr, ]
summary(first.mr.df)
first.mr.df[first.mr.df$sex == "female",]
indexes <- which(data.combined$new.title == "Mr." &
                data.combined$sex == "female")
data.combined$new.title[indexes] <- "Mrs."
length(which(data.combined$sex == "female" &
             (data.combined$new.title == "Master." |
              data.combined$new.title == "Mr.")))
indexes.first.mr <- which(data.combined$new.title == "Mr." & data.combined$pclass ==
"1")
first.mr.df <- data.combined[indexes.first.mr, ]
summary(first.mr.df[first.mr.df$survived == "1",])
View(first.mr.df[first.mr.df$survived == "1",])
indexes <- which(data.combined$ticket == "PC 17755" |
                data.combined$ticket == "PC 17611" |
                data.combined$ticket == "113760")
View(data.combined[indexes,])
ggplot(first.mr.df, aes(x = fare, fill = survived)) +
  geom_density(alpha = 0.5) +
  ggtitle("1st Class 'Mr.' Survival Rates by fare")
ticket.party.size <- rep(0, nrow(data.combined))
avg.fare <- rep(0.0, nrow(data.combined))
tickets <- unique(data.combined$ticket)
for (i in 1:length(tickets)) {
  current.ticket <- tickets[i]
  party.indexes <- which(data.combined$ticket == current.ticket)
```

```

current.avg.fare <- data.combined[party.indexes[1], "fare"] / length(party.indexes)

for (k in 1:length(party.indexes)) {
  ticket.party.size[party.indexes[k]] <- length(party.indexes)
  avg.fare[party.indexes[k]] <- current.avg.fare
}
}

data.combined$ticket.party.size <- ticket.party.size
data.combined$avg.fare <- avg.fare
first.mr.df <- data.combined[indexes.first.mr, ]
summary(first.mr.df)

ggplot(first.mr.df[first.mr.df$survived != "None",], aes(x = ticket.party.size, fill =
survived)) +
  geom_density(alpha = 0.5) +
  ggtitle("Survival Rates 1st Class 'Mr.' by ticket.party.size")

ggplot(first.mr.df[first.mr.df$survived != "None",], aes(x = avg.fare, fill = survived)) +
  geom_density(alpha = 0.5) +
  ggtitle("Survival Rates 1st Class 'Mr.' by avg.fare")
summary(data.combined$avg.fare)
data.combined[is.na(data.combined$avg.fare), ]
indexes <- with(data.combined, which(pclass == "3" & title == "Mr." & family.size == 1
&
                                ticket != "3701"))
similar.na.passengers <- data.combined[indexes,]
summary(similar.na.passengers$avg.fare)
data.combined[is.na(avg.fare), "avg.fare"] <- 7.840
preproc.data.combined <- data.combined[, c("ticket.party.size", "avg.fare")]
preProc <- preProcess(preproc.data.combined, method = c("center", "scale"))
postproc.data.combined <- predict(preProc, preproc.data.combined)
cor(postproc.data.combined$ticket.party.size, postproc.data.combined$avg.fare)
indexes <- which(data.combined$pclass == "1")
cor(postproc.data.combined$ticket.party.size[indexes],

```

```
postproc.data.combined$avg.fare[indexes])
features <- c("pclass", "new.title", "family.size", "ticket.party.size", "avg.fare")
rpart.train.3 <- data.combined[1:891, features]
rpart.3.cv.1 <- rpart.cv(94622, rpart.train.3, rf.label, ctrl.3)
rpart.3.cv.1
prp(rpart.3.cv.1$finalModel, type = 0, extra = 1, under = TRUE)
test.submit.df <- data.combined[892:1309, features]
rpart.3.preds <- predict(rpart.3.cv.1$finalModel, test.submit.df, type = "class")
table(rpart.3.preds)
submit.df <- data.frame(PassengerId = rep(892:1309), Survived = rpart.3.preds)
write.csv(submit.df, file = "RPART_SUB_20160619_1.csv", row.names = FALSE)
features <- c("pclass", "new.title", "ticket.party.size", "avg.fare")
rf.train.temp <- data.combined[1:891, features]
set.seed(1234)
rf.temp <- randomForest(x = rf.train.temp, y = rf.label, ntree = 1000)
rf.temp
test.submit.df <- data.combined[892:1309, features]
rf.preds <- predict(rf.temp, test.submit.df)
table(rf.preds)
submit.df <- data.frame(PassengerId = rep(892:1309), Survived = rf.preds)
write.csv(submit.df, file = "RF_SUB_FINAL.csv", row.names = FALSE)
```

7.7.8 Files in Module 7

```
train <- read.csv("train.csv", header = TRUE)
test <- read.csv("test.csv", header = TRUE)
test.survived <- data.frame(survived = rep("None", nrow(test)), test[,])
data.combined <- rbind(train, test.survived)
str(data.combined)
data.combined$survived <- as.factor(data.combined$survived)
data.combined$pclass <- as.factor(data.combined$pclass)
table(data.combined$survived)
table(data.combined$pclass)
```

```
library(ggplot2)
train$pclass <- as.factor(train$pclass)
ggplot(train, aes(x = pclass, fill = factor(survived)))
geom_bar() +
  xlab("Pclass") +
  ylab("Total Count") +
  labs(fill = "Survived")
head(as.character(train$name)) # Data is in the format- Last Name, Title_First Name
length(unique(as.character(data.combined$name)))
dup.names <-
as.character(data.combined[which(duplicated(as.character(data.combined$name))),
"name"])
data.combined[which(data.combined$name %in% dup.names),]
library(stringr)
misses <- data.combined[which(str_detect(data.combined$name, "Miss.")),]
misses[1:5,]
mrses <- data.combined[which(str_detect(data.combined$name, "Mrs.")), ]
mrses[1:5,]
males <- data.combined[which(train$sex == "male"), ]
males[1:5,]
extractTitle <- function(name) {
  name <- as.character(name)
  if (length(grep("Miss.", name)) > 0) {
    return ("Miss.")
  } else if (length(grep("Master.", name)) > 0) {
    return ("Master.")
  } else if (length(grep("Mrs.", name)) > 0) {
    return ("Mrs.")
  } else if (length(grep("Mr.", name)) > 0) {
    return ("Mr.")
  } else {
    return ("Other")
  }
}
```



```
}  
}  
titles <- NULL  
for (i in 1:nrow(data.combined)) {  
  titles <- c(titles, extractTitle(data.combined[i,"name"]))  
}  
data.combined$title <- as.factor(titles)  
ggplot(data.combined[1:891,], aes(x = title, fill = survived)) + # fill means colour code it  
on the basis of value of survived  
  stat_count(width = 0.5) +  
  facet_wrap(~pclass) +  
  ggtitle("Pclass") +  
  xlab("Title") +  
  ylab("Total Count") +  
  labs(fill = "Survived")  
table(data.combined$sex)  
ggplot(data.combined[1:891,], aes(x = sex, fill = survived)) +  
  stat_count(width = 0.5) +  
  facet_wrap(~pclass) +  
  ggtitle("Pclass") +  
  xlab("Sex") +  
  ylab("Total Count") +  
  labs(fill = "Survived")  
summary(data.combined$age)  
summary(data.combined[1:891,"age"])  
ggplot(data.combined[1:891,], aes(x = age, fill = survived)) +  
  facet_wrap(~sex + pclass) +  
  geom_histogram(binwidth = 10) +  
  xlab("Age") +  
  ylab("Total Count")  
boys <- data.combined[which(data.combined$title == "Master."),]  
summary(boys$age)
```

```
ggplot(boys[boys$survived != "None",], aes(x = age, fill = survived)) +  
  facet_wrap(~pclass) +  
  geom_histogram(binwidth = 5) +  
  ggtitle("Age for 'Master' by Pclass") +  
  xlab("Age") +  
  ylab("Total Count")  
misses <- data.combined[which(data.combined$title == "Miss."),]  
summary(misses$age)  
ggplot(misses[misses$survived != "None",], aes(x = age, fill = survived)) +  
  facet_wrap(~pclass) +  
  geom_histogram(binwidth = 5) +  
  ggtitle("Age for 'Miss.' by Pclass") +  
  xlab("Age") +  
  ylab("Total Count")  
misses.alone <- misses[which(misses$sibsp == 0 & misses$parch == 0),]  
summary(misses.alone$age)  
length(which(misses.alone$age <= 14.5)) # Which is = to 4, hence we have very less  
female children.  
summary(data.combined$sibsp)  
length(unique(data.combined$sibsp))  
data.combined$sibsp <- as.factor(data.combined$sibsp)  
ggplot(data.combined[1:891,], aes(x = sibsp, fill = survived)) +  
  stat_count(width = 1) +  
  facet_wrap(~pclass + title) +  
  ggtitle("Pclass, Title") +  
  xlab("SibSp") +  
  ylab("Total Count") +  
  ylim(0,300) +  
  labs(fill = "Survived")  
data.combined$parch <- as.factor(data.combined$parch)  
ggplot(data.combined[1:891,], aes(x = parch, fill = survived)) +  
  stat_count(width = 1) +
```

```
facet_wrap(~pclass + title) +
ggtitle("Pclass, Title") +
xlab("ParCh") +
ylab("Total Count") +
ylim(0,300) +
labs(fill = "Survived")
temp.sibsp <- c(train$sibsp, test$sibsp)
temp.parch <- c(train$parch, test$parch)
data.combined$family.size <- as.factor(temp.sibsp + temp.parch + 1)
ggplot(data.combined[1:891,], aes(x = family.size, fill = survived)) +
  stat_count(width = 1) +
  facet_wrap(~pclass + title) +
  ggtitle("Pclass, Title") +
  xlab("family.size") +
  ylab("Total Count") +
  ylim(0,300) +
  labs(fill = "Survived")
ggplot(data.combined[1:891,], aes(x = family.size, fill = survived)) +
  stat_count(width = 1) +
  facet_wrap(~pclass) +
  ggtitle("Pclass") +
  xlab("family.size") +
  ylab("Total Count") +
  ylim(0,300) +
  labs(fill = "Survived")
str(data.combined$ticket)
data.combined$ticket <- as.character(data.combined$ticket)
data.combined$ticket[1:20]
ticket.first.char <- ifelse(data.combined$ticket == "", " ", substr(data.combined$ticket, 1,
1)) # substr(x, start, stop)
unique(ticket.first.char) # Find the unique values in ticket.first.char variable
data.combined$ticket.first.char <- as.factor(ticket.first.char)
```

```
ggplot(data.combined[1:891,], aes(x = ticket.first.char, fill = survived)) +  
  geom_bar() +  
  ggtitle("Survivability by ticket.first.char") +  
  xlab("ticket.first.char") +  
  ylab("Total Count") +  
  ylim(0,350) +  
  labs(fill = "Survived")  
  
ggplot(data.combined[1:891,], aes(x = ticket.first.char, fill = survived)) +  
  geom_bar() +  
  facet_wrap(~pclass) +  
  ggtitle("Pclass") +  
  xlab("ticket.first.char") +  
  ylab("Total Count") +  
  ylim(0,150) +  
  labs(fill = "Survived")  
  
ggplot(data.combined[1:891,], aes(x = ticket.first.char, fill = survived)) +  
  geom_bar() +  
  facet_wrap(~pclass + title) +  
  ggtitle("Pclass, Title") +  
  xlab("ticket.first.char") +  
  ylab("Total Count") +  
  ylim(0,200) +  
  labs(fill = "Survived")  
  
summary(data.combined$fare) # The distribution is scewed to the high end since mean >  
median  
  
length(unique(data.combined$fare))  
  
ggplot(data.combined, aes(x = fare)) +  
  stat_count(width = 5) +  
  ggtitle("Combined Fare Distribution") +  
  xlab("Fare") +  
  ylab("Total Count") +  
  ylim(0,200)
```

```
ggplot(data.combined[1:891,], aes(x = fare, fill = survived)) +
  stat_count(width = 5) +
  facet_wrap(~pclass + title) +
  ggtitle("Pclass, Title") +
  xlab("fare") +
  ylab("Total Count") +
  ylim(0,50) +
  labs(fill = "Survived")
str(data.combined$cabin)
data.combined$cabin <- as.character(data.combined$cabin)
data.combined$cabin[1:100] # we use 100 because lost of cabin entries are empty
data.combined[which(data.combined$cabin == ""), "cabin"] <- "U"
data.combined$cabin[1:100]
cabin.first.char <- as.factor(substr(data.combined$cabin, 1, 1))
str(cabin.first.char)
levels(cabin.first.char)
data.combined$cabin.first.char <- cabin.first.char
ggplot(data.combined[1:891,], aes(x = cabin.first.char, fill = survived)) +
  geom_bar() +
  ggtitle("Survivability by cabin.first.char") +
  xlab("cabin.first.char") +
  ylab("Total Count") +
  ylim(0,750) +
  labs(fill = "Survived")
ggplot(data.combined[1:891,], aes(x = cabin.first.char, fill = survived)) +
  geom_bar() +
  facet_wrap(~pclass) +
  ggtitle("Survivability by cabin.first.char") +
  xlab("Pclass") +
  ylab("Total Count") +
  ylim(0,500) +
  labs(fill = "Survived")
```

```
ggplot(data.combined[1:891,], aes(x = cabin.first.char, fill = survived)) +  
  geom_bar() +  
  facet_wrap(~pclass + title) +  
  ggtitle("Pclass, Title") +  
  xlab("cabin.first.char") +  
  ylab("Total Count") +  
  ylim(0,500) +  
  labs(fill = "Survived")  
data.combined$cabin.multiple <- as.factor(ifelse(str_detect(data.combined$cabin, " "),  
"Y", "N"))  
ggplot(data.combined[1:891,], aes(x = cabin.multiple, fill = survived)) +  
  geom_bar() +  
  facet_wrap(~pclass + title) +  
  ggtitle("Pclass, Title") +  
  xlab("cabin.multiple") +  
  ylab("Total Count") +  
  ylim(0,350) +  
  labs(fill = "Survived")  
ggplot(data.combined[1:891,], aes(x = cabin.multiple, fill = survived)) +  
  geom_bar() +  
  facet_wrap(~pclass) +  
  ggtitle("Pclass") +  
  xlab("cabin.multiple") +  
  ylab("Total Count") +  
  ylim(0,350) +  
  labs(fill = "Survived")  
str(data.combined$embarked)  
levels(data.combined$embarked)  
ggplot(data.combined[1:891,], aes(x = embarked, fill = survived)) +  
  geom_bar() +  
  facet_wrap(~pclass + title) +  
  ggtitle("Pclass, Title") +
```

```
xlab("embarked") +  
ylab("Total Count") +  
ylim(0,300) +  
labs(fill = "Survived")  
ggplot(data.combined[1:891,], aes(x = embarked, fill = survived)) +  
  geom_bar() +  
  facet_wrap(~pclass) +  
  ggtitle("Pclass") +  
  xlab("embarked") +  
  ylab("Total Count") +  
  ylim(0,300) +  
  labs(fill = "Survived")
```

7.7.8.1 init.R

```
install.packages("shiny",clean=T)  
install.packages("randomForest",clean=T)  
install.packages("Rook",clean=T)  
install.packages("rpart",clean=T)  
install.packages("rattle",clean=T)  
install.packages("rpart.plot",clean=T)  
library('shiny')  
print("All packages installed...")
```

7.7.8.2 server.R

```
library(shiny)  
library(rpart)  
library(rattle)  
library(rpart.plot)  
library(RColorBrewer)  
library(ggplot2)  
source("preprocessTitanicData.R")  
shinyServer(function(input, output) {  
  output$dataBoxPlot <- renderPlot({
```

```

    #ggplot() +
    # geom_point(data = train, aes(x = gp, y = y)) +
    #geom_point(data = ds, aes(x = gp, y = mean),
    #          colour = 'red', size = 3) +
    # geom_errorbar(data = ds, aes(x = gp, y = mean,
    #                               ymin = mean - sd, ymax = mean + sd),
    #              colour = 'red', width = 0.4)
    qplot(Age, data=train, geom="density", fill=Pclass, alpha=I(.5),
          main="Age distribution by class", xlab="Age",
          ylab="Density")
  })

  output$dataPlot1 <- renderPlot({
    q<-ggplot(train, aes_string(x=input$x, y=input$y, shape=input$toPlot,
    color=input$toPlot), facets=paste(input$facets, collapse="~"))
    q<-q + geom_point(size=I(3), xlab="XX", ylab="YY") +
      facet_grid(paste(input$facets, collapse="~"), scales="free", space="free")
    print(q)
  })

  output$ageHist <- renderPlot({
    x <- train$Age
    bins <- seq(min(x, na.rm=T), max(x, na.rm=T), length.out = input$ageBins + 1)

    # draw the histogram with the specified number of bins
    hist(x, breaks = bins, col = 'darkgray', border = 'white')
  })

  fit <- reactive({
    variables <- paste(input$treeVariables, collapse="+")
    #avoid error if no box is checked
    if (nchar(variables)<2){
      variables<-"Sex"

```



```

    }
    args <- list(paste("as.factor(Survived) ~ ", variables))

    args$data<-train
    args$method<-"class"# computing the decision tree
    do.call(rpart, args)
  })
  output$decisionTree <- renderPlot({
    fancyRpartPlot(fit())
  })
  output$didHeSurvive <- renderText({
    FamilyID2<-paste(input$FamilyName,input$FamilySize)
    toTest <- data.frame(Sex=input$Sex, Age=input$Age, Pclass=input$Pclass,
SibSp=input$Siblings,
                        Fare=input$Fare, Embarked=input$Embarked, Title=input$Title,
input$FamilySize, FamilyID2=FamilyID2)
    toTest$Pclass<-factor(toTest$Pclass, levels=c(1,2,3), labels=c("First class",
"Second class", "Third class"))
    #fit is shared bw the "did he survive" and "decision tree" panels
    Prediction <- predict(fit(), toTest, type="class")
    write.csv(Prediction, "prediction.csv")
    return(as.character(Prediction))
  })
})

```

7.7.8.3 ui.R

```

library(shiny)
shinyUI(fluidPage(
  #titlePanel("Running R Code in the cloud"),

  tabsetPanel("Running R Code in the cloud",
    tabPanel("Context",

```

```

    h2("The story"),
    p("The sinking of the RMS Titanic is one of the most infamous
shipwrecks in history.

    On April 15, 1912, during her maiden voyage, the Titanic sank after
colliding with an iceberg,

    killing 1502 out of 2224 passengers and crew. This sensational
tragedy shocked

    the international community and led to better safety regulations for
ships."),
    img(src="RMS_Titanic.jpg")
),
"Useful graphs",

tabPanel("The data",
  sidebarLayout(

    sidebarPanel(
      selectInput("x", label="x-axis",
        choices=list("Age" = "Age", "Fare"="Fare", "Family
Size"="FamilySize", "Number of Siblings on board"="SibSp"),
        selected=list("Age")),
      selectInput("y", label="y-axis",
        choices=list("Age" = "Age", "Fare"="Fare", "Family
Size"="FamilySize", "Number of Siblings on board"="SibSp"),
        selected=list("Fare")),
      selectInput("toPlot", label="Variable to distinguish on the plot",
        choices=list("Sex" = "Sex", "Class" = "Pclass",
          "Embarkment Port"="Embarked", "Family
Size"="FamilySize", "Family Name"="FamilyID2",
          "Title"="Title", "Number of Siblings on
board"="SibSp"),
        selected=list("Pclass"))

```

```

checkboxGroupInput("facets", label="Facets",
                choices = list("Survived"="Survived", "Sex" = "Sex",
"Class" = "Pclass", "Embarkment Port"="Embarked",
                                "Family Size"="FamilySize", "Family
Name"="FamilyID2",
                                "Title"="Title", "Number of Siblings on
board"="SibSp"),
                selected = list("Sex", "Survived"))

),

mainPanel(
  h2("Data exploration"),
  plotOutput("dataPlot1")
)
)),

tabPanel("Age repartition",
  sidebarLayout(
    sidebarPanel(sliderInput("ageBins", "Number of age ranges to
consider", min=1, max=30, value=8)),

    mainPanel(
      h2("Survival by Age range"),
      plotOutput("ageHist"),
      plotOutput("dataBoxPlot")
    )
  )),

"Modeling",

```

```

    tabPanel("Decision tree",
      sidebarLayout(

        sidebarPanel(
          checkboxGroupInput("treeVariables", label = h4("Variables to take
into account"),
            choices = list("Sex" = "Sex", "Age" = "Age", "Class" =
"Pclass", "Fare"="Fare",
              "Embarkment Port"="Embarked", "Family
Size"="FamilySize", "Family Name"="FamilyID2",
              "Title"="Title", "Number of Siblings on
board"="SibSp"),
            selected = list("Sex", "Age")
          )),

        mainPanel(
          h2("Decision tree obtained with rpart"),
          plotOutput("decisionTree")
        )
      )),
    tabPanel("Did he survive ?",
      sidebarLayout(
        sidebarPanel(
          "Select the characteristic of the passenger :",
          radioButtons("Sex", "Sex",
            choices=list("Male"="male", "Female"="female"), selected="male"),
          sliderInput("Age", "Age", min=0, max=100, value = 25),
          radioButtons("Pclass", "Class", choices=list("First"=1, "Second"=2,
            "Third"=3), selected=2),
          sliderInput("Fare", "Fare", min=1, max=100, value=10),

```

```

        selectInput("Embarked", "Embarked at", choices=list("Cherbourg"="C",
"Queenstown"="Q", "Southampton"="S"), selected="S"),
        sliderInput("FamilySize", "Family size", min=1, max=15, value=3),
        textInput("FamilyName", "Family name", value="Smith"),
        selectInput("Title", "Title", choices=list("Mr"="Mr", "Mrs"="Mrs",
"Miss"="Miss", "Master"="Master", "Dr"="Dr",
"Rev"="Rev", "Sir"="Sir", "Lady"="Lady",
"Col"="Col"), selected="Mr"),
        sliderInput("Siblings", "Number of Siblings on board", min=0, max=10,
val=2)

    ),
    mainPanel(
        h2("Prediction according to the current decision tree"),
        "According to the currently computed decision tree, this passenger must
have",
        textOutput("didHeSurvive", container=strong), " !",
        br()
    )
))
)
))

```

7.7.8.4 preprocessTitanicData.R.R

```

train <- read.csv("data/train.csv")
test <- read.csv("data/test.csv")
test$Survived <- NA
combi <- rbind(train, test)
combi$Name = as.character(combi$Name)
combi$Title<-sapply(combi$Name, FUN=function(x){strsplit(x, split='[.]')[[1]][2]})
combi$Title<-sub(' ', "", combi$Title)

```

```
combi$Title[combi$Title=='Mlle']<-'Miss'
combi$Title[combi$Title %in% c('Mme', 'Ms')]<-'Mrs'
combi$Title[combi$Title %in% c('Capt', 'Don', 'Major', 'Sir')]<-'Sir'
combi$Title[combi$Title %in% c('Dona', 'Lady', 'the Countess', 'Jonkheer')]<-'Lady'
combi$Title<-factor(combi$Title)
combi$FamilySize <-combi$SibSp+combi$Parch
combi$Surname<-sapply(combi$Name, FUN=function(x){strsplit(x, split='[,.]')[[1]][1]})
combi$FamilyID<-paste(combi$Surname,as.character(combi$FamilySize), sep="_")
famIDs<-data.frame(table(combi$FamilyID))
famIDsFreq1<-famIDs[famIDs$Freq==1,]
combi$FamilyID[combi$FamilyID %in% famIDsFreq1$Var1] <-'Alone'
famIDsFreq2<-famIDs[famIDs$Freq==2,]
combi$FamilyID[combi$FamilyID %in% famIDsFreq2$Var1] <-'Small'
combi$FamilyID<-factor(combi$FamilyID)
Agefit <- rpart(Age ~ Pclass + Sex +SibSp + Parch + Fare + Title + FamilySize,
               data=combi[!is.na(combi$Age),],
               method="anova")
combi$Age[is.na(combi$Age)]<-predict(Agefit, combi[is.na(combi$Age),])
combi$Embarked[which(combi$Embarked == "")]<-"S"
combi$Embarked<-factor(combi$Embarked)
combi$Fare[which(is.na(combi$Fare))]<- median(combi$Fare, na.rm=TRUE)
combi$FamilyID2 <- combi$FamilyID
combi$FamilyID2 <- as.character(combi$FamilyID2)
combi$FamilyID2[combi$FamilySize <=1] <- "Alone"
combi$FamilyID2[combi$FamilySize >1 & combi$FamilySize <= 2] <- "Small"
combi$FamilyID2 <- factor(combi$FamilyID2)
train<-combi[1:891,]
test<-combi[892:1309,]
train$Survived<-factor(train$Survived, levels=c(0,1), labels=c("Died", "Survived"))
train$Pclass<-factor(train$Pclass, levels=c(1,2,3), labels=c("First class", "Second class",
"Third class"))
```

```
fit <- rpart(Survived ~ as.factor(Pclass) + Sex + Age, data = train, method="class")
```

Chapter 8

RESULT ANALYSIS

8.1 Results

- The female passenger's survived chance is about 3 times than male.
- The first class passenger survived chance is 3 times than 2nd class and 1.5 times than 3rd class.
- The sex is more important factor than Pclass.
- The OOB estimate of error rate for our final random forest model is 16.16%. In other words, our model has an accuracy of 83.84%, which is really good. After Kaggle submission, our training data has an accuracy of 80.861%.
- The Shiny Web-App that we have created is giving the survived or perished value for a given set of input variables in real time, in addition to different real-time graphs and decision trees.

8.2 Graphs

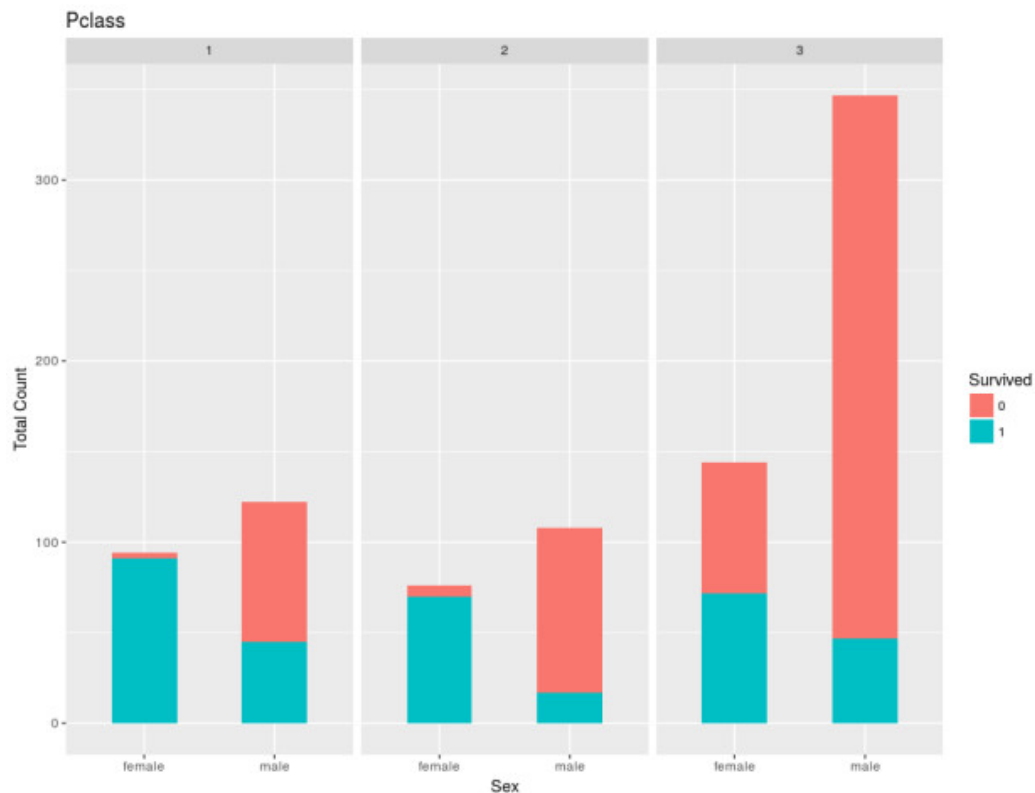


Fig 8.2.1 Graph of survived variable for males vs female

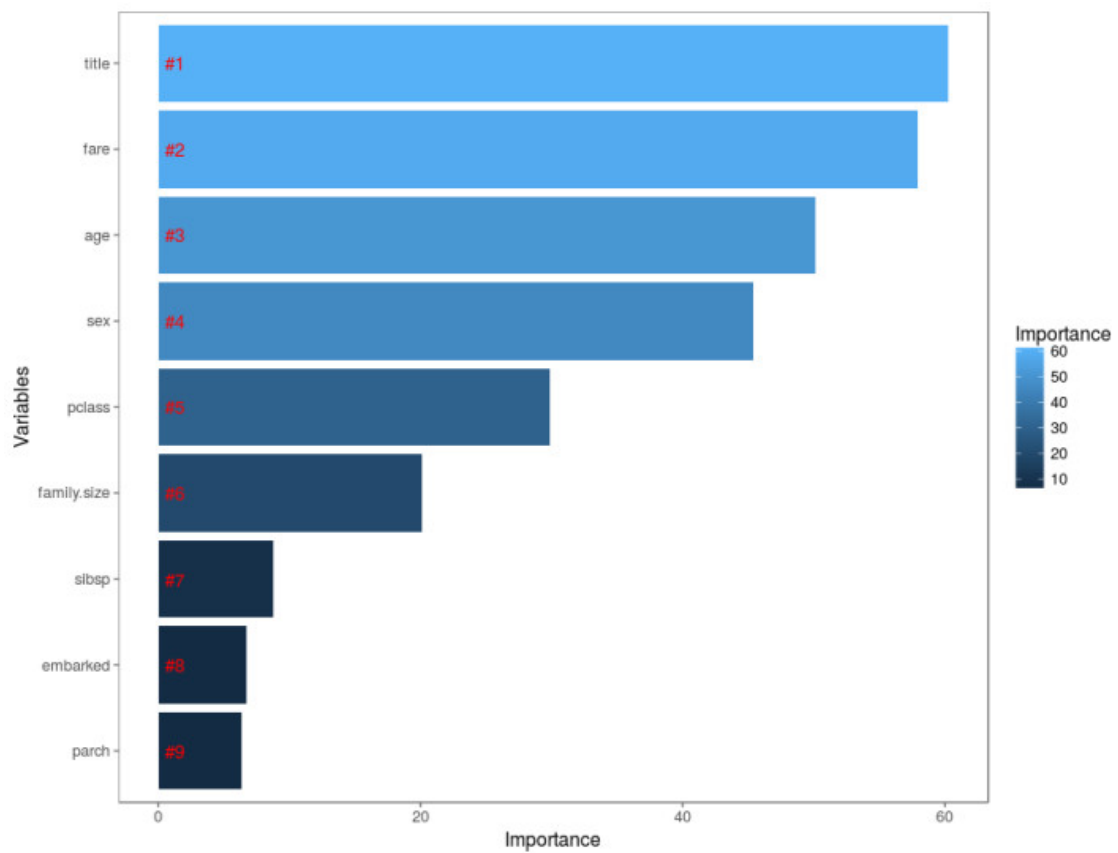


Fig. 8.2.2 Graph for the importance factor for different variables in Random Forests

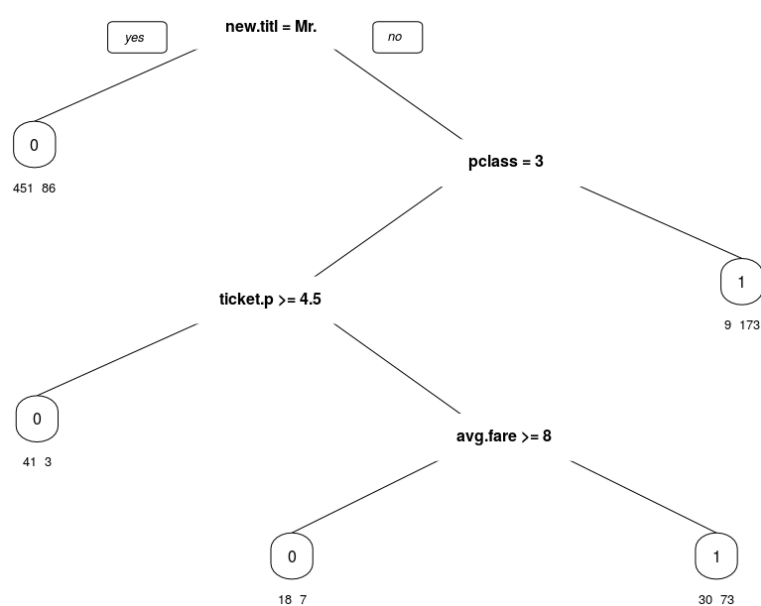


Fig. 8.2.3 The Most Accurate Decision Tree used for Final Random Forests Mode

8.3 Accuracy of Final Model

The OOB estimate of error rate for our final random forest model is 16.16%. In other words, our model has an accuracy of 83.84%, which is really good. After Kaggle submission, we find that our training data has an accuracy of 80.861%.

8.4 Screenshot of Shiny Web UI

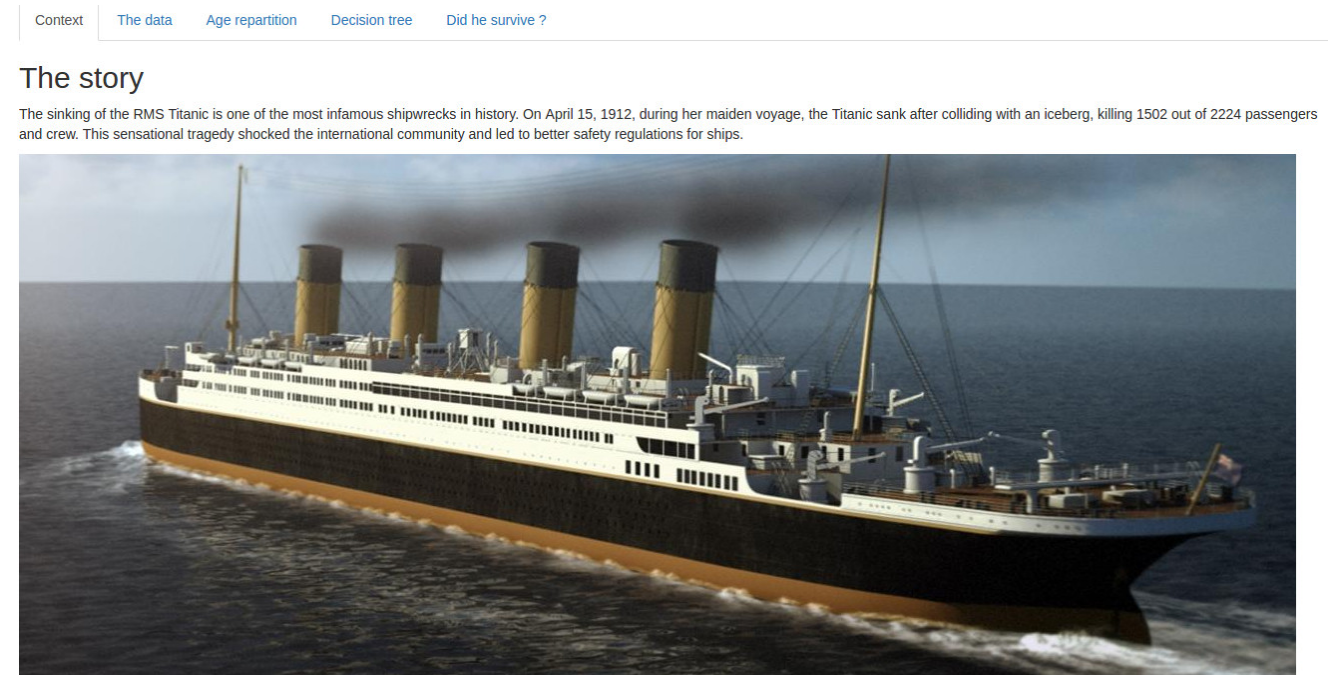


Fig 8.4.1 First Tab



Fig 8.4.2 Second Tab



Fig 8.4.3 Third Tab

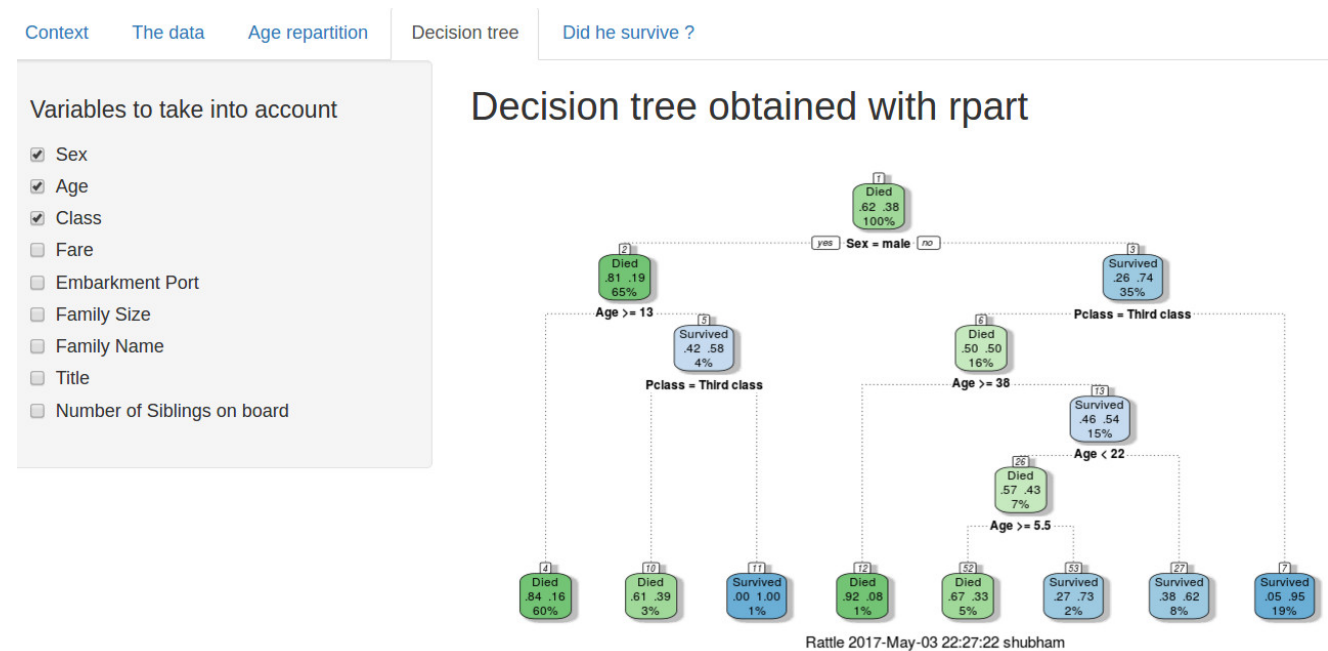


Fig 8.4.4 Fourth Tab

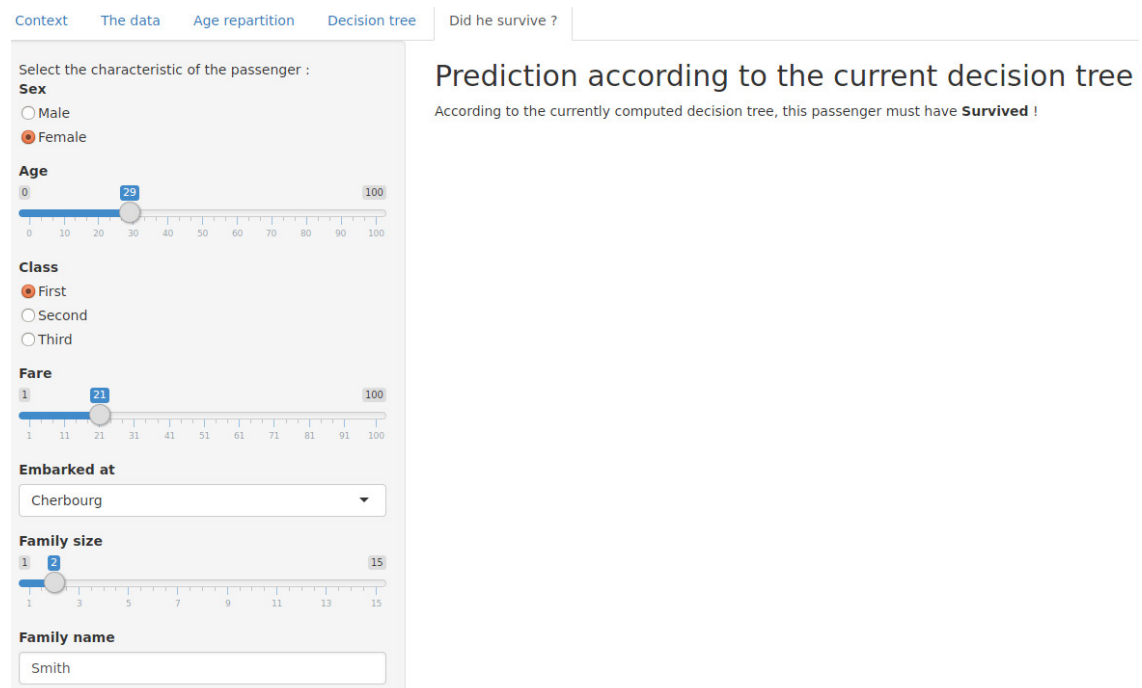


Fig 8.4.5 Fifth Tab

Chapter 9

CONCLUSION

9.1 Conclusion and Future Work

After several iterations of exploring and conditioning on the data, we have built a useful algorithm for predicting the survival of each passenger aboard the RMS Titanic. The technique applied in this project is a manual implementation of a simple machine learning model, the random forest. This model is quite accurate, and can be used in future project of the same domain. We have also seen that Random Forest, which is more accurate than SVM in this case, can be improved using decision trees and feature engineering. Also, this project has helped us in getting a deep understanding of different machine learning algorithms, like SVM or K-Fold Cross Validation.

We can use the similar approach in other data analytics project, like analyzing data for 9/11 terrorist attack or Bhopal Gas Tragedy.

As we all know, decision tree is just one of many models that come from supervised learning. In supervised learning, we attempt to use features of the data to predict or model things with objective outcome labels. That is to say, each of our data points has a known outcome value, such as a categorical, discrete label like 'Survived', or a numerical, continuous value like predicting the price of a house.

Self driving car is an example where supervised learning could be used. Previously shown photos or videos being recorded while driving the car manually could be used as data to train it and teach how to drive on its own.

We can further improve this model by performing different kinds of feature engineering, which may in turn give us some better results.

9.2 Limitations

- There are about 170 passengers have no value on age. The percentage is 19.9% in whole population. It might be have impact to the result conclusion but not too huge impact. There is no appropriate value for these NaN value

- We are only making the initial conclusion that gender and pclass is correlate to the survived rate. But when we create a decision tree for it, we can know how far these two variable dependent.
- There also might be some missing columns that are not been included. Such as Passenger income, job, etc.
- We did not test for staitical significance of different variables, but rather tried too find some trends and patterns via a mostly visual EDA.
- While we analyzed the importance of some general factors, many influential factors are missing from this dataset. for example, the position of cabins in relation to the hole in the ship, number of peopl ein each cabin, where each person were at the time of accident, and the family ties between the passenger IDs.
- This is only data of 891 passengers out of 2300 on board. We can't conclude results with confidence % unless we perform t stat.

GLOSSARY

| | |
|-----|------------------------|
| CV | Cross Validation |
| RF | Random Forest |
| OOB | Out of Bound |
| SVM | Support Vector Machine |
| DT | Decision Trees |