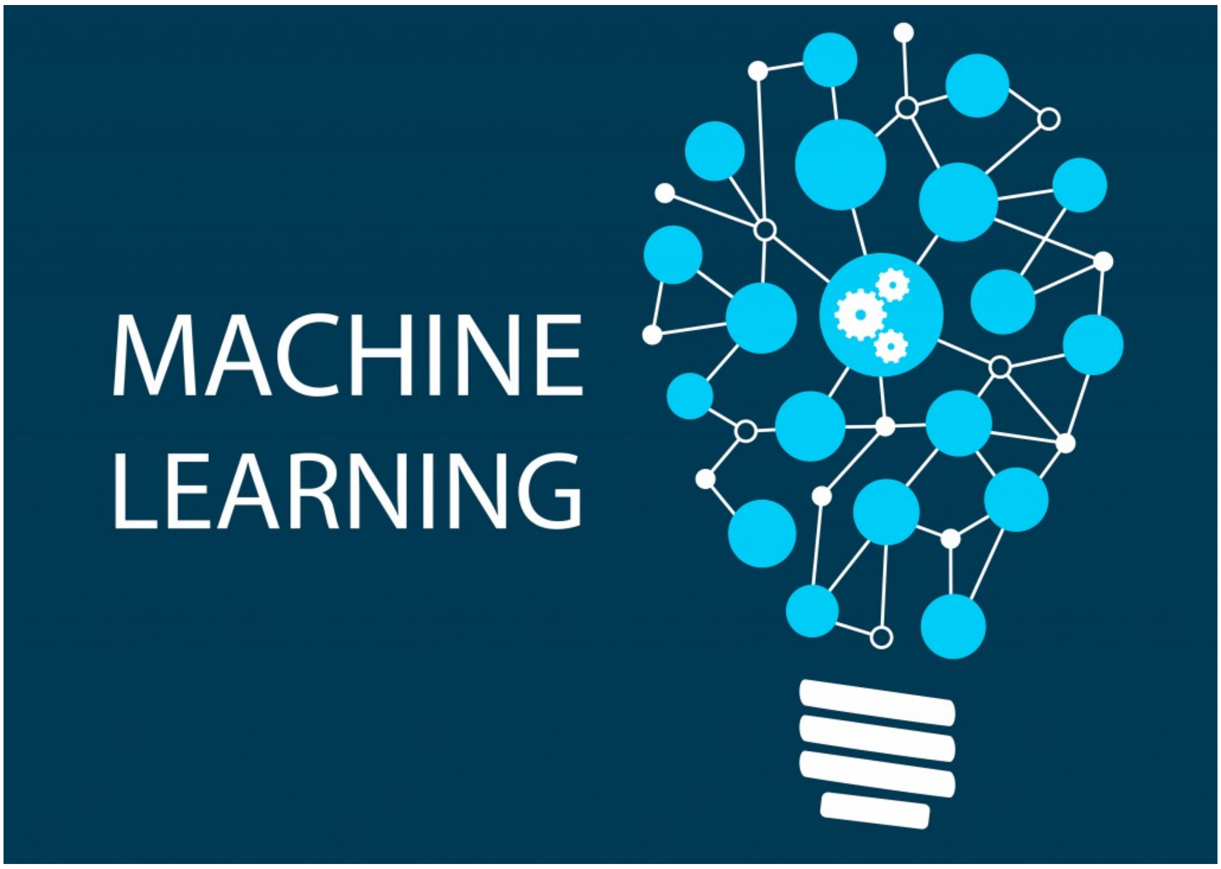
Basic introduction decision tree and randomforest technique in machine learning

Kexin Wan

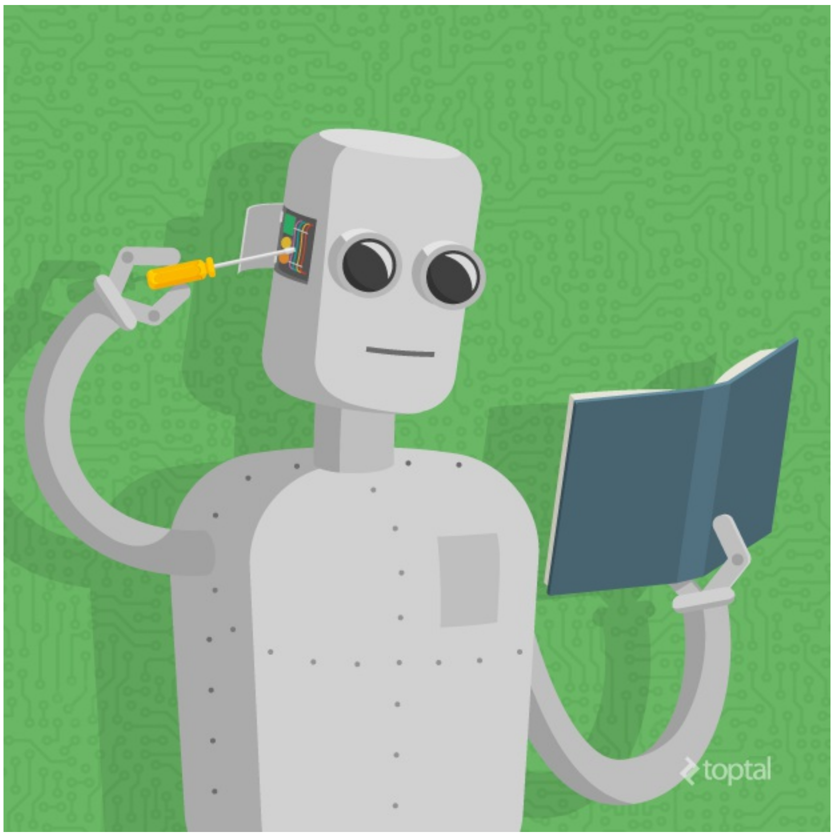
2017/11/27

Introduction:



In the 21st century, I believe most of us have heard about the heated concept of Machine learning. But what is it? Machine learning is a method of data analysis that automates analytical model building. It enables us to quickly and automatically produce models that can analyze bigger, more complex data and deliver faster, more accurate results. In this post, I am going to focus my discussion of machine learning within the field of data analytics. In the field of data analytics, machine learning is usually a method used to devise useful models for prediction. There are many approaches in machine learning that can help us build effective predictive model. But today, I am going to talk about the one that is the easiest to manipulate for entry-level machine learning learner – decision tree and randomForest. Plz Plz don't be intimidated by our topic. We don't need to write complex algorithm for the predictive computation. Instead, in this post, I am going to teach you how to use readily available packages and functions in R to build the model directly. All we need to learn is how to use the two functions: rpart() and randomForest() and interpret the statistics through plots and tables. Though I am only going to touch on the surface of machine learning, these two techniques would still be quite useful in making prediction. Hopefully, after this post, you would be able to make prediction based on other data set by yourself!

Motivation:



I have always wanted to learn some basic machine learning techniques by myself. However, as an entry-level programmer, I am intimidated by the complexity involved in building the machine learning model. After learning the readily available packages in R, I can now build predictive modeling to predict interesting results, without writing complex algorithm that I could hardly manage at my current level. I think it's a really useful technique to learn for those of us who are interested in machine learning, data mining, or just want to build some basic predictive model during our spare time. It not only introduces us to some basic concepts in machine learning, but also enables us to generate useful and interesting prediction with available data.

Background:

Machine learning is a field of computer science that gives computers the ability to learn without being explicitly programmed. Arthur Samuel, an American pioneer in the field of computer gaming and artificial intelligence, coined the term "Machine Learning" in 1959 while at IBM. Evolved from the study of pattern recognition and computational learning theory in artificial intelligence, machine learning explores the study and construction of algorithms that can learn from and make predictions on data. Decision tree learning is one of the important approaches employed in machine learning. It uses a decision tree as a predictive model, which maps observations about an item to conclusions about the item's target value. The random forest learning works by creating more number of decision trees. By creating more number of decision trees to build the predictive model, we avoid the problem of overfitting and gain more accurate prediction result.

Content:

Content introduction:

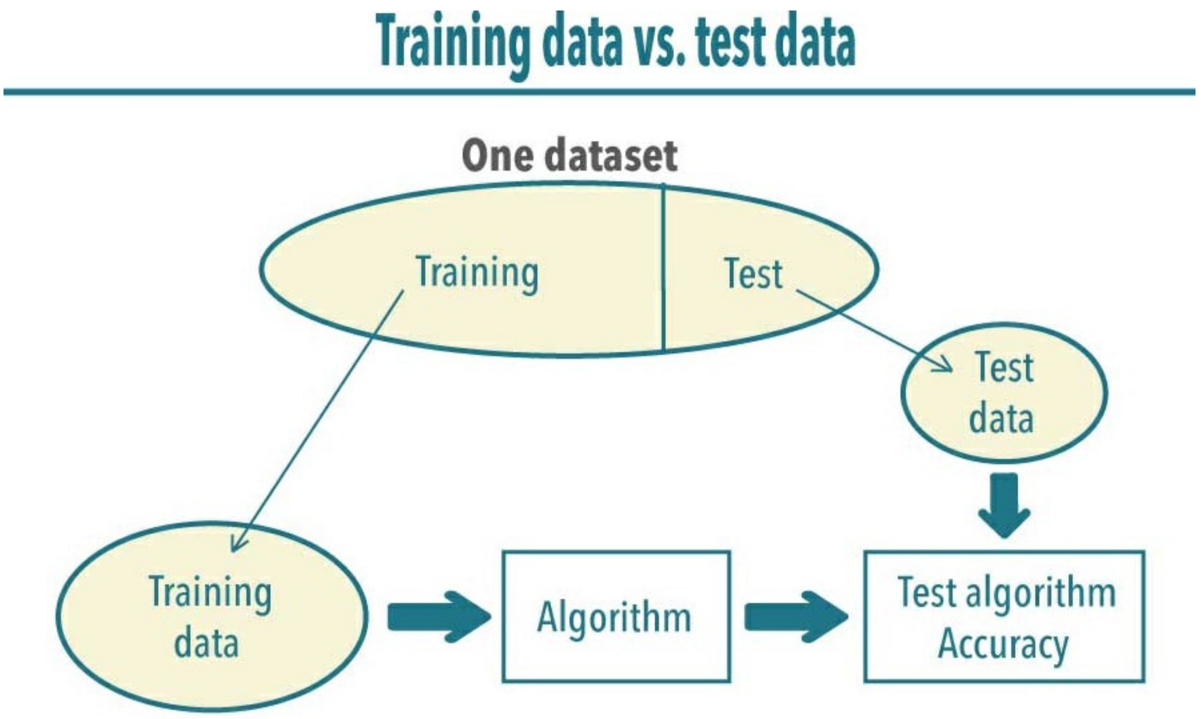
I believe those of us who have seen the movie "Titanic" know that in this movie, some individuals were more likely to survive the sinking (lucky Rose) than others (poor Jack). In this post content I am going to teach how to apply basic machine learning model, using basic techniques such as decision tree and random forest to predict a passenger's chance of surviving.

In this content, the packages we will use include:

1. "rpart" for building decision tree.
2. "rpart.plot" for visualizing decision tree.
3. "randomForest" for building randomForest.

Part 1: Importing the data

Concepts: Training Set, Test Set and their importance



- Training Set: In machine learning, a training set is a dataset used to train a model. In training the model, specific features are picked out from the training set. These features are then incorporated into the model.
- Test Set: The test set is a dataset used to measure how well the model performs at making predictions on that test set.
- The importance of training set and test set: Training Sets and Test Sets are the most important part of machine learning. Original dataset in training data can help us train our model, but we also need a test to see if the model really makes successful predictions. The test set enables us to know whether or not our model really fits the training data. And if not, the predictions of result from the test set can act as a guide for us to make revisions on our training model.
- Example:

In sentiment analysis, data scientists tend to use n-grams as features, i.e. single words or sequences of 2 or 3 consecutive words in tweets as features. Thereby, if the training set is labeled correctly, the model should be able to learn something from these features, i.e. the degree to which each feature affects the sentiment of a sentence. A test set for sentimental analysis is a dataset of tweets that are distinct from the tweets in the training set. If the prediction scores (sentiment scores) for the test set are unreasonable, data scientists always make some adjustments to the model and try again.

Applying the concepts in Titanics:

The training dataset we will use to train and build the machine learning model is the Titanic passengers raw data, which can be accessed here. This training set contains the outcome for each passenger and also their individual features. Our model will be based on “features” like passenger’s gender and class. We can also employ some basic feature engineering techniques to create new features.

The testing dataset we will use to see how well our model performs on unseen data can be accessed here. This testing dataset only contains the features of each passenger, without the outcomes for them. It is our job to use the machine learning model we have trained to predict whether or not they could survive in the sinking of the Titanic.

```r
# Read both the training and testing dataset
train_url <- "https://storage.googleapis.com/kaggle-competitions-data/kaggle/3136/train.csv?GoogleAccessId=competi
tions-data@kaggle-161607.iam.gserviceaccount.com&Expires=1512025280&Signature=Ca95bqus6rjC9yAPD1yaKv9hLqRu6h3gr06S
%2FFshp%2FsgOMNzlp95jmyKqlUam9AHdjjVkqNGNG8ztMtLORzhUrWiAcln9FqP%2Ft%2Flrn9lKZUQKJ%2FQaNmwS%2FtmeYeZQHOmKwLJ2d9waD
ZPiaxzvAhlq0lsPUoDSrTvSQK5HqL5YWjxX091CiULuDWlO2IGJWaGzhA8YXrxum621r9NFNDLhqgZ2M7OHI9X%2F20X2c8lu%2Bwo7h%2F8ERaedW
O8Omy4HFBoQvKz%2Flls790Muz3rtcTofZ61ZPh4SqHi9m1PsScEUxGqDtGaku3rZWj1ueB7ZVSlXUwlhszWRp3bRgSEGhOc%2BQ%3D%3D"

test_url <- "https://storage.googleapis.com/kaggle-competitions-data/kaggle/3136/test.csv?GoogleAccessId=competiti
ons-data@kaggle-161607.iam.gserviceaccount.com&Expires=1512025717&Signature=H1WZ2nmwa%2B4ew368DJD1CKH7BrUB%2F9vvmA
EM2ZsWzNpAJrq%2FR2XWrwQCHbJzpiKHPAMpG3c92bgnSSBr5mc9APka41CKuTOPt4sANySvLHignfSnWJKOhsrj86HhbSbr0ZOat%2BmdKeU0JoKT
REHfMjrj56Noi6OZXBTorjJayUrAkXseO51TIHff%2BLevGOcF9xe2WqhKpIimkFEJOywn0uDtu%2BPozT2SZB2Lx851m%2B%2FeHeAjQta9tjOhpm
rfxMVKjED%2BUivVmsFTmtFB7CSHhERYSn3G2uEzFsa38ZSOnb4NZ9gaa%2B5Ix%2FPVs1EkT4Zl8dYMwzTkpl6YzSIvKR5ICg%3D%3D"

train_location <- "~/desktop/titanic_project/training.csv"

test_location <- "~/desktop/titanic_project/testing.csv"

download.file(url = train_url, destfile = train_location)

download.file(url = test_url, destfile = test_location)

training_set <- read.csv(file = "~/desktop/titanic_project/training.csv")

testing_set <- read.csv(file = "~/desktop/titanic_project/testing.csv")
```

After importing the two datasets let's have a look at the structure of the two datasets.

```r
# The structure of the training dataset
str(training_set)
```

```
## 'data.frame':    891 obs. of  12 variables:
##  $ PassengerId: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Survived   : int  0 1 1 1 0 0 0 0 1 1 ...
##  $ Pclass     : int  3 1 3 1 3 3 1 3 3 2 ...
##  $ Name       : Factor w/ 891 levels "Abbing, Mr. Anthony",..: 109 191 354 273 16 555 516 625 413 577 ...
##  $ Sex        : Factor w/ 2 levels "female","male": 2 1 1 1 2 2 2 2 1 1 ...
##  $ Age        : num  22 38 26 35 35 NA 54 2 27 14 ...
##  $ SibSp      : int  1 1 0 1 0 0 0 3 0 1 ...
##  $ Parch      : int  0 0 0 0 0 0 0 1 2 0 ...
##  $ Ticket     : Factor w/ 681 levels "110152","110413",..: 524 597 670 50 473 276 86 396 345 133 ...
##  $ Fare       : num  7.25 71.28 7.92 53.1 8.05 ...
##  $ Cabin      : Factor w/ 148 levels "","A10","A14",..: 1 83 1 57 1 1 131 1 1 1 ...
##  $ Embarked   : Factor w/ 4 levels "","C","Q","S": 4 2 4 4 4 3 4 4 4 2 ...
```

```r
# The structure of the testing dataset
str(testing_set)
```

```
## 'data.frame':    418 obs. of  11 variables:
##  $ PassengerId: int  892 893 894 895 896 897 898 899 900 901 ...
##  $ Pclass     : int  3 3 2 3 3 3 3 2 3 3 ...
##  $ Name       : Factor w/ 418 levels "Abbott, Master. Eugene Joseph",..: 207 404 270 409 179 367 85 58 5 104 ..
.
##  $ Sex        : Factor w/ 2 levels "female","male": 2 1 2 2 1 2 1 2 1 2 ...
##  $ Age        : num  34.5 47 62 27 22 14 30 26 18 21 ...
##  $ SibSp      : int  0 1 0 0 1 0 0 1 0 2 ...
##  $ Parch      : int  0 0 0 0 1 0 0 1 0 0 ...
##  $ Ticket     : Factor w/ 363 levels "110469","110489",..: 153 222 74 148 139 262 159 85 101 270 ...
##  $ Fare       : num  7.83 7 9.69 8.66 12.29 ...
##  $ Cabin      : Factor w/ 77 levels "","A11","A18",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ Embarked   : Factor w/ 3 levels "C","Q","S": 2 3 2 3 3 3 2 3 1 3 ...
```

As we can see from the displayed structure, the testing set has one less variable than the training set – the survival of the passenger. Other than that, all of the other variables (containing information about individual passenger's attribute) are the same.

Part 2: Explore the relationship between passenger's attribute and survival result

Before we get started on learning real machine learning techniques, let's first explore the relationship between different passenger's feature and the survival result. This job can be accomplished by using table and prop.table, two functions that allow us to do a two-way comparison and therefore to detect which variable possess predictive value. So far in this course we have already learn about how to use table and prop.table to detect frequency and relative frequency of one variable. Now let't see how to use these two functions for two-way comparison.

For example, if I want to see the relationship between classes and the survival rate, I can do a two-way comparison like this:

```r
# See the frequency distribution in a two-way comparison
table(training_set$Pclass,training_set$Survived)
```

```
##
##         0   1
##    1   80 136
##    2   97  87
##    3  372 119
```

```
# See the proportion in a two-way comparison
prop.table(table(training_set$Pclass,training_set$Survived), margin = 1)
```

```
##
##              0         1
##    1 0.3703704 0.6296296
##    2 0.5271739 0.4728261
##    3 0.7576375 0.2423625
```

In the above code, notice there is a new argument in the function prop.table. When doing a two-way comparison using prop.table, margin is used to whether you want to have a row-wise or column-wise proportion. For row-wise proportion we use margin = 1, for column-wise proportion we use margin = 2.

From the two-way comparison between class and survival result above, we can see that the passengers in the first class have a much higher rate of survival than the passengers in the second class, while the passengers of second class have a much higher rate of survival than the passengers in the third class. Therefore, class is a variable that possess predictive value for the survival of the passenger. This information can be used in later feature engineering to build machine learning model.

Your turn

Use prop.table to do a two-way comparison between gender and survival result. To get useful information, we need row-wise proportion. See from the result if gender also possess predictive values in deciding the survival of passenger.

Answer

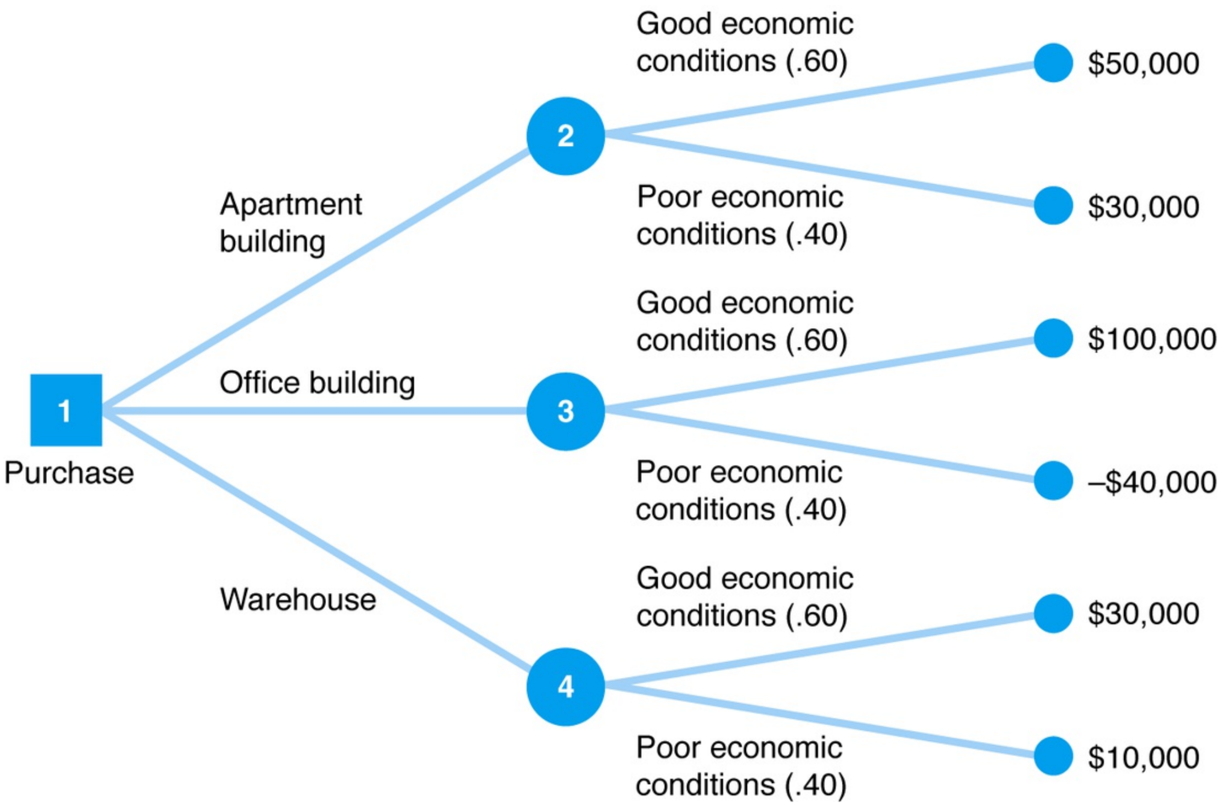
```
prop.table(table(training_set$Sex,training_set$Survived), margin = 1)
```

```
##
##                   0         1
##    female 0.2579618 0.7420382
##    male   0.8110919 0.1889081
```

From the two-way comparison table above, female possessed a much higher survival rate than male. Therefore, gender also possesses predictive values in determining the survival result.

Part 3: Learning how to use decision tree to predict for new observation

Concepts: Decision Tree and their nodes


Good economic conditions (.60)
$50,000
2
Poor economic conditions (.40)
$30,000
Apartment building
Good economic conditions (.60)
$100,000
Office building
1
3
Purchase
Poor economic conditions (.40)
–$40,000
Good economic conditions (.60)
Warehouse
$30,000
4
Poor economic conditions (.40)
$10,000

- Decision Tree: A decision tree is a map of the possible outcomes of a series of related choices. It allows an individual or organization to weigh possible actions against one another based on their costs, probabilities, and benefits. They can be used either to drive informal discussion or to map out an algorithm that predicts the best choice mathematically. Conceptually, the decision tree algorithm starts with all the data at the root node and scans all the variables for the best one to split on. After it branches into each possible outcome, those outcomes would then lead to additional nodes, which branch off into other possibilities. This gives it a treelike shape. The final nodes at the bottom of the decision tree are known as end nodes, and the majority vote of the observations in that node determine how to predict for new observations that end up in that terminal node.

- Chance nodes, decision nodes and end nodes: There are three different types of nodes: chance nodes, decision nodes, and end nodes. A chance node, represented by a circle, shows the probabilities of certain results. A decision node, represented by a square, shows a decision to be made. An end node shows the final outcome of a decision path.
- Example: As can be seen from the above diagram, a square (a deicison node) indicating a decision to be made, a circle (a chance node) indicates possible decision to be made and the probability of their result, and a triangle (an end node) shows the final outcome of the path.

Applying the Concepts in Titanics:

R provides a way for us to enjoy the convenience of decision tree. Instead of writing a decision tree algorithm by ourselves, we can use the readily available R package rpart to build a decision tree.

```r
# Importing rpart package
library(rpart)
```

The rpart() function inside rpart is the most important function in helping us build the decision tree. The function has the following structure:

> rpart(formula, data, weights, subset, na.action = na.rpart, method, model = FALSE, x = FALSE, y = TRUE, parms, control, cost, …)

Some of the most commonly used arguments include formula, data and method.

- Formula: specifying variable we are interested in predicting (Predicted_ Variable) and variables we will use to build model for prediction (Model_ Variable). Formula typically has the form of **formula = Predicted_variable ~ Model_Variable_1 + Model_Variable_2 + … Model_Variable_n
- Data: the training data set we will use to build the decision tree.
- method: method would be one of “anova”, “poisson”, “class” or “exp”. The choice depends on the type of prediction you want. If the predicted variable is a survival object, then method = “exp” is assumed, if the predicted variable has 2 columns then method = “poisson” is assumed, if the predicted variable is a factor then method = “class” is assumed, otherwise method = “anova” is assumed.
- cp: This argument is put inside rpart.control, which contains various parameters that control aspects of the rpart fit. complexity parameter. Any split that does not decrease the overall lack of fit by a factor of cp is not attempted. The main role of this parameter is to save computing time by pruning off splits that are obviously not worthwhile.
- minsplit: This argument is put inside rpart.control, which contains various parameters that control aspects of the rpart fit. It is the minimum number of observations that must exist in a node in order for a split to be attempted.

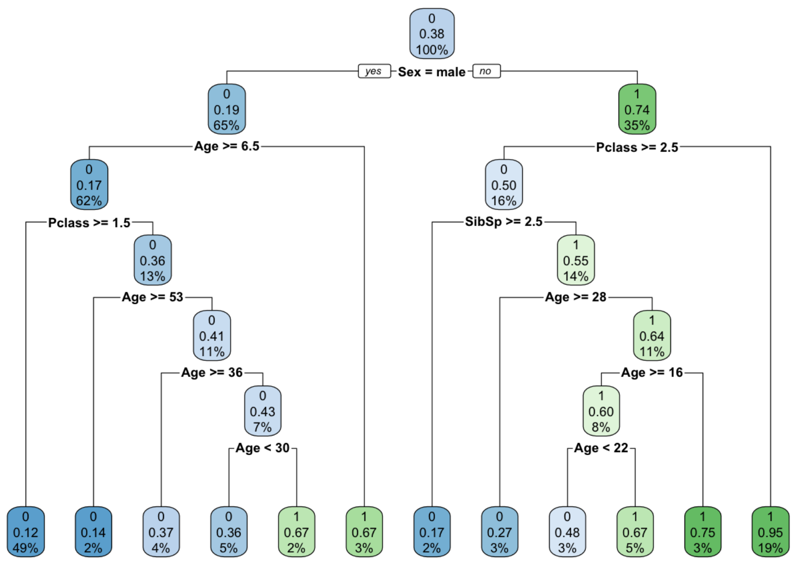
Okay, with some understanding of how rpart function works, now let’s try to build a decision tree predicting the survival of passenger based on four variables: Pclass, sex, age and SibSp. In this decision tree, I will use a cp of 0 so that there is no stopping of splits and a minsplit of 40.

```r
# Building our first decision tree
tree_1 <- rpart(Survived ~ Pclass + Sex + Age + SibSp, training_set, method = "class", control = rpart.control(cp
= 0, minsplit = 40))
```

After building our first decision tree, let’s try to use the helpful rpart.plot() function to visualize it.

```r
library(rpart.plot)
```

```r
# Visualizing our first decision tree using rpart.plot()
rpart.plot(tree_1)
```



In the above example, notice that we use a minsplit of 40. You might wonder why don’t we use a minsplit of 10 or even smaller to get a more accurate result. The reason is because by creating too specified rules we overfitted the tree. That is, the model is only specific to the training set we are using. It cannot be generalized to other data set to get good and accurate result.

Don't worry if you don't fully understand the graph now, we will do the explanations later. Now, let's do some exercises to see if you have managed the concepts.

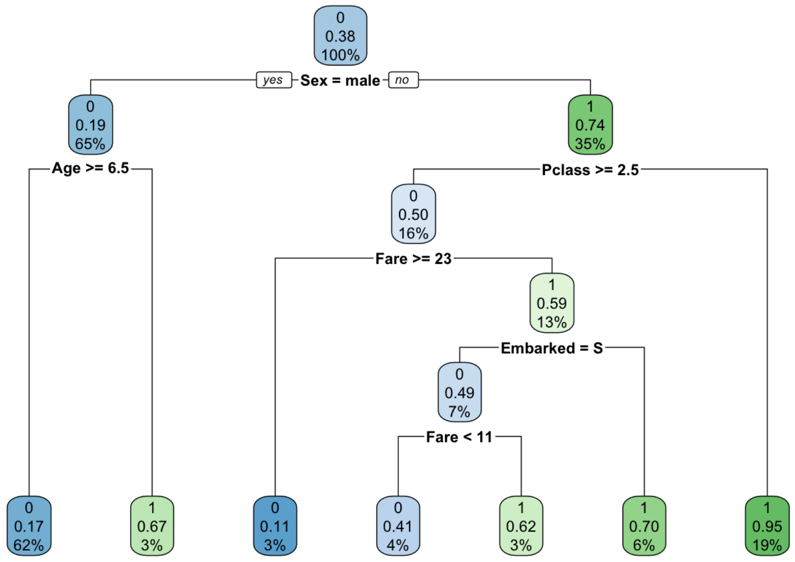
Your turn: using rpart to build a tree two, which also predict the survival of a passenger but this time based on more variables: Pclass, Sex, Age, SibSp, Parch, Fare and Embarked. For control, we will use a minsplit of 50 and cp of 0.

```
# Put the code inside
```

Answer:

```
# Building the second decision tree
tree_2 <- rpart(Survived ~ Pclass + Sex + Age + SibSp + Parch + Fare + Embarked, training_set, method = "class", c
ontrol = rpart.control(cp = 0, minsplit = 50))

# Visualizing the second decision tree
rpart.plot(tree_2)
```



Now let's take a look at the graph and try to understand it.

The number (0 or 1) on the top of each square indicates the fate of the majority represented in this node. For example, at the root node, it shows a 0, given that only 38% of our population survive. However, in the node corresponding to sex = female, the number is 1, since among the female population 74% survive.

The number below 0/1 indicates the way that the node is voting. For example, on the root node, 0.38 indicates that among the total population, only 0.38 survive while 0.62 die. On the node corresponding to sex=female, Pclass > 2.5 and Fare >= 23, only 0.11 of the population in this node survive while 0.89 die. Therefore, any passenger in the testing set that satisifies this condition will have a predicted result of death (remember that decision tree making decision based on majority vote).

The percentage below the decimal number indicates the proportion of the population that resides in this node, or bucket (here at the top level it is everyone, 100%).

The final nodes at the bottom of the decision tree are known as terminal nodes, or sometimes as leaf nodes. After all the boolean choices have been made for a given passenger, they will end up in one of the leaf nodes, and the majority vote of all passengers in that bucket determine how we will predict for new passengers with unknown fates.

After building the decision tree, we can now use the tree to predict the survival of each passenger (Don't forget that's the reason we want to build the tree!)

To make the prediction we will use the predict() function, which typically takes in three arguments: The tree or randomforest (the model we built) that will be used for predicting the result for training data set, the testing dataset, and the type of the variable we will predict (recall the method argument in rpart()).

```
# Making prediction using the predict() function.
testing_set_tree <- testing_set
tree_prediction <- predict(tree_2, testing_set_tree, type= "class")
# Combine the prediction result with passenger ID
tree_result <- data.frame(testing_set$PassengerId, tree_prediction)
```

```
# Take a look at the resulk
tree_result[1:30,]
```

```
##    testing_set.PassengerId tree_prediction
## 1                      892               0
## 2                      893               0
## 3                      894               0
## 4                      895               0
## 5                      896               1
## 6                      897               0
## 7                      898               1
## 8                      899               0
## 9                      900               1
## 10                     901               0
## 11                     902               0
## 12                     903               0
## 13                     904               1
## 14                     905               0
## 15                     906               1
## 16                     907               1
## 17                     908               0
## 18                     909               0
## 19                     910               0
## 20                     911               1
## 21                     912               0
## 22                     913               0
## 23                     914               1
## 24                     915               0
## 25                     916               1
## 26                     917               0
## 27                     918               1
## 28                     919               0
## 29                     920               0
## 30                     921               0
```

Part 4: Random forest – the machine learning technique that overcomes overfitting

Concepts: Random Forest and its condition

- Random Forest: Random forests are an ensemble learning method for classification, 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. Random forest grows multiple classification trees using the training set. At the time of prediction, each tree is used to come up with a prediction and every outcome is counted as a vote. For example, if you have trained 10 trees with 9 saying a passenger in the test set will survive and 1 says he will not, the passenger will be classified as a survivor.

- Condition for random forest: The most important condition to employ random forest technique is no missing value in the data frame. So before we start to build machine learning model using random forest, the first step is always to clear all of the missing values in the data frame. And usually we can use the decision tree to help us decide the value of the missing variable using existing variables.

Applying the Concepts in Titanics:

Before applying the random forest technique on our data set, let's first clearing all of the missing values in our training dataset. By using is.na(tesing_set) to check, we find that there are missing values in Embarked, Fare and Age.

To deal with these missing values, we can either directly replace them with the mode for factor variable, or median for numeric variable. Or more precisely, we can build a specific decison tree to predict the missing value.

```
testing_set$Survived <- NA
all_data <- rbind(training_set,testing_set)
# replace the missing embarked values with the mode of the embarked variable
Mode <- function(x) {
  ux <- unique(x)
  ux[which.max(table(match(x, ux)))]
}
replace <- Mode(all_data$Embarked)
training_set$Embarked[is.na(training_set$Embarked)] <- replace

# replace the missing Fare value with the median of the fare variable
replace_2 <- median(all_data$Fare,na.rm=TRUE)
training_set$Fare[is.na(training_set$Fare)] <- replace_2

# Build a decision tree to predict missing value for age
predicted_age <- rpart(Age ~ Pclass + Sex + SibSp + Parch + Fare + Embarked, data = all_data[!is.na(all_data$Age),
], method = "anova")
all_data$Age[is.na(all_data$Age)] <- predict(predicted_age, all_data[is.na(all_data$Age),])

# Spliting all_date into training and testing data without any missing values
train <- all_data[1:891,]
test <- all_data[892:1309,]
```

After clearing all of the missing values, we can now move to the next step – to build a random forest and use the random forest to predict the survival of each passenger. Before doing that let's first import the randomForest package.

```
library(randomForest)
```

```
## randomForest 4.6-12
```

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

Now let's use the randomForest function inside the package to build a simple randomForest. The function has the following structure:

> randomForest(x, y=NULL, xtest=NULL, ytest=NULL, ntree=500, mtry=if

```
             > (!is.null(y) && !is.factor(y)), max(floor(ncol(x)/3), 1) else
               > floor(sqrt(ncol(x))), replace=TRUE, classwt=NULL, cutoff, strata,

             This might look complicated, but don't worry, we only need to understand some of the most commonly us
ed arguments, which include formula, data and ntree.
```

- formula: The format of the formula here is similar to the format in the decision tree. But since there is no argument class here to tell the function that our predicted variable is a categorical variable we need to turn the variable into factor when necessary.
- data: Similar to the rpart() function the data argument takes in the training data set.
- ntree: The ntree argument specifies the number of tree we want the forest to grow. This should not be set to too small a number, to ensure that every input row gets predicted at least a few times.

Now let's start to build a random Forest predicting the survival of each passenger using variables Pclass, Sex, Age, SibSp, Parch, Fare and Embarked.

```
# Build a randomforest
the_forest <- randomForest(formula=as.factor(Survived) ~ Pclass + Sex + Age + SibSp + Parch + Fare + Embarked, dat
a=train, ntree=1000)

# Use the built randomforest to predict the survival of each passenger in the test
the_prediction <- predict(the_forest, test)

# Combine the prediction with passenger ID to see the predicted result
result <- data.frame(test$PassengerId,the_prediction, row.names = test$PassengerId)
```

Good job! We have succeeded in predicting the result. It's time to take a look at it!

```
result[1:30,]
```

```
##     test.PassengerId the_prediction
## 892              892              0
## 893              893              0
## 894              894              0
## 895              895              0
## 896              896              0
## 897              897              0
## 898              898              0
## 899              899              0
## 900              900              1
## 901              901              0
## 902              902              0
## 903              903              0
## 904              904              1
## 905              905              0
## 906              906              1
## 907              907              1
## 908              908              0
## 909              909              0
## 910              910              0
## 911              911              0
## 912              912              1
## 913              913              0
## 914              914              1
## 915              915              0
## 916              916              1
## 917              917              0
## 918              918              1
## 919              919              0
## 920              920              0
## 921              921              0
```

Your turn: Since randomforest uses randomization to generate the result, we need to set a seed whenever we are creating a random Forest. Now, set the seed to make a randomforest again by yourself.

```
# Put the code inside
```

Answer:

```
set.seed(232)
the_forest_2 <- randomForest(formula=as.factor(Survived) ~ Pclass + Sex + Age + SibSp + Parch + Fare + Embarked, d
ata=train, ntree=1000)
```

Conclusion

Congratulations! Now we have completed building the predictive machine learning model for our Titanic passenger testing dataset. Take a look at the result to see if the result really matches your expectation. In this post, you learn a lot of useful techniques in machine learning: * The concepts of training set and testing set and their importance * Using prop.table and table for two-way comparison to see if variable possess predictive value. * Learning the concept of decision tree and their nodes. * Learning how to use rpart() function to build decision tree and predict() to predict the target value for the testing dataset. * Use the rpart.plot() function to visualize the decision tree and understand the information displayed in the graph. * Learning the concept of random forest, its condition and why it can avoid overfitting. * Learning how to clean data before applying randomforest technique and how to use randomForest() function to build model for prediction.

Take Home Message

Rpart and randomForest are two useful techniques for us to build predictive machine learning model. With helpful graphs coming with these packages we are not only able to predict accurate target values but also understand the result statistically.

References

- Nexosis Training Set vs. Test Set
- Lucid Chart: What is a decision tree?
- R documentary: function rpart()
- Understanding decision tree
- Plotting with rpart.plot
- What is a random forest
- R documentary: function randomForest()