TABLE OF CONTENTS

UNIT 1: INTRODUCTION TO MACHINE LEARNING 3

WHY LEARN MACHINE LEARNING 4

DIFFERENCE BETWEEN ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING 5

APPLICATION OF MACHINE LEARNING 5

UNIT 2: GETTING STARTED: DATA PREPROCESSING 6

GET THE DATASET 6

IMPORT THE LIBRARIES 6

IMPORT THE DATASET 7

MISSING DATA 7

CATEGORICAL DATA 8

TRAINING SET AND TEST SET 8

FEATURE SCALING 9

UNIT 3: REGRESSION 10

SIMPLE LINEAR REGRESSION 10

ASSUMPTIONS OF LINEAR REGRESSION 15

MULTIPLE LINEAR REGRESSION 21

POLYNOMIAL REGRESSION 25

SUPPORT VECTOR REGRESSION (SVR) 28

DECISION TREE REGRESSION 30

RANDOM FOREST REGRESSION 34

INTERPRETING COEFFICIENT OF REGRESSION 36

EVALUATING REGRESSION MODELS SPECIFICATION 37

OTHER TYPES OF REGRESSION MODELS 37

CONCLUSION 39

UNIT 4: CLASSIFICATION 40

LOGISTIC REGRESSION 41

K-NEAREST NEIGHBORS (K-NN) 45

SUPPORT VECTOR MACHINE (SVM) 49

KERNEL SVM 51

NAIVE BAYES 52

DECISION TREE CLASSIFICATION 54
RANDOM FOREST CLASSIFICATION 55
EVALUATING CLASSIFICATION MODELS PERFORMANCE 57
UNIT 5: CLUSTERING 61
K-MEANS CLUSTERING 61
PARTITIONING AROUND MEDOIDS (PAM) 64
HIERARCHICAL CLUSTERING 66
UNIT 6: ASSOCIATION RULE LEARNING 72
APRIORI 72
ECLAT 72
UNIT 7: REINFORCEMENT LEARNING 73
UPPER CONFIDENCE BOUND 73
THOMPSON SAMPLING 73
UNIT 8: NATURAL LANGUAGE PROCESSING 74
UNIT 9: DEEP LEARNING 77
ARTIFICAL NEURAL NETWORKS 77
CONVOLUTIONAL NEURAL NETWORKS 77
UNIT 10: APPLICATION: RECOMMENDATION SYSTEM 78
UNIT 11: APPLICATION: FORECASTING ALGORITHMS 79
UNIT 12: APPLICATION: FACE RECOGNITION ALGORITHM 80
UNIT 13: APPLICATION: SOCIAL MEDIA ANALYTICS 81
UNIT 14: CONCLUSION 82
REGRESSION 82
CLASSIFICATION 84
CLUSTERING 85
HOW TO EVALUATE MACHINE LEARNING ALGORITHMS? 86

UNIT 1: INTRODUCTION TO MACHINE LEARNING

It is no doubt that machine learning is increasingly gaining popularity and has become the hottest trend in the tech industry. Machine learning is incredibly powerful to make predictions or calculated suggestions based on large amounts of data. So, what is a machine learning? Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it learn for themselves. How does a system learn?

A Computer program is said to learn from Experience "E" with respect to some task "T" and some performance measure "P", if its performance on "T", as measured by "P", improves with "E".

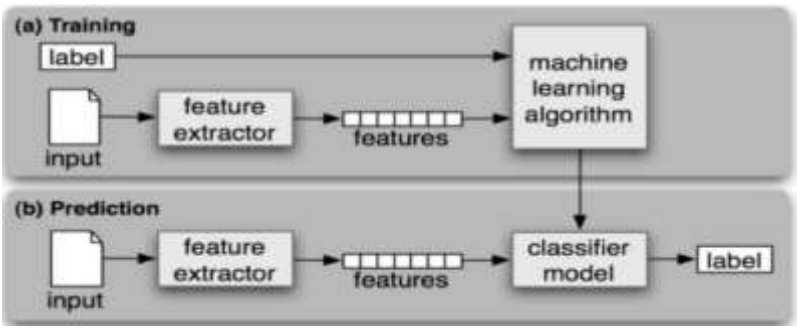


Figure 1: Machine Learning Workflow

The process of learning begins with observations or data (Training Data), such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide. The primary aim is to allow the computers learn automatically without human intervention or assistance and adjust actions accordingly.

Machine learning algorithms are often categorized as ***supervised*** or ***unsupervised***.

Supervised machine learning algorithms can apply what has been learned in the past to new data using labeled examples to predict future events. Starting from the analysis of a known training dataset, the learning algorithm produces an inferred function to make predictions about the output values. The system is able to provide targets for any new input after sufficient training. The learning algorithm can also compare its output with the correct, intended output and find errors in order to modify the model accordingly.

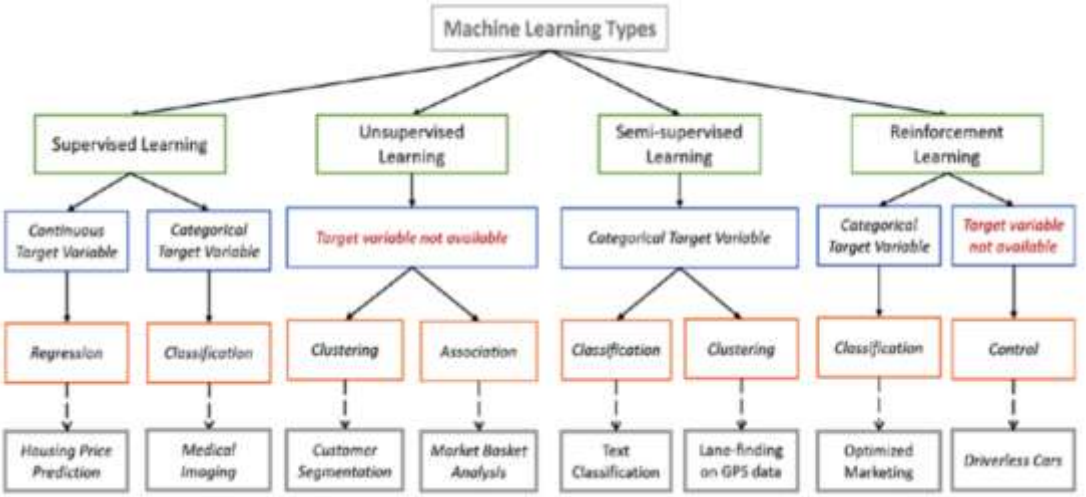
Unsupervised machine learning algorithms are used when the information used to train is neither classified nor labeled. Unsupervised learning studies how systems can infer a function to describe a hidden structure from unlabeled data. The system doesn't figure out the right output, but it explores the data and can draw inferences from datasets to describe hidden structures from unlabeled data.

Semi-supervised machine learning algorithms fall somewhere in between supervised and unsupervised learning, since they use both labeled and unlabeled data for training – typically a small amount of labeled data and a large amount of unlabeled data. The systems that use this

method are able to considerably improve learning accuracy. Usually, semi-supervised learning is chosen when the acquired labeled data requires skilled and relevant resources in order to train it / learn from it.

Reinforcement machine learning algorithms is a learning method that interacts with its environment by producing actions and discovers errors or rewards. Trial and error search and delayed reward are the most relevant characteristics of reinforcement learning. Simple reward feedback is required for the agent to learn which action is best; this is known as the reinforcement signal.

Still no cleared about these methods? Not to worry. We will learn and practice in the chapters to come. One thing we should understand is Machine learning enables analysis of massive quantities of data and generally delivers faster, more accurate results in order to identify profitable opportunities or dangerous risks, it may also require additional time and resources to train it properly.



WHY LEARN MACHINE LEARNING

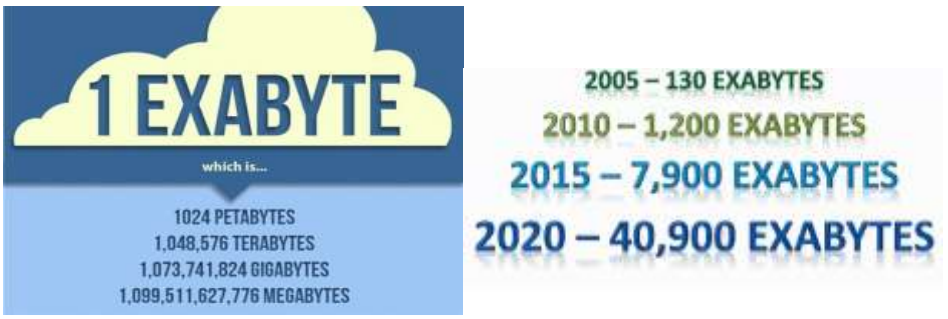


Figure 2: Exabyte and growth of data. (Source: IDC)

We live in 21st century and the data is everywhere. Every second tons of data are produced, if could be the text messages you are sending or posting a pic on Instagram. Since the dawn of time until 2005, humans had created 130 Exabytes of data. By 2020, its expected to reach

40,900 Exabytes. To understand this, we know that one letter takes about 1 byte of space. This is a phenomenal growth of the data we create. This is the reality of the world we live in. Our capacity to process this data is very less and even though machine can process much more data but still it will not be possible to process all these data. Machine learning provides us with that opportunity. Machine Learning algorithm can help us to step us to analyze all these data and help us to create value out of it.

DIFFERENCE BETWEEN ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

Artificial Intelligence (AI) and Machine Learning (ML) are two very hot buzzwords, and are often seem to be used interchangeably. They are not quite the same thing. Let's understand the difference between the two:

- Artificial Intelligence is the broader concept of machines being able to carry out tasks in a way that we would consider “smart”.
- Machine Learning is a current application of AI based around the idea that we should really just be able to give machines access to data and let them learn for themselves.

Artificial Intelligences – devices designed to act intelligently – are often classified into one of two fundamental groups – applied or general. Applied AI is far more common – systems designed to intelligently trade stocks and shares, or maneuver an autonomous vehicle would fall into this category.

Engineers and Scientists have realized that rather than teaching computers and machines how to do everything, it would be far more efficient to code them to think like human beings, and then plug them into the internet to give them access to all of the information in the world. This is machine learning. A Neural Network is a computer system designed to work by classifying information in the same way a human brain does. It can be taught to recognize, for example, images, and classify them according to elements they contain. Essentially it works on a system of probability – based on data fed to it, it is able to make statements, decisions or predictions with a degree of certainty. The addition of a feedback loop enables “learning” – by sensing or being told whether its decisions are right or wrong, it modifies the approach it takes in the future.

APPLICATION OF MACHINE LEARNING

Some of the most common examples of machine learning are:

1. Netflix’s algorithms to make movie suggestions
2. Amazon’s algorithms that recommend books based on books you have bought before.
3. Self-driving car
4. Knowing what customers are saying about you on Twitter?
5. Fraud detection? One of the more obvious, important uses in our world today.
6. Speech recognition, Natural language processing and Computer vision
7. Computational biology and Medical outcomes analysis
8. Virtual Reality (VR) games, etc

Readers can add more such applications to the list.

UNIT 2: GETTING STARTED: DATA PREPROCESSING

Data preprocessing is an umbrella term that covers an array of operations data scientists will use to get their data into a form more appropriate for what they want to do with it. For example, before performing sentiment analysis of twitter data, you may want to strip out any html tags, white spaces, expand abbreviations and split the tweets into lists of the words they contain. When analyzing spatial data, you may scale it so that it is unit-independent, that is, so that your algorithm doesn't care whether the original measurements were in miles or centimeters.
Preprocessing is important step and we have to do it before we start machine learning to make sure that there is no error that gets into the model because of the data we have. This may be the most boring part but very crucial so that we get our analysis right! I have purposely put the headers as a separate section because we will have to repeat them every time we work on performing analysis. Let's get started.

GET THE DATASET

Step 1 is to get the dataset to work on the analysis. The dataset to be worked with this book has been placed on the Github location:
https://github.com/swapnilsaurav/MachineLearning

We will mention the filename of the dataset that can be downloaded from the above location for each of the exercise as we go along. For PreProcessing exercise, download the file - 1_Data_PreProcessing.csv from the above location. This dataset contains four columns – Regions (Region), number of Salesperson (Salesperson) in that region, Quotation that was given for a contract (Quotation) and if the team was awarded the contract or not (Win). The data from multiple contract is presented together hence you will see the regions are repeated. We have 14 observations. Before we proceed with the analysis, we will have to differentiate between dependent variables and the independent variables. In this example, the independent variables are the first 3 columns – Region, Salesperson and Quotation and dependent variable is Win. In our learning of the machine learning, we will use the independent variable to predict the dependent variable. We will use Region, Salesperson and Quotation column to predict if the contract can be won or not.

Region	Salesperson	Quotation	Win
North	44	72000	No
South	27	48000	Yes
East	30	54000	No

Figure 3: Dataset snapshot

IMPORT THE LIBRARIES

We will use R Studio to do our R programs. Create a new R file where we will perform all our preprocessing steps.
Step 1 is to install the library that are required for our work. A library is a tool that you can use to make a specific job. Packages are collections of R functions, data, and compiled code in

a well-defined format. The directory where packages are stored is called the library. R comes with a standard set of packages. Others are available for download and installation. Once installed, they have to be loaded into the session to be used. In this section, we will talk about specific libraries that can be used for specific algorithms for machine learning.
For preprocessing steps we do not need any libraries but let's understand how to install and include libraries when required.
ggplot2 is a plotting system for R, based on the grammar of graphics. It takes care of many of the fiddly details that make plotting a hassle (like drawing legends) as well as providing a powerful model of graphics that makes it easy to produce complex multi-layered graphics.

To install a package, in the console, type: install.packages("ggplot2 ") and hit enter.

```
install.packages("ggplot2 ")
```

IMPORT THE DATASET

Before importing the dataset, set the working directory. You can get the current working directory using the command: getwd(). To set a directory different than this or point it to the directory containing the dataset, use the following command: setwd("D:\\MachineLearning")
Remember "\" is the escape character hence we have "\\" here (refer R tutorial)
Now lets read the dataset from the current directory:

```
setwd("D:\\MachineLearning")
dataset = read.csv("1_Data_PreProcessing.csv")
```

To view the imported data, say print(dataset) or click on the dataset variable under the global environment.

	Region	Salesperson	Quotation	Win
1	North	44	72000	No
2	South	27	48000	Yes
3	East	30	54000	No
4	South	38	61000	No
5	East	40	NA	Yes

Figure 4: Snapshot of the dataset imported into R studio

MISSING DATA

One of the first problem you will face is handling the missing values. This happens very frequently while working with the real time dataset hence you need to learn the trick to handle missing data and make it look good so that machine learning algorithm can run correctly. As you can see in the current dataset, we have 2 missing data – one under Quotation and other under Salesperson. One option is to remove the missing data altogether from the analysis but that's not the right approach. Another option which is the most common option is to be replace the missing data with the mean of all the other value. Let's use this strategy for our exercise.
We will replace the value from both Salesperson and Quotation columns where missing values are present using ifelse() function as below:

```
dataset$Salesperson = ifelse(is.na(dataset$Salesperson),
                    ave(dataset$Salesperson,FUN = function(x) mean(x, na.rm=TRUE)),
                    dataset$Salesperson)
```

Add similar code for Quotation column also. We have replaced all the missing data from the column.

CATEGORICAL DATA

Categorical variables represent types of data which may be divided into groups. In the current dataset we have 2 such variables – Region (categories are: North, South, East, West) and Win (categories are: Yes, No). Its important to convey to the machine learning algorithms about categorical values so that it doesn't treat them as regular values. We will use *factor()* in R to convert them into categories.

```
dataset$Region = factor(dataset$Region)
print(dataset$Region)
```

print function will show that there are four levels – East, North, South, West (set in alphabetical order). But its better for analysis purpose if we convert them into numbers rather than the characters, so let's rewrite above statement to convert into numerical levels.

```
dataset$Region = factor(dataset$Region,
                    levels = c('East', 'North', 'South', 'West'),
                    labels = c(1,2,3,4))
```

TRAINING SET AND TEST SET

We need to split our dataset into training set and test set for machine learning algorithm. We need to split the dataset because we want the machine to learn the algorithm and make prediction. We are going to build our machine learning algorithm on the training dataset and test it on the test set to know how well it has "learnt" the correlation.
Next question that is asked is how much of the data should be Training set and Test set. Best practice is to choose 80% of the data as Training set and 20% as Test set. In this case, we have
We need to import ***catools*** library which will make our job easier and then activate it using library function.

```
install.packages("caTools ")        #Package name is with quotes
library(caTools)       #Package name is without quotes
```

Algorithm can use any random number to process the data so every time you run the same code you will notice slight variation in the output. Hence we will use ***set.seed(seednumber)*** function for now. This is not suggested when you do actual analysis. The seed number you choose is the starting point used in the generation of a sequence of random numbers, which is why (provided you use the same pseudo-random number generator) you'll obtain the same results given the same seed number. Do not set the seed too often.

```
install.packages("caTools ")        #Package name is with quotes
library(caTools)       #Package name is without quotes
```

sample.split() will split the dataset into training and test dataset. First parameter is the dependent variable (Win in the given dataset). Next we give is split ratio for training set.

```
set.seed(123456789)   #Any number
split = sample.split(dataset$Win, SplitRatio = 0.8)   #Split ratio for Training Set
```

The split variable will return TRUE or FALSE for each row. TRUE means that the data will go to Training set and FALSE means that the data will go to Test set.

```
> print(split)
 [1]  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
```

Now we will introduce 2 variables, once each for Training set and Test set which will be subset of split.

```
training_set = subset(dataset, split==TRUE)
test_set = subset(dataset, split==FALSE)
```

FEATURE SCALING

Looking at the dataset, the Salesperson independent variable value varies from 27 to 48 and the Quotation value varies from 40000 to 90000. In scenarios like these, owing to mere greater numeric range, the impact on response variables by the feature having greater numeric range could be more than the one having less numeric range, and this could, in turn, impact prediction accuracy. The objective is to improve predictive accuracy and not allow a particular feature impact the prediction due to large numeric value range. Thus, we may need to normalize or scale values under different features such that they fall under common range.

There are couple of ways to scale the value:
Min-Max Normalization: Data frame could be normalized using Min-Max normalization technique which specify following formula to be applied on each value of features to be normalized: (X - min(X))/(max(X) - min(X))
Z-Score Normalization: The disadvantage with min-max normalization technique is that it tends to bring data towards the mean. If there is a need for outliers to get weighted more than the other values, z-score standardization technique suits better. Formula is:
(X - mean(X))/(Std. Dev.(X))

In order to achieve z-score standardization, one could use R's built-in ***scale()*** function:

```
training_set = scale(training_set)
```

```
Error in colMeans(x, na.rm = TRUE) : 'x' must be numeric
```

This will throw error because scale expects all the data to be numeric type but remember we have converted Win and Regions as Factors. Factors are not numeric. Also, logically we should not be including Factors in Scaling. We need to scale only the columns: Salesperson (Column Index 2) and Quotation (Column Index 3). Above code will be re-written as:

```
training_set[, 2:3] = scale(training_set[, 2:3])
test_set[, 2:3] = scale(test_set[, 2:3])
```

That was the last step in Data Preprocessing and now our data is ready to be used by Machine Learning algorithm. We have learnt all the basic steps in data preprocessing. We are not going to use all these steps always. It depends on the given dataset. Lets learn the models now.

UNIT 3: REGRESSION

Regression models (both linear and non-linear) are used for predicting a real value, like salary for example. If your independent variable is time, then you are forecasting future values, otherwise your model is predicting present but unknown values. Regression technique vary from Linear Regression to SVR and Random Forests Regression.
In this part, you will understand and learn how to implement the following Machine Learning Regression models:

- Simple Linear Regression
- Multiple Linear Regression
- Polynomial Regression
- Support Vector for Regression (SVR)
- Decision Tree Classification
- Random Forest Classification

SIMPLE LINEAR REGRESSION

Simple linear regression is a statistical method that allows us to summarize and study relationships between two continuous (quantitative) variables:

- One variable, x, is regarded as the predictor, explanatory, or independent variable.
- The other, denoted y, is regarded as the response, outcome, or dependent variable.

Simple linear regression gets its adjective "simple," because it concerns the study of only one predictor variable. In contrast, multiple linear regression, which we study in the next section, gets its adjective "multiple," because it concerns the study of two or more predictor variables.

Download *2_Marks_Data.csv* from: https://github.com/swapnilsaurav/MachineLearning

Dataset contains number of hour students have studied per week and the marks obtained in the final examination. Problem statement is to find the correlation between the number of hour and the marks obtained and then we will be able to predict the marks student can get if we know the number of hours he or she spend studying.

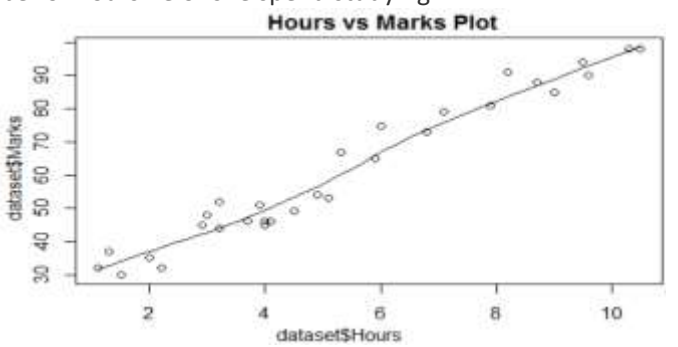


Figure 5: Scatter Plot (Hours v Marks)

Linear regression consists of finding the best-fitting straight line through the points. The best-fitting line is called a regression line. Scatter Plot which represents the relationship between two variables can be drawn between the X and Y variable to see the relationship using the function ***scatter.smooth()***.

```
dataset = read.csv("2_Marks_Data.csv")
scatter.smooth(x=dataset$Hours, y=dataset$Marks, main="Hours vs Marks Plot"
```

The diagonal line in the scatter plot is the regression line and consists of the predicted score on Y for each possible value of X. The points located away from the regression line represent the errors of prediction. A line that fits the data "best" will be one for which the n prediction errors — one for each observed data point — are as small as possible in some overall sense. One way to achieve this goal is to invoke the "least squares criterion," which says to "minimize the sum of the squared prediction errors." That is:

- The equation of the best fitting line is: y=b0+b1xi
- b0 is the intercept and b1 represents the slope of the line
- We just need to find the values b0 and b1 that make the sum of the squared prediction errors the smallest it can be.

Let's now see how we can solve it using R. Complete code with explanation given in the comment section is given below:

```
dataset = read.csv("2_Marks_Data.csv")
scatter.smooth(x=dataset$Hours, y=dataset$Marks, main="Hours vs Marks Plot"
#install.packages("caTools ")    #Install is required only once
library(caTools)
#Splitting the dataset into Training set and Test set
set.seed(123456789)   #Any number
split = sample.split(dataset$Marks, SplitRatio = 0.8)   #Split is based on Dependent variable
training_set = subset(dataset, split==TRUE)
test_set = subset(dataset, split==FALSE)

#Feature Scaling is not required because
#the package we will use for analysis takes care of Feature Scaling.
#Next Step: Fitting Simple Regression to the training set
regressor = lm(formula= Marks~Hours,
          data = training_set)
#To read the output, call summary function and
#see the details it displays on the console
summary(regressor)
```

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  20.7583     1.8059   11.49 4.08e-12 ***
Hours         7.5675     0.3009   25.15  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Figure 6: Simple Linear Regression Output

Lot of information is displayed but we are particularly interested in above values. Estimate gives us the value of the equation, equation formed here is: Y = 20.7583 + 7.5675X
It also tells us about the statistical significance. 3 stars indicate highly statically significant. Number of stars varies from 0 to 3 (3 is the highest).

Detailed explanation:

Formula Call: It shows the formula we used to calculate the model.

Residuals: Residuals are essentially the difference between the actual observed response values (Marks) and the response values that the model predicted. When assessing how well the model fit the data, you should look for a symmetrical distribution across these points on the mean value zero (0).

Coefficients: Theoretically, in simple linear regression, the coefficients are two unknown constants that represent the intercept and slope terms in the linear model. If we wanted to predict the Marks obtained given the hours of study, we would get a training set and produce estimates of the coefficients to then use it in the model formula.

Coefficient - Estimate: Without studying, one can score on average 20.7583 marks - that's Intercept. The second row in the Coefficients is the slope, saying that for every 1-hour increase in the study, the marks goes up by 7.5675.

Coefficient - Standard Error: The coefficient Standard Error measures the average amount that the coefficient estimates vary from the actual average value of our response variable. We'd ideally want a lower number relative to its coefficients. In our example, we've previously determined that 1-hour increase in the study, the marks goes up by 7.5675. The Standard Error can be used to compute an estimate of the expected difference in case we ran the model again and again. In other words, we can say that the Marks obtained can vary by 0.3009. The Standard Errors can also be used to compute confidence intervals and to statistically test the hypothesis of the existence of a relationship between Hours of study and marks obtained.

Coefficient - t value: The coefficient t-value is a measure of how many standard deviations our coefficient estimate is far away from 0. We want it to be far away from zero as this would indicate we could reject the null hypothesis - that is, we could declare a relationship between Hours and Marks exist. In our example, the t-statistic values are relatively far away from zero and are large relative to the standard error, which could indicate a relationship exists. In general, t-values are also used to compute p-values.

Coefficient - Pr(>t): The Pr(>t) acronym found in the model output relates to the probability of observing any value equal or larger than t. A small p-value indicates that it is unlikely we will observe a relationship between the predictor (Hours) and response (Marks) variables due to chance. Typically, a p-value of 5% or less is a good cut-off point. In our model example, the p-values are very close to zero. Note the 'signif. Codes' associated to each estimate. Three stars (or asterisks) represent a highly significant p-value. Consequently, a small p-value for the intercept and the slope indicates that we can reject the null hypothesis which allows us to conclude that there is a relationship between speed and distance.

Residual Standard Error: Residual Standard Error is measure of the quality of a linear regression fit. Theoretically, every linear model is assumed to contain an error term E. Due to the presence of this error term, we are not capable of perfectly predicting our response variable (Marks) from the predictor (Hours) one. The Residual Standard Error is the average amount that the response (Marks) will deviate from the true regression line. In our example, the actual distance required to stop can deviate from the true regression line by approximately 4.599, on average. Simplistically, degrees of freedom are the number of data points that went into the estimation of the parameters used after taking into account these

parameters (restriction). *Degree of Freedom* is given by the difference between the number of observations in the sample and the number of variables in the model.

Multiple R-squared, Adjusted R-squared: The R-squared (R2R2) statistic provides a measure of how well the model is fitting the actual data. It takes the form of a proportion of variance. R2R2 is a measure of the linear relationship between our predictor variable (Hours) and our response / target variable (Marks). It always lies between 0 and 1 (i.e.: a number near 0 represents a regression that does not explain the variance in the response variable well and a number close to 1 does explain the observed variance in the response variable). In our example, the R2R2 we get is 0.9576. Or roughly 95% of the variance found in the response variable (Marks) can be explained by the predictor variable (Hours). It's hard to define what level of R2R2 is appropriate to claim the model fits well. Essentially, it will vary with the application and the domain studied.
A side note: In multiple regression settings, the R2R2 will always increase as more variables are included in the model. That's why the adjusted R2R2 is the preferred measure as it adjusts for the number of variables considered.

F-Statistic: F-statistic is a good indicator of whether there is a relationship between our predictor and the response variables. The further the F-statistic is from 1 the better it is. However, how much larger the F-statistic needs to be depends on both the number of data points and the number of predictors. Generally, when the number of data points is large, an F-statistic that is only a little bit larger than 1 is already sufficient to reject the null hypothesis (H0 : There is no relationship between Hours and Marks). The reverse is true as if the number of data points is small, a large F-statistic is required to be able to ascertain that there may be a relationship between predictor and response variables. In our example the F-statistic is 632.4 which is relatively larger than 1 given the size of our data.

Predicting the Test set result using Predict function

```
Y_predict = predict(regressor, newdata = test_data)
print(y_predict)
```

Output:

```
> y_predict
       5        6        8       19       23       26
37.40667 42.70389 44.97412 65.40624 80.54114 88.86533
```

Figure 7: Output of test data set

Analyze the result

Combine the Predict output to the test dataset and we analyze variance to calculate the accuracy. The explanation is given in the comment section of the code.

```
# Combine the test data and predict value to another variable
analyze_data <- cbind(test_set, y_predict)
print(analyze_data)
```

Analyze: Measures of Forecast Error – lets look at the some of the measures of calculating forecast error/accuracy.

Mean Square Error(MSE): The smaller the means squared error, the closer you are to finding the line of best fit. Calculation:

- Subtract the calculated value from the original to get the error.
- Square the errors.

- Add up the errors.
- Find the mean.

```
analyze.mse <- mean((analyze_data$Marks - analyze_data$y_predict)^2)
print(analyze.mse)
```

Mean Absolute Percent Error (MAPE): The mean absolute percentage error (MAPE), also known as mean absolute percentage deviation (MAPD), is a measure of prediction accuracy of a forecasting method. It usually expresses accuracy as a percentage, and is defined by the formula:

$$\mathrm{M} = \frac{100}{n}\sum_{t=1}^{n}\left|\frac{A_t - F_t}{A_t}\right|$$, where At is the actual value and Ft is the forecast value.

The difference between At and Ft is divided by the Actual value At again. The absolute value in this calculation is summed for every forecasted point in time and divided by the number of fitted points n. Multiplying by 100 makes it a percentage error.

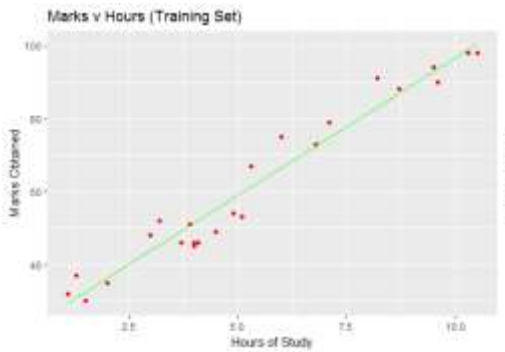
MLmetrics Package: MLmetrics provides a collection of evaluation metrics, including loss, score and utility functions, that measure regression, classification and ranking performance.

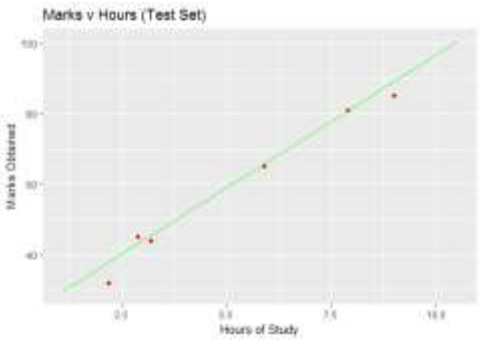
```
#install.packages("MLmetrics")
library(MLmetrics)
#Mean Absolute Percent Error (MAPE):
# MAPE(y_pred, y_true) : Syntax
analyze.mape <- MAPE(analyze_data$y_predict, analyze_data$Marks)

#Convert MAPE value into percentage
paste("Forecast Error: ", round(100*analyze.mape, 2), "%", sep="")
```

What is displayed is Forecast Error. Forecast Accuracy is (1-Forecast Error).

Visualization

We will use GGPLOT2 library for visualization. Please refer my book title “Learn and Practice R Programming ” for the tutorial on GGPLOT2.






```
#Visualization
#install.packages("ggplot2")
library(ggplot2)
#Step by step plotting of all the data
ggplot() +
  geom_point(aes(x= training_set$Hours, y= training_set$Marks),
```

```
        color = 'red') +  # Observation points
 geom_line(aes(x= training_set$Hours, y= predict(regressor, newdata = training_set)),
        color = 'green') +   #Training set predicted salary
 ggtitle("Marks v Hours (Training Set)") +   # Adding Title for the plot
 xlab("Hours of Study") +   # X axis label
 ylab("Marks Obtained")
```

Now lets see how we can predict for new dataset. We will re-write the code for test_set:

```
#Step by step plotting of all the test data set
ggplot() +
 geom_point(aes(x= test_set$Hours, y= test_set$Marks),
        color = 'red') +  # Observation points
 geom_line(aes(x= training_set$Hours, y= predict(regressor, newdata = training_set)),
        color = 'green') +   #Training set predicted salary
 ggtitle("Marks v Hours (Test Set)") +   # Adding Title for the plot
 xlab("Hours of Study") +   # X axis label
 ylab("Marks Obtained")
```

Note that we have not changed the variable names in geom_line() code because the linear regression line is based on the training set. Comparing two graphs we see that Green line doesn't change.

Now, lets dive into Multiple Linear Regression, we will have several independent variables but before that look at the assumptions we make while we work with Linear Models.

ASSUMPTIONS OF LINEAR REGRESSION

Note: R Studio comes bundled with many datasets. In this section, we will use one such pre bundled dataset named **cars**. It has two columns distance and speed.

There are assumptions we make when we work with Linear Regression:

Assumption 1 Linear relationship: Linear regression needs the relationship between the independent and dependent variables to be linear. It is also important to check for outliers since linear regression is sensitive to outlier effects.

How to test: The linearity assumption can best be tested with scatter plots. If the scatter plot follows a linear pattern (i.e. not a curvilinear) that shows that linearity assumption is met.

Assumption 2 The mean of residuals is (or very close to) zero: This is default unless you explicitly make amends, such as setting the intercept term to zero.

How to test: Run the mean of Output_Variable$residuals code and check the value:

```
mod <- lm(dist ~ speed, data=cars)
mean(mod$residuals)
```

Since the mean of residuals is approximately zero, this assumption holds true for this model.

Assumption 3 Homoscedasticity of residuals (equal variance): The data are needs to be homoscedastic (meaning the residuals are equal across the regression line).

How to test: Once the regression model is built, set par(mfrow=c(2, 2)), then, plot the model using plot(lm.mod). This produces Residual plots, a set of four plots. The top-left and bottom-left plots shows how the residuals vary as the fitted values increase.

```
par(mfrow=c(2,2))  # set 2 rows and 2 column plot layout
mod <- lm(dist ~ speed, data=cars)
```

```
plot(mod)
```

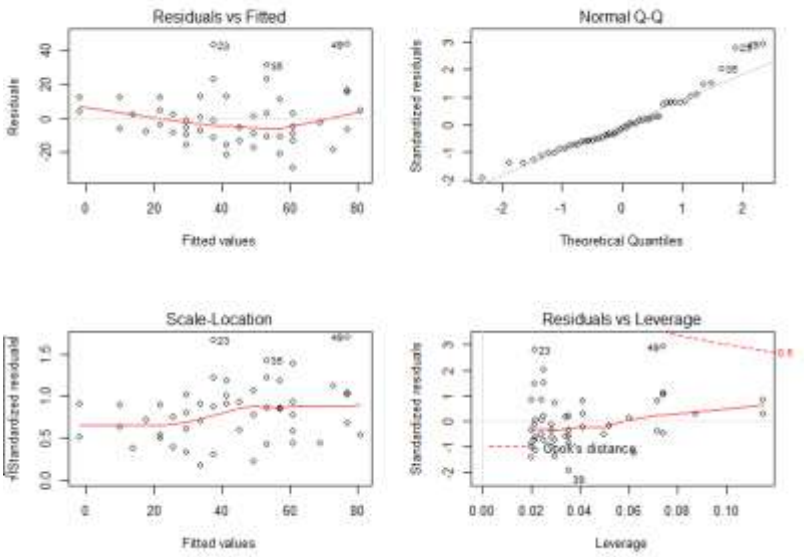


Figure 8: Homoscedasticity example

Line looks pretty flat (almost), with negligible increasing or decreasing trend. So, the condition of homoscedasticity can be accepted. Compare this with another dataset which comes pre-build with R Studio ***mtcars*** (plot output is shown figure below):

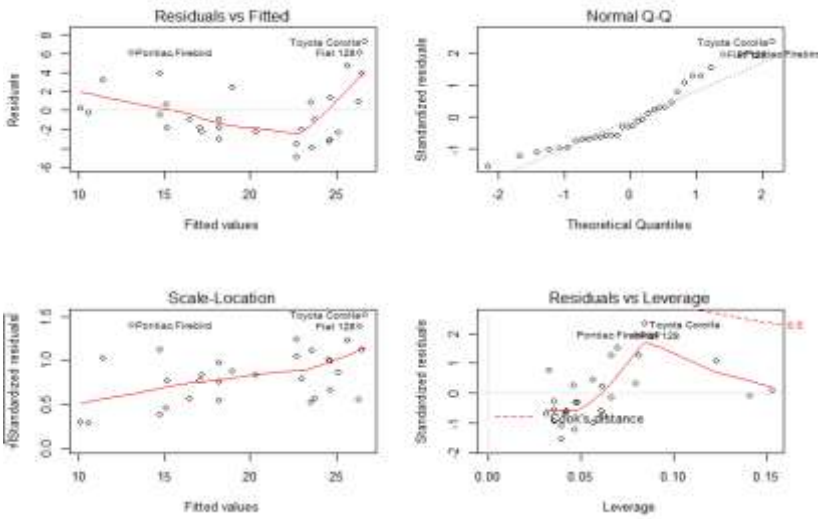


Figure 9: Heteroscedasticity example

```
mod_1 <- lm(mpg ~ disp, data=mtcars)  # linear model
plot(mod_1)
```

From the first plot (top-left), as the fitted values along x increase, the residuals decrease and then increase. The plot on the bottom left also checks this. In this case, there is a definite pattern noticed. So, there is heteroscedasticity (opposite of Homoscedasticity).

Let's discuss about these four plots in details. Readers are advised to refer other sources for more information.
Residual Plots: Residuals plots are an easy way to avoid biased models and can help you make adjustments. For instance, residual plots display patterns when an underspecified regression equation is biased, which can indicate the need to model curvature. The simplest model that creates random residuals is a great contender for being reasonably precise and unbiased.

Residuals vs Fitted plot: When conducting a residual analysis, a "residuals versus fits plot" is the most frequently created plot. It is a scatter plot of residuals on the y axis and fitted values (estimated responses) on the x axis. The plot is used to detect non-linearity, unequal error variances, and outliers.

Normal Q-Q: This is a scatterplot created by plotting two sets of quantiles- the sample quantiles against the distribution quantiles. If both sets of quantiles came from the same distribution, we should see the points forming a line that's roughly straight. The Normal Q-Q plot is used to check if our residuals follow Normal distribution or not. The residuals are normally distributed if the points follow the dotted line closely.

Scale Location: The scale-location plot shows the square root of the standardized residuals (sort of a square root of relative error) as a function of the fitted values. Scale location plot indicates spread of points across predicted values range. One of the assumptions for Regression is Homoscedasticity i.e. variance should be reasonably equal across the predictor range. A horizontal red line is ideal and would indicate that residuals have uniform variance across the range. As residuals spread wider from each other the red spread line goes up.

Residuals vs Leverage: Lets understand few terminologies before we understand this plot.

- In statistics, Cook's distance or Cook's D is a commonly used estimate of the influence of a data point when performing a least-squares regression analysis. In a practical ordinary least squares analysis, Cook's distance can be used in several ways: to indicate influential data points that are particularly worth checking for validity; or to indicate regions of the design space where it would be good to be able to obtain more data points. Data points with large residuals (outliers) and/or high leverage may distort the outcome and accuracy of a regression. Cook's distance measures the effect of deleting a given observation. Points with a large Cook's distance are considered to merit closer examination in the analysis.
- Influence: The Influence of an observation can be thought of in terms of how much the predicted scores would change if the observation is excluded. Cook's Distance is a pretty good measure of influence of an observation.
- Leverage: The leverage of an observation is based on how much the observation's value on the predictor variable differs from the mean of the predictor variable. The more the leverage of an observation, the greater potential that point has in terms of influence.

In the Residuals vs Leverage plot the dotted red lines are cook's distance and the areas of interest for us are the ones outside dotted line on top right corner or bottom right corner. If any point falls in that region, we say that the observation has high leverage or potential for influencing our model is higher if we exclude that point. Its not always the case though that all outliers will have high leverage or vice versa.

Assumption 4 Normality of residuals: The residuals should be normally distributed. This can be visually checked using the *qqnorm*() plot (top right plot in figure 8 and figure 9). If points lie exactly on the line, it is perfectly normal distribution. However, some deviation is to be expected, particularly near the ends (note the upper right), but the deviations should be small.

Assumption 5 No autocorrelation of residuals: Autocorrelation occurs when the residuals are not independent from each other. In other words, when the value of y(x+1) is not independent from the value of y(x).

How to test: Method 1- Run: Using Auto Correlation Functions (acf)

```
library(ggplot2)
lmMod <- lm(speed ~ dist, data=cars)
acf(lmMod$residuals)

#Now we will test for another inbuilt dataset Economics
data(economics)
lmMod <- lm(pce ~ pop, data=economics)
acf(lmMod$residuals, main="pop v pce (Economics Dataset)")
```

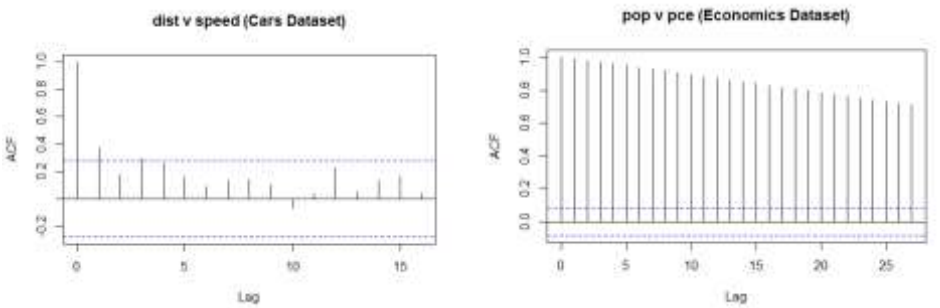


Figure 10: acf graphs for Cars and Economics dataset

The X axis corresponds to the lags of the residual, increasing in steps of 1. The very first line (to the left) shows the correlation of residual with itself (Lag 0), therefore, it will always be equal to 1. In the Cars dataset, immediate line next to Lag 0 should drop to a near zero value below the dashed blue line (significance level), In this case, its still marginally higher than the blue line, so we can conclude that the residuals are not autocorrelated (marginally but). In the *Economincs* dataset, Lag 1, Lag 2 are closer to 1 hence the residuals are autocorrelated.

Method 2- Run: Using Durbin-Watson test (Need to pre-install ***lmtest*** package)

```
lmMod_2 <- lm(speed ~ dist, data=cars)
lmtest::dwtest(lmMod_2)
```

Output:

```
        Durbin-Watson test
data:  lmMod_2
DW = 1.1949, p-value = 0.0009159
alternative hypothesis: true autocorrelation is greater than 0
```

With a p-value = 0.0009159, we cannot reject the null hypothesis. Therefore, we can safely assume that residuals are not autocorrelated.

How to rectify Economics dataset?

Add lag1 of residual as an X variable to the original model. This can be conveniently done using the slide function in DataCombine package.

```
#install.packages('DataCombine')
library(DataCombine)
lmMod <- lm(pce ~ pop, data=economics)
econ_data <- data.frame(economics, resid_mod1=lmMod$residuals)
econ_data_1 <- slide(econ_data, Var="resid_mod1", NewVar = "lag1", slideBy = -1)
econ_data_2 <- na.omit(econ_data_1)
lmMod2 <- lm(pce ~ pop + lag1, data=econ_data_2)
# Test for Autocorrelation with Durbin-Watson test
lmtest::dwtest(lmMod2)
```

Output:

```
	Durbin-Watson test
data:  lmMod2
DW = 2.0431, p-value = 0.6672
alternative hypothesis: true autocorrelation is greater than 0
```

With a high p value of 0.667, we cannot reject the null hypothesis that true autocorrelation is zero. So the assumption that residuals should not be autocorrelated is satisfied by this model.

Assumption 6: The X variables and residuals are uncorrelated. Do a correlation test on the X variable and the residuals.

```
mod.lm <- lm(dist ~ speed, data=cars)
cor.test(cars$speed, mod.lm$residuals)
```

Correlation value is `8.058406e-17`, close to zero hence we can ignore the correlation.

Assumption 7: The number of observations must be greater than number of Xs (Independent variables). This can be directly observed by looking at the data.

Assumption 8: The variability in X values is positive.

```
var(cars$speed)
```

Output:

```
[1] 27.95918
```

The variance in the X variable above is much larger than 0. So, this assumption is satisfied.

Assumption 9: The regression model is correctly specified. This means that if the Y and X variable has an inverse relationship, the model equation should be specified appropriately:

$$Y - \beta 1 + \beta 2 * \left(\frac{1}{X}\right)$$

Assumption 10 No perfect multicollinearity: There is no perfect linear relationship between explanatory variables.

How to check? Using Variance Inflation factor (VIF). VIF is a metric computed for every X variable that goes into a linear model. If the VIF of a variable is high, it means the information in that variable is already explained by other X variables present in the given model, which means, more redundant is that variable. So, lower the VIF (<2) the better. VIF for a X var is calculated as:

$VIF = \frac{1}{(1 - R_{sq})}$, Rsq is the Rsq term for the model with given X as response against all other Xs that went into the model as predictors.

Practically, if two of the X's have high correlation, they will likely have high VIFs. Generally, VIF for an X variable should be less than 4 in order to be accepted as not causing multi-collinearity. The cutoff is kept as low as 2, if you want to be strict about your X variables. Lets

see an example below. We are using inbuilt dataset *mtcars*. We would like to select which all factors impacts mpg (Miles per Gallon) given 10 different variables.

```
#install.packages('car')
library(car)
#mpg is dependent variable and all other represent independent variable
mod2 <- lm(mpg ~ ., data=mtcars)
vif(mod2)
```

Output:

```
     cyl     disp       hp     drat       wt     qsec       vs       am     gear     carb
15.373833 21.620241  9.832037 3.374620 15.164887  7.527958 4.965873 4.648487 5.357452 7.908747
```

Two ways to rectify:

- Either iteratively remove the X var with the highest VIF or,
- See correlation between all variables and keep only one of all highly correlated pairs.

We will use *carrplot* package to plot correlation, see correlation between all variables and keep only one of all highly correlated pairs. Corrplot (Visualization of a Correlation Matrix): A graphical display of a correlation matrix or general matrix.

```
#install.packages('corrplot')
library(corrplot)
corrplot(cor(mtcars[, -1]),  type =”upper”, method = "circle")     # Will plot a Correlogram
```

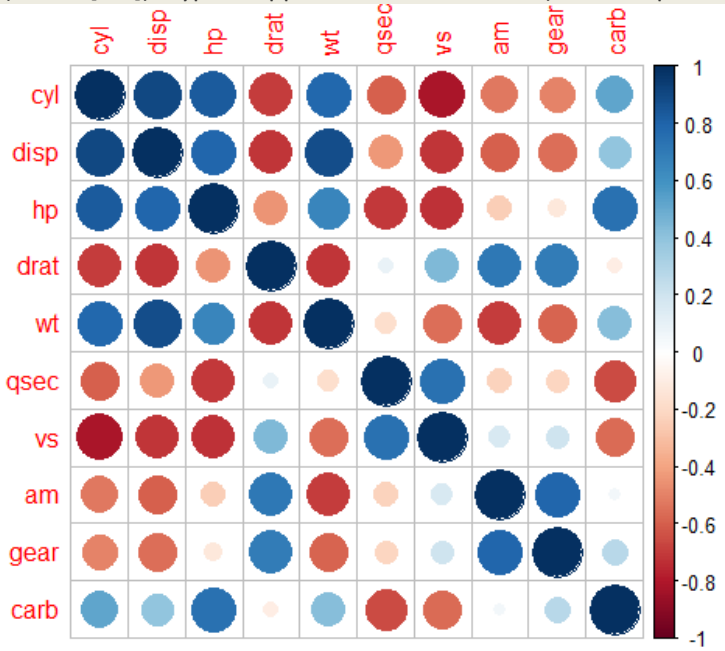


Figure 11: Correlogram (Correlation matrix plotted)

Check plot for following values of method: aquare, ellipse, number, shade, color, pie

Interpreted from above plot:

Positive correlations are displayed in blue and negative correlations in red color. Color intensity and the size of the circle are proportional to the correlation coefficients. In the right side of the correlogram, the legend color shows the correlation coefficients and the corresponding colors.

Let’s select cyl and gear as independent factors for our analysis.

```
mod <- lm(mpg ~ cyl + gear, data=mtcars)
vif(mod)
```

The convention is, the VIF should not go more than 4 for any of the X variables. That means we are not letting the RSq of any of the Xs (the model that was built with that X as a response variable and the remaining Xs are predictors) to go more than 75%. => 1/(1-0.75) = 1/0.25 =>4.

MULTIPLE LINEAR REGRESSION

Multiple linear regression (MLR) is a statistical technique that uses several explanatory variables to predict the outcome of a response variable. A linear regression model that contains more than one predictor variable is called a multiple linear regression model. The goal of multiple linear regression (MLR) is to model the relationship between the explanatory and response variables.
The model for MLR, given n observations, is:

$$y_i = B_0 + B_1x_{i1} + B_2x_{i2} + ... + B_px_{ip} + E \text{ where } i = 1,2, ..., n$$

The word "linear" in "multiple linear regression" refers to the fact that the model meets all the criteria discussed in the previous session.

Dataset

Download *3_Startups.csv* from: https://github.com/swapnilsaurav/MachineLearning

The dataset has 5 columns which contains extract from the Profit and Loss statement of 50 start up companies. This tells about the companies R&D, Admin and Marketing spend, the state in which these companies are based and also profit that the companies realized in that year. A venture capitalist (VC) would be interested in such a data and would to see if factors like R&D Spend, Admin expenses, Marketing spend and State has any role to play on the profitability of a startup. This analysis would help VC to make investment decisions in future. Profit is the dependent variable and other variables are independent variables.

1. Read the dataset for Multiple Linear Regression from the file location

```
# Importing the dataset
dataset = read.csv('D:/MachineLearning/3_Startups.csv')
```

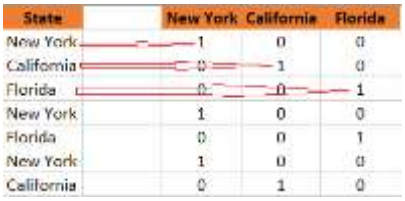
DUMMY VARIABLES

Let's look at the dataset we have for this example:

R&D Spend	Administration	Marketing Spend	State	Profit
165349.2	136897.8	471784.1	New York	192261.83
162597.7	151377.59	443898.53	California	191792.06
153441.51	101145.55	407934.54	Florida	191050.39
144372.41	118671.85	383199.62	New York	182901.99

Figure 12: Dataset for Multiple Linear Regression

One challenge we would face while building the linear model is on handling the State variable. State column has a categorical value and can not be treated as like any other numeric value. We need to add dummy variables for each categorical value like below:



State	New York	California	Florida
New York	1	0	0
California	0	1	0
Florida	0	0	1
New York	1	0	0
Florida	0	0	1
New York	1	0	0
California	0	1	0

Add 3 columns for each categorical value of state. Add 1 to the column where row value of state matches to the column header. Row containing New York will have 1 against the column header New York and rest of the values in that column will be zero.

Figure 13: Added dummy columns

Similarly, we need to modify California and Florida columns too. Three additional columns that we added are called dummy variables and these will be used in our model building. State column can be ignored. We can also ignore Florida column from analysis because row which has zero under New York and California implicitly implies Florida will have a value of 1. We always use 1 less dummy variable compared to total factors to avoid dummy variable trap. Since these variables are highly correlated, we drop one variable.

Lets implement it in R, factor will handle it on its own.

```
# Encoding categorical data
dataset$State = factor(dataset$State,
                       levels = c('New York', 'California', 'Florida'),
                       labels = c(1, 2, 3))
# Splitting the dataset into the Training set and Test set
# install.packages('caTools')
library(caTools)
set.seed(123)
split = sample.split(dataset$Profit, SplitRatio = 0.8)
training_set = subset(dataset, split == TRUE)
test_set = subset(dataset, split == FALSE)

# Feature Scaling
# training_set = scale(training_set)
# test_set = scale(test_set)

# Fitting Multiple Linear Regression to the Training set
regressor = lm(formula = Profit ~ .,
               data = training_set)

# Predicting the Test set results
y_pred = predict(regressor, newdata = test_set)
```

How many independent variables to consider?

We need to be careful to choose which ones we need to keep for input variables. We do not want to include all the variables for mainly 2 reasons:

1. GIGO: If we feed garbage to our model we will get garbage out so we need to feed in right set of data
2. Justifying the input: Can we justify the inclusion of all the data, if no then we should not include them.

There are 5 methods to build a multiple linear model:

1. Select all in
2. Backward Elimination

3. Forward Selection
4. Bidirectional Elimination

Select-all-in: We select all the independent variables because we know that all variables impact the result or you have to because business leaders want you to include them.

Backward Elimination:

1. Select a significance level to stay in the model (e..g. SL =0.05)
2. Fit the full model with all possible predictors.
3. Consider the predictor with the highest P-value. If P>SL, go to step 4 otherwise goto 5
4. Remove the predictor and refit the model and Go to step 3
5. Your model is ready!

Forward Selection:

1. Select a significance level to stay in the model (e..g. SL =0.05)
2. Fit all the simple regression models, Select the one with the lowest P-value.
3. Keep this variable and fit all possible models with one extra predictor added to the ones you already have. Now Run with 2 variable linear regressions.
4. Consider the predictor with the lowest P-value. If P<SL, go to Step 3, otherwise go to next step.
5. Keep the previous model!

Bi-directional Selection: It is a combination of Forward selection and backward elimination:

1. Select a significant level to enter and stay in the model (SLE = SLS = 0.05)
2. Perform the next step of Forward selection (new variables must have P<SLE)
3. Perform all the step of Backward elimination (old variables must have P<SLS)
4. Iterate between 2 & 3 till no new variables can enter and no old variables can exit.

For more information on Variable Selection, refer *MultiLinear VariableSelection.pdf* document information available on the Github location:
https://github.com/swapnilsaurav/MachineLearning

Let's implement Backward Elimination in R, similarly it can be done for Forward selection also. We need to remove the variables which are not statically significant and still get amazing result. We will employ Backward elimination on the *3_Startups.csv* dataset and use same Multiple linear regression that we have used earlier. Run the below code:

```
# Building the optimal model using Backward elimination
# Step 1 use all the variables
regressor = lm(formula = Profit ~ R.D.Spend + Administration + Marketing.Spend +State,
               data = dataset)
# Run summary and verify the R value
summary(regressor)
```

Let's evaluate the coefficients values.

```
Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)      4.965e+04  7.637e+03   6.501 1.94e-07 ***
R.D.Spend        7.986e-01  5.604e-02  14.251 6.70e-16 ***
Administration  -2.942e-02  5.828e-02  -0.505    0.617
Marketing.Spend  3.268e-02  2.127e-02   1.537    0.134
State2           1.213e+02  3.751e+03   0.032    0.974
State3           2.376e+02  4.127e+03   0.058    0.954
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Figure 14: Multiple input values run for regression

We see that, R function has created 2 state variables for 3 different variables. This is what we learnt under adding dummy variables. As per our algorithm, the variable with highest P value should be removed. Let's run regression again after removing State variable. You can even look at the last column which doesn't have a name but will show the number of stars. The meaning is given at the bottom of the coefficients under the row name: Significant Codes – If the P value is between o and 0.001 (0.01%), statistically its very significant. P values between 0.001 and 0.01 gives 2 stars, 0.01 to 0.05 will give 1 star and value between 0.05 and 0.1 gives us . (dot) – each meaning decreasing order of significance. Values higher than 0.1 is blank meaning there is no impact of that input.

In the next step, we will re-run again after removing State variable. In this case, you will find that Administration has the highest P value (about 60%), we will run again after removing the Administration variable from the formula. What do we see now? Marketing Spend P value has come below 0.1 (0.06 to be exact). It is still greater than 0.05 but now it demonstrates some significance. Its up to us to include or exclude this value from the analysis.

Alternate method to use:

The function *regsubsets()* in the library "leaps" can be used for regression subset selection. Thereafter, one can view the ranked models according to different scoring criteria by plotting the results of regsubsets().

```
dataset = read.csv('3_Startups.csv')
library(leaps)
leaps=regsubsets(Profit~R.D.Spend + Administration + Marketing.Spend +State, data=dataset, nbest=10)
```

To view the ranked models according to the adjusted R-squared criteria and BIC, respectively, type:

```
plot(leaps, scale="adjr2")
plot(leaps, scale="bic")
```

Note: Perform above plots and compare the result. Here black indicates that a variable is included in the model, while white indicates that they are not. The model containing all variables minimizes the adjusted Rsquare criteria (left), while the model including Size, Lot and Taxes minimizes the BIC (right). Looking at the values on the y-axis of the plot indicates that the top four models have roughly the same adjusted R-square and BIC values, thus possibly explaining the discrepancy in the results.

Automatic methods are useful when the number of explanatory variables is large and it is not feasible to fit all possible models. In this case, it is more efficient to use a search algorithm (e.g., Forward selection, Backward elimination and Stepwise regression) to find the best

model. The R function step() can be used to perform variable selection. To perform forward selection we need to begin by specifying a starting model and the range of models which we want to examine in the search.

```
null=lm(Price~1, data=Housing)
null
full=lm(Price~., data=Housing)
full
```

We can perform forward selection using the command:

```
step(null, scope=list(lower=null, upper=full), direction="forward")
```

This tells R to start with the null model and search through models lying in the range between the null and full model using the forward selection algorithm. It gives rise to the following output:

```
Start:  AIC=1061.42
Profit ~ 1

                  Df  Sum of Sq        RSS     AIC
+ R.D.Spend        1 7.5349e+10 4.2560e+09  916.98
+ Marketing.Spend  1 4.4511e+10 3.5094e+10 1022.46
+ Administration   1 3.2071e+09 7.6398e+10 1061.36
<none>                          7.9605e+10 1061.42
+ State            2 1.9006e+09 7.7704e+10 1064.21
```

Figure 15: Step 1 Output

```
Coefficients:
    (Intercept)        R.D.Spend  Marketing.Spend
      4.698e+04        7.966e-01        2.991e-02
```

Figure 16: Final Output

According to this procedure, the best model is the one that includes the variables R.D.Spend and Marketing.Spend. We can perform backward elimination on the same data set using the command:

```
step(full, data=Housing, direction="backward")
```

and stepwise regression using the command:

```
step(null, scope = list(upper=full), data=Housing, direction="both")
```

Both algorithms give rise to results that are equivalent to the forward selection procedure.

POLYNOMIAL REGRESSION

Linear vs Non-Linear models

A model is linear when each term is either a constant or the product of a parameter and a predictor variable. A linear equation is constructed by adding the results for each term. This constrains the equation to just one basic form:

*Response = constant + parameter * predictor + ... + parameter * predictor*

In statistics, a regression equation (or function) is linear when it is linear in the parameters. Where as you can transform the predictor variables in ways that produce curvature.

$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \varepsilon$. This model is still linear in the parameters even though the predictor variable is squared. You can also use log and inverse functional forms that are linear in the parameters to produce different types of curves.

While a linear equation has one basic form, nonlinear equations can take many different forms. The easiest way to determine whether an equation is nonlinear is to find that it's not linear. If the equation doesn't meet the criteria for a linear equation, it's nonlinear. That covers many different forms, which is why nonlinear regression provides the most flexible curve-fitting functionality. Examples of non-linear:

- *Weibull growth*: Theta1 + (Theta2 - Theta1) * exp(-Theta3 * X^Theta4)
- *Fourier*: Theta1 * cos(X + Theta4) + (Theta2 * cos(2*X + Theta4) + Theta3

In statistics, polynomial regression is a form of regression analysis in which the relationship between the independent variable x and the dependent variable y is modelled as an nth degree polynomial in x. We will see this in the program below.
Although polynomial regression is technically a special case of multiple linear regression, the interpretation of a fitted polynomial regression model requires a somewhat different perspective. It is often difficult to interpret the individual coefficients in a polynomial regression fit, since the underlying monomials can be highly correlated. For example, x and x^2 (square) have correlation around 0.97 when x is uniformly distributed on the interval (0, 1).

A data set contains measurements of yield from an experiment done at five different temperature levels. The variables are y = yield and x = temperature in degrees Fahrenheit. The figures below give a scatterplot of the raw data with lines pertaining to a linear fit and a quadratic fit overlayed. Obviously the trend of this data is better suited to a quadratic fit.

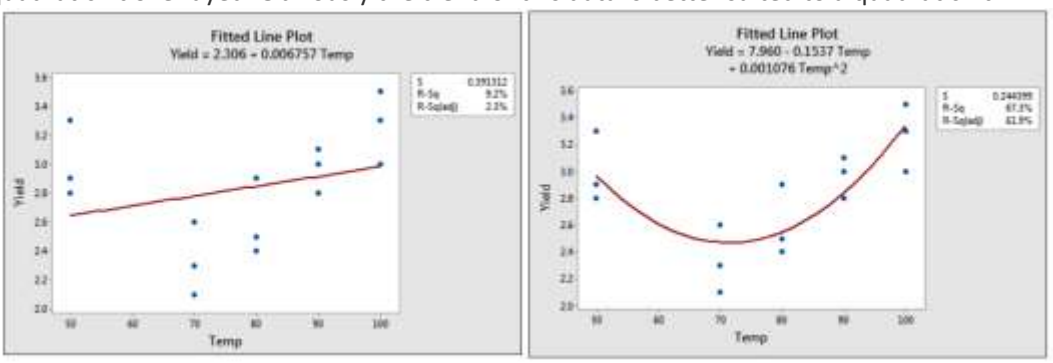


Figure 17: Example of Polynomial Regression

Modelling and Solving Polynomial Regression

We have got a dataset of employees' job level/position and the respective salary at that position. We will then predict the salary of a person belonging to certain position. Lets say that the person is at position 5 for last 2 years. If we know that it takes 4 years to get promoted from Position 5 to 6 then we can assume the current position of the person to be 5.5 and predict the salary for this position. As usual first step is to perform data preprocessing. Download *4_Position_Salaries.csv* from: www.github.com/swapnilsaurav/MachineLearning. Please read through the code to understand the steps involved.

```
# Building Polynomial Regression
# Importing the dataset
dataset = read.csv('D:/MachineLearning/4_Position_Salaries.csv')
dataset = dataset[2:3]

# Splitting the dataset Not needed as per business requirement
# Feature Scaling is performed inbuilt
```

```
#Let's perform Linear Regression to see the result with the Polynomial result:
# Fitting Linear Regression to the dataset
lin_reg = lm(formula = Salary ~ .,
             data = dataset)

# Fitting Polynomial Regression to the dataset
dataset$Level2 = dataset$Level^2
dataset$Level3 = dataset$Level^3
dataset$Level4 = dataset$Level^4
poly_reg = lm(formula = Salary ~ .,
              data = dataset)

# Visualising the Linear Regression results
# install.packages('ggplot2')
library(ggplot2)
ggplot() +
  geom_point(aes(x = dataset$Level, y = dataset$Salary),
             colour = 'red') +
  geom_line(aes(x = dataset$Level, y = predict(lin_reg, newdata = dataset)),
            colour = 'blue') +
  ggtitle('Linear Regression Validation') +
  xlab('Level') +
  ylab('Salary')

# Visualising the Polynomial Regression results
# install.packages('ggplot2')
library(ggplot2)
ggplot() +
  geom_point(aes(x = dataset$Level, y = dataset$Salary),
             colour = 'red') +
  geom_line(aes(x = dataset$Level, y = predict(poly_reg, newdata = dataset)),
            colour = 'blue') +
  ggtitle('Polynomial Regression Validation') +
  xlab('Level') +
  ylab('Salary')

# Visualising the Regression Model results (for higher resolution and smoother curve)
# install.packages('ggplot2')
library(ggplot2)
x_grid = seq(min(dataset$Level), max(dataset$Level), 0.1)
ggplot() +
  geom_point(aes(x = dataset$Level, y = dataset$Salary),
             colour = 'red') +
  geom_line(aes(x = x_grid, y = predict(poly_reg,
                                        newdata = data.frame(Level = x_grid,
                                                             Level2 = x_grid^2,
                                                             Level3 = x_grid^3,
```

```
                                        Level4 = x_grid^4))),
        colour = 'blue') +
 ggtitle('Polynomial Regression Validation') +
 xlab('Level') +
 ylab('Salary')

#Now lets predict the value for a given number of years of experience
given_exp = 6.5

# Predicting a new result with Linear Regression
predict(lin_reg, data.frame(Level = given_exp))

# Predicting a new result with Polynomial Regression
predict(poly_reg, data.frame(Level = given_exp,
                             Level2 = given_exp^2,
                             Level3 = given_exp^3,
                             Level4 = given_exp^4))
```

SUPPORT VECTOR REGRESSION (SVR)

Support Vector Machine" (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In this example we will see how it can be used in regression models.

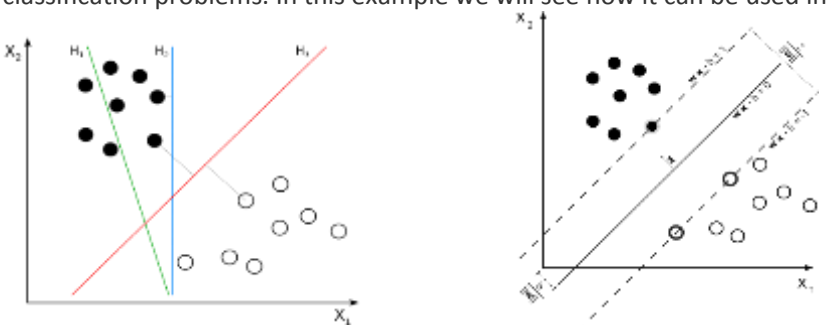


Figure 18: Support Vector Machine (Regression)

A support vector machine constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks like outlier detection. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class (so-called functional margin), since in general the larger the margin the lower the generalization error of the classifier.

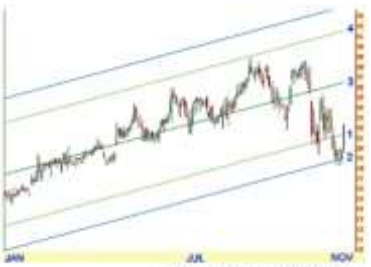
In SVR, you want to find a function such that all points are within a certain distance from this function. Again, if points are outside this distance ["**ε** -tube"], there is a penalty or loss. The linear **ε**-insensitive loss function ignores errors that are within **ε** distance of the observed value by treating them as equal to zero. The loss is measured based on the distance between observed value y and the **ε** boundary. This is formally described by:

$$L_\varepsilon = \begin{cases} 0 & \text{if } |y - f(x)| \le \varepsilon \\ |y - f(x)| - \varepsilon & \text{otherwise} \end{cases}$$

Our goal is to is to find a function f(x) that has at most ε deviation from actually obtained target yi, for all the training data, and at the same time is as flat as possible. In other words, the data points lie in between the two borders of the margin which is maximized under suitable conditions will avoid outlier inclusion.

Modelling and Solving SVR Regression

In order to create a SVR model with R you will need the package e1071. So be sure to install it and to add the library(e1071) line at the start of your file. Download *4_Position_Salaries.csv* from: www.github.com/swapnilsaurav/MachineLearning. Please read through the comments in the code to understand the steps involved.



Stock price prediction

Step 1: Prepare the dataset

```
############
# SVR Model building
# Importing the dataset
dataset = read.csv('D:/MachineLearning/4_Position_Salaries.csv')
dataset = dataset[2:3]
# Splitting the dataset into the Training set and Test set
# Not required here
```

Step 2: Fitting SVR for the dataset and predicting the output

The SVR performs linear regression in a higher (infinite) dimensional space. A simple way to think of it is as if each data point in your training set represents it's own dimension. When you evaluate your kernel between a test point and a point in your training set, the resulting value gives you the coordinate of your test point in that dimension. The vector we get when we evaluate the test point for all points in the training set, is the representation of the test point in the higher dimensional space. The form of the kernel tells you about the geometry of that higher dimensional space.

```
# Fitting SVR to the dataset
# install.packages('e1071')
library(e1071)
regressor = svm(formula = Salary ~ .,
                data = dataset,
                type = 'eps-regression',
                kernel = 'radial')

# Predicting a new result
y_pred = predict(regressor, data.frame(Level = 6.5))
```

Regressor value explanation:

- Type: Two types of regression available are: `eps-regression` and `nu-regression`. The original SVM formulations for Regression (SVR) used parameters C [0, inf) and epsilon[0, inf) to apply a penalty to the optimization for points which were not correctly predicted. An alternative version of both SVM regression was later developed where the epsilon penalty parameter was replaced by an alternative

parameter, nu [0,1], which applies a slightly different penalty. The main motivation for the nu versions of SVM is that it has a has a more meaningful interpretation. This is because nu represents an upper bound on the fraction of training samples which are errors (badly predicted) and a lower bound on the fraction of samples which are support vectors. Some users feel nu is more intuitive to use than C or epsilon. Epsilon or nu are just different versions of the penalty parameter. The same optimization problem is solved in either case. Thus it should not matter which form of SVM you use, epsilon or nu.

- Kernel: Data is rarely clean and simple. Many times we do not get a clear hyperplane and the dataset will often look more like the jumbled balls. In order to classify a dataset like that it's necessary to move away from a 2d view of the data to a 3d view. That's when we use non-linear kernel like polynomial, radial or sigmoid. Linear kernel is used when dataset can be linearly separated.

More details are available under Support Vector Machine section of Classification chapter.

Step 3: Visualize the output

```
# Visualising the SVR results
# install.packages('ggplot2')
library(ggplot2)
ggplot() +
  geom_point(aes(x = dataset$Level, y = dataset$Salary),
             colour = 'red') +
  geom_line(aes(x = dataset$Level, y = predict(regressor, newdata = dataset)),
            colour = 'blue') +
  ggtitle(' SVR Model Design') +
  xlab('Level') +
  ylab('Salary')

# Visualising the SVR results (for higher resolution and smoother curve)
# install.packages('ggplot2')
library(ggplot2)
x_grid = seq(min(dataset$Level), max(dataset$Level), 0.1)
ggplot() +
  geom_point(aes(x = dataset$Level, y = dataset$Salary),
             colour = 'red') +
  geom_line(aes(x = x_grid, y = predict(regressor, newdata = data.frame(Level = x_grid))),
            colour = 'blue') +
  ggtitle(' SVR Model Design') +
  xlab('Level') +
  ylab('Salary')
```

DECISION TREE REGRESSION

Decision tree builds regression or classification models in the form of a tree structure. It brakes down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf

nodes. A decision node (e.g., Outlook) has two or more branches (e.g., Sunny, Overcast and Rainy), each representing values for the attribute tested. Leaf node (e.g., Hours Played) represents a decision on the numerical target. The topmost decision node in a tree which corresponds to the best predictor called root node. Decision trees can handle both categorical and numerical data.

Figure 19: Decision tree example

The goal of the Decision Tre is to create a model that predicts the value of a target variable based on several input variables. An example is shown in the diagram above. Each interior node corresponds to one of the input variables; there are edges to children for each of the possible values of that input variable. Each leaf represents a value of the target variable given the values of the input variables represented by the path from the root to the leaf.

The term Classification And Regression Tree (CART) analysis is an umbrella term used to refer to both Decision Tree Regression and Decision Tree Classification procedures. Trees used for regression and trees used for classification have some similarities - but also some differences, such as the procedure used to determine where to split.

Regression versus Classification

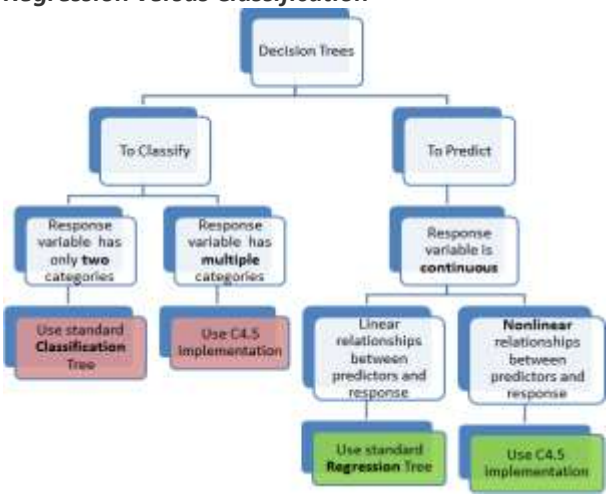


Figure 20: Decision Tree procedures

In a standard classification tree, the idea is to split the dataset based on homogeneity of data. Lets say for example we have two variables: age and weight that predict if a person is going to sign up for a gym membership or not. In our training data if it showed that 90% of the people

who are older than 40 signed up, we split the data here and age becomes a top node in the tree. We can almost say that this split has made the data "90% pure".
In a regression tree the idea is this: since the target variable does not have classes, we fit a regression model to the target variable using each of the independent variables. Then for each independent variable, the data is split at several split points. At each split point, the "error" between the predicted value and the actual values is squared to get a "Sum of Squared Errors (SSE)". The split point errors across the variables are compared and the variable/point yielding the lowest SSE is chosen as the root node/split point. This process is recursively continued.

When to use classification vs regression tree
Classification trees, as the name implies are used to separate the dataset into classes belonging to the response variable. Usually the response variable has two classes: Yes or No (1 or 0). If the target variable has more than 2 categories, then a variant of the algorithm, called C4.5, is used. For binary splits however, the standard CART procedure is used. Thus classification trees are used when the response or target variable is categorical in nature.
Regression trees are needed when the response variable is numeric or continuous. For example, the predicted price of a consumer good. Thus regression trees are applicable for prediction type of problems as opposed to classification. Keep in mind that in either case, the predictors or independent variables may be categorical or numeric. It is the target variable that determines the type of decision tree needed.

Modelling and Solving Decision Tree Regression
Download *4_Position_Salaries.csv* from: www.github.com/swapnilsaurav/MachineLearning.
Below steps are to read the dataset into R, these steps are same as what we did for other regression models.
Step 1: Reading DataSet

```
 # Decision Tree Regression

# Importing the dataset
dataset = read.csv('D:/MachineLearning/4_Position_Salaries.csv')
dataset = dataset[2:3]
```

Step 2: Fitting Decision Tree Regression Model
rpart package has the rpart function which we will use to run Decision Tree Regression model

```
# Fitting Decision Tree Regression to the dataset
# install.packages('rpart')
library(rpart)
regressor = rpart(formula = Salary ~ .,
                  data = dataset)
```

Step 3: Predicting the new result using the regressor

```
# Predicting a new result with Decision Tree Regression
y_pred = predict(regressor, data.frame(Level = 6.5))
print(y_pred)
```

Printing y_pred gives us the value corresponding to 6.5 level. But is this the correct output? Lets view the plot and see if that looks correct. We will see more version of this graph in the next section titled: *Improving the Decision Tree model*

Step 4: Visualize the Decision Tree result

```
# Visualizing the Decision Tree Regression results (higher resolution)
# install.packages('ggplot2')
library(ggplot2)
x_grid = seq(min(dataset$Level), max(dataset$Level))
ggplot() +
 geom_point(aes(x = dataset$Level, y = dataset$Salary),
            colour = 'red') +
 geom_line(aes(x = x_grid, y = predict(regressor, newdata = data.frame(Level = x_grid))),
           colour = 'blue') +
 ggtitle('Validate (Decision Tree Regression)') +
 xlab('Level') +
 ylab('Salary')
```

Step 5: Plot the Decision Tree

```
# Plotting the tree
plot(regressor)
text(regressor)
```

Improving the Decision Tree Model

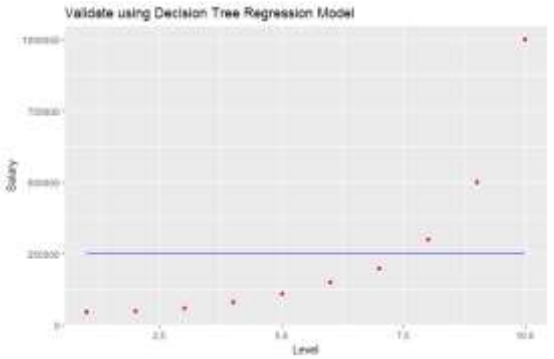


Figure 21: Output of Decision Tree Model using step 2

Why did we get a straight line? Its because we do not have any split hence the model has taken the average of all the data and estimating same value for all the input data. More conditions we have more splits will be made. So this model is not useful to us. To modify we will add a parameter to rpart algorithm. We will add an optional parameter called *control*. We will pass internal function control with minsplit=1. This will fix the Decision Tree model.

Improving Decision Tree Regression Model by adding minsplit

```
regressor = rpart(formula = Salary ~ .,
                  data = dataset,
                  control = rpart.control(minsplit = 1))
```

Run the visualize code again. What do we get? We see the split as shown in the figure below:

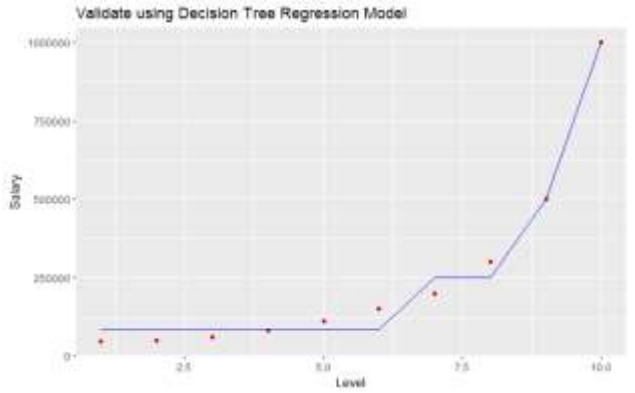


Figure 22: Plot with rpart function with Control value

But should this be the output? Based on the Decision Tree concepts we should see conditions as either vertical or horizontal lines but not as inclined as shown. Inclined line could mean we get infinite number of splits between two points but that's not the case here. Between given two points there is no split as we have the values incremented by 1, so between two groups there is nothing to plot resulting in drawing a simple line joining two points. This belongs to non-linear non-continuous regression models. Decision tree is predicting for each discrete variable and if there is no prediction between 2 variables, it will simply join them. To plot in the interval, we will create smaller x-grid size (or to make it high resolution). Smaller the x-grid higher will be the resolution.

```
x_grid = seq(min(dataset$Level), max(dataset$Level),0.01)
```

From the definition of Decision tree, we know that the predicted value is the average of the values in the split so this model will predict same value of $250,000 for any value of level between 6.5 to 8.5.

RANDOM FOREST REGRESSION

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 forest is like bootstrapping algorithm with Decision tree (CART) model. Say, we have 1000 observation in the complete population with 10 variables. Random forest tries to build multiple CART model with different sample and different initial variables. For instance, it will take a random sample of 100 observations and 5 randomly chosen initial variables to build a CART model. It will repeat the process (say) 10 times and then make a final prediction on each observation. Final prediction is a function of each prediction. This final prediction can simply be the mean of each prediction.

Steps involved in the process can be described as below:

Step 1: Pick at random K data points from the training set

Step 2: Build the Decision Tree associated to these K data points

Step 3: Choose the number of trees (Ntree) you want to build and repeat steps 1 & 2

Step 4: Use all of them to predict: For a new data point, make each one of your Ntree trees predict the value of Y for the given input, and assign the new data point average of all the predicted Y values.

Default values is set to 500 trees so we get minimum of 500 output values and the final output is average of these output. It improves accuracy of the process. Regression Trees are known to be very unstable, in other words, a small change in your data may drastically change your model. The Random Forest uses this instability as an advantage resulting on a very stable model. This is also called as Ensemble machine learning paradigm where multiple learners are trained to solve the same problem – you can be using same algorithm multiple time (as in this case) or use multiple algorithm to solve same problem.

Modelling and Solving Random Tree Regression

Step 1: Setting up the data set

```
# Random Forest Regression
# Importing the dataset
dataset = read.csv('4_Position_Salaries.csv')
dataset = dataset[2:3]
```

Step 2: Using randomForest package

```
# install.packages('randomForest')
library(randomForest)
set.seed(123456789)
regressor = randomForest(x = dataset[1],  y = dataset$Salary, ntree = 500)
```

Step 3: Predict the value

```
# Predicting a new result with Random Forest Regression
y_pred = predict(regressor, data.frame(Level = 6.5))
```

Step 4: Visualize the data

```
# Visualising the Random Forest Regression results (higher resolution)
# install.packages('ggplot2')
library(ggplot2)
x_grid = seq(min(dataset$Level), max(dataset$Level), 0.01)
ggplot() +
 geom_point(aes(x = dataset$Level, y = dataset$Salary),
            colour = 'red') +
 geom_line(aes(x = x_grid, y = predict(regressor, newdata = data.frame(Level = x_grid))),
           colour = 'blue') +
 ggtitle('Validate - Random Forest Regression') +
 xlab('Level') +
 ylab('Salary')
```

Change the value of ntree and see how to output changes. Change the resolution (x_grid) value and see how the graph changes. When we increase the number of trees it doesn't necessarily mean that we will have more steps in the output because the more we add trees the more the average of the different predictions made by the trees is converging to the same average. That's al we have under Regressions models. Let's understand some concepts related to regression in the next section.

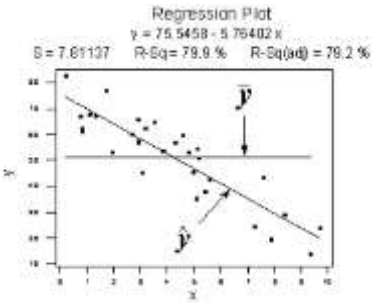
INTERPRETING COEFFICIENT OF REGRESSION

Coefficient of Correlation: Correlation coefficients are used in statistics to measure how strong a relationship is between two variables. There are several types of correlation coefficient: Pearson's correlation (also called Pearson's R) is a correlation coefficient commonly used in linear regression. When anyone refers to the correlation coefficient, they are usually talking about Pearson's.

$$r = \frac{n(\Sigma xy) - (\Sigma x)(\Sigma y)}{\sqrt{[\,n\Sigma x^2 - (\Sigma x)^2\,]\,[\,n\Sigma y^2 - (\Sigma y)^2\,]}}$$

Correlation coefficient formulas are used to find how strong a relationship is between data. The formulas return a value between -1 and 1, where:

- 1 indicates a strong positive relationship.
- -1 indicates a strong negative relationship.
- A result of zero indicates no relationship at all.



Coefficient of Determination (R Squared) Intuition: The coefficient of determination can be thought of as a percent. It gives you an idea of how many data points fall within the results of the line formed by the regression equation. The higher the coefficient, the higher percentage of points the line passes through when the data points and line are plotted. If the coefficient is 0.80, then 80% of the points should fall within the regression line. Values of 1 or 0 would indicate the regression line represents all or none of the data, respectively. A higher coefficient is an indicator of a better goodness of fit for the observations.

Two trend lines, one represented by Average (horizontal line) of all the data set and the regression line (slope line) are compared and R squared is calculated as:

$$SS_{res} = SUM\ (y_i - \hat{y_i})^2$$
$$SS_{tot} = SUM\ (y_i - y_{ave})^2$$
$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

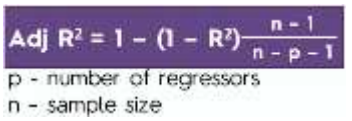
1- (Sum of the errors on regression line / Sum of the errors on average line).

2- Can R square be negative? Answer is yes, that's when Average value fits the trend better than regression line.

The usefulness of R2 is its ability to find the likelihood of future events falling within the predicted outcomes. The idea is that if more samples are added, the coefficient would show the probability of a new point falling on the line. Even if there is a strong connection between the two variables, determination does not prove causality.

The Adjusted Coefficient of Determination (Adjusted R-squared) is an adjustment for the Coefficient of Determination that takes into account the number of variables in a data set. It also penalizes you for points that don't fit the model. You might be aware that few values in a data set (a too-small sample size) can lead to misleading statistics, but you may not be aware that too many data points can also lead to problems. Every time you add a data point in regression analysis, R^2 will increase. R^2 never decreases. Therefore, the more points you add, the better the regression will seem to "fit" your data. If your data doesn't quite fit a line, it can be tempting to keep on adding data until you have a better fit. Some of the points you add will be significant (fit the model) and others will not. R^2 doesn't care about the insignificant points. The more you add, the higher the coefficient of determination.

Formula to calculate Adjusted R^2



The adjusted R^2 can be used to include a more appropriate number of variables, thwarting your temptation to keep on adding variables to your data set.
The adjusted R^2 will increase only if a new data point improves the regression more than you would expect by chance. R^2 doesn't include all data points, is always lower than R^2 and can be negative (although it's usually positive). Negative values will likely happen if R^2 is close to zero — after the adjustment, the value will dip below zero a little.

EVALUATING REGRESSION MODELS SPECIFICATION

Model specification is the process of determining which independent variables to include and exclude from a regression equation. How do you choose the best regression model? The world is complicated, and trying to explain it with a small sample doesn't help. Often, the variable selection process is a mixture of statistics, theory, and practical knowledge.

Statistical Methods for Model Specification
We have already discussed about Adjusted R-squared and Predicted R-squared. Typically, we want to select models that have larger adjusted and predicted R-squared values.
P-values for the independent variables: In regression, p-values less than the significance level indicate that the term is statistically significant.
Residual Plots: During the specification process, check the residual plots.
Ultimately, statistical measures can't tell you which regression equation is best. They just don't understand the fundamentals of the subject-area. Your expertise is always a vital part of the model specification process! Choosing the correct regression model is one issue, while choosing the right type of regression analysis for your data is an entirely different matter.

OTHER TYPES OF REGRESSION MODELS

Regression analysis mathematically describes the relationship between a set of independent variables and a dependent variable. There are numerous types of regression models we have seen but there are more advanced variants. Some of these variants are mentioned here for your knowledge. Choosing the right model often depends on the kind of data you have for the dependent variable and the type of model that provides the best fit. Lets do some revision:

Regression Analysis with Continuous Dependent Variables: Continuous variables are a measurement on a continuous scale, such as weight, time, and length. Some of the examples are:

- Linear regression: Linear regression, also known as ordinary least squares (OLS) and linear least squares, is the real workhorse of the regression world.
- Advanced types of linear regression: OLS has several weaknesses, including a sensitivity to both outliers and multi-collinearity, and it is prone to overfitting. To address these problems, statisticians have developed several advanced variants:
    - Ridge regression allows you to analyze data even when severe multi-collinearity is present and helps prevent overfitting. This type of model reduces the large, problematic variance that multi-collinearity causes by

introducing a slight bias in the estimates. The procedure trades away much of the variance in exchange for a little bias, which produces more useful coefficient estimates when multi-collinearity is present.

- Lasso regression (least absolute shrinkage and selection operator) performs variable selection that aims to increase prediction accuracy by identifying a simpler model. It is similar to Ridge regression but with variable selection.
- Partial least squares (PLS) regression is useful when you have very few observations compared to the number of independent variables or when your independent variables are highly correlated. PLS decreases the independent variables down to a smaller number of uncorrelated components, similar to Principal Components Analysis. Then, the procedure performs linear regression on these components rather the original data. PLS emphasizes developing predictive models and is not used for screening variables. Unlike OLS, you can include multiple continuous dependent variables. PLS uses the correlation structure to identify smaller effects and model multivariate patterns in the dependent variables.

- Nonlinear regression: Nonlinear regression also requires a continuous dependent variable, but it provides a greater flexibility to fit curves than linear regression.

Regression Analysis with Categorical Dependent Variables: A categorical variable has values that you can put into a countable number of distinct groups based on a characteristic. Logistic regression transforms the dependent variable and then uses Maximum Likelihood Estimation, rather than least squares, to estimate the parameters.

- Binary Logistic Regression: Use binary logistic regression to understand how changes in the independent variables are associated with changes in the probability of an event occurring. This type of model requires a binary dependent variable. A binary variable has only two possible values, such as pass and fail.
- Ordinal Logistic Regression: Ordinal logistic regression models the relationship between a set of predictors and an ordinal response variable. An ordinal response has at least three groups which have a natural order, such as hot, medium, and cold.
- Nominal Logistic Regression: Nominal logistic regression models the relationship between a set of independent variables and a nominal dependent variable. A nominal variable has at least three groups which do not have a natural order, such as scratch, dent, and tear.

Regression Analysis with Count Dependent Variables: If your dependent variable is a count of items, events, results, or activities, you might need to use a different type of regression model. Counts are nonnegative integers.

- Poisson regression: Count data frequently follow the Poisson distribution, which makes Poisson Regression a good possibility. Poisson variables are a count of something over a constant amount of time, area, or another consistent length of observation. With a Poisson variable, you can calculate and assess a rate of occurrence. A classic example of a Poisson dataset is provided by Ladislaus Bortkiewicz, a Russian economist, who analyzed annual deaths caused by horse kicks in the Prussian Army from 1875-1984.

- Alternatives to Poisson regression for count data: Not all count data follow the Poisson distribution because this distribution has some stringent restrictions. Fortunately, there are alternative analyses you can perform when you have count data.
    - Negative binomial regression: Poisson regression assumes that the variance equals the mean. When the variance is greater than the mean, your model has overdispersion. A negative binomial model, also known as NB2, can be more appropriate when overdispersion is present.
    - Zero-inflated models: Your count data might have too many zeros to follow the Poisson distribution. In other words, there are more zeros than the Poisson regression predicts. Zero-inflated models assume that two separate processes work together to produce the excessive zeros. One process determines whether there are zero events or more than zero events. The other is the Poisson process that determines how many events occur, some of which some can be zero.

CONCLUSION

Readers can explore these algorithms on themselves. Author is also coming up with a separate book on just Regression analysis which will cover additional information along with more of these algorithms. You can reach out to our team on ekapresshyderabad@gmail.com for more information.

We end our Regression discussion here and in the next chapter we will see Classification models. Readers are encouraged to read articles and blogs on these topics to understand more about the models we discussed in this chapter.

UNIT 4: CLASSIFICATION

Classification is the problem of identifying to which of a set of categories (sub-populations) a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known. In the terminology of machine learning, classification is considered an instance of supervised learning, i.e. learning where a training set of correctly identified observations is available. The corresponding unsupervised procedure is known as clustering, and involves grouping data into categories based on some measure of inherent similarity or distance.

Some examples of Classification problems are:

- Text categorization (e.g., spam filtering), Optical character recognition
- Fraud detection
- Machine vision (e.g., face detection)
- Natural-language processing (e.g., spoken language understanding)
- Market segmentation (e.g.: predict if customer will respond to promotion)
- Bioinformatics (e.g., classify proteins according to their function)

Let's implement now...

R Implementation of Classification algorithms

For all the classification algorithms discussed here, we will be using *5_Ads_Success.csv.*

Download *5_Ads_Success.csv* from: www.github.com/swapnilsaurav/MachineLearning.

The given dataset consists of user profile data such as user ID, Gender, Age, Estimated Salary and if the user has made a purchase or not under the column Purchased. Purchased is our dependent variable here. Goal of the algorithms is to predict if a customer is going to make a purchase or not based on the values from Age and Salary. Age and Salary becomes our independent variables.

Let's discuss the preprocessing of the data here in common section. Remember, you will have to run this code prior to running any of the algorithm discussed below.

Step 1: Import the dataset

```
# Importing the dataset
dataset = read.csv('D:/MachineLearning/5_Ads_Success.csv')
dataset = dataset[3:5]
```

Or, incase you want to upload the dataset file:

```
### To upload a file:
# Train <- read.csv(file.choose())
dataset = dataset[3:5]
```

Step 2: Encoding the Target variable as Factor

```
# Encoding the target feature as factor
dataset$Purchased = factor(dataset$Purchased, levels = c(0, 1))
```

Step 3: Splitting the dataset into the Training set and Test set

```
# Splitting the dataset into the Training set and Test set
# install.packages('caTools')
```

```
library(caTools)
set.seed(123)
split = sample.split(dataset$Purchased, SplitRatio = 0.75)
training_set = subset(dataset, split == TRUE)
test_set = subset(dataset, split == FALSE)
```

Step 4: Scale the dataset

```
# Feature Scaling
training_set[-3] = scale(training_set[-3])
test_set[-3] = scale(test_set[-3])
```

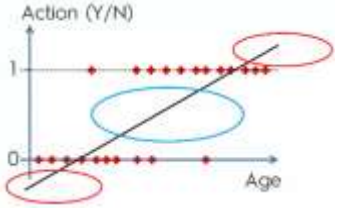
Now let's look at the remaining part of the implementation under each algorithm header.

LOGISTIC REGRESSION

Logistics Regression is a classification not a regression algorithm. It is used to estimate discrete values (Binary values like 0/1, yes/no, true/false) based on given set of independent variable(s). In simple words, it predicts the probability of occurrence of an event by fitting data to a logit function. Hence, it is also known as logit regression. Since, it predicts the probability, its output values lie between 0 and 1.

Understanding Logistics Regression

We will be using *5_Ads_Success.csv* dataset to work on Logistics Classification. For understanding Logistics Regression, we will look at only one independent variable i.e. Age in this explanation. For the implementation we will take 2 variables – Age and Salary to predict.



Plotting Age on x-axis and Purchased on y-axis, we might get a graph as shown above, which indicate that the younger people are more likely not purchase the product but older age group people tend to buy the product. We can draw a regression line as shown above which will divide the data into two sets. However, there are some issues with the regression line. We have only two values possible – Yes/No or probability 1/0. The regression line, we see, extends beyond the limit 0 and 1 which is not possible. To avoid this, we use Sigmoid function as shown in the figure below. A sigmoid function is a mathematical function having a characteristic "S"-shaped curve or sigmoid curve. A sigmoid function is a bounded differentiable real function that is defined for all real input values and has a non-negative derivative at each point.

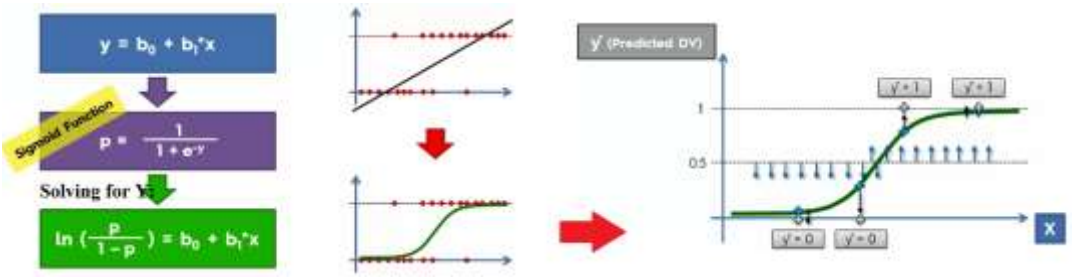


Figure 23: Interpretation of Logistics Regression plot

Another issue with the linear plot is that we will have to find a way to interpret the points of the line between 0 and 1 as the output probability can range from 0 to 1 but the output has to be either 0 or 1. To handle this, we can define our own reference value above which the output will be 1 and below it would be 0. In our example we have use 0.5 as the reference value but remember you can define your own based on the business requirements.

Binary logistic regression major assumptions

- The dependent variable should be dichotomous in nature (e.g., presence vs. absent).
- There should be no outliers in the data, which can be assessed by converting the continuous predictors to standardized scores, and removing values below -3.29 or greater than 3.29.
- There should be no high correlations (multicollinearity) among the predictors. This can be assessed by a correlation matrix among the predictors. Tabachnick and Fidell (2013) suggest that as long correlation coefficients among independent variables are less than 0.90 the assumption is met.
- At the center of the logistic regression analysis is the task estimating the log odds of an event. Mathematically, logistic regression estimates a multiple linear regression function defined as:

$$\text{logit}(p) = \log\left(\frac{p(y=1)}{1-(p=1)}\right) = \beta_0 + \beta_1 x_0 + \beta_2 \cdot x_0 + \_ + \beta_p \cdot x_n$$

for i = 1..n .

Overfitting: Just like multiple regression, adding independent variables to a logistic regression model will always increase the amount of variance explained in the log odds (expressed as R^2). However, adding more and more variables to the model can result in overfitting, which reduces the generalizability of the model beyond the data on which the model is fit. Numerous pseudo- R^2 values have been developed for binary logistic regression. A better approach is to present any of the goodness of fit tests available: Hosmer-Lemeshow is a commonly used measure of goodness of fit based on the Chi-square test.

Implementing Logistics Regression

Step 1: Pre-processing the data set as shown under the Classification section

Step 2: Fitting Logistics Regression

This dataset has a binary response (Yes, No) variable called Purchased. There are two predictor variables: Age and Salary.

```
# # Fitting Logistic Regression to the Training set
classifier = glm(formula = Purchased ~ .,
                 family = binomial,
                 data = training_set)
```

Step 3: Get the result

In order to get the results, we use the summary command:

```
 # # Fitting Logistic Regression to the Training set
summary(classifier)
```

```
Call:
glm(formula = Purchased ~ ., family = binomial, data = training_set)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.0753  -0.5235  -0.1161   0.3224   2.3977

Coefficients:
                Estimate Std. Error z value Pr(>|z|)
(Intercept)      -1.1923     0.2018  -5.908 3.47e-09 ***
Age               2.6324     0.3461   7.606 2.83e-14 ***
EstimatedSalary   1.3947     0.2326   5.996 2.03e-09 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 390.89  on 299  degrees of freedom
Residual deviance: 199.78  on 297  degrees of freedom
AIC: 205.78

Number of Fisher Scoring iterations: 6
```

Figure 24: Output from glm() function

Observations

- Call: In the output above, the first thing we see is the Input info.
- Deviance residuals: It is a measure of model fit. This shows the distribution of the deviance residuals for individual cases used in the model.
- Coefficients, their standard errors, the z-statistic (sometimes called a Wald z-statistic), and the associated p-values. Both Age and EstimatedSalary are statistically significant, as are the three terms for rank. The logistic regression coefficients give the change in the log odds of the outcome for a one-unit increase in the predictor variable.
  - For every one-unit change in Age, the log odds of admission (versus non-admission) increases by 2.634.
  - For one-unit increase in EstimatedSalary, the log odds of being admitted to graduate school increases by 0.804.
- Fit indices: Includes the null and deviance residuals and the AIC. The Residual deviance is a measure of the lack of fit of your model taken as a whole, whereas the Null deviance is such a measure for a reduced model that only includes the intercept.
  In linear regression, residuals can be defined as yi − yˆi
  where yi is the observed dependent variable for the i th subject, and ˆyi the corresponding prediction from the model.
  The same concept applies to logistic regression, where y(i) is necessarily equal to either 1 or 0, and

$$\hat{y}_i = \hat{\pi}_i(x_i) = \frac{\exp(\alpha + \beta_1 x_{i1} + \ldots + \beta_p x_{ip})}{1 + \exp(\alpha + \beta_1 x_{i1} + \ldots + \beta_p x_{ip})}$$

  - X^2 distribution with n − (p + 1) degrees of freedom can be derived from “deviance residuals”
  - Suppose that we have a statistical model of some data. Let k be the number of estimated parameters in the model. Let $\hat{L}$ be the maximum value of the likelihood function for the model. Then the AIC value of the model is the following:

$$\mathrm{AIC} = 2k - 2\ln(\hat{L})$$

  - The AIC is another measure of goodness of fit that takes into account the ability of the model to fit the data. This is very useful when comparing two models where one may fit better but perhaps only by virtue of being more flexible and thus better able to fit any data.

- The reference to Fisher scoring iterations has to do with how the model was estimated.

Step 4: Predict the classification and create the Confusion Matrix

```
# # Predicting the Test set results
prob_pred = predict(classifier, type = 'response', newdata = test_set[-3])
y_pred = ifelse(prob_pred > 0.5, 1, 0)
```

Step 5: Predict the classification and create the Confusion Matrix

We will now evaluate the prediction using Confusion matrix. This will count the number of correct and incorrect predictions.

```
# Making the Confusion Matrix
cm = table(test_set[, 3], y_pred)
```

Step 6: Visualizing the training set data

```
## Visualising the Training set results
#install.packages("ElemStatLearn")
library(ElemStatLearn)
set = training_set
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('Age', 'EstimatedSalary')
prob_set = predict(classifier, type = 'response', newdata = grid_set)
y_grid = ifelse(prob_set > 0.5, 1, 0)
plot(set[, -3],
     main = 'Logistic Regression (Training set)',
     xlab = 'Age', ylab = 'Estimated Salary',
     xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'springgreen3', 'tomato'))
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))
```

Based on the age, the classifier predicts if the user will be buying the product or not by classifying the green and red areas respectively. Using this data, the makers of the products can efficiently use their budget by targeting the age group belonging green area. There are some exceptions though. The red dot in the green area says that our model predicted that the person would buy the product but the person has not bought it.

One more important concepts to understand here is that the two regions are separated by a straight line. This straight line is called Prediction Boundary. In logistic regression, this line will always be straight and indicate it's a linear classifier.

Step 7: Visualizing the test set data

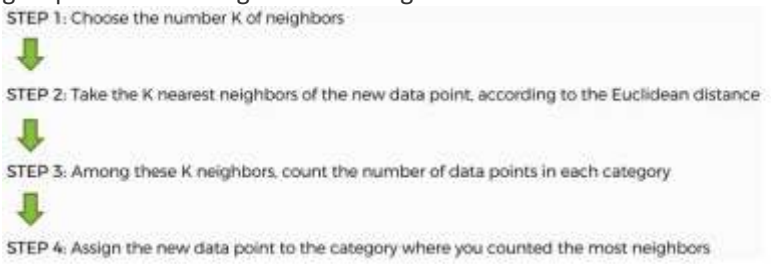
```
# # Visualising the Test set results
library(ElemStatLearn)
set = test_set
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
```

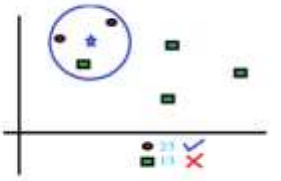
```
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('Age', 'EstimatedSalary')
prob_set = predict(classifier, type = 'response', newdata = grid_set)
y_grid = ifelse(prob_set > 0.5, 1, 0)
plot(set[, -3],
     main = 'Logistic Regression (Test set)',
     xlab = 'Age', ylab = 'Estimated Salary',
     xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'springgreen3', 'tomato'))
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))
```

We see that test data prediction is also very accurate based on the plot.

K-NEAREST NEIGHBORS (K-NN)

K-Nearest Neighbours (KNN) is one of the most basic yet essential classification algorithms in Machine Learning. It finds intense application in pattern recognition, data mining and intrusion detection. If we plot data points on a graph, we may be able to locate some clusters, or groups. Now, given an unclassified point, we can assign it to a group by observing what group its nearest neighbours belong to.





Predictions are made for a new instance (x) by searching through the entire training set for the K (given) most similar instances (the neighbors) and summarizing the output variable for those K instances. The choice of the parameter K is very crucial in this algorithm.

In this example, K =3 and the new dataset has two red circles around it hence we say that the new dataset belongs to red circle group and not to green square group.

To determine which of the K instances in the training dataset are most similar to a new input a distance measure is used. For real-valued input variables, the most popular distance measure is Euclidean distance. Euclidean distance is calculated as the square root of the sum of the squared differences between a new point (x) and an existing point (xi) across all input attributes j.

- EuclideanDistance(x, xi) = sqrt(sum((xj – xij)^2))

Other popular distance measures include:

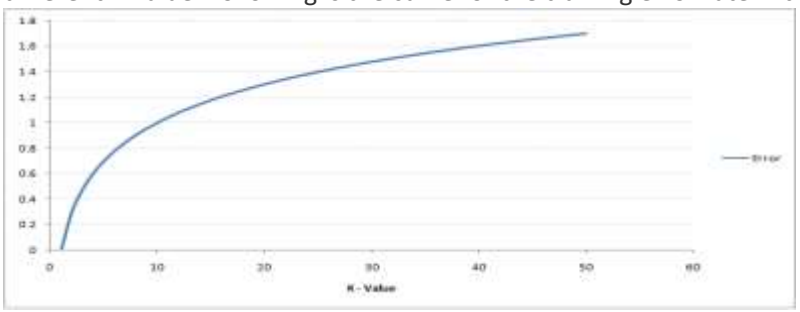
- Hamming Distance: Calculate the distance between binary vectors (more).
- Manhattan Distance: Calculate the distance between real vectors using the sum of their absolute difference. Also called City Block Distance (more).
- Minkowski Distance: Generalization of Euclidean and Manhattan distance

Best Prepare Data for KNN

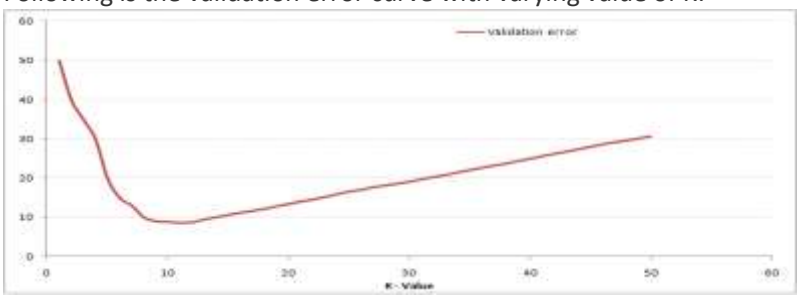
- Rescale Data: KNN performs much better if all of the data has the same scale. Normalizing your data to the range [0, 1] is a good idea. It may also be a good idea to standardize your data if it has a Gaussian distribution.
- Address Missing Data: Missing data will mean that the distance between samples can not be calculated. These samples could be excluded or the missing values could be imputed.
- Lower Dimensionality: KNN is suited for lower dimensional data. You can try it on high dimensional data (hundreds or thousands of input variables) but be aware that it may not perform as well as other techniques. KNN can benefit from feature selection that reduces the dimensionality of the input feature space.

How do we choose K factors?

With K increasing to infinity it becomes one single set of data depending on the total majority. The training error rate and the validation error rate are two parameters we need to access on different K-value. Following is the curve for the training error rate with varying value of K:



As you can see, the error rate at K=1 is always zero for the training sample. This is because the closest point to any training data point is itself. Hence the prediction is always accurate with K=1. If validation error curve would have been similar, our choice of K would have been 1. Following is the validation error curve with varying value of K:



This makes the story clearer. At K=1, we were overfitting the boundaries. Hence, error rate initially decreases and reaches minima. After the minima point, it then increases with increasing K. To get the optimal value of K, you can segregate the training and validation from the initial dataset. Now plot the validation error curve to get the optimal value of K. This value of K should be used for all predictions.

Implementation in R

Step 1: Pre-processing the data set as shown under the Classification section

```
# # K-Nearest Neighbors (K-NN)
# Fitting K-NN to the Training set and Predicting the Test set results
```

```
library(class)
y_pred = knn(train = training_set[, -3],
             test = test_set[, -3],
             cl = training_set[, 3],
             k = 5,                  # Choose odd number for K value to avoid a tie
             prob = TRUE)

# Making the Confusion Matrix
cm = table(test_set[, 3], y_pred)

# Visualising the Training set results
library(ElemStatLearn)
set = training_set
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('Age', 'EstimatedSalary')
y_grid = knn(train = training_set[, -3], test = grid_set, cl = training_set[, 3], k = 5)
plot(set[, -3],
     main = 'K-NN (Training set)',
     xlab = 'Age', ylab = 'Estimated Salary',
     xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'springgreen3', 'tomato'))
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))

# Visualising the Test set results
library(ElemStatLearn)
set = test_set
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('Age', 'EstimatedSalary')
y_grid = knn(train = training_set[, -3], test = grid_set, cl = training_set[, 3], k = 5)
plot(set[, -3],
     main = 'K-NN (Test set)',
     xlab = 'Age', ylab = 'Estimated Salary',
     xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'springgreen3', 'tomato'))
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))
```

Second method – using train() function of Caret package

```
# # Run k-NN:
#install.packages("caret")
library(caret)
set.seed(400)
ctrl <- trainControl(method="repeatedcv",repeats = 3)
```

```
knnFit <- train(Purchased ~ .,
          data = training_set,
          method = "knn",
          trControl = ctrl)
knnFit

#Use plots to see optimal number of clusters:
#Plotting yields Number of Neighbours Vs accuracy (based on repeated cross validation)
plot(knnFit)
```

```
ctrl <- trainControl(method="repeatedcv",repeats = 3)
```

This is the train control function of Caret package. Here we choose repeated cross validation. Repeated 3 means we do everything 3 times for consistency.

```
knnFit <- train(Purchased ~ .,  data = training_set,  method = "knn",   trControl = ctrl)
```

train function sets up a grid of tuning parameters for a number of classification and regression routines, fits each model and calculates a resampling based performance measure.

```
k-Nearest Neighbors

300 samples
  2 predictor
  2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 270, 271, 270, 270, 270, 269, ...
Resampling results across tuning parameters:

  k  Accuracy   Kappa
  5  0.9013608  0.7860953
  7  0.9180349  0.8247975
  9  0.9113274  0.8092740

Accuracy was used to select the optimal model using  the largest value.
The final value used for the model was k = 7.
```

Kappa Statistic: Kappa can be used to assess the performance of kNN algorithm. It compares multiple raters accuracy. Kappa can be formally expressed by the following equation:

$$kappa = \frac{P(A) - P(E)}{1 - P(E)}$$

```
   y_pred
     0  1
0   58  6
1   4  32
```

where P(A) is the relative observed accuracy, and P(E) is the expected accuracy.

Observed Accuracy is simply the number of instances that were classified correctly throughout the entire confusion matrix. We simply add the number of instances that the machine learning classifier agreed with the ground truth label, and divide by the total number of instances. For this confusion matrix, where K=7, this would be 0.9 ((58 + 32) / 100 = 0.9).

Expected Accuracy: The marginal frequency for a certain class by a certain "rater" is just the sum of all instances the "rater" indicated were that class. In our case, 62 (58 + 4 = 62) instances were labeled as Not Purchased (0) by the ground truth, and 64 (58 + 6 = 64) instances were classified as Not Purchased (0) by the KNN classifier. This results in a value of 39.68 (62 * 64 / 100 = 39.68). This is then done for the second class (Purchased) as well. We get: (4+32) * (6+32)/100 = 13.68. The last step is to add all these values together, and finally divide again by the total number of instances, resulting in an Expected Accuracy of 0.5336 ((39.68 + 13.68) / 100 = 0.5336).

Kappa = (0.9 – 0.5336) / (1-0.5336) = 0.7856

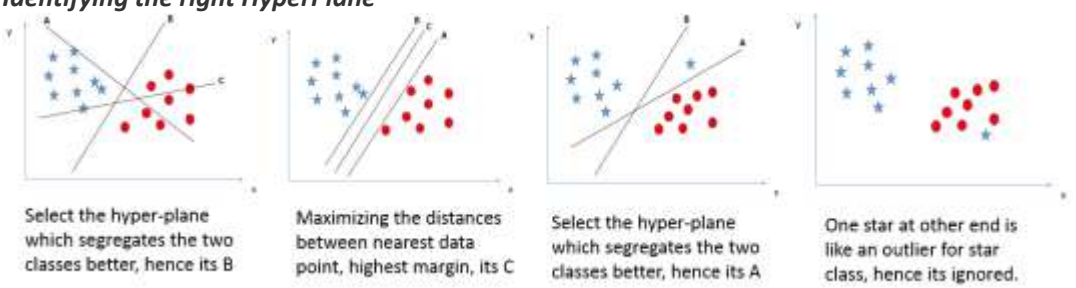
Curse of Dimensionality

KNN works well with a small number of input variables (p), but struggles when the number of inputs is very large. Each input variable can be considered a dimension of a p-dimensional input space. For example, if you had two input variables x1 and x2, the input space would be 2-dimensional. As the number of dimensions increases the volume of the input space increases at an exponential rate. In high dimensions, points that may be similar may have very large distances. All points will be far away from each other and our intuition for distances in simple 2 and 3-dimensional spaces breaks down. This might feel unintuitive at first, but this general problem is called the “Curse of Dimensionality".

SUPPORT VECTOR MACHINE (SVM)

In machine learning, support vector machines (SVMs, also support vector networks) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting). An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

Identifying the right HyperPlane



Support Vectors are the points that support the Hyperplane. Even if we get rid of all the other points the Hyperplane will not change. Its called vectors because in more than 2 dimensional scenario we would get vectors. Maximum Margin Hyperplane or Maximum Margin Classifier is considered for the analysis. Margin on the right of the Hyperplane is classified as positive hyperplane and Margin on the left is called the negative Hyperplane.

Why SVM algorithms are popular?

SVM unlike other types of algorithm, looks at the extreme types of data and tries to create the Hyperplane based on that.

Step 1: Load the data set, create factors, split the dataset into Training and Test and perform feature scaling (as shown under Classification section)

Step 2: To use SVM in R, we have a package e1071. The syntax of svm package is quite similar to linear regression. We use svm function here. Default type of algorithm to use is C-classification. For this example, lets use kernel value as linear and check for the result.

```
# Fitting SVM to the Training set
# install.packages('e1071')
library(e1071)
classifier = svm(formula = Purchased ~ .,
                 data = training_set,
                 type = 'C-classification',
                 kernel = 'linear')

# Predicting the Test set results
y_pred = predict(classifier, newdata = test_set[-3])

# Making the Confusion Matrix
cm = table(test_set[, 3], y_pred)
```

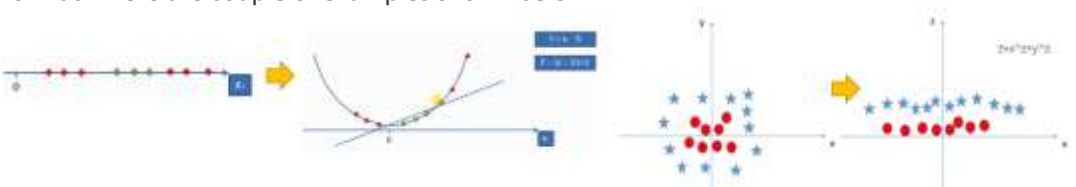
Step 3: Visualize the output

```
# Visualising the Training set results
library(ElemStatLearn)
set = training_set
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('Age', 'EstimatedSalary')
y_grid = predict(classifier, newdata = grid_set)
plot(set[, -3],
     main = 'SVM (Training set)',
     xlab = 'Age', ylab = 'Estimated Salary',
     xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)

#Add some colors now
points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'springgreen3', 'tomato'))
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))

# Visualising the Test set results
library(ElemStatLearn)
set = test_set
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('Age', 'EstimatedSalary')
y_grid = predict(classifier, newdata = grid_set)
plot(set[, -3], main = 'SVM (Test set)',
     xlab = 'Age', ylab = 'Estimated Salary',
     xlim = range(X1), ylim = range(X2))
#Add some colors now
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'springgreen3', 'tomato'))
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))
```

The cases where data is not linearly separable, we will have to solve using different method. Next example shows how non-linear separable data can be classified using SVM. In such cases we can not just draw a line to separate the data but use special functions to do so.

KERNEL SVM

When dataset is not linear we need to find functions that can separate the dataset into linear format. There are couple of examples shown below:



In the above examples we see that we can solve such non-linear problems by mapping to a higher dimensional space but mapping to a higher dimensional is highly compute-intensive. Its easy to find a lost stick on a 100-yard line but its difficult when we have to find same stick in a two dimensional 100 yard * 100 yard. Its almost impossible when we go to a 3-D plane so we need to find a better way to do things. This is called kernel trick.

Types of Kernel Functions

Implementation:
Step 1: Setting up dataset remain same compare to previous SVM example.
Step 2: Fitting Kernel to the training set. We need to provide non-linear type value to kernel to handle non-linearly separable values.
Step 3: Plot using previous SVM example.

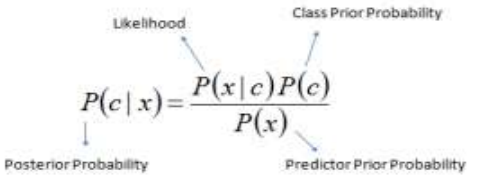
```
# Fitting Kernel SVM to the Training set
# install.packages('e1071')
library(e1071)
classifier = svm(formula = Purchased ~ .,
                 data = training_set,
                 type = 'C-classification',
                 kernel = 'radial')

# Predicting the Test set results
y_pred = predict(classifier, newdata = test_set[-3])

# Making the Confusion Matrix
cm = table(test_set[, 3], y_pred)
```

Type variables: C-classification and nu-classification is for binary classification usage. Say if you want to build a model to classify cat vs. dog based on features for animals, i.e., prediction target is a discrete variable/label. One-classification is for "outlier detection", where you only have one classes data. For example, you want to detect "unusual" behaviors of one user's account. But you do not have "unusual behavior" to train the model.

Kernel variables: This method is used to classify the non-linearly separable dataset. Best fit high dimensional functions like Polynomial, Radial (Gaussian) or Sigmoid are used.

NAIVE BAYES

Naive Bayes classifiers are a family of simple probabilistic classifiers. A Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as 'Naive'. Classifier is based on Bayes theorem, explained in book: Statistics for Machine Learning.



$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

How Naïve Bayes theorem

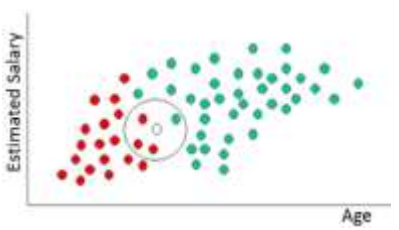
To understand the algorithm, we will use same dataset 5_Ads_Success.csv and select EstimatedSalary and Age as the independent variables to predict Purchase decision.

Step 1: Convert the data into frequency distribution table.

Step 2: Create Likelihood table by finding the probabilities

Step 3: Use Naive Bayesian equation to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome of prediction.

Let's plot the EstimatedSalary on the Y-axis and Age on the X-axis:



Problem:

- Green circle represents the training set data indicating Purchased
- Red circle represents the training set data indicating NOT Purchased
- White circle is the input data which we need to classify if Purchased or Not.

To measure this likelihood, we draw a circle around X which encompasses a number (to be chosen a priori) of points irrespective of their class labels.

Step 1: To find Posterior probability if Purchased

a. Likelihood that new data is Purchased (look into the circle and ignore Not Purchased) = Similar points / Total points = 1/20
b. Prior Probability of Purchased = Total Green/Total circle = 20/30
c. Predictor Prior Probability of Purchased (We will look into the circle assuming the points in the circle have the same characteristics) = Similar points/Total = 4/30

d. Posterior probability of purchased = a * b / c = 1/20*20/30*30/4 = ¼= 0.25

Step 2: To find Posterior probability if Not Purchased

a. Likelihood that new data is Not Purchased = 3/10
e. Prior Probability of Not Purchased = Total Red/Total circle = 10/30
b. Predictor Prior Probability of Not Purchased = Similar points/Total = 4/30
c. Posterior probability of Not purchased = a * b / c =3/10*10/30*30/4 = ¾=0.75

Step 3: Compare the probabilities

- The new data-point is more likely to exhibit Not Purchased behavior (75%).

Implementation in R

Step 1: Preparing the dataset as shown under Classification section

Step 2: Implementing Naïve bayes

naiveBayes function assumes independence of the predictor variables, and Gaussian distribution (given the target class) of metric predictors.

Parameters:

- The formula is traditional Y~X1+X2+…+Xn
- The data is typically a dataframe of numeric or factor variables.
- laplace provides a smoothing effect. Whatever positive integer this is set to will be added into for every class. The bigger the Laplace smoothing value, the more you are making the models the same.
- subset lets you use only a selection subset of your data based on some boolean filter
- na.action lets you determine what to do when you hit a missing value in your dataset.

```
### Fitting Naïve Bayes to the Training set
# install.packages('e1071')
library(e1071)
classifier = naiveBayes(x = training_set[-3],
                        y = training_set$Purchased)
# Predicting the Test set results
y_pred = predict(classifier, newdata = test_set[-3])
# Making the Confusion Matrix
cm = table(test_set[, 3], y_pred)
```

Step 3: Visualizing the Plots

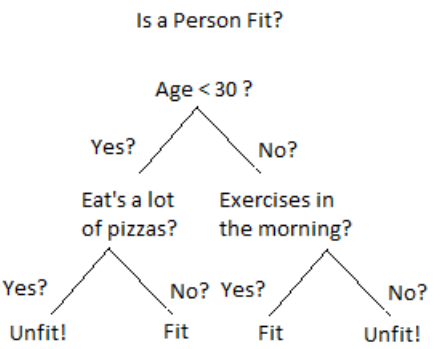
```
# Visualising the Training set results
library(ElemStatLearn)
set = training_set
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('Age', 'EstimatedSalary')
y_grid = predict(classifier, newdata = grid_set)
plot(set[, -3],
     main = 'Naive Bayes (Training set)',
     xlab = 'Age', ylab = 'Estimated Salary',
     xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'springgreen3', 'tomato'))
```

```
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))

# Visualising the Test set results
library(ElemStatLearn)
set = test_set
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('Age', 'EstimatedSalary')
y_grid = predict(classifier, newdata = grid_set)
plot(set[, -3], main = 'Naive Bayes (Test set)',
     xlab = 'Age', ylab = 'Estimated Salary',
     xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'springgreen3', 'tomato'))
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))
```

DECISION TREE CLASSIFICATION

Decision Tree Classifier, repetitively divides the working area(plot) into sub part by identifying lines. Decision tree finds the optimal set of partition which will create maximized set of partitions. It works repetitively because there may be two distant regions of same class divided by other as shown in image.

The tree can be explained by two entities, namely decision nodes and leaves. The leaves are the decisions or the final outcomes. And the decision nodes are where the data is split.



Implementation in R

```
# # Decision Tree Classification – Pre processing part to be copied from Classification section

# Fitting Decision Tree Classification to the Training set
# install.packages('rpart')
library(rpart)
classifier = rpart(formula = Purchased ~ .,
                   data = training_set)

# Predicting the Test set results
y_pred = predict(classifier, newdata = test_set[-3], type = 'class')

# Making the Confusion Matrix
cm = table(test_set[, 3], y_pred)

# Visualising the Training set results
library(ElemStatLearn)
set = training_set
```

```
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('Age', 'EstimatedSalary')
y_grid = predict(classifier, newdata = grid_set, type = 'class')
plot(set[, -3],
     main = 'Decision Tree Classification (Training set)',
     xlab = 'Age', ylab = 'Estimated Salary',
     xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'springgreen3', 'tomato'))
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))

# Visualising the Test set results
library(ElemStatLearn)
set = test_set
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('Age', 'EstimatedSalary')
y_grid = predict(classifier, newdata = grid_set, type = 'class')
plot(set[, -3], main = 'Decision Tree Classification (Test set)',
     xlab = 'Age', ylab = 'Estimated Salary',
     xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'springgreen3', 'tomato'))
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))

# Plotting the tree
plot(classifier)
text(classifier)
```

Now we define few terms related to decision tree:

1. Impurity: Impurity is when we have a traces of one class division into other. This can arise due to following reason
   - We run out of available features to divide the class upon.
   - We tolerate some percentage of impurity (we stop further division) for faster performance. (There is always trade off between accuracy and performance).
2. Entropy: Entropy is degree of randomness of elements or in other words it is measure of impurity.
3. Information Gain: Decision tree at every stage selects the one that gives best information gain. When information gain is 0 means the feature does not divide the working set at all.

RANDOM FOREST CLASSIFICATION

Random Forest Classifier is ensemble algorithm. Ensemble algorithms are those which combines more than one algorithms of same or different kind for classifying objects. Random forest classifier creates a set of decision trees from randomly selected subset of training set. It

then aggregates the votes from different decision trees to decide the final class of the test object.

Understanding in simple term: Suppose training set is given as: [X1, X2, X3, X4] with corresponding labels as [L1, L2, L3, L4], random forest may create three decision trees taking input of subset for example,

[X1, X2, X3]
[X1, X2, X4]
[X2, X3, X4]

So finally, it predicts based on the majority of votes from each of the decision trees made.

This works well because a single decision tree may be prone to a noise, but aggregate of many decision trees reduce the effect of noise giving more accurate results. Alternatively, the random forest can apply weight concept for considering the impact of result from any decision tree. Tree with high error rate are given low weight value and vise versa. This would increase the decision impact of trees with low error rate.

Implementation in R

```
# # Random Forest Classification- Pre processing part to be copied from Classification section
# Fitting Random Forest Classification to the Training set
# install.packages('randomForest')
library(randomForest)
set.seed(123)
classifier = randomForest(x = training_set[-3],
                          y = training_set$Purchased,
                          ntree = 500)

# Predicting the Test set results
y_pred = predict(classifier, newdata = test_set[-3])

# Making the Confusion Matrix
cm = table(test_set[, 3], y_pred)

# Visualising the Training set results
library(ElemStatLearn)
set = training_set
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('Age', 'EstimatedSalary')
y_grid = predict(classifier, grid_set)
plot(set[, -3],
     main = 'Random Forest Classification (Training set)',
     xlab = 'Age', ylab = 'Estimated Salary',
     xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'springgreen3', 'tomato'))
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))

# Visualising the Test set results
library(ElemStatLearn)
```

```
set = test_set
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('Age', 'EstimatedSalary')
y_grid = predict(classifier, grid_set)
plot(set[, -3], main = 'Random Forest Classification (Test set)',
    xlab = 'Age', ylab = 'Estimated Salary',
    xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'springgreen3', 'tomato'))
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))
# Choosing the number of trees
plot(classifier)
```

EVALUATING CLASSIFICATION MODELS PERFORMANCE

Let's understand ***False positive and False negative*** concepts. A false positive error, or in short a false positive, commonly called a "false alarm", is a result that indicates a given condition exists, when it does not. For example, in the case of "The Boy Who Cried Wolf", the condition tested for was "is there a wolf near the herd?"; the shepherd at first wrongly indicated there was one, by calling "Wolf, wolf!". A false positive error is a type I error where the test is checking a single condition, and wrongly gives an affirmative (positive) decision.
A false negative error, or in short a false negative, is a test result that indicates that a condition does not hold, while in fact it does. I.e., erroneously no effect has been inferred. An example is a truly guilty prisoner who is acquitted of a crime. The condition "the prisoner is guilty" holds (the prisoner is guilty). But the test (a trial in a court of law) failed to realize this, and wrongly decided the prisoner was not guilty, falsely concluding a negative about the condition. A false negative error is a type II error occurring in a test where a single condition is checked for and the result of the test is erroneously that the condition is absent.

Confusion Matrix: A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known. The confusion matrix itself is relatively simple to understand, but the related terminology can be confusing.

n=165	Predicted: NO	Predicted: YES
Actual: NO	50	10
Actual: YES	5	100

n=165	Predicted: NO	Predicted: YES	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

What can we learn from this matrix?

- There are two possible predicted classes: "yes" and "no". If we were predicting the presence of a disease, for example, "yes" would mean they have the disease, and "no" would mean they don't have the disease.

- The classifier made a total of 165 predictions (e.g., 165 patients were being tested for the presence of that disease).
- Out of those 165 cases, the classifier predicted "yes" 110 times, and "no" 55 times.
- In reality, 105 patients in the sample have the disease, and 60 patients do not.

Let's now define the most basic terms, which are whole numbers (not rates):

- true positives (TP): These are cases in which we predicted yes (they have the disease), and they do have the disease.
- true negatives (TN): We predicted no, and they don't have the disease.
- false positives (FP): We predicted yes, but they don't actually have the disease. (Also known as a "Type I error.")
- false negatives (FN): We predicted no, but they actually do have the disease. (Also known as a "Type II error.")

This is a list of rates that are often computed from a confusion matrix for a binary classifier:

- Accuracy: Overall, how often is the classifier correct?
  - (TP+TN)/total = (100+50)/165 = 0.91
- Misclassification Rate: Overall, how often is it wrong?
  - (FP+FN)/total = (10+5)/165 = 0.09
  - equivalent to 1 minus Accuracy
  - also known as "Error Rate"
- True Positive Rate (***Sensitivity***): When it's actually yes, how often does it predict yes?
  - TP/actual yes = 100/105 = 0.95
  - also known as "Sensitivity" or "Recall"
- False Positive Rate: When it's actually no, how often does it predict yes?
  - FP/actual no = 10/60 = 0.17
- ***Specificity***: When it's actually no, how often does it predict no?
  - TN/actual no = 50/60 = 0.83
  - equivalent to 1 minus False Positive Rate
- ***Precision***: When it predicts yes, how often is it correct?
  - TP/predicted yes = 100/110 = 0.91
- ***Prevalence***: How often does the yes condition actually occur in our sample?
  - actual yes/total = 105/165 = 0.64

Accuracy Paradox

n=200	Predicted: NO	Predicted: YES
Actual: NO	25	65
Actual: YES	10	100

Classification model based solution
Accuracy =125/200 = 60.25%

n=200	Predicted: NO	Predicted: YES
Actual: NO	0	35
Actual: YES	0	165

No model used, default Prediction is YES
Predicted Yes for all values
Accuracy = 165/200 = 82.5%

In the above example, the second table gives better accuracy which doesn't use any model instead predicts YES for all data. This shouldn't have been the case. Hence we can not rely on just accuracy metric. This is a good time to introduce few other terms:

Positive Predictive Value: This is very similar to precision, except that it takes prevalence into account. In the case where the classes are perfectly balanced (meaning the prevalence is 50%), the positive predictive value (PPV) is equivalent to precision.

$$\text{PPV} = \frac{\text{number of true positives}}{\text{number of true positives} + \text{number of false positives}} = \frac{\text{number of true positives}}{\text{number of positive calls}} = \frac{\text{sensitivity} \times \text{prevalence}}{\text{sensitivity} \times \text{prevalence} + (1 - \text{specificity}) \times (1 - \text{prevalence})}$$

The complement of the PPV is the false discovery rate (FDR): 1-PPV

Cumulative Accuracy Profile (CAP) Curve: Imagine that you as a data scientist work in a company that want to promote its new product so they will send an email with their offer to all the customers and usually 10% of the customer responses and actually buys the product so they though that that will be the case for this time and that scenario is called the Random Scenario.

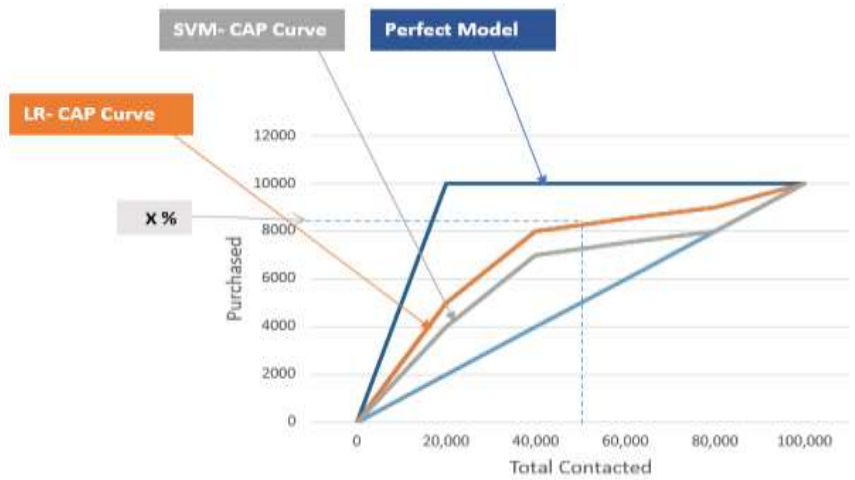
Inspect your historical data and take a group of customers who actually bought the offer and try to extract those information [browsing device type (mobile or laptop), Age, Salary, Savings]. Measure those factors and try to discover which of them affects the number of Purchased products or in other words fit the data to a Logistic Regression model. Make a prediction of which customers are more likely to purchase the product. Then by measuring the response of those targeted group represented in that curve 'CAP Curve'.

We definitely can notice the improvement; when you contacted 20,000 targeted customers you got about 5,000 positive responses where in scenario#1 by contacting the same number of customers, you got only 2,000 positive responses.

The idea here is to compare your model to the random scenario and you can take it to the next level by building another model maybe a Support Vector Machine (SVM)/ Kernel SVM model to compare it with your current logistic regression model.

Hypothetically we can draw the so called The Perfect Model which represents a model which is kind of impossible to build unless you have some sort of a Crystal Ball. It shows that when sending the offer to 10,000 possible customers you got a perfect positive response where all contacted people bought the product.

Now let's draw them on the plot.



There are 2 approaches to analyze the graph:

First —

- Calculate area under the Perfect Model Curve (aP)
- Calculate area under the Perfect Model Curve (aR)

- Calculate Accuracy rate(AR) = aR/ aP; as (AR)~1 (The better is your model) and as (AR)~0 (The worse is your model)

Second —

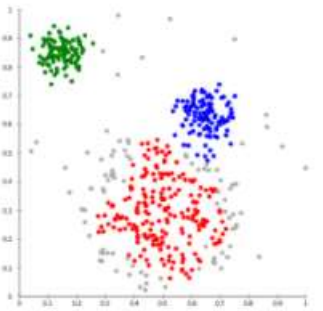
- Draw a line from the 50% point (50,000) in the Total Contacted axis up to the Model CAP Curve
- Then from that intersection point, Project it to the Purchased axis
- This X% value represents how good your model is:
- If X < 60% then you have a rubbish model
- If 60% < X < 70% then you have a poor model
- If 70% < X < 80% then you have a good model
- If 80% < X < 90% then you have a very good model
- If 90% < X < 100% then your model is too good to be true! This usually happens due Overfitting which is definitely not a good thing as your model will be good in classifying only the data it is trained on but very poor with new unseen instances.

By now, I am sure, you would have an idea of classification machine learning algorithms. If you are keen to master machine learning, start right away. Take up problems, develop a physical understanding of the process, apply these codes and see the fun!

UNIT 5: CLUSTERING

The general idea of a clustering algorithm is to partition a given dataset into distinct, exclusive clusters so that the data points in each group are quite similar to each other and are meaningful. Clustering falls into the unsupervised learning algorithms. Meaningfulness purely depends on the intention behind the purpose of the group's formations.

Suppose we have 100 articles and we want to group them into different categories. Let's consider the below categories: Sports articles, Business articles and Entertainment articles. When we group all the 100 articles into the above 3 categories. All the articles belong to the sports category will be same, In the sense, the content in the sports articles belongs to sports category. When you pick an article from sports category and the other article from business articles. Content-wise they will be completely different. This summarizes the rule of thumb condition to form clusters.



Much of the history of cluster analysis is concerned with developing algorithms that were not too computer intensive, since early computers were not nearly as powerful as they are today. Accordingly, computational shortcuts have traditionally been used in many cluster analysis algorithms. These algorithms have proven to be very useful, and can be found in most computer software.

More recently, many of these older methods have been revisited and updated to reflect the fact that certain computations that once would have overwhelmed the available computers can now be performed routinely. In R, a number of these updated versions of cluster analysis algorithms are available through the cluster library, providing us with a large selection of methods to perform cluster analysis, and the possibility of comparing the old methods with the new to see if they really provide an advantage. Lets look at some of the examples.

K-MEANS CLUSTERING

K-Means clustering approach is very popular in a variety of domains. In biology it is often used to find structure in DNA-related data or subgroups of similar tissue samples to identify cancer cohorts. In marketing, K-Means is often used to create market/customer/product segments. One of the first steps in building a K-Means clustering work is to define the number of clusters to work with. Subsequently, the algorithm assigns each individual data point to one of the clusters in a random fashion. Details Steps involved in K-means are:

1. Choose the K-number of clusters. Each data point is randomly assigned to a cluster.
2. Select random K points, the initial set of centroids (not necessarily from the dataset)
3. Assign each data point to the closet centroid that forms K clusters
4. Compute and place the new centroids of each cluster

5. Reassign each data point to the new closest centroid.
6. If there is any reassignment of clusters, go to Step 4 or else Stop.

The underlying idea of the algorithm is that a good cluster is the one which contains the smallest possible within-cluster variation of all observations in relation to each other. The most common way to define this variation is using the squared Euclidean distance. This process of identifying groups of similar data points can be a relatively complex task since there is a very large number of ways to partition data points into clusters.
Let's have a look at an example in R using the Chatterjee-Price Attitude Data from the library(datasets) package. The dataset is a survey of clerical employees of a large financial organization. The data are aggregated from questionnaires of approximately 35 employees for each of 30 (randomly selected) departments. The numbers give the percent proportion of favorable responses to seven questions in each department. For more details, see ?attitude.

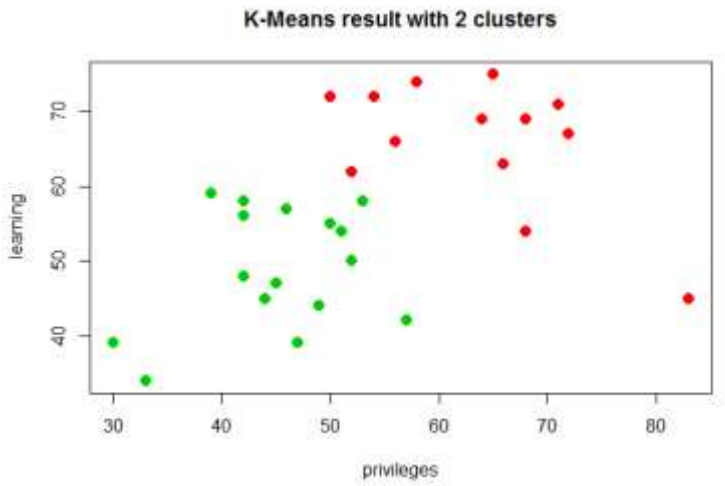
```
# # K-Means Clustering
# Importing the dataset
# Load necessary libraries
library(datasets)

# Inspect data structure
str(attitude)
# Summarise data
summary(attitude)

# Splitting the dataset into the Training set and Test set Not required in Clustering
# Feature Scaling Not required
```

This data gives the percent of favorable responses for each department. For example, one department had only 30% of responses favorable when it came to assessing 'privileges' and one department had 83% of favorable responses when it came to assessing 'privileges', and a lot of other favorable response levels in between.
When performing clustering, some important concepts like the data in hand should be standardized, whether the number of clusters obtained are truly representing the underlying pattern found in the data, whether there could be other clustering algorithms or parameters to be taken, etc. It is often recommended to perform clustering algorithms with different approaches and preferably test the clustering results with independent datasets. Particularly, it is very important to be careful with the way the results are reported and used.
For simplicity, we'll take a subset of the attitude dataset and consider only two variables in our K-Means clustering exercise. So imagine that we would like to cluster the attitude dataset with the responses from all 30 departments when it comes to 'privileges' and 'learning' and we would like to understand whether there are commonalities among certain departments when it comes to these two variables.

```
# # Subset the attitude data
dat = attitude[,c(3,4)]
# Plot subset data
plot(dat, main = "% of favourable responses to
    Learning and Privilege", pch =20, cex =2)
```

Now we can apply K-Means clustering to this data set and try to assign each department to a specific number of clusters that are “similar”. Let’s use the kmeans function from R base stats package:

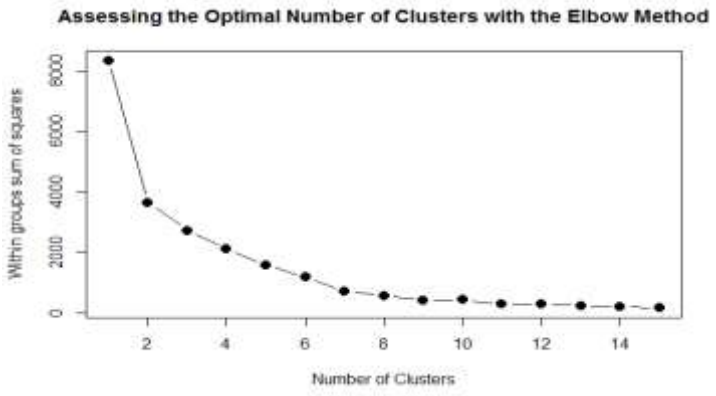
```
# # Perform K-Means with 2 clusters
set.seed(123)
km1 = kmeans(x =dat, centers = 2, nstart=100)
# Plot results
plot(dat, col =(km1$cluster +1) , main="K-Means result with 2 clusters", pch=20, cex=2)
```



As mentioned before, one of the key decisions to be made when performing K-Means clustering is to decide on the numbers of clusters to use. In practice, there is no easy answer and it’s important to try different ways and numbers of clusters to decide which options is the most useful, applicable or interpretable solution. However, one solution often used to identify the optimal number of clusters is called the Elbow method and it involves observing a set of possible numbers of clusters relative to how they minimize the within-cluster sum of squares. In other words, the Elbow method examines the within-cluster dissimilarity as a function of the number of clusters.

```
# # Using the elbow method to find the optimal number of clusters
mydata <- dat
wss <- (nrow(mydata)-1)*sum(apply(mydata,2,var))
for (i in 2:15) wss[i] <- sum(kmeans(mydata,
                                     centers=i)$withinss)
plot(1:15, wss, type="b", xlab="Number of Clusters",
   ylab="Within groups sum of squares",
   main="Assessing the Optimal Number of Clusters with the Elbow Method",
   pch=20, cex=2)
```

With the Elbow method, the solution criterion value (within groups sum of squares) will tend to decrease substantially with each successive increase in the number of clusters. Simplistically, an optimal number of clusters is identified once a “kink” in the line plot is observed and this is very subjective.



But from the example above, we can say that after 6 clusters the observed difference in the within-cluster dissimilarity is not substantial. Consequently, we can say with some reasonable confidence that the optimal number of clusters to be used is 6.

```
# # Perform K-Means with the optimal number of clusters identified from the Elbow method
set.seed(321)
km2 = kmeans(dat, 6, nstart=100)

# Examine the result of the clustering algorithm
km2
# # Plot results
plot(dat, col =(km2$cluster +1) , main="K-Means result with 6 clusters", pch=20, cex=2)
```

From the results above we can see that there is a relatively well defined set of groups of departments that are relatively distinct when it comes to answering favorably around Privileges and Learning in the survey. It is only natural to think the next steps from this sort of output. One could start to devise strategies to understand why certain departments rate these two different measures the way they do and what to do about it. But we will leave this to another exercise.

PARTITIONING AROUND MEDOIDS (PAM)

The k-means technique is fast, and doesn't require calculating all of the distances between each observation and every other observation. It can be written to efficiently deal with very large data sets, so it may be useful in cases where other methods fail. On the down side, if you rearrange your data, it's very possible that you'll get a different solution every time you change the ordering of your data.

The R cluster library provides a modern alternative to k-means clustering, known as pam, which is an acronym for "Partitioning around Medoids". The term medoid refers to an observation within a cluster for which the sum of the distances between it and all the other members of the cluster is a minimum. pam requires that you know the number of clusters that you want (like k-means clustering), but it does more computation than k-means in order to insure that the medoids it finds are truly representative of the observations within a given cluster.

Implementation in R

```
# pam: Advanced version of K-Means algorithm
# Importing the dataset
# Load necessary libraries
library(datasets)
library(cluster)
dat.pam = attitude[,c(3,4)]
set.seed(123)
cluster.pam = pam(dat.pam,2)
names(cluster.pam)
cluster.pam
```

Like most R objects, you can use the names function to see what else is available. Further information can be found in the help page for pam.object.

```
names(cluster.pam)
```

```
[1] "medoids"    "id.med"     "clustering" "objective"  "isolation"  "clusinfo"   "silinfo"
[8] "diss"       "call"       "data"
```

Plot the result

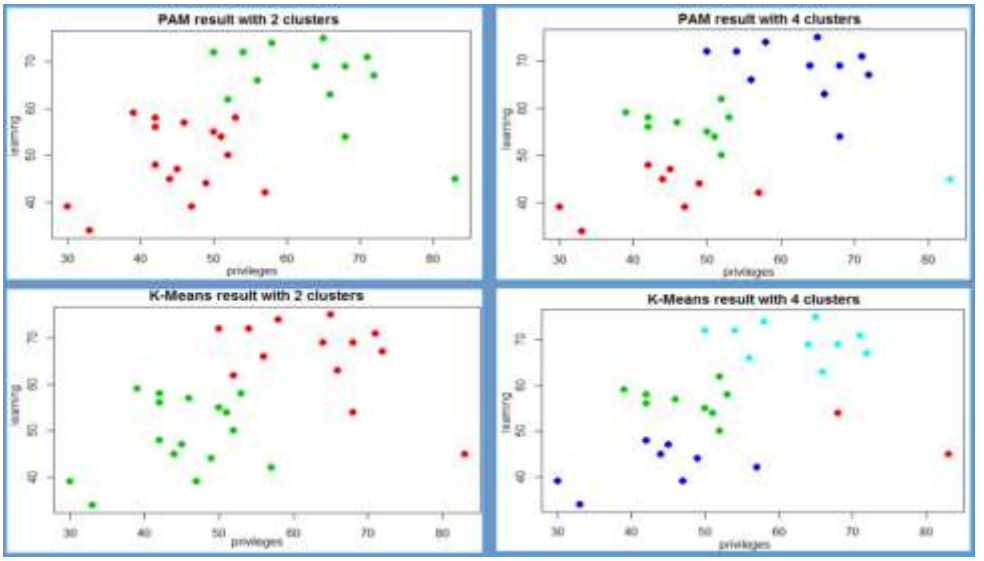
```
# Plot results
plot(dat, col =(cluster.pam$cluster +1) , main="PAM result with 2 clusters", pch=20, cex=2)
```

Using table function to compare the result:

```
#We can use table to compare the results of the kmeans and pam solutions:
table(km1$cluster,cluster.pam$clustering)
```

Analysis of the result

Below are confusion matrix and plots, first one was run with 2 clusters and second one was run with 4 clusters:



The solutions seem to agree, except for 1 observation.

HIERARCHICAL CLUSTERING

Hierarchical clustering is an alternative approach to k-means clustering for identifying groups in the dataset and does not require to pre-specify the number of clusters to generate. It refers to a set of clustering algorithms that build tree-like clusters by successively splitting or merging them. This hierarchical structure is represented using a tree. Hierarchical clustering methods use a distance similarity measure to combine or split clusters. The recursive process continues until there is only one cluster left or we cannot split more clusters. We can use a dendrogram to represent the hierarchy of clusters. Hierarchical classifications produced by either Agglomerative (Bottom up approach) or Divisive (Top down approach).

Agglomerative clustering: It's also known as AGNES (Agglomerative Nesting). It works in a bottom-up manner. Each object is initially considered as a single-element cluster (leaf). At each step of the algorithm, the two clusters that are the most similar are combined into a new bigger cluster (nodes). This procedure is iterated until all points are member of just one single big cluster (root). The result is a tree which can be plotted as a dendrogram.

Divisive hierarchical clustering: It's also known as DIANA (Divise Analysis) and it works in a top-down manner. The algorithm is an inverse order of AGNES. It begins with the root, in which all objects are included in a single cluster. At each step of iteration, the most heterogeneous cluster is divided into two. The process is iterated until all objects are in their own cluster.

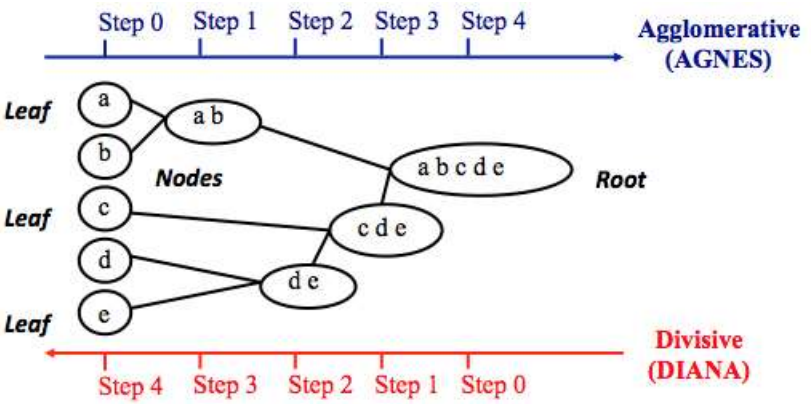


Figure 25: Agglomerative and Divisive clustering algorithms

Note that agglomerative clustering is good at identifying small clusters. Divisive hierarchical clustering is good at identifying large clusters.

How do we measure the dissimilarity between two clusters of observations? A number of different methods have been developed. The most common types methods are:

- Maximum or complete linkage clustering: It computes all pairwise dissimilarities between the elements in cluster 1 and the elements in cluster 2, and considers the largest value (i.e., maximum value) of these dissimilarities as the distance between the two clusters. It tends to produce more compact clusters.
- Minimum or single linkage clustering: It computes all pairwise dissimilarities between the elements in cluster 1 and the elements in cluster 2, and considers the smallest of these dissimilarities as a linkage criterion. It tends to produce long, "loose" clusters.

- Mean or average linkage clustering: It computes all pairwise dissimilarities between the elements in cluster 1 and the elements in cluster 2, and considers the average of these dissimilarities as the distance between the two clusters.
- Centroid linkage clustering: It computes the dissimilarity between the centroid for cluster 1 (a mean vector of length p variables) and the centroid for cluster 2.
- Ward's minimum variance method: It minimizes the total within-cluster variance. This method does not directly define a measure of distance between two points or clusters. It is an ANOVA based approach. At each stage, those two clusters merge, which provides the smallest increase in the combined error sum of squares from one-way univariate ANOVAs that can be done for each variable with groups defined by the clusters at that stage of the process.

The different approaches produce different dendrograms. Its left to the readers to plot and check different dendrograms. The code for the plotting and interpreting the dendrogram is given later in this section.

Let's see the implementation in R. Step 1 is to prepare for the dataset. We'll use the built-in R data set *USArrests*, which contains statistics in arrests per 100,000 residents for assault, murder, and rape in each of the 50 US states in 1973. It includes also the percent of the population living in urban areas. Below code is common for both Agglomerative and Divisive hierarchical clustering algorithms.

1. *Preparing the data set:*

```
# # Common code for both types of algorithm
# Libraries required:
library(cluster)            # clustering algorithms
library(factoextra)       # clustering visualization
library(dendextend)    # for comparing two dendrograms
# Read in-built dataset
library(Matrix)
df <- USArrests
#To remove any missing value:
df <- na.omit(df)
#Scaling the dataset
df <- scale(df)
head(df)
```

AGGLOMERATIVE HIERARCHICAL CLUSTERING

Hierarchical agglomerative clustering methods, starts out by putting each observation into its own separate cluster. It then examines all the distances between all the observations and pairs together the two closest ones to form a new cluster. This is a simple operation, since hierarchical methods require a distance matrix, and it represents exactly what we want - the distances between individual observations. So finding the first cluster to form simply means looking for the smallest number in the distance matrix and joining the two observations that the distance corresponds to into a new cluster. Now there is one less cluster than there are observations. To determine which observations will form the next cluster, we need to come up with a method for finding the distance between an existing cluster and individual observations.

Performing the clustering: The commonly used functions are: hclust [in stats package] and agnes [in cluster package] for agglomerative hierarchical clustering

First, we compute the dissimilarity values with dist and then feed these values into hclust and specify the agglomeration method to be used (i.e. “complete”, “average”, “single”, “ward.D”). We can plot the dendrogram after this.

```
# #####Method 1:  agglomerative HC with hclust    ####
# Dissimilarity matrix
diss.at <- dist(df, method = "euclidean")

# Hierarchical clustering using Complete Linkage
hc.hclust <- hclust(diss.at, method = "complete" )

# Plot the obtained dendrogram
plot(hc.hclust, cex = 0.6, hang = -1)
######## End of Method 1 :   agglomerative HC with hclust    ####
```

Dendogram will be analyzed later in this chapter.

Alternatively, we can use the agnes function. These functions behave very similarly; however, with the agnes function, we can also get the agglomerative coefficient, which measures the amount of clustering structure found (values closer to 1 suggest strong clustering structure).

```
# #####Method 2:  agglomerative HC with agnes    ####
# Compute with agnes
hc2 <- agnes(df, method = "complete")

# Agglomerative coefficient
hc2$ac

# Plot the obtained dendrogram
pltree(hc2, cex = 0.6, hang = -1, main = "Dendrogram of agnes")
######## End of Method 2 :   agglomerative HC with agnes  ####
```

```
> hc.agnes$ac
[1] 0.8531583
```

Agglomerative coefficient allows us to find certain hierarchical clustering methods that can identify stronger clustering structures. In the below example, we see that Ward’s method identifies the strongest clustering structure of the four methods assessed:

```
# # methods to assess
m <- c( "average", "single", "complete", "ward")
# function to compute coefficient
ac <- function(x) {
  ac.cal <- agnes(df, method = x)
  cat(x, " : " , ac.cal$ac)
}
#Calling the function
ac(m[1])       #Average
ac(m[2])      #Single
ac(m[3])      # Complete method
ac(m[4])      #Ward’s method
```

```
> ac(m[1])
average  :  0.7379371
> ac(m[2])
single  :  0.6276128
> ac(m[3])
complete  :  0.8531583
> ac(m[4])
ward  :  0.934621
```

DIVISIVE HIERARCHICAL CLUSTERING

Divisive clustering is called top-down clustering or divisive clustering. We start at the top with all documents in one cluster. The cluster is split using a flat clustering algorithm. This procedure is applied recursively until each document is in its own singleton cluster. The basic principle of divisive clustering was published as the DIANA (DIvisive ANAlysis Clustering) algorithm. Initially, all data is in the same cluster, and the largest cluster is split until every object is separate. DIANA chooses the object with the maximum average dissimilarity and then moves all objects to this cluster that are more similar to the new cluster than to the remainder.

Step 1: Preparing the data set:

```
# # DIVISIVE HIERARCHICAL CLUSTERING
# Read in-built dataset
df <- USArrests
#To remove any missing value:
df <- na.omit(df)
#Scaling the dataset
df <- scale(df)
head(df)
```

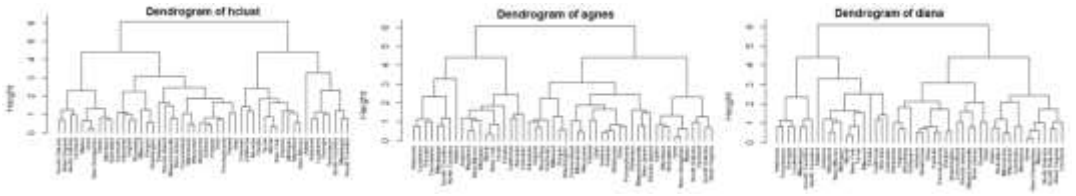
Step 2: Performing the clustering: The R function diana provided by the cluster package allows us to perform divisive hierarchical clustering. diana works similar to agnes; however, there is no method to provide.

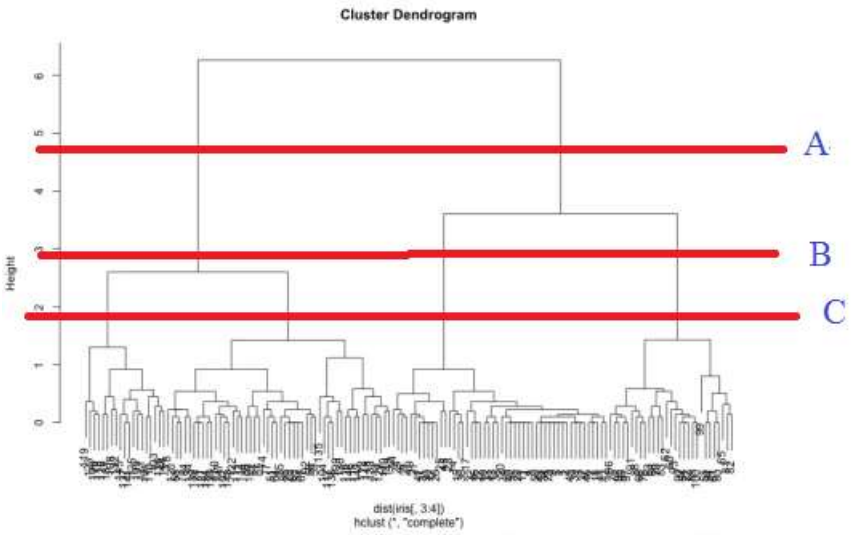
```
## # compute divisive hierarchical clustering
hc.diana <- diana(df)

# Divise coefficient; amount of clustering structure found
hc.diana$dc
## [1] 0.8514345

# plot dendrogram
pltree(hc.diana, cex = 0.6, hang = -1, main = "Dendrogram of diana")
```

WORKING WITH DENDROGRAMS

Let's look at the dendograms created by our last 3 algorithms. In the dendrogram displayed below, each leaf corresponds to one observation. As we move up the tree, observations that are similar to each other are combined into branches, which are themselves fused at a higher height. The height of the fusion, provided on the vertical axis, indicates the (dis)similarity

between two observations. The higher the height of the fusion, the less similar the observations are. We can use the analysis to come up with a good number of clusters. Let's see how.






```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

UNIT 6: ASSOCIATION RULE LEARNING

APRIORI

```
#
```

ECLAT

```
#
```

UNIT 7: REINFORCEMENT LEARNING

UPPER CONFIDENCE BOUND

```
#
```

THOMPSON SAMPLING

```
#
```

UNIT 8: NATURAL LANGUAGE PROCESSING

Implementation in R

Download the file 8_Restaurant_Reviews.csv from:
https://github.com/swapnilsaurav/MachineLearning
This file has 2 columns: Review and Liked. Review has the text description of the review for a restaurant whereas the Liked column has value 1 for positive comment and 0 for negative comment. The values are separated by Tab and not Comma because the description can also contain Comma. We will use below code to read the file.

```
# # Natural Language Processing

# Importing the dataset
library(readr)
dataset_original <- read_delim("D:/MachineLearning/8_Restaurant_Reviews.csv",
                               "\t", escape_double = FALSE, trim_ws = TRUE)
View(dataset_original)
```

```
# # Cleaning the texts
# install.packages('tm')
# install.packages('SnowballC')
library(tm)
library(SnowballC)
corpus = VCorpus(VectorSource(dataset_original$Review))
corpus = tm_map(corpus, content_transformer(tolower))
corpus = tm_map(corpus, removeNumbers)
corpus = tm_map(corpus, removePunctuation)
corpus = tm_map(corpus, removeWords, stopwords())
corpus = tm_map(corpus, stemDocument)
corpus = tm_map(corpus, stripWhitespace)
```

```
# # Creating the Bag of Words model
dtm = DocumentTermMatrix(corpus)
dtm = removeSparseTerms(dtm, 0.999)
dataset = as.data.frame(as.matrix(dtm))
dataset$Liked = dataset_original$Liked
```

```
# Encoding the target feature as factor
dataset$Liked = factor(dataset$Liked, levels = c(0, 1))

# Splitting the dataset into the Training set and Test set
# install.packages('caTools')
library(caTools)
set.seed(123)
split = sample.split(dataset$Liked, SplitRatio = 0.8)
training_set = subset(dataset, split == TRUE)
test_set = subset(dataset, split == FALSE)
```

```
# # Fitting Random Forest Classification to the Training set
# install.packages('randomForest')
library(randomForest)
classifier = randomForest(x = training_set[-692],
                          y = training_set$Liked,
                          ntree = 10)

# Predicting the Test set results
y_pred = predict(classifier, newdata = test_set[-692])

# Making the Confusion Matrix
cm = table(test_set[, 692], y_pred)
```

```
#
```

```
#
```

UNIT 9: DEEP LEARNING

Deep learning (also known as deep structured learning or hierarchical learning) is part of a broader family of machine learning methods based on learning data representations, as opposed to task-specific algorithms. Learning can be supervised, semi-supervised or unsupervised.

ARTIFICAL NEURAL NETWORKS

```
#
```

CONVOLUTIONAL NEURAL NETWORKS

```
#
```

UNIT 10: APPLICATION: RECOMMENDATION SYSTEM

```
#
```

UNIT 11: APPLICATION: FORECASTING ALGORITHMS

```
#
```

UNIT 12: APPLICATION: FACE RECOGNITION ALGORITHM

```
#
```

UNIT 13: APPLICATION: SOCIAL MEDIA ANALYTICS

```
#
```

```
#
```

UNIT 14: CONCLUSION

Thank you for being with me till the end. Before we finish lets take a mini tour and revisit all the algorithms we have covered till now and try to understand tradeoffs of each algorithm. We'll discuss the advantages and disadvantages of each algorithm. Categorizing machine learning algorithms is tricky, and there are several reasonable approaches; they can be grouped into generative/discriminative, parametric/non-parametric, supervised/ unsupervised, and so on.

One thing we need to understand is that no one algorithm works best for every problem, and it's especially relevant for supervised learning (i.e. predictive modeling). For example, you can't say that neural networks are always better than decision trees or vice-versa. There are many factors at play, such as the size and structure of your dataset. As a result, you should try many different algorithms for your problem, while using a hold-out "test set" of data to evaluate performance and select the winner. Having said that the algorithms you try must be appropriate for your problem, which is where picking the right machine learning task comes in. Let's go over one by one now.

REGRESSION

We now know that Regression is the supervised learning task for modeling and predicting continuous, numeric variables. Examples include predicting real-estate prices, stock price movements, or student test scores.

(Regularized) Linear Regression

Linear regression is one of the most common algorithms for the regression task. In its simplest form, it attempts to fit a straight hyperplane to your dataset (i.e. a straight line when you only have 2 variables). As you might guess, it works well when there are linear relationships between the variables in your dataset.

In practice, simple linear regression is often outclassed by its regularized counterparts (LASSO, Ridge, and Elastic-Net). Regularization is a technique for penalizing large coefficients in order to avoid overfitting, and the strength of the penalty should be tuned.

- Strengths: Linear regression is straightforward to understand and explain, and can be regularized to avoid overfitting. In addition, linear models can be updated easily with new data using stochastic gradient descent.
- Weaknesses: Linear regression performs poorly when there are non-linear relationships. They are not naturally flexible enough to capture more complex patterns, and adding the right interaction terms or polynomials can be tricky and time-consuming.
- Implementations: using package *glmnet*: Lasso and Elastic-Net Regularized Generalized Linear Models

Regression Tree (Ensembles)

Regression trees (or decision trees) learn in a hierarchical fashion by repeatedly splitting your dataset into separate branches that maximize the information gain of each split. This branching structure allows regression trees to naturally learn non-linear relationships. Ensemble methods, such as Random Forests (RF) and Gradient Boosted Trees (GBM), combine predictions from many individual trees. In practice, RF's often perform very well out-of-the-box while GBM's are harder to tune but tend to have higher performance ceilings.

- Strengths: Decision trees can learn non-linear relationships, and are fairly robust to outliers. Ensembles perform very well in practice, winning many classical (i.e. non-deep-learning) machine learning competitions.
- Weaknesses: Unconstrained, individual trees are prone to overfitting because they can keep branching until they memorize the training data. However, this can be alleviated by using ensembles.
- Implementations:
  - Random Forest - Package randomForest: Breiman and Cutler's Random Forests for Classification and Regression
  - Gradient Boosted Tree - Package gbm: Generalized Boosted Regression Models

Deep Learning

Deep learning refers to multi-layer neural networks that can learn extremely complex patterns. They use "hidden layers" between inputs and outputs in order to model intermediary representations of the data that other algorithms cannot easily learn.

- Strengths: Deep learning is the current state-of-the-art for certain domains, such as computer vision and speech recognition. Deep neural networks perform very well on image, audio, and text data, and they can be easily updated with new data using batch propagation. Their architectures (i.e. number and structure of layers) can be adapted to many types of problems, and their hidden layers reduce the need for feature engineering.
- Weaknesses: Deep learning algorithms are usually not suitable as general-purpose algorithms because they require a very large amount of data. In fact, they are usually outperformed by tree ensembles for classical machine learning problems. In addition, they are computationally intensive to train, and they require much more expertise to tune (i.e. set the architecture and hyperparameters).

Honorable Mention: Nearest Neighbors

Nearest neighbors algorithms are "instance-based," which means that that save each training observation. They then make predictions for new observations by searching for the most similar training observations and pooling their values.

These algorithms are memory-intensive, perform poorly for high-dimensional data, and require a meaningful distance function to calculate similarity. In practice, training regularized regression or tree ensembles are almost always better uses of your time.

CLASSIFICATION

Classification is the supervised learning task for modeling and predicting categorical variables. Examples include predicting employee churn, email spam, financial fraud, etc.

(Regularized) Logistic Regression

Logistic regression is the classification counterpart to linear regression. Predictions are mapped to be between 0 and 1 through the logistic function, which means that predictions can be interpreted as class probabilities.

- Strengths: Outputs have a nice probabilistic interpretation, and the algorithm can be regularized to avoid overfitting.
- Weaknesses: Logistic regression tends to underperform when there are multiple or non-linear decision boundaries.

Package - **glmnet**: Lasso and Elastic-Net Regularized Generalized Linear Models

Classification Tree (Ensembles)

Classification trees are the classification counterparts to regression trees. They are both commonly referred to as "decision trees" or by the umbrella term "classification and regression trees (CART)."

- Strengths: As with regression, classification tree ensembles also perform very well in practice. They are robust to outliers, scalable, and able to naturally model non-linear decision boundaries thanks to their hierarchical structure.
- Weaknesses: Unconstrained, individual trees are prone to overfitting, but this can be alleviated by ensemble methods.

packages **gbm**: Generalized Boosted Regression Models
randomForest: Breiman and Cutler's Random Forests for Classification and Regression

Deep Learning

To continue the trend, deep learning is also easily adapted to classification problems. In fact, classification is often the more common use of deep learning, such as in image classification.

- Strengths: Deep learning performs very well when classifying for audio, text, and image data.
- Weaknesses: As with regression, deep neural networks require very large amounts of data to train, so it's not treated as a general-purpose algorithm.

Support Vector Machines

Support vector machines (SVM) use a mechanism called kernels, which essentially calculate distance between two observations. The SVM algorithm then finds a decision boundary that maximizes the distance between the closest members of separate classes.
For example, an SVM with a linear kernel is similar to logistic regression. Therefore, in practice, the benefit of SVM's typically comes from using non-linear kernels to model non-linear decision boundaries.

- Strengths: SVM's can model non-linear decision boundaries, and there are many kernels to choose from. They are also fairly robust against overfitting, especially in high-dimensional space.
- Weaknesses: However, SVM's are memory intensive, trickier to tune due to the importance of picking the right kernel, and don't scale well to larger datasets. Currently in the industry, random forests are usually preferred over SVM's.

kernlab: Kernel-Based Machine Learning Lab

Naive Bayes

Essentially, Naive Bayes (NB) model is actually a probability table that gets updated through the training data. To predict a new observation, you'd simply "look up" the class probabilities in your "probability table" based on its feature values. It's called "naive" because its core assumption of conditional independence (i.e. all input features are independent from one another) rarely holds true in the real world.

- Strengths: Even though the conditional independence assumption rarely holds true, NB models actually perform surprisingly well in practice, especially for how simple they are. They are easy to implement and can scale with your dataset.
- Weaknesses: Due to their sheer simplicity, NB models are often beaten by models properly trained and tuned using the previous algorithms listed.

naivebayes: High Performance Implementation of the Naive Bayes Algorithm

CLUSTERING

Clustering is an unsupervised learning task for finding natural groupings of observations (i.e. clusters) based on the inherent structure within your dataset. Examples include customer segmentation, grouping similar items in e-commerce, and social network analysis.

K-Means

K-Means is a general purpose algorithm that makes clusters based on geometric distances (i.e. distance on a coordinate plane) between points. The clusters are grouped around centroids, causing them to be globular and have similar sizes. This is recommended algorithm for beginners because it's simple, yet flexible enough to get reasonable results for most problems.

- Strengths: K-Means is hands-down the most popular clustering algorithm because it's fast, simple, and surprisingly flexible if you pre-process your data and engineer useful features.
- Weaknesses: The user must specify the number of clusters, which won't always be easy to do. In addition, if the true underlying clusters in your data are not globular, then K-Means will produce poor clusters.

Affinity Propagation

Affinity Propagation is a relatively new clustering technique that makes clusters based on graph distances between points. The clusters tend to be smaller and have uneven sizes.

- Strengths: The user doesn't need to specify the number of clusters (but does need to specify 'sample preference' and 'damping' hyperparameters).
- Weaknesses: The main disadvantage of Affinity Propagation is that it's quite slow and memory-heavy, making it difficult to scale to larger datasets. In addition, it also assumes the true underlying clusters are globular.

apcluster: Affinity Propagation Clustering

Hierarchical / Agglomerative

Hierarchical clustering, a.k.a. agglomerative clustering, is a suite of algorithms based on the same idea: (1) Start with each point in its own cluster. (2) For each cluster, merge it with another based on some criterion. (3) Repeat until only one cluster remains and you are left with a hierarchy of clusters.

- Strengths: The main advantage of hierarchical clustering is that the clusters are not assumed to be globular. In addition, it scales well to larger datasets.
- Weaknesses: Much like K-Means, the user must choose the number of clusters (i.e. the level of the hierarchy to "keep" after the algorithm completes).

DBSCAN

DBSCAN is a density based algorithm that makes clusters for dense regions of points. There's also a recent new development called HDBSCAN that allows varying density clusters.

- Strengths: DBSCAN does not assume globular clusters, and its performance is scalable. In addition, it doesn't require every point to be assigned to a cluster, reducing the noise of the clusters.
- Weaknesses: The user must tune the hyperparameters 'epsilon' and 'min_samples,' which define the density of clusters. DBSCAN is quite sensitive to these hyperparameters.

dbscan: Density Based Clustering of Applications with Noise (DBSCAN) and Related Algorithms

HOW TO EVALUATE MACHINE LEARNING ALGORITHMS?

What algorithm should you use on your dataset?

This is the most common question in applied machine learning. It's a question that can only be answered by trial and error. No one can tell you what algorithm to use on your dataset to get the best results. If you or anyone knew what algorithm gave the best results for a specific dataset, then you probably would not need to use machine learning in the first place because of your deep knowledge of the problem.

We need a strategy to find the best algorithm for our dataset. One way that you could choose an algorithm for a problem is to reply on experience. But, the most robust way to discover good or even best algorithms for your dataset is by trial and error. Evaluate a diverse set of algorithms on your dataset and see what works and drop what doesn't.

Next, let's take a look at how we can evaluate multiple machine algorithms on a dataset in R.

Dataset

The test problem used in this example is a binary classification dataset from the UCI Machine Learning Repository call the *Pima Indians dataset*. The data describes medical details for female patients and boolean output variable as to whether they had an onset of diabetes within five years of their medical evaluation.

If you have a large dataset, take some different random samples and one simple model (glm) and see how long it takes to train. Select a sample size that falls within the sweet spot. There are only 768 instances in this case, so we will use all of the data.

Step 1: Let's load libraries and our diabetes dataset. It is distributed with the mlbench package, so we can just load it up.

```
# # load libraries
library(mlbench)
library(caret)
# load data
data(PimaIndiansDiabetes)
# rename dataset to keep code below generic
dataset <- PimaIndiansDiabetes
```

Step 2: Test Options

Test options refers to the technique used to evaluate the accuracy of a model on unseen data. They are often referred to as resampling methods in statistics.

In this book so far, we have used *Train/Test split*, which is recommended when we have a lot of data and we need to determine a dataset to build accurate models. Other methods are:

- Cross Validation: 5 folds or 10 folds provide a commonly used tradeoff of speed of compute time and generalize error estimate.
- Repeated Cross Validation: 5- or 10-fold cross validation and 3 or more repeats to give a more robust estimate, only if you have a small dataset and can afford the time.

In this case we will use 10-fold cross validation with 3 repeats.

```
# control <- trainControl(method="repeatedcv", number=10, repeats=3)
seed <- 123
```

Note that we assigning a random number seed to a variable, so that we can re-set the random number generator before we train each algorithm. This is important to ensure that each algorithm is evaluated on exactly the same splits of data, allow for true apples to apples comparisons later.

Step 3: Test Metric

There are many possible evaluation metrics to chose from. Caret provides a good selection and you can use your own if needed.

For Classification problems, we recommend:

- Accuracy: x correct divided by y total instances. Easy to understand and widely used.
- Kappa: easily understood as accuracy that takes the base distribution of classes into account.

For Regression problems:

- RMSE: root mean squared error. Again, easy to understand and widely used.
- Rsquared: the goodness of fit or coefficient of determination.

Other popular measures include ROC and LogLoss.

The evaluation metric is specified the call to the train() function for a given model, so we will define the metric now for use with all of the model training later.

```
# define the metric now for use with all of the model training later
metric <- "Accuracy"
```

Step 4: Running the algorithms

It is important to have a good mix of algorithm representations (lines, trees, instances, etc.) as well as algorithms for learning those representations. At least 10-to-20 different algorithms we need to choose. Almost all machine learning algorithms are parameterized, requiring that you specify their arguments. In this case, when we are comparing different algorithms, there is not need to try variations of algorithm parameters, that comes later when improving results.

```
# ##The most useful transform is to scale and center the data
preProcess=c("center", "scale")
# Linear Discriminant Analysis
set.seed(seed)
fit.lda <- train(diabetes~., data=dataset, method="lda", metric=metric, preProc=c("center", "scale"), trControl=control)
# Logistic Regression
set.seed(seed)
fit.glm <- train(diabetes~., data=dataset, method="glm", metric=metric, trControl=control)
# GLMNET
set.seed(seed)
fit.glmnet <- train(diabetes~., data=dataset, method="glmnet", metric=metric, preProc=c("center", "scale"), trControl=control)
# SVM Radial
set.seed(seed)
fit.svmRadial <- train(diabetes~., data=dataset, method="svmRadial", metric=metric, preProc=c("center", "scale"), trControl=control, fit=FALSE)
# kNN
set.seed(seed)
fit.knn <- train(diabetes~., data=dataset, method="knn", metric=metric, preProc=c("center", "scale"), trControl=control)
# Naive Bayes
set.seed(seed)
fit.nb <- train(diabetes~., data=dataset, method="nb", metric=metric, trControl=control)
# CART
set.seed(seed)
fit.cart <- train(diabetes~., data=dataset, method="rpart", metric=metric, trControl=control)
# C5.0
set.seed(seed)
fit.c50 <- train(diabetes~., data=dataset, method="C5.0", metric=metric, trControl=control)
# Bagged CART
set.seed(seed)
fit.treebag <- train(diabetes~., data=dataset, method="treebag", metric=metric, trControl=control)
# Random Forest
```

```
set.seed(seed)
fit.rf <- train(diabetes~., data=dataset, method="rf", metric=metric, trControl=control)
# Stochastic Gradient Boosting (Generalized Boosted Modeling)
set.seed(seed)
fit.gbm <- train(diabetes~., data=dataset, method="gbm", metric=metric, trControl=control, verbose=FALSE)
```

Step 6: Select the right Model

Now that we have trained a large and diverse list of models, we need to evaluate and compare them. The goal now is to select a handful, perhaps 2-to-5 diverse and well performing algorithms to investigate further

```
# Running all the models
results <- resamples(list(lda=fit.lda, logistic=fit.glm, glmnet=fit.glmnet,
                          svm=fit.svmRadial, knn=fit.knn, nb=fit.nb, cart=fit.cart, c50=fit.c50,
                          bagging=fit.treebag, rf=fit.rf, gbm=fit.gbm))
# Table comparison
summary(results)
```

Output is summarized as a table below:

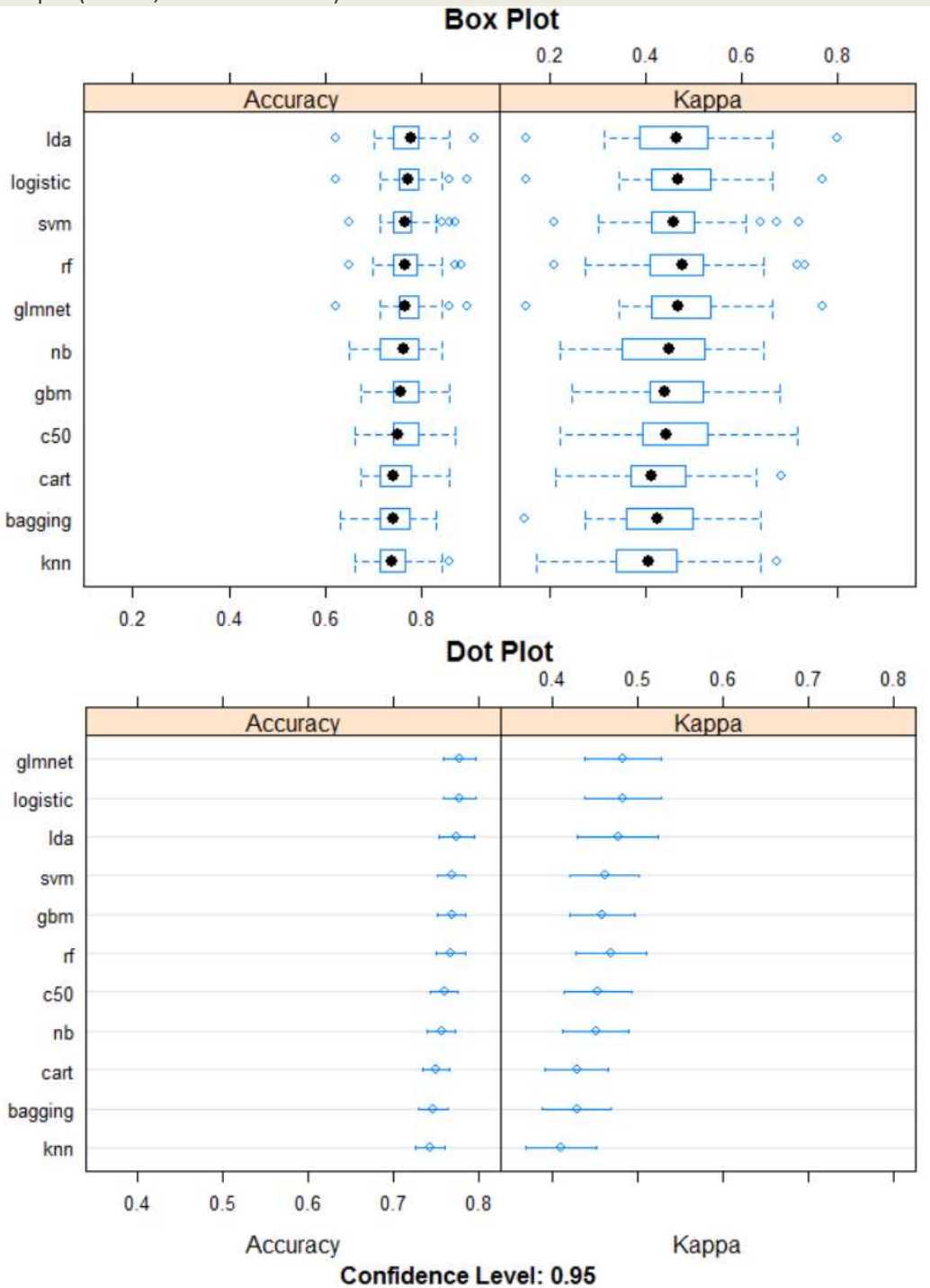
```
Accuracy
            Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
lda      0.6234  0.7427 0.7792 0.7743  0.7922 0.9091    0
logistic 0.6234  0.7532 0.7713 0.7769  0.7922 0.8961    0
glmnet   0.6234  0.7532 0.7662 0.7769  0.7922 0.8961    0
svm      0.6494  0.7403 0.7662 0.7686  0.7792 0.8701    0
knn      0.6623  0.7143 0.7386 0.7434  0.7655 0.8571    0
nb       0.6494  0.7143 0.7632 0.7561  0.7915 0.8442    0
cart     0.6753  0.7166 0.7403 0.7505  0.7760 0.8571    0
c50      0.6623  0.7403 0.7516 0.7596  0.7890 0.8701    0
bagging  0.6316  0.7166 0.7403 0.7465  0.7763 0.8312    0
rf       0.6494  0.7427 0.7662 0.7673  0.7895 0.8831    0
gbm      0.6753  0.7435 0.7582 0.7682  0.7890 0.8571    0

Kappa
            Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
lda      0.1513  0.3933 0.4645 0.4763  0.5279 0.7987    0
logistic 0.1513  0.4197 0.4668 0.4826  0.5337 0.7679    0
glmnet   0.1513  0.4209 0.4686 0.4823  0.5337 0.7679    0
svm      0.2098  0.4148 0.4590 0.4610  0.4980 0.7196    0
knn      0.1741  0.3414 0.4053 0.4100  0.4617 0.6723    0
nb       0.2234  0.3563 0.4491 0.4504  0.5225 0.6457    0
cart     0.2133  0.3713 0.4111 0.4281  0.4821 0.6836    0
c50      0.2229  0.3939 0.4435 0.4531  0.5235 0.7148    0
bagging  0.1502  0.3649 0.4247 0.4280  0.4910 0.6385    0
rf       0.2098  0.4104 0.4784 0.4680  0.5179 0.7319    0
gbm      0.2480  0.4086 0.4390 0.4580  0.5105 0.6781    0
```

Step 7: Use visualization techniques to understand the data better

It is also useful to review the results using a few different visualization techniques to get an idea of the mean and spread of accuracies.

```
# # boxplot comparison
```

```
bwplot(results, main="Box Plot")
# Dot-plot comparison
dotplot(results, main="Dot Plot")
```



From these results, it looks like linear methods do well on this problem. I would probably investigate logistic, lda, glmnet, and gbm further.

Below are some tips that you can use to get good at evaluating machine learning algorithms in R:

- Speed: Get results fast. Use small samples of your data and simple estimates for algorithm parameters.
- Diversity: Use a diverse selection of algorithms including representations and different learning algorithms for the same type of representation.
- Scale-up: Don't be afraid to schedule follow-up spot-check experiments with larger data samples.
- Short-list: Your goal is to create a shortlist of algorithms to investigate further, not optimize accuracy (atleast not at this stage).
- Heuristics: Best practice algorithm configurations and algorithms known to be suited to problems like ours an excellent place to start. Use default parameters as some algorithms only start to show that they are accurate with specific parameter configurations

I want to share the 5 steps process which I follow for my predictive modeling problems:
Step 1: Define your problem.
Step 2: Prepare your data.
Identify Outliers in your Data
Improve Model Accuracy with Data Pre-Processing
Discover Feature Selection
Manage Data Leakage in Machine Learning
Step 3: Spot-check algorithms.
Evaluate Machine Learning Algorithms
Choose The Right Test Options When Evaluating ML Algorithms
Step 4: Improve results.
Step 5: Present results.

And finally, few suggestions from our end:

Practice, practice, practice… Remember true mastery comes with practice.
Master the fundamentals... There are dozens of algorithms not covered in the book, and some of them can be quite effective but what we have covered will provide you a strong foundation for applied machine learning.
Take part in competitions… you'll develop practical intuition, which unlocks the ability to pick up almost any algorithm and apply it effectively.
Better data beats fancier algorithms… Garbage in will give you garbage out, right kind of data, effective exploratory analysis, data cleaning, and feature engineering can significantly boost your results.