

414458: COMPUTER LABORATORY – VII

PART B: Machine Learning Applications

B.E. (Information Technology) 2015 Course Semester I

Teaching Scheme

Practical : 4 Hrs/Week
Theory : 3 Hr/Week

Examination Scheme

Term Work : 50 Marks
Practical : 50 Marks

DEPARTMENT OF INFORMATION TECHNOLOGY
2021-2022

INDEX

<u>Sr.No</u>	<u>Title</u>	<u>Page No</u>
<u>1</u>	Study of platform for Implementation of Assignments Download the open source software of your interest. Document the distinct features and functionality of the software platform. You may choose WEKA and R and Python	<u>4-13</u>
<u>2</u>	Supervised Learning - Regression (Using R) Dataset (Advertising.csv) Generate a proper 2-D data set of N points. Split the data set into Training Data set and Test Data set. i) Perform linear regression analysis with Least Squares Method. ii) Plot the graphs for Training MSE and Test MSE and comment on Curve Fitting and Generalization Error. iii) Verify the Effect of Data Set Size and Bias-Variance Tradeoff. iv) Apply Cross Validation and plot the graphs for errors. v) Apply Subset Selection Method and plot the graphs for errors. vi) Describe your findings in each case	<u>14-20</u>
<u>3</u>	Create Association Rules for the Market Basket Analysis for the given Threshold. (Using R) Dataset (Groceries)	<u>21-27</u>
<u>4</u>	Implement K-Means algorithm for clustering to create a Cluster on the given data. (Using Python) Dataset (Mall_Customers.csv)	<u>28-36</u>
<u>5</u>	Implement SVM for performing classification and find its accuracy on the given data. (Using Python) Dataset (Social_Network_Ads.csv)	<u>37-44</u>
<u>6</u>	Creating & Visualizing Neural Network for the given data. (Using Python) Dataset (Churn_Modelling.csv)	<u>45-56</u>
<u>7</u>	On the given data perform the performance measurements using Simple Naïve Bayes algorithm such as Accuracy, Error rate, precision, Recall, TPR,FPR,TNR,FPR etc. (Using Weka API through JAVA) Dataset (weather.csv)	<u>57-62</u>
<u>8</u>	Principal Component Analysis-Finding Principal Components, Variance and Standard Deviation calculations of principal components. (Using R) Dataset (Iris.csv)	<u>63-68</u>

<u>Sr.No</u>	<u>Title</u>	<u>No of Hours</u>	<u>Week</u>
<u>1</u>	Study of platform for Implementation of Assignments Download the open source software of your interest. Document the distinct features and functionality of the software platform. You may choose WEKA and R and Python	2	1
<u>2</u>	Supervised Learning - Regression (Using R) Generate a proper 2-D data set of N points. Split the data set into Training Data set and Test Data set. i) Perform linear regression analysis with Least Squares Method. ii) Plot the graphs for Training MSE and Test MSE and comment on Curve Fitting and Generalization Error. iii) Verify the Effect of Data Set Size and Bias-Variance Tradeoff. iv) Apply Cross Validation and plot the graphs for errors. v) Apply Subset Selection Method and plot the graphs for errors. vi) Describe your findings in each case	4	2,3
<u>3</u>	Create Association Rules for the Market Basket Analysis for the given Threshold. (Using R)	2	4
<u>4</u>	Implement K-Means algorithm for clustering to create a Cluster on the given data.(Using Python)	2	5
<u>5</u>	Implement SVM for performing classification and find its accuracy on the given data. (Using Python)	2	6
<u>6</u>	Creating & Visualizing Neural Network for the given data. (Using Python)	2	7
<u>7</u>	On the given data perform the performance measurements using Simple Naïve Bayes algorithm such as Accuracy, Error rate, precision, Recall, TPR,FPR,TNR,FPR etc. (Using Weka API through JAVA)	2	8
<u>8</u>	Principal Component Analysis-Finding Principal Components, Variance and Standard Deviation calculations of principal components. (Using R)	2	9

Class and Subject: BEIT Information Technology (2015)
Subject: Computing Laboratory VII (Part B)

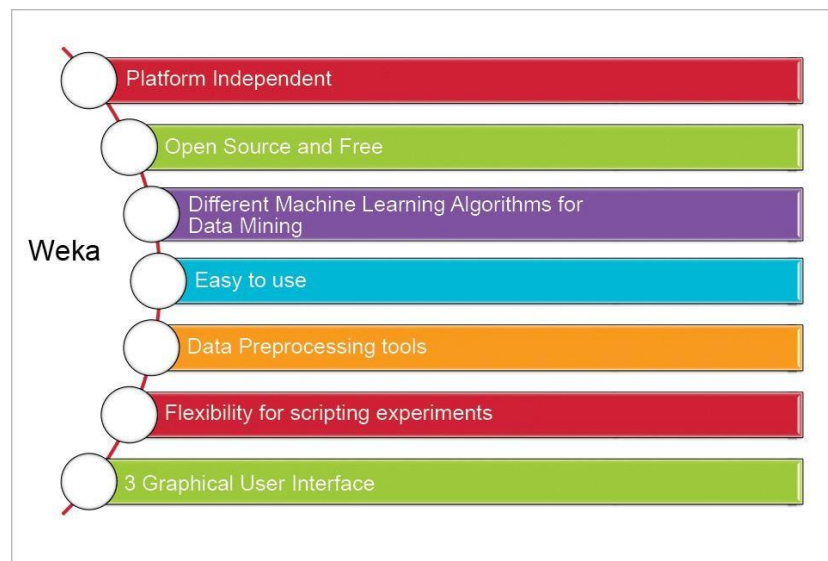
MLA Lab 1

Aim: Study of platform for Implementation of Assignments. Download the open source software of your interest. Document the distinct features and functionality of the software platform. You may choose WEKA and R and Python

A) WEKA Platform Setup

Web Site - <https://www.cs.waikato.ac.nz/ml/weka/>

Weka is a set of machine learning algorithms that can be applied to a data set directly, or called from your own Java code. Weka contains tools for data pre-processing, classification, regression, clustering, association rules, and visualisation.

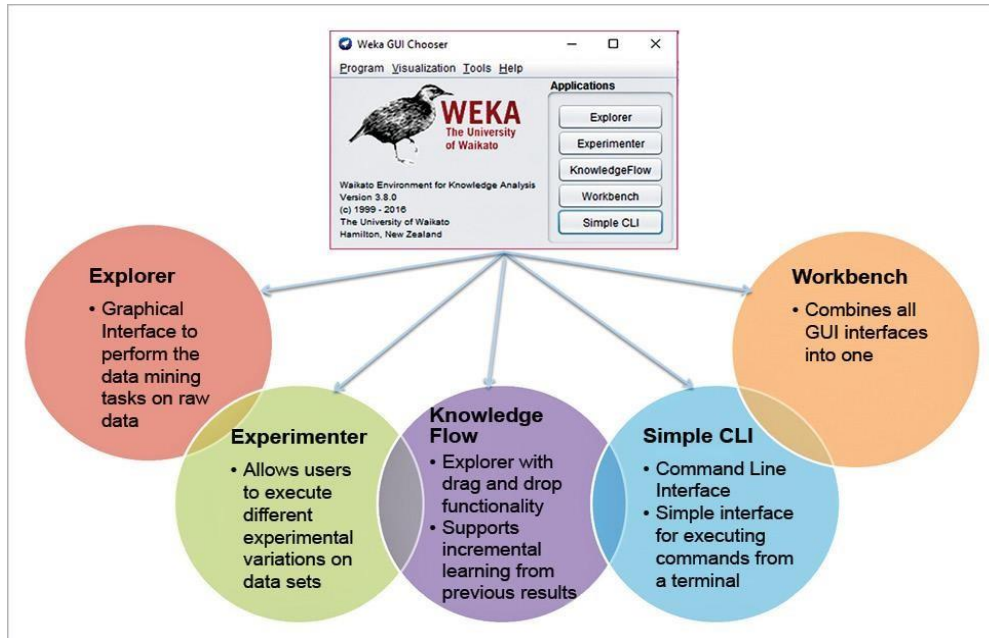


1. Make sure that you have installed JVM environment before installing following weka.

Click **here** to download a self-extracting executable for 64-bit Windows that includes Oracle's 64-bit Java VM 1.8
(weka-3-8-2jre-x64.exe; 265.4 MB)

2. If JVM is not installed then install following version of weka

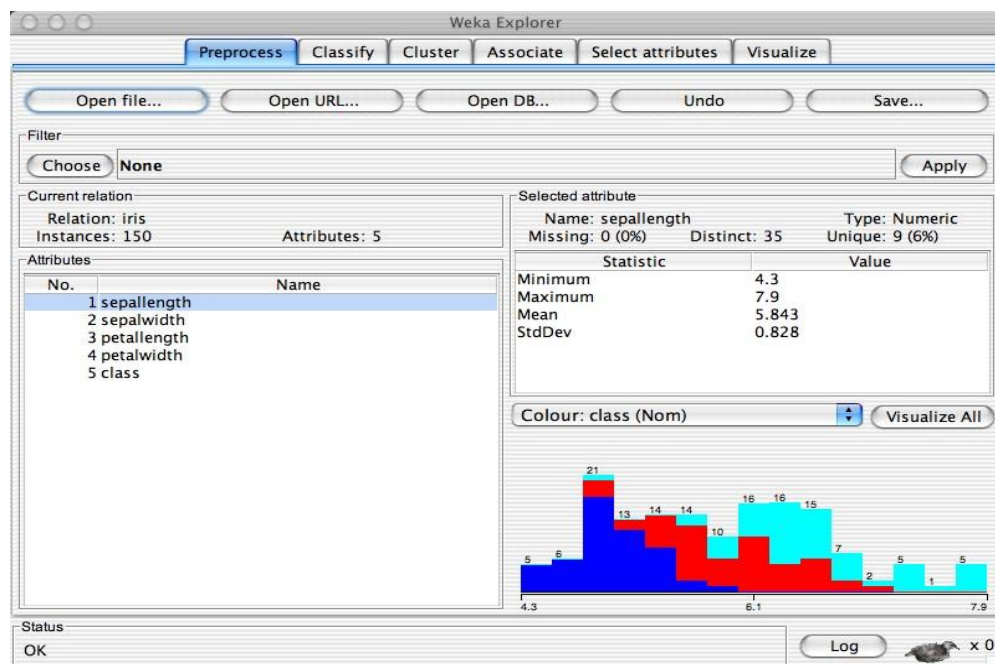
Click **here** to download a self-extracting executable for 64-bit Windows without a Java VM
(weka-3-8-2-x64.exe; 50.8 MB)



3. Weka's GUI allows you to:

- Preprocess data
- Choose learning algorithms
- Evaluate the results
- Build simple visualizations
- Form an interpretation of the results
- Export some output

So, if you want to start machine learning algorithms without much of a coding background WEKA is the tool for you. Finally, after you run the weka, will are set to experiments in machine learning as follows..



B) R & Rstudio for Machine Learning

1. **Download R from <http://cran.us.r-project.org/>.**
2. Click on **Download R for Windows**. Click on **base**. Click on **Download R 3.3.2 for Windows** (or a newer version that appears).
3. Install R. Leave all default settings in the installation options.
4. Download RStudio Desktop for windows from <http://rstudio.org/download/desktop> (it should be called something like RStudio 1.0.136—Windows Vista/7/8/10).
Choose default installation options

4. Install the packages (Optional)

If your need to use R requires a particular package/library to be installed in R-studio. You can follow the instructions below to do so

1. Run R studio
2. Click on the Packages tab in the bottom-right section and then click on install. The following dialog box will appear
3. In the Install Packages dialog, write the package name you want to install under the Packages field and then click install. This will install the package you searched for or give you a list of matching package based on your package text.

RStudio is a separate piece of software that works with R to make R much more user friendly and also adds some helpful features.

What can RStudio do for you?

First, RStudio gives R a point and click interface for a few of its features.

Second, RStudio adds a number of features that make your R programming easier and more efficient.

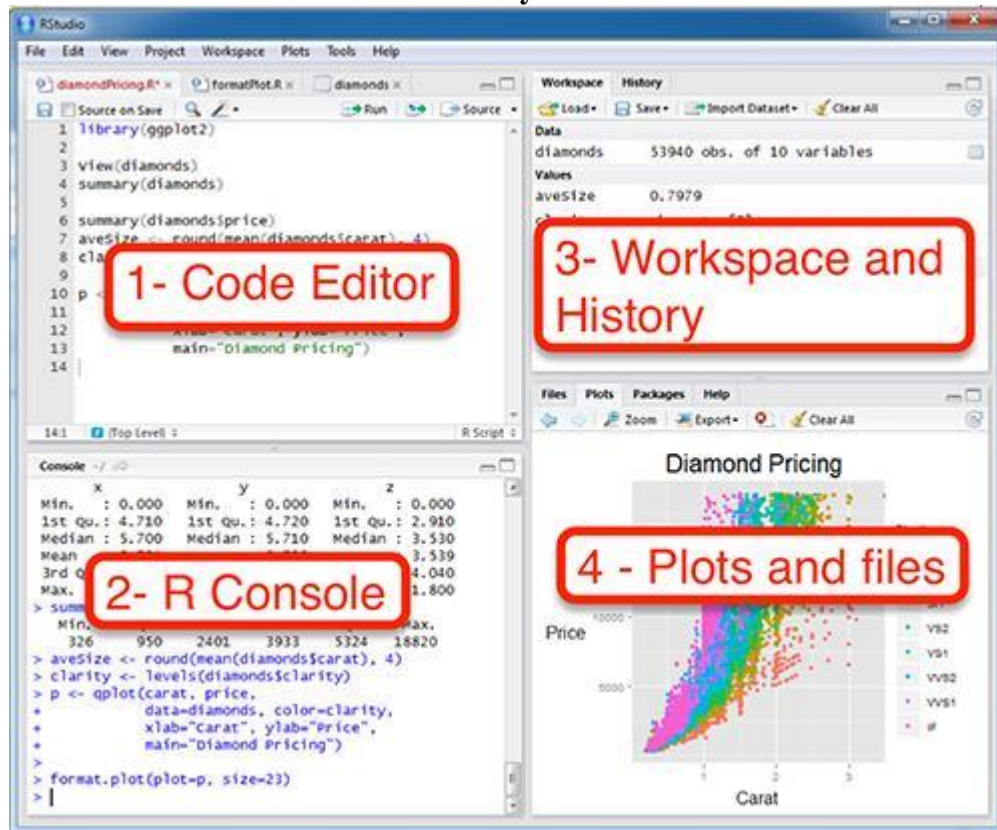
Here are the benefits, right from RStudio's website:

- Syntax highlighting, code completion, and smart indentation
- Execute R code directly from the source editor
- Easily manage multiple working directories using projects
- Quickly navigate code using type ahead search and go to definition

An IDE built for R

- Workspace browser and data viewer
- Plot history, zooming, and flexible image and PDF export
- Integrated R help and documentation
- Searchable command history

IDE looks like as follows after you install and start RStudio



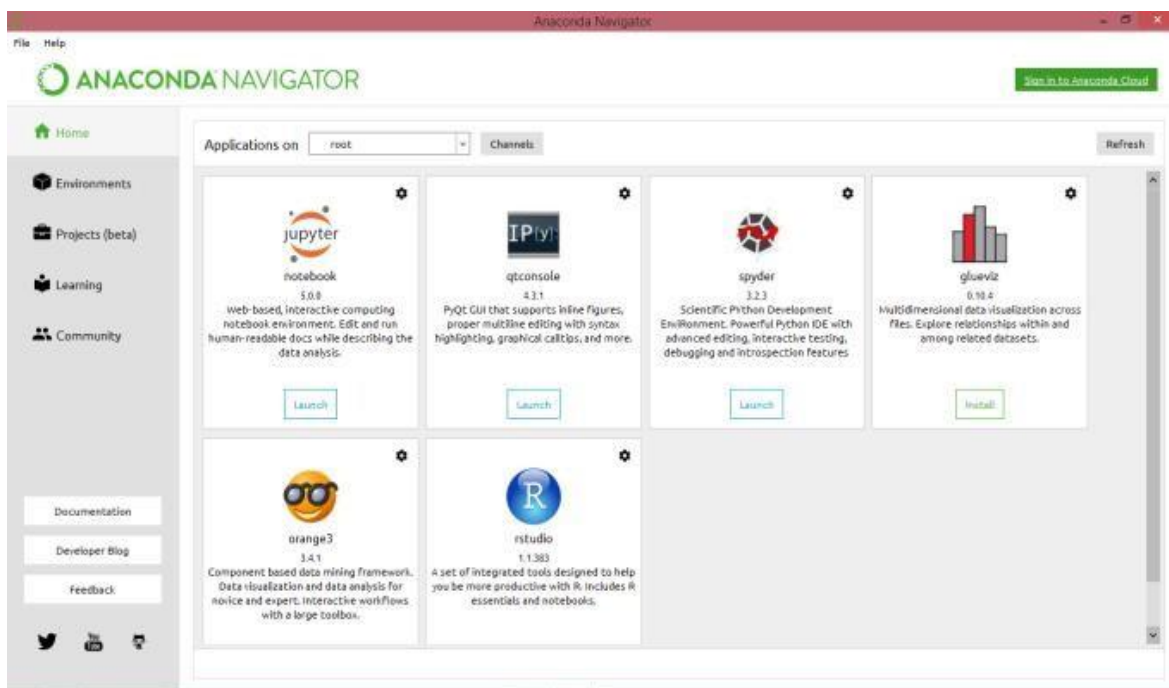
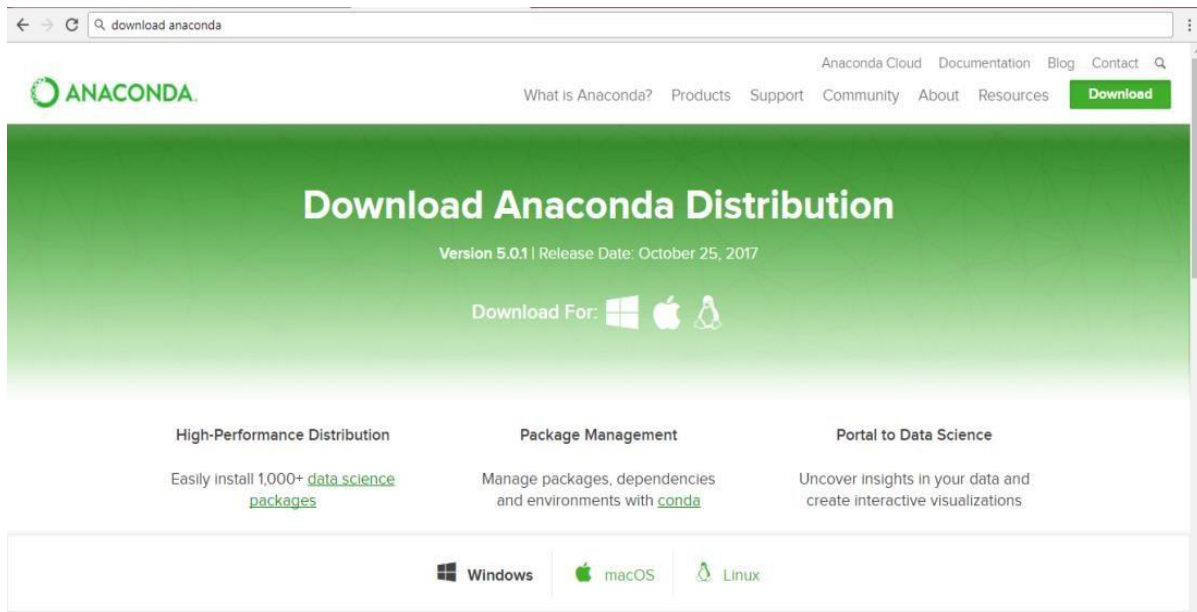
C) Anaconda, Python, Jupyter notebook for Machine Learning

There are a lot of environments that are available for free in the internet, where you could straight away go and start Programming, but Anaconda powered by Continuum Analytics is an environment that anyone could use to program in Python and R.

If you use wish to program in Python, then Jupyter Notebook is your place, where you have access to a lot of scientific and Numeric Libraries and if you are more of an Analytical Person than solving scientific problems then there is R studio where a ready environment is available for persons who wish to code in R

Installing Anaconda:

- click on this link to get Anacondas in the web, <https://www.anaconda.com/download/>
- Select the required OS that you have in your PC, (Linux, Windows, Mac)
- After downloading your file, **install** your software in the system.
- This will be your main screen of your Anaconda Prompt. This is just like a chrome where you get stuffed with a lot of webpages, but here with a lot of different tools that Anaconda is offering to us such as Jupyter, R studio, Orange, Spyder, etc..



Congratulations for setting up your Environment to code your Machine Learning Algorithm.

Python and Jupyter Notebook (Alternate installation) 64bit or 32 bit

This section shows downloading and installing Python 3.6.2 on Windows 10. **You should download and install the latest version of Python.** The current latest is Python 3.6.4.

The Python download requires about 30 Mb of disk space; keep it on your machine, in case you need to re-install Python. When installed, Python requires about an additional 90 Mb of disk space.

Downloading


1. Click Python Download.

The following page will appear in your browser.



2. Click the **Download Python 3.6.2** button.

The file named **python-3.6.2.exe** should start downloading into your standard download folder. This file is about 30 Mb so it might take a while to download fully if you are on a slow internet connection (it took me about 10 seconds over a cable modem).

The file should appear as  python-3.6.2.exe

3. Move this file to a more permanent location, so that you can install Python (and reinstall it easily later, if necessary).
4. Feel free to explore this webpage further; if you want to just continue the installation, you can terminate the tab browsing this webpage.
5. Start the **Installing** instructions directly below.

Installing

1. Double-click the icon labeling the file **python-3.6.2.exe**.

An **Open File - Security Warning** pop-up window will appear.



2. Click **Run**.

A **Python 3.6.2 (32-bit) Setup** pop-up window will appear.



Ensure that the **Install launcher for all users (recommended)** and the **Add Python 3.6 to PATH** checkboxes at the bottom are checked.

If the Python Installer finds an earlier version of Python installed on your computer, the **Install Now** message will instead appear as **Upgrade Now** (and the checkboxes will not appear).

3. Highlight the **Install Now** (or **Upgrade Now**) message, and then click it.

A **User Account Control** pop-up window will appear, posing the question **Do you want to allow the following program to make changes to this computer?**

4. Click the **Yes** button.

A new **Python 3.6.2 (32-bit) Setup** pop-up window will appear with a **Setup Progress** message and a progress bar.

During installation, it will show the various components it is installing and move the progress bar towards completion. Soon, a new **Python 3.6.2 (32-bit) Setup** pop-up window will appear with a **Setup was successfully** message.

5. Click the **Close** button.

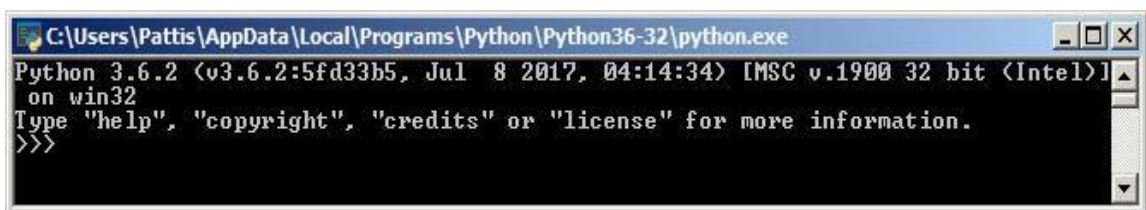
Python should now be installed.

Verifying

To try to verify installation,

1. Navigate to the directory **C:\Users\Pattis\AppData\Local\Programs\Python\Python36- 32** (or to whatever directory Python was installed: see the pop-up window for Installing step 3).
2. Double-click the icon/file **python.exe**.

The following pop-up window will appear.



A pop-up window with the title **C:\Users\Pattis\AppData\Local\Programs\Python\Python36-32** appears, and inside the window; on the first line is the text **Python 3.6.2 ...** (notice that it should also say 32 bit). Inside the window, at the bottom left, is the prompt **>>>**: type **exit()** to this prompt and press **enter** to terminate Python.

Open the command prompt and use following commands (make sure python scripts directory in path, set environment variable if python not running from command prompt.

1. C:>Pip list //should list all the libraries and tools
2. C:>Pip install scikit-learn // Machine learning
3. C:>Pip install scipy //library for scientific and technical computing
4. C:>Pip install nltk //natural languages ML
5. C:>Pip install matplotlib //for visualization
6. C:>Pip install jupyter // installs jupyter notebook

Now run the jupyter notebook with command

C:>Jupyter notebook

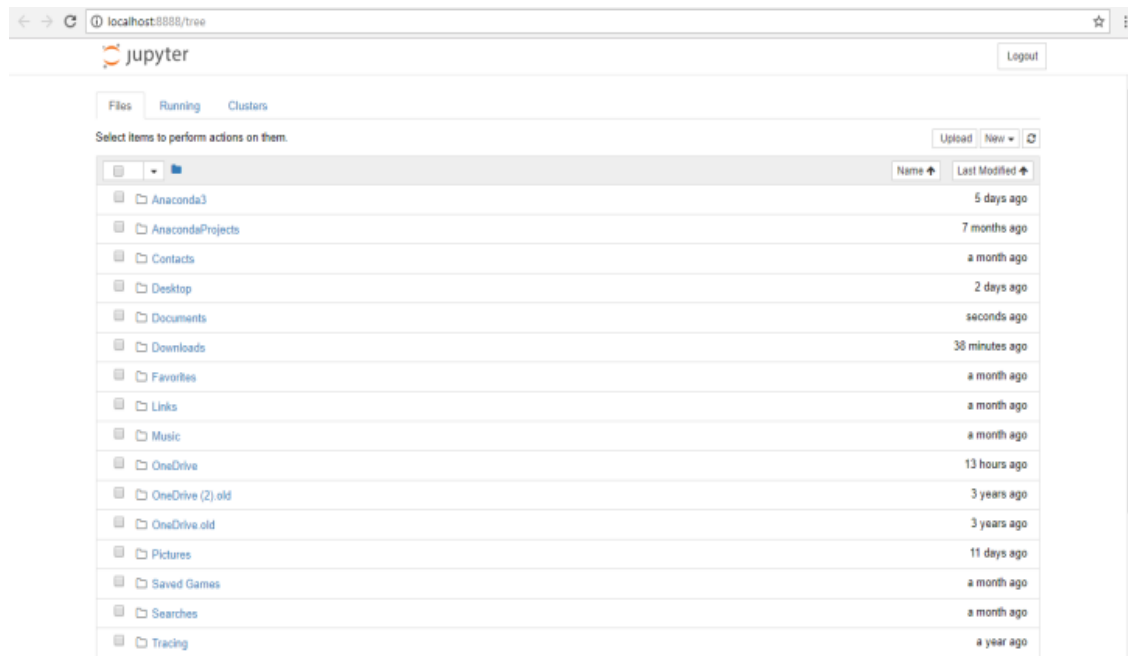
And you should see the IDE shown in next section

Or Running Jupyter Notebook with anaconda

Note that Jupyter Notebook runs on your browser, so each time you open your notebook you will be directed to your default browser. Use Chrome but some prefer Mozilla Firefox because of its timeliness.

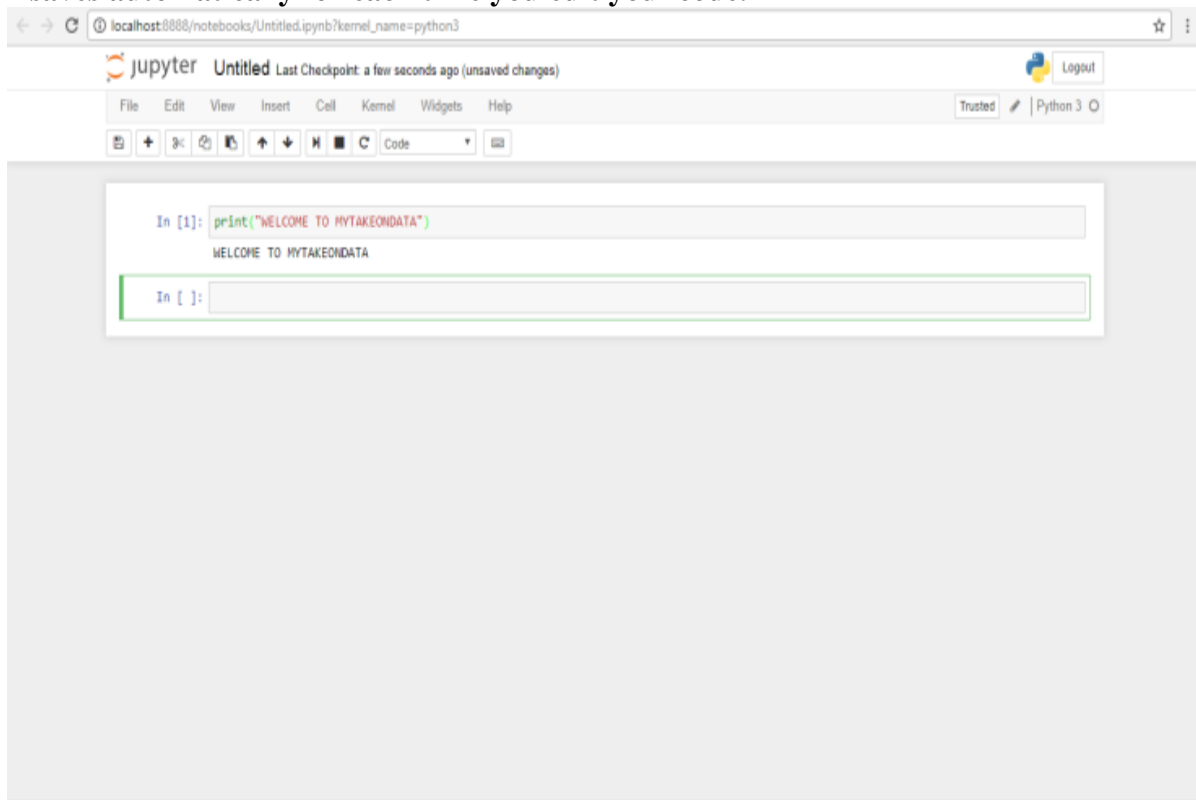
Click On the Launch button for Jupyter Notebook available in your Anaconda GUI

After Clicking Launch, you will be directed to your default browser which displays web gui like this.



- A list of folders available in your directory will be shown here.

- To enter your code in python click **New -> python3**
- Now you are in your first page of your python environment. Here's where you can execute your interactive code and save it on the run time. The best thing about Jupyter Notebooks is that **it saves automatically for each time you edit your code.**



You are ready with Weka, R and python environments in your computer and good luck for machine learning programming. Follow various ML tutorials with these tools

Class and Subject: BEIT Information Technology (2015)
Subject: Computing Laboratory VII (Part B)

MLA Lab 2

Aim: Supervised Learning - Regression (Using R)

Generate a proper 2-D data set of N points. Split the data set into Training Data set and Test Data set. (Use Advertising dataset)

- i) Perform linear regression analysis with Least Squares Method.
- ii) Plot the graphs for Training MSE and Test MSE and comment on Curve Fitting and Generalization Error.
- iii) Verify the Effect of Data Set Size and Bias-Variance Tradeoff.
- iv) Apply Cross Validation and plot the graphs for errors.
- v) Apply Subset Selection Method and plot the graphs for errors.
- vi) Describe your findings in each case

Theory background [Ref: Medium.com]

Mean Squared Error: General steps to calculate MSE the from a set of X and Y values:

1. Find the regression line.
2. Insert your X values into the linear regression equation to find the new Y values (Y').
3. Subtract the new Y value from the original to get the error.
4. Square the errors.
5. Add up the errors.
6. Find the mean.

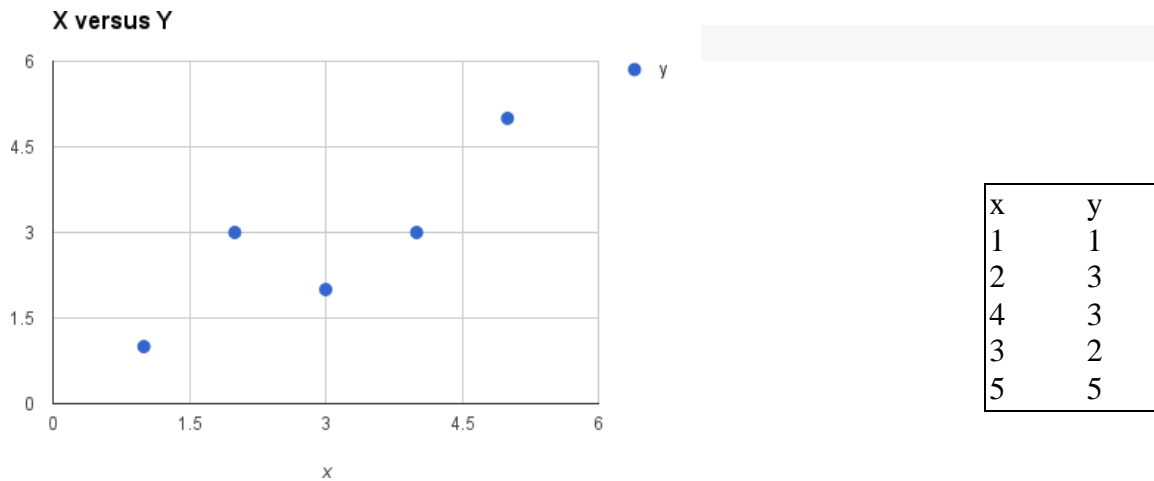
Tutorial Data Set

The data set we are using is completely made up.

Below is the raw data.

The attribute x is the input variable and y is the output variable that we are trying to predict. If we got more data, we would only have x values and we would be interested in predicting y values.

Below is a simple scatter plot of x versus y.



Plot of the Dataset for Simple Linear Regression

We can see the relationship between x and y looks kind of linear. As in, we could probably draw a line somewhere diagonally from the bottom left of the plot to the top right to generally describe the relationship between the data.

This is a good indication that using linear regression might be appropriate for this little dataset.

Simple Linear Regression

When we have a single input attribute (x) and we want to use linear regression, this is called simple linear regression.

If we had multiple input attributes (e.g. x1, x2, x3, etc.) This would be called multiple linear regression. The procedure for linear regression is different and simpler than that for multiple linear regression, so it is a good place to start.

In this section we are going to create a simple linear regression model from our training data, then make predictions for our training data to get an idea of how well the model learned the relationship in the data.

With simple linear regression we want to model our data as follows:

$$y = B0 + B1 * x$$

This is a line where y is the output variable we want to predict, x is the input variable we know and B0 and B1 are coefficients that we need to estimate that move the line around.

Technically, B0 is called the intercept because it determines where the line intercepts the y-axis. In machine learning we can call this the bias, because it is added to offset all predictions that we make. The B1 term is called the slope because it defines the slope of the line or how x translates into a y value before we add our bias.

The goal is to find the best estimates for the coefficients to minimize the errors in predicting y from x.

Simple regression is great, because rather than having to search for values by trial and error or calculate them analytically using more advanced linear algebra, we can estimate them directly from our data.

We can start off by estimating the value for B1 as:

$$B1 = \frac{\sum (Xi - \bar{X}) * (Yi - \bar{Y})}{\sum (Xi - \bar{X})^2}$$

Where mean() is the average value for the variable in our dataset. The xi and yi refer to the fact that we need to repeat these calculations across all values in our dataset and i refers to the i'th value of x or y.

We can calculate B0 using B1 and some statistics from our dataset, as follows:

$$B0 = \bar{Y} - (B1 * \bar{X})$$

Not that bad right? We can calculate these right in our spreadsheet.

Estimating Slope (B1)

Let's start with the top part of the equation, the numerator.

First we need to calculate the mean value of x and y. The mean is calculated as: sum(x) / n Where n is the number of values (5 in this case). Let's calculate the mean value of our x and y variables:

$$\bar{X} = 3, \quad \bar{Y} = 2.8$$

We now have the parts for calculating the numerator. All we need to do is multiple the error for each x with the error for each y and calculate the sum of these multiplications.

1	x - mean(x)	y - mean(y)	Multiplication
2	-2	-1.8	3.6
3	-1	0.2	-0.2
4	1	0.2	0.2
5	0	-0.8	0
6	2	2.2	4.4

Summing the final column we have calculated **our numerator as 8**.

Now we need to calculate the bottom part of the equation for calculating B1, or the denominator. This is calculated as the sum of the squared differences of each x value from the mean.

We have already calculated the difference of each x value from the mean, all we need to do is square each value and calculate the sum.

Calculating the sum of these squared values gives us up **denominator of 10**
 Now we can calculate the value of our slope.

$$B1 = 8 / 10 \text{ so further } B1 = 0.8$$

Estimating Intercept (B0)

This is much easier as we already know the values of all of the terms involved.

$$B0 = \bar{Y} - (B1 * \bar{X})$$

or

$$B0 = 2.8 - 0.8 * 3, \text{ or further } B0 = 0.4$$

Making Predictions

We now have the coefficients for our simple linear regression equation.

$$y = B0 + B1 * x$$

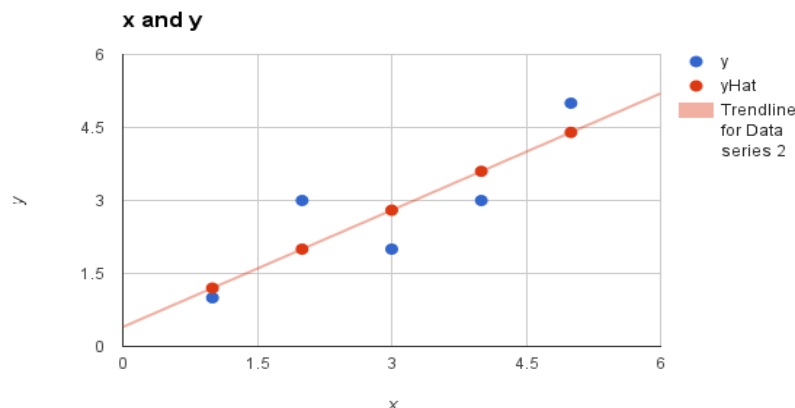
or

$$y = 0.4 + 0.8 * x$$

Let's try out the model by making predictions for our training data.

	x	y	predicted y
1	1	1	1.2
2	2	3	2
3	4	3	3.6
4	3	2	2.8
5	5	5	4.4

We can plot these predictions as a line with our data. This gives us a visual idea of how well the line models our data.



Simple Linear Regression Model

Estimating Error

We can calculate a error for our predictions called the Root Mean Squared Error or RMSE.

$$MSE = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2.$$

$$RMSE = \sqrt{\frac{\sum_{i=1}^N (Predicted_i - Actual_i)^2}{N}}$$

Where $\text{sqrt}()$ is the square root function, p is the predicted value and y is the actual value, i is the index for a specific instance, n is the number of predictions, because we must calculate the error across all predicted values.

First we must calculate the difference between each model prediction and the actual y values. We can easily calculate the square of each of these error values ($\text{error} \times \text{error}$ or error^2).

	pred-y	y	error	Squared error
1				
2	1.2	1	0.2	0.04
3	2	3	-1	1
4	3.6	3	0.6	0.36
5	2.8	2	0.8	0.64
6	4.4	5	-0.6	0.36

The sum of these errors is 2.4 units, dividing by n and taking the square root gives us: RMSE = 0.692 Or, each prediction is on average wrong by about 0.692 units.

//Python code for linear regression

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
data=pd.read_csv("headbrain.csv")
#print(data)

#Let us divide data set in training and testing data sets
X_train =data[:200]
X_test =data[200:]
#print("Training data set as follows")
#print(X_train)
#print("Testing data set as follows")
#print(X_test)

X = X_train['Head Size(cm^3)'].values
Y = X_train['Brain Weight(grams)'].values
mean_x = np.mean(X)
mean_y = np.mean(Y)
#print(X)
#print(Y)
print("Printing mean x")
print(mean_x)
print("Printing mean y")
print(mean_y)
```

```

m = len(X)
print("Number of samples in training set")
print(m)
numer = 0
denom = 0
for i in range(m):
    numer += (X[i] - mean_x) * (Y[i] - mean_y)
    denom += (X[i] - mean_x) ** 2
b1 = numer / denom
b0 = mean_y - (b1 * mean_x)
print("Coefficient and bias is as follows")
print(b1, b0)

# Plotting Values and Regression Line
max_x = np.max(X)
min_x = np.min(X)

# Calculating line values x and y
x = np.linspace(min_x, max_x)
y = b0 + b1 * x

# Plotting Line
plt.plot(x, y, color='YELLOW', label='Regression Line')
# Plotting Scatter Points
plt.scatter(X, Y, c='GREEN', label='Scatter Plot headsize vs brain wt')

plt.xlabel('Head Size in cm3')
plt.ylabel('Brain Weight in grams')
plt.legend()
plt.show()

# Calculating Root Mean Squares Error
sse = 0
for i in range(m):
    y_pred = b0 + b1 * X[i]    #y_pred i.e brain wt= b0+b1* head size
    sse += (Y[i] - y_pred) ** 2    #Y[i] means brain weight

print("Mean square error of brain wt of train data",sse)
mse = sse/m
rmse = np.sqrt(mse)
print("Root Mean square error is",rmse)

# Score of determination starts here
#The coefficient of determination (denoted by R2) is a key output of
regression analysis

```

```

#The coefficient of determination is the square of the correlation (r)
between predicted y scores and actual y scores;
#thus, it ranges from 0 to 1.
ss_t = 0
ss_r = 0
for i in range(m):
    y_pred = b0 + b1 * X[i]
    ss_t += (Y[i] - mean_y) ** 2
    ss_r += (Y[i] - y_pred) ** 2
scorer2 = 1 - (ss_r/ss_t)
print("R^2 score for training data is",scorer2)

```

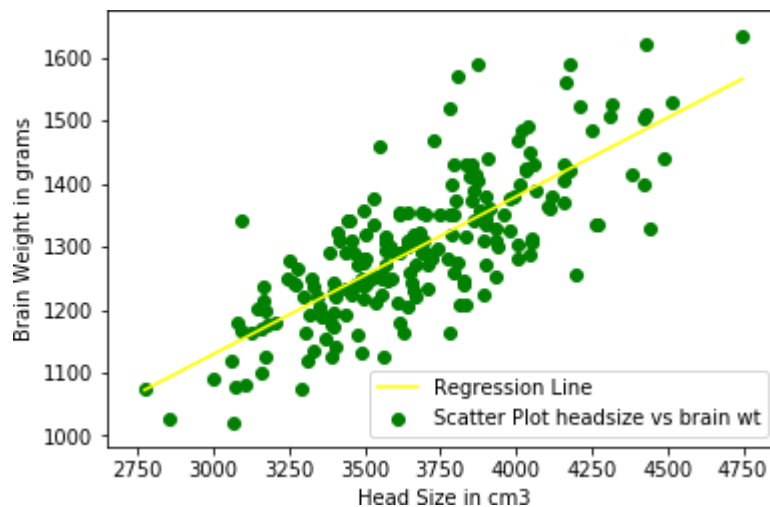
Output as follows

Printing mean x 3679.225

Printing mean y 1299.01

Number of samples in training set 200

Coefficient and bias is as follows 0.24984027563731726
379.79141186829145



Mean square error of brain wt of train data 1069588.9925686093

Root Mean square error is 73.12964489755879

R^2 score for training data is 0.5949551398852462

Class and Subject: BEIT Information Technology (2015)
Subject: Computing Laboratory VII (Part B)

MLA Lab 3

Aim: Create Association Rules for the Market Basket Analysis for the given Threshold.
(Using R) (Use Groceries Dataset)

Market Basket Analysis with R

Ref: Salesmarfi.com [Salem](#)



Source: [Paul's Health Blog](#)

Association Rules

There are many ways to see the similarities between items. These are techniques that fall under the general umbrella of **association**. The outcome of this type of technique, in simple terms, is a set of rules that can be understood as “if this, then that”.

Applications

So what kind of items are we talking about?

There are many applications of association:

- Product recommendation – like Amazon’s “customers who bought that, also bought this”
- Music recommendations – like Last FM’s artist recommendations
- Medical diagnosis – like with diabetes [really cool stuff](#)
- Content optimisation – like in magazine websites or blogs

In this post we will focus on the retail application – it is simple, intuitive, and the dataset comes packaged with R making it repeatable.

The Groceries Dataset

Imagine 10000 receipts sitting on your table. Each receipt represents a transaction with items that were purchased. The receipt is a representation of stuff that went into a customer's basket – and therefore 'Market Basket Analysis'.

That is exactly what the Groceries Data Set contains: a collection of receipts with each line representing 1 receipt and the items purchased. Each line is called a *transaction* and each column in a row represents an *item*. You can download the [Groceries data set](#) to take a look at it, but this is not a necessary step.

A little bit of Math

We already discussed the concept of Items and Item Sets.

We can represent our items as an item set as follows:

$$I = \{ i_1, i_2, i_3 \dots i_n \} \qquad I =$$

Therefore a transaction is represented as follows:

$$t_n = \{ i_j, i_k, \dots i_n \} \qquad t_n$$

This gives us our rules which are represented as follows:

$$\{i_1, i_2\} \Rightarrow i_k$$

Which can be read as "if a user buys an item in the item set on the left hand side, then the user will likely buy the item on the right hand side too". A more human readable example is:

$$\{\text{Coffee, Sugar}\} \Rightarrow \text{milk}$$

If a customer buys coffee and sugar, then they are also likely to buy milk.

With this we can understand three important ratios; the support, confidence and lift. We describe the significance of these in the following bullet points, but if you are interested in a formal mathematical definition you can find it on Wikipedia.

- **Support:** The fraction of which our item set occurs in our dataset.
- **Confidence:** probability that a rule is correct for a new transaction with items on the left.
- **Lift:** The ratio by which by the confidence of a rule exceeds the expected confidence.

Note: if the lift is 1 it indicates that the items on the left and right are independent.

Apriori Recommendation with R

So lets get started by loading up our libraries and data set.

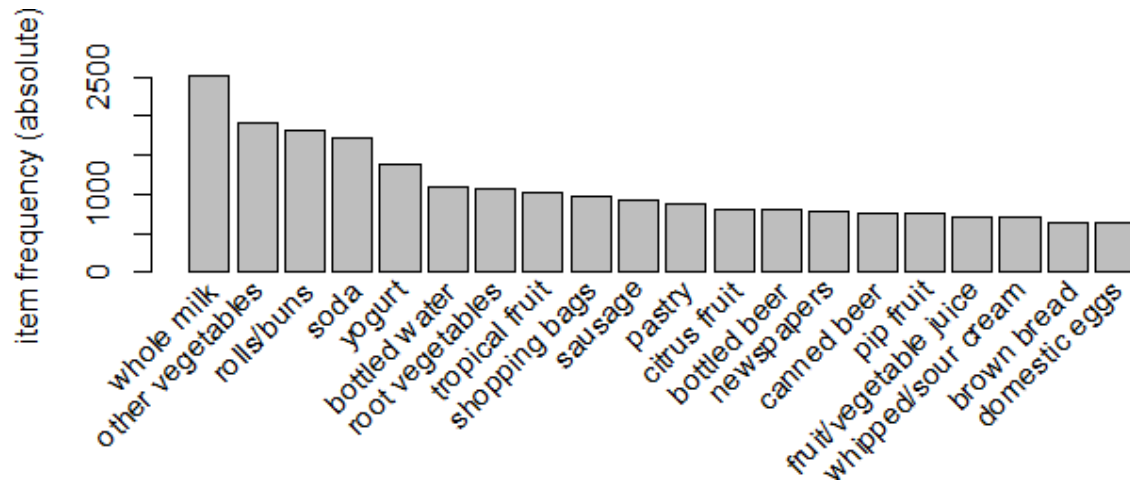
```
# Load the libraries
library(arules)
```

```
library(arulesViz)
library(datasets)

# Load the data set
data(Groceries)
```

Lets explore the data before we make any rules:

```
# Create an item frequency plot for the top 20 items
itemFrequencyPlot(Groceries, topN=20, type="absolute")
```



We are now ready to mine some rules!

You will always have to pass the minimum required **support** and **confidence**.

- We set the minimum support to 0.001
- We set the minimum confidence of 0.8
- We then show the top 5 rules

```
# Get the rules
rules <- apriori(Groceries, parameter = list(supp = 0.001, conf = 0.8))

# Show the top 5 rules, but only 2 digits
options(digits=2)
inspect(rules[1:5])
```

The output we see should look something like this

	lhs	rhs	support	confidence	lift
1	{liquor, red/blush wine}	=> {bottled beer}	0.0019	0.90	11.2
2	{curd, cereals}	=> {whole milk}	0.0010	0.91	3.6
3	{yogurt, cereals}	=> {whole milk}	0.0017	0.81	3.2
4	{butter, jam}	=> {whole milk}	0.0010	0.83	3.3
5	{soups, bottled beer}	=> {whole milk}	0.0011	0.92	3.6

This reads easily, for example: if someone buys yogurt and cereals, they are 81% likely to buy whole milk too.

We can get summary info. about the rules that give us some interesting information such as:

- The number of rules generated: 410
- The distribution of rules by length: Most rules are 4 items long

- The summary of quality measures: interesting to see ranges of support, lift, and confidence.
- The information on the data mined: total data mined, and minimum parameters.

```
set of 410 rules

rule length distribution (lhs + rhs): sizes
  3   4   5   6
29 229 140  12

summary of quality measures:
      support      conf.      lift
Min.   :0.00102   Min.   :0.80   Min.   : 3.1
1st Qu.:0.00102   1st Qu.:0.83   1st Qu.: 3.3
Median :0.00122   Median :0.85   Median : 3.6
Mean   :0.00125   Mean   :0.87   Mean   : 4.0
3rd Qu.:0.00132   3rd Qu.:0.91   3rd Qu.: 4.3
Max.   :0.00315   Max.   :1.00   Max.   :11.2

mining info:
      data      n      support      confidence
Groceries  9835    0.001      0.8
```

Sorting stuff out

The first issue we see here is that the rules are not sorted. Often we will want the most relevant rules first. Lets say we wanted to have **the most likely** rules. We can easily sort by confidence by executing the following code.

```
rules<-sort(rules, by="confidence", decreasing=TRUE)
```

Now our top 5 output will be sorted by confidence and therefore the most relevant rules appear.

	lhs	rhs	support	conf.	lift
1	{rice,sugar}	=> {whole milk}	0.0012	1	3.9
2	{canned fish,hygiene articles}	=> {whole milk}	0.0011	1	3.9
3	{root vegetables,butter,rice}	=> {whole milk}	0.0010	1	3.9
4	{root vegetables,whipped/sour cream,flour}	=> {whole milk}	0.0017	1	3.9
5	{butter,soft cheese,domestic eggs}	=> {whole milk}	0.0010	1	3.9

Rule 4 is perhaps excessively long. Lets say you wanted more concise rules. That is also easy to do by adding a "maxlen" parameter to your apriori function:

```
rules <- apriori(Groceries, parameter = list(supp = 0.001, conf = 0.8,maxlen=3))
```

Redundancies

Sometimes, rules will repeat. Redundancy indicates that one item might be a given. As an analyst you can elect to drop the item from the dataset. Alternatively, you can remove redundant rules generated.

We can eliminate these repeated rules using the follow snippet of code:

```
subset.matrix <- is.subset(rules, rules)
subset.matrix[lower.tri(subset.matrix, diag=T)] <-NA
redundant <- colSums(subset.matrix, na.rm=T) >= 1
rules.pruned <- rules[!redundant]
rules<-rules.pruned
```


Targeting Items

Now that we know how to generate rules, limit the output, lets say we wanted to target items to generate rules. There are two types of targets we might be interested in that are illustrated with an example of “whole milk”:

- What are customers likely to buy before buying whole milk
- What are customers likely to buy if they purchase whole milk?

This essentially means we want to set either the Left Hand Side and Right Hand Side. This is not difficult to do with R!

Answering the first question we adjust our `apriori()` function as follows:

```
rules<-apriori(data=Groceries, parameter=list(supp=0.001,conf = 0.08),
               appearance = list(default="lhs",rhs="whole milk"),
               control = list(verbose=F))
rules<-sort(rules, decreasing=TRUE,by="confidence")
inspect(rules[1:5])
```

The output will look like this:

	lhs	rhs	supp.	conf.	lift
1	{rice,sugar}	=> {whole milk}	0.0012	1	3.9
2	{canned fish,hygiene articles}	=> {whole milk}	0.0011	1	3.9
3	{root vegetables,butter,rice}	=> {whole milk}	0.0010	1	3.9
4	{root vegetables,whipped/sour cream,flour}	=> {whole milk}	0.0017	1	3.9
5	{butter,soft cheese, domestic eggs}	=> {whole milk}	0.0010	1	3.9

Likewise, we can set the left hand side to be “whole milk” and find its antecedents.

Note the following:

- We set the confidence to 0.15 since we get no rules with 0.8
- We set a minimum length of 2 to avoid empty left hand side items

```
rules<-apriori(data=Groceries, parameter=list(supp=0.001,conf = 0.15,minlen=2),
               appearance = list(default="rhs",lhs="whole milk"),
               control = list(verbose=F))
rules<-sort(rules, decreasing=TRUE,by="confidence")
inspect(rules[1:5])
```

Now our output looks like this:

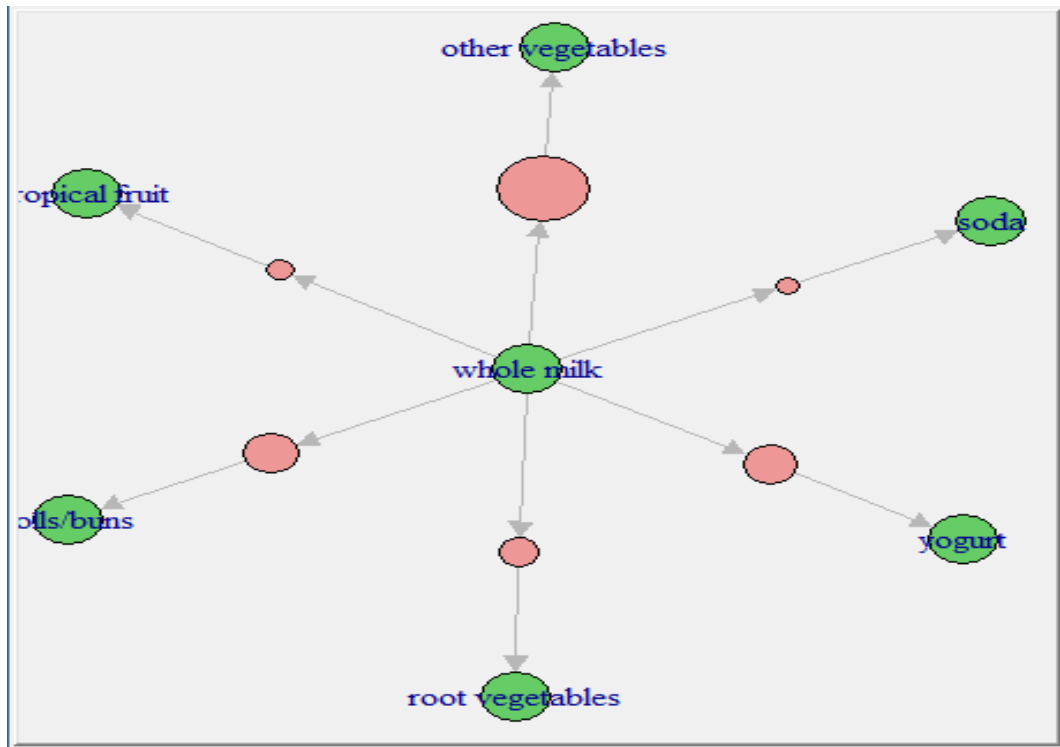
	lhs	rhs	support	confidence	lift
1	{whole milk}	=> {other vegetables}	0.075	0.29	1.5
2	{whole milk}	=> {rolls/buns}	0.057	0.22	1.2
3	{whole milk}	=> {yogurt}	0.056	0.22	1.6
4	{whole milk}	=> {root vegetables}	0.049	0.19	1.8
5	{whole milk}	=> {tropical fruit}	0.042	0.17	1.6
6	{whole milk}	=> {soda}	0.040	0.16	0.9

Visualization

The last step is visualization. Lets say you wanted to map out the rules in a graph. We can do that with another library called “arulesViz”.

```
library(arulesViz)
plot(rules,method="graph",interactive=TRUE,shading=NA)
```

You will get a nice graph that you can move around to look like this:



References

- [Snowplow Market Basket Analysis](#)
- [Discovering Knowledge in Data: An Introduction to Data Mining](#)
- [RDatamining.com](#)

```
# Load the libraries for apriori algorithm, visualizations and for required data set
```

```
library(arules)
library(arulesViz)
library(datasets)
```

```
# Load the data set
data(Groceries)
```

```
# Lets explore the data before we make any rules:
# Create an item frequency plot for the top 20 items
itemFrequencyPlot(Groceries,topN=20,type="absolute")
```

```
# You will always have to pass the minimum required support and confidence.
# We set the minimum support to 0.001
# We set the minimum confidence of 0.8
```

```
# Get the rules
rules <- apriori(Groceries, parameter = list(supp = 0.001, conf = 0.8))
```

```
# Show the top 5 rules, but only 2 digits
options(digits=2)
inspect(rules[1:5])
```

```
# Sorting Rules by confidence
rules <- sort(rules, by="confidence", decreasing=TRUE)
rules <- apriori(Groceries, parameter = list(supp = 0.001, conf = 0.8, maxlen=3))
```

```
# Redundancies
subset.matrix <- is.subset(rules, rules)
subset.matrix[lower.tri(subset.matrix, diag=T)] <- NA
redundant <- colSums(subset.matrix, na.rm=T) >= 1
rules.pruned <- rules[!redundant]
rules <- rules.pruned
```

```
# Targeting items
rules <- apriori(data=Groceries, parameter=list(supp=0.001,conf = 0.08),
  appearance = list(default="lhs",rhs="whole milk"),
  control = list(verbose=F))
rules <- sort(rules, decreasing=TRUE,by="confidence")
inspect(rules[1:5])
```

```
rules <- apriori(data=Groceries, parameter=list(supp=0.001,conf = 0.15,minlen=2),
  appearance = list(default="rhs",lhs="whole milk"),
  control = list(verbose=F))
rules <- sort(rules, decreasing=TRUE,by="confidence")
inspect(rules[1:5])
```

```
# Visualization
library(arulesViz)
plot(rules,method="graph",interactive=TRUE,shading=NA)
```

Class and Subject: BEIT Information Technology (2015)
Subject: Computing Laboratory VII (Part B)

MLA Lab 4

Aim: Implement K-Means algorithm for clustering to create a Cluster on the given data. (Using Python)
(Use Mall_Customers Dataset)

K-Means Clustering [Ref: <https://www.analyticsvidhya.com>]

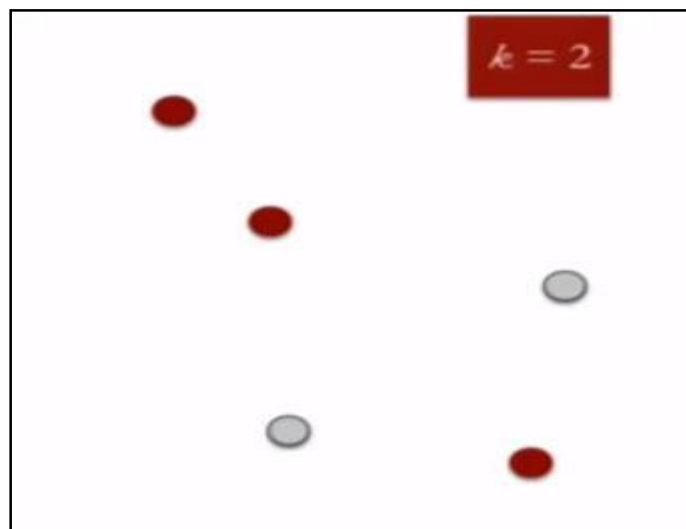
One of the simplest and most widely used unsupervised learning algorithm. It involves a simple way to classify the data set into fixed no. of **K** clusters. The idea is to define **K** centroids, one for each cluster.

The final clusters depend on the initial configuration of centroids. So, they should be initialized as far from each other as possible.

- K-Means is *iterative* in nature and *easy* to implement.

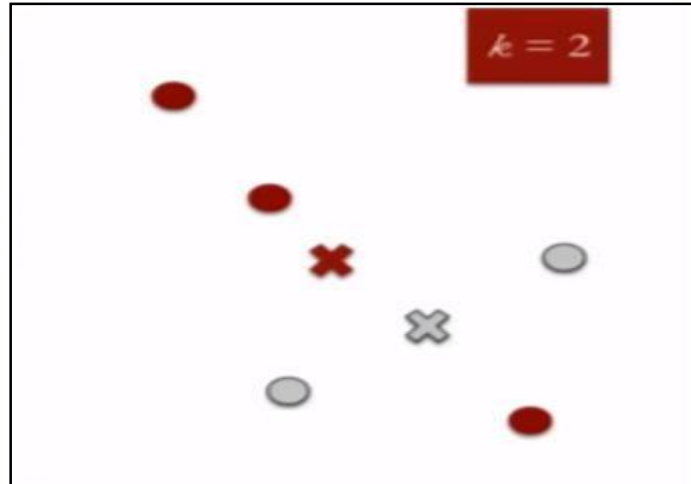
Algorithm Explained

- Let there be **N** data points. At first, **K** centroids are initialised in our data set representing **K** different clusters.



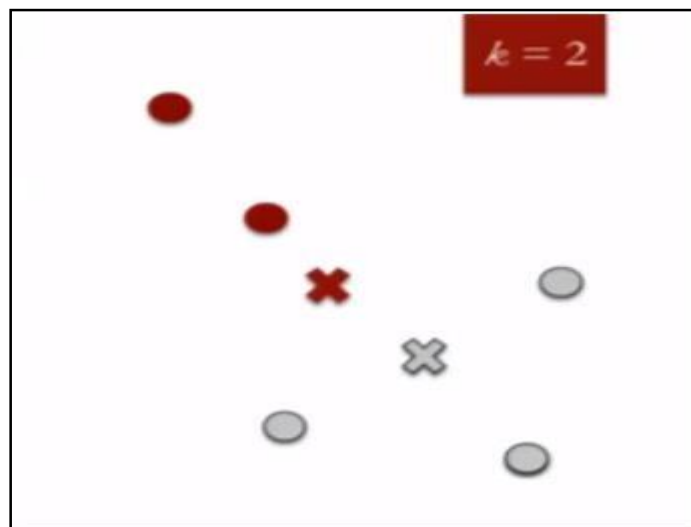
Step 1: $N = 5$, $K = 2$

- Now, each of the **N** data points are assigned to closest centroid in the data set and merged with that centroid as a single cluster. In this way, every data point is assigned to one of the centroids.



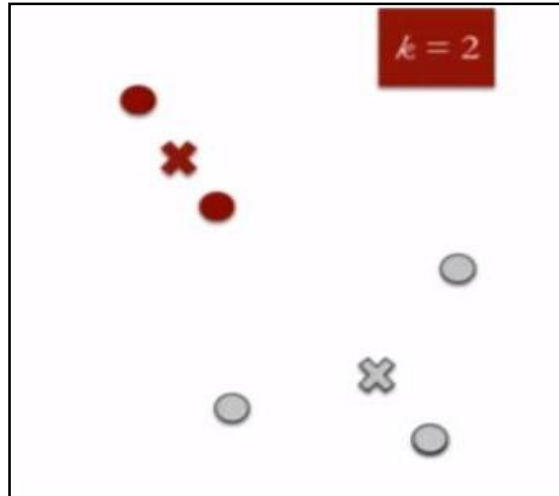
Step 2: Calculating the centroid of the 2 clusters

- Then, K cluster centroids are recalculated and again, each of the N data points are assigned to the nearest centroid.



Step 3: Assigning all the data points to the nearest cluster centroid

- Step 3 is repeated until no further improvement can be made.



Step 4: Recalculating the cluster centroid. After this step, no more improvement can be made.

In this process, a loop is generated.

- *K centroids change their location step by step until no more change is possible.*

This algorithm aims at minimising the **objective function**:

$$J = \sum_{j=1}^k \sum_{i=1}^n \|x_i^{(j)} - c_j\|^2$$

It represent the sum of **euclidean distance** of all the data points from the cluster centroid which is minimised.

How to initialize Kcentroids?

1. **Forgy:** Randomly assigning K centroid points in our data set.
2. **Random Partition:** Assigning each data point to a cluster randomly, and then proceeding to evaluation of centroid positions of each cluster.
3. **KMeans++:** Used for **small** data sets.
4. **Canopy Clustering:** Unsupervised pre-clustering algorithm used as preprocessing step for K-Means or any Hierarchical Clustering. It helps in speeding up clustering operations on **large data sets**.

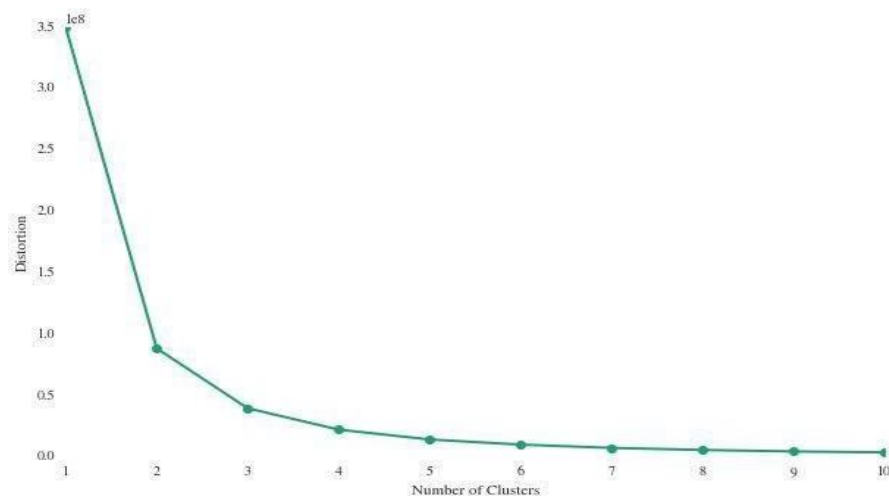
How to calculate centroid of a cluster?

Simply the mean of all the data points within that cluster.

How to find value of K for the dataset?

In K-Means Clustering, value of **K** has to be specified beforehand. It can be determined by any of the following methods:

- **Elbow Method:** Clustering is done on a dataset for varying values of and **SSE (Sum of squared errors)** is calculated for each value of **K**.
- Then, a graph between **K** and SSE is plotted. Plot formed assumes the shape of an arm. There is a point on the graph where SSE does not decrease significantly with increasing **K**.
- This is represented by elbow of the arm and is chosen as the value of **K**. (OPTIMUM)



K can be 3 or 4.

K-Means v/s Hierarchical

1. For **big data**, **K-Means** is better!
Time complexity of K-Means is linear, while that of hierarchical clustering is quadratic.
2. Results are reproducible in **Hierarchical**, and not in K-Means, as they depend on initialization of centroids.
3. K-Means requires prior and proper knowledge about the data set for specifying **K**.
In **Hierarchical**, we can choose no. of clusters by interpreting dendrogram.

```
# Kmeans Clustering implementation using python.  
# Hardcoded Dataset is declared of 19 values
```

```
# In[1]:  
from copy import deepcopy  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt
```

```
# In[2]:  
# Data set is given below  
df = pd.DataFrame({  
    'x': [12, 20, 28, 18, 29, 33, 24, 45, 45, 52, 51, 52, 55, 53, 55, 61, 64, 69, 72],  
    'y': [39, 36, 30, 52, 54, 46, 55, 59, 63, 70, 66, 63, 58, 23, 14, 8, 19, 7, 24]  
})
```

```
# generate random numbers  
np.random.seed(200)  
k = 3
```

```
#centroids[i] = [x, y]  
# Return random integers  
centroids = {  
    i+1:[np.random.randint(0, 80), np.random.randint(0, 80)]  
    for i in range(k)  
}  
print("Centroid=",centroids)
```

```
# In[3]:  
plt.scatter(df['x'], df['y'], color='BLACK')  
colmap = {1: 'r', 2: 'g', 3: 'b'}  
  
for i in centroids.keys():  
    plt.scatter(*centroids[i], color=colmap[i]) #represent color centroid ..  
# .keys() returns a view  
object that displays a list of all the keys.  
plt.xlim(0, 80)  
plt.ylim(0, 80)  
plt.show()
```

```
# In[4]: Repeat Assignment Stage  
def assignment(df, centroids):  
    for i in centroids.keys(): # start from 1 to 3 everytime  
        #  $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$   
        df['distance_from_{}'.format(i)] =  
            ( np.sqrt( (df['x'] - centroids[i][0]) ** 2 +  
                        (df['y'] - centroids[i][1]) ** 2 ) )  
    centroid_distance_cols = ['distance_from_{}'.format(i) for i in  
centroids.keys()]  
#idmin) finding the nearest data in the dataframe
```



```

df['closest'] = df.loc[:, centroid_distance_cols].idxmin(axis=1)
df['closest'] = df['closest'].map(lambda x: int(x.lstrip('distance_from_')))
df['color'] = df['closest'].map(lambda x: colormap[x])

return df
df = assignment(df, centroids)
print(df)

```

In[5]:

```

plt.scatter(df['x'], df['y'], color=df['color'], alpha=0.5, edgecolor='k')
for i in centroids.keys():
    plt.scatter(*centroids[i], color=colormap[i])
plt.xlim(0, 80)
plt.ylim(0, 80)
plt.show()

```

In[6]:

```

import copy
old_centroids = copy.deepcopy(centroids) # create bindings between a target
and an object.

```

```

def update(k):
    for i in centroids.keys():
        centroids[i][0] = np.mean(df[df['closest'] == i]['x'])
        centroids[i][1] = np.mean(df[df['closest'] == i]['y'])
    return k

```

```

centroids = update(centroids)
print("Updtaed centroids ", centroids)

```

In[7]:

```

plt.scatter(df['x'], df['y'], color=df['color'], alpha=0.3) # alpha value for
intensity
for i in centroids.keys():
    plt.scatter(*centroids[i], color=colormap[i])
plt.xlim(0, 80)
plt.ylim(0, 80)
plt.show()

```

In[8]: Continue till assigned clusters are not changed

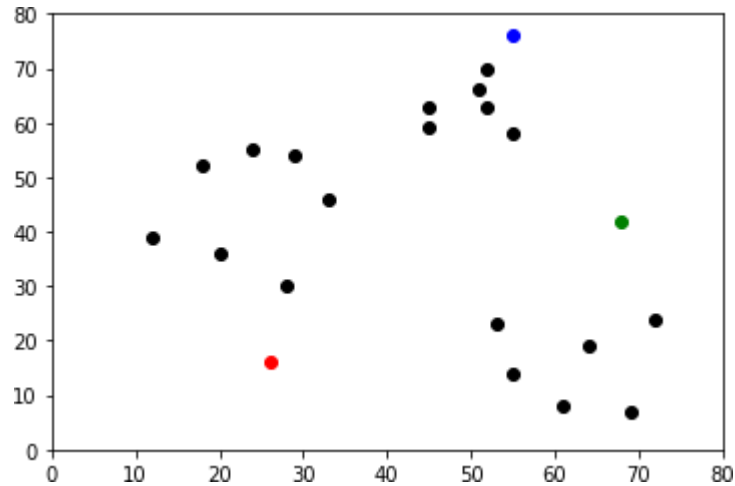
```

while True:
    closest_centroids = df['closest'].copy(deep=True)
    centroids = update(centroids)
    df = assignment(df, centroids)
    if closest_centroids.equals(df['closest']):
        break

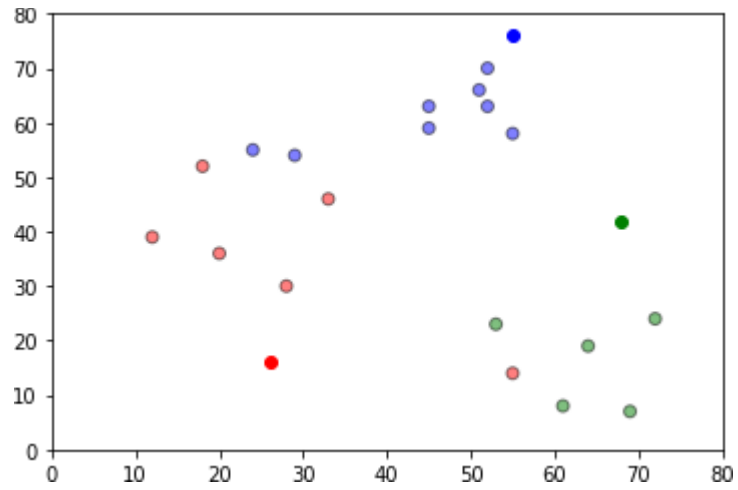
```

```
plt.scatter(df['x'], df['y'], color=df['color'], alpha=0.5)
for i in centroids.keys():
    plt.scatter(*centroids[i], color=colmap[i])
plt.xlim(0, 80)
plt.ylim(0, 80)
plt.show()
```

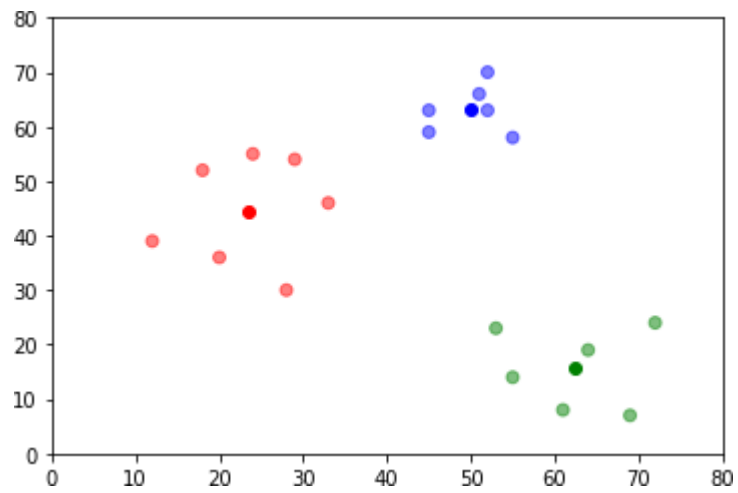
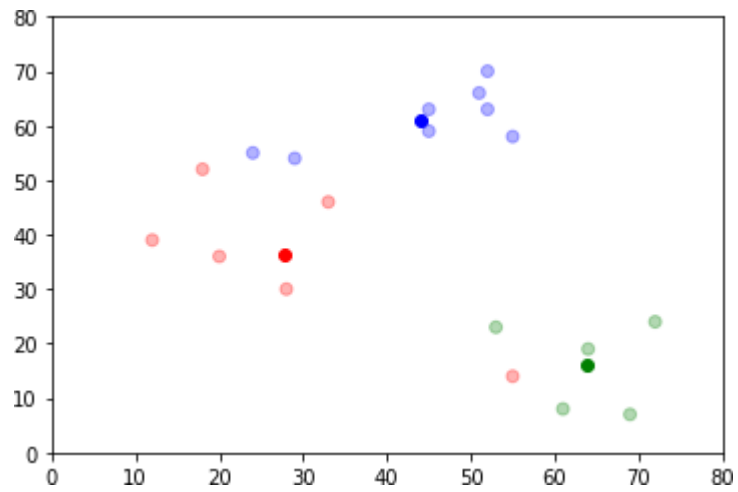
Centroid= {1: [26, 16], 2: [68, 42], 3: [55, 76]}



	x	y	distance_from_1	distance_from_2	distance_from_3	closest	color
0	12	39	26.925824	56.080300	56.727418	1	r
1	20	36	20.880613	48.373546	53.150729	1	r
2	28	30	14.142136	41.761226	53.338541	1	r
3	18	52	36.878178	50.990195	44.102154	1	r
4	29	54	38.118237	40.804412	34.058773	3	b
5	33	46	30.805844	35.227830	37.202150	1	r
6	24	55	39.051248	45.880279	37.443290	3	b
7	45	59	47.010637	28.600699	19.723083	3	b
8	45	63	50.695167	31.144823	16.401219	3	b
9	52	70	59.933296	32.249031	6.708204	3	b
10	51	66	55.901699	29.410882	10.770330	3	b
11	52	63	53.712196	26.400758	13.341664	3	b
12	55	58	51.039201	20.615528	18.000000	3	b
13	53	23	27.892651	24.207437	53.037722	2	g
14	55	14	29.068884	30.870698	62.000000	1	r
15	61	8	35.902646	34.713110	68.264193	2	g
16	64	19	38.118237	23.345235	57.706152	2	g
17	69	7	43.931765	35.014283	70.405966	2	g
18	72	24	46.690470	18.439089	54.708317	2	g



Updtaed centroids {1: [27.666666666666668, 36.166666666666664], 2: [63.8, 16.2], 3: [44.125, 61.0]}



Some things to take note of clustering as follows

- k-means clustering is very sensitive to scale due to its reliance on Euclidean distance so be sure to normalize data if there are likely to be scaling problems.
- If there are some symmetries in your data, some of the labels may be mis-labelled
- It is recommended to do the same k-means with different initial centroids and take the most common label.

Reference

[1] <http://benalexkeen.com/k-means-clustering-in-python/>

[2] <https://www.analyticsvidhya.com>

MLA Lab 5

Aim: Implement SVM for performing classification and find its accuracy on the given data. (Using Python) (Use Social_Network_Ads Dataset)

Support Vector Machines with Scikit-learn

In this experiment, you'll learn about Support Vector Machines, one of the most popular and widely used supervised machine learning algorithms.

SVM offers very high accuracy compared to other classifiers such as logistic regression, and decision trees. It is known for its kernel trick to handle nonlinear input spaces. It is used in a variety of applications such as face detection, intrusion detection, classification of emails, news articles and web pages, classification of genes, and handwriting recognition. In this experiment, you will be using scikit-learn in Python.

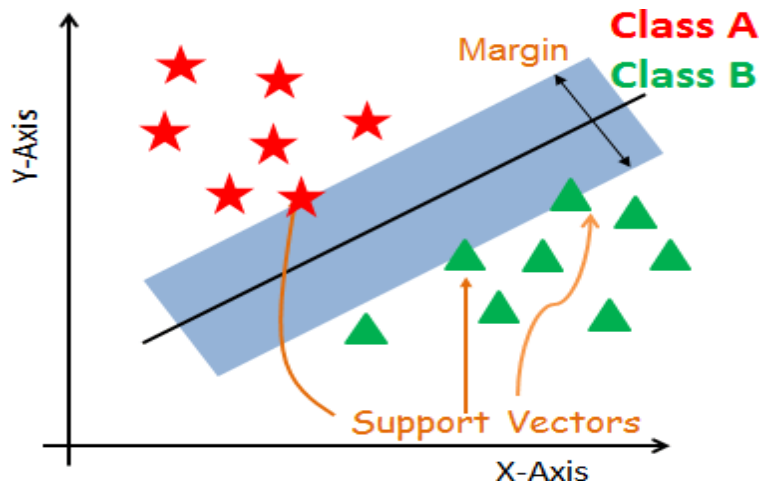
SVM is an exciting algorithm and the concepts are relatively simple. The classifier separates data points using a hyperplane with the largest amount of margin. That's why an SVM classifier is also known as a discriminative classifier. SVM finds an optimal hyperplane which helps in classifying new data points.

In this experiment, you are going to cover following topics:

- Support Vector Machines
- SVM Process
- Kernels
- Classifier building in Scikit-learn

Support Vector Machines

Generally, Support Vector Machines is considered to be a classification approach, it but can be employed in both types of classification and regression problems. It can easily handle multiple continuous and categorical variables. SVM constructs a hyperplane in multidimensional space to separate different classes. SVM generates optimal hyperplane in an iterative manner, which is used to minimize an error. The core idea of SVM is to find a maximum marginal hyperplane(MMH) that best divides the dataset into classes.



Support Vectors

Support vectors are the data points, which are closest to the hyperplane. These points will define the separating line better by calculating margins. These points are more relevant to the construction of the classifier.

Hyperplane

A hyperplane is a decision plane which separates between a set of objects having different class memberships.

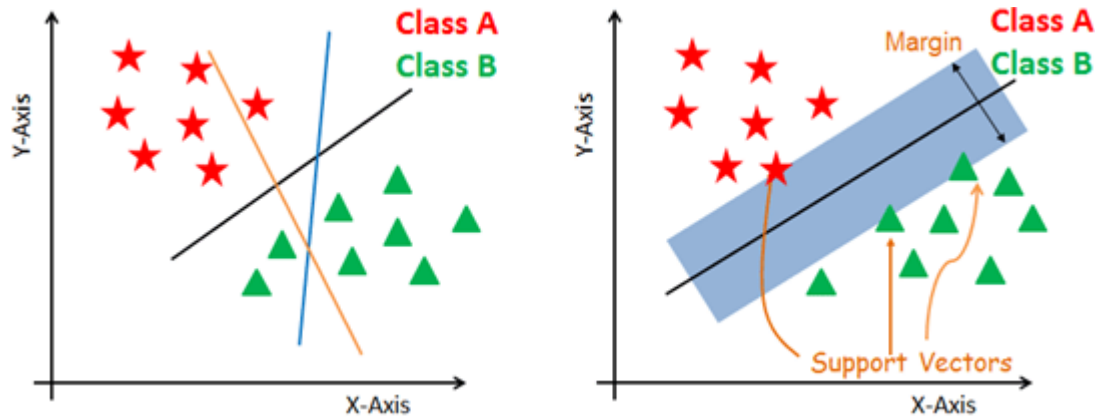
Margin

A margin is a gap between the two lines on the closest class points. This is calculated as the perpendicular distance from the line to support vectors or closest points. If the margin is larger in between the classes, then it is considered a good margin, a smaller margin is a bad margin.

SVM Process

The main objective is to segregate the given dataset in the best possible way. The distance between the either nearest points is known as the margin. The objective is to select a hyperplane with the maximum possible margin between support vectors in the given dataset. SVM searches for the maximum marginal hyperplane in the following steps:

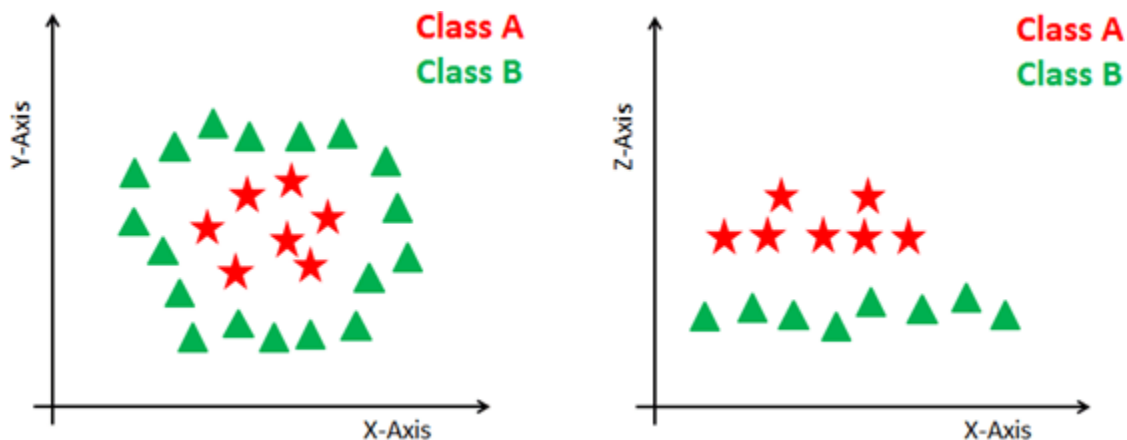
1. Generate hyperplanes which segregates the classes in the best way. Left-hand side figure showing three hyperplanes black, blue and orange. Here, the blue and orange have higher classification error, but the black is separating the two classes correctly.
2. Select the right hyperplane with the maximum segregation from the either nearest data points as shown in the right-hand side figure.



Dealing with non-linear and inseparable planes

Some problems can't be solved using linear hyperplane, as shown in the figure below (left-hand side).

In such situation, SVM uses a kernel trick to transform the input space to a higher dimensional space as shown on the right. The data points are plotted on the x-axis and z-axis (Z is the squared sum of both x and y: $z=x^2+y^2$). Now you can easily segregate these points using linear separation.



SVM Kernels

The SVM algorithm is implemented in practice using a kernel. A kernel transforms an input data space into the required form. SVM uses a technique called the kernel trick. Here, the kernel takes a low-dimensional input space and transforms it into a higher dimensional space. In other words, you can say that it converts nonseparable problem to separable problems by adding more dimension to it. It is most useful in non-linear separation problem. Kernel trick helps you to build a more accurate classifier.

- **Linear Kernel** A linear kernel can be used as normal dot product any two given observations. The product between two vectors is the sum of the multiplication of each pair of input values.

$$K(x, x_i) = \text{sum}(x * x_i)$$

- **Polynomial Kernel** A polynomial kernel is a more generalized form of the linear kernel. The polynomial kernel can distinguish curved or nonlinear input space.

$$K(x, x_i) = 1 + \text{sum}(x * x_i)^d$$

Where d is the degree of the polynomial. d=1 is similar to the linear transformation. The degree needs to be manually specified in the learning algorithm.

The most applicable machine learning algorithm for our problem is Linear SVC. Before hopping into Linear SVC with our data, we're going to show a very simple example that should help solidify your understanding of working with Linear SVC.

The objective of a Linear SVC (Support Vector Classifier) is to fit to the data you provide, returning a "best fit" hyperplane that divides, or categorizes, your data. From there, after getting the hyperplane, you can then feed some features to your classifier to see what the "predicted" class is. This makes this specific algorithm rather suitable for our uses, though you can use this for many situations. Let's get started.

First, we're going to need some basic dependencies:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import style
style.use("ggplot")
from sklearn import svm
```

Matplotlib here is not truly necessary for Linear SVC. The reason why we're using it here is for the eventual data visualization. Typically, you won't be able to visualize as many dimensions as you will have features, but, it's worth visualizing at least once to understand how linear svc works.

Other than the visualization packages we're using, you will just need to import svm from sklearn and numpy for array conversion.

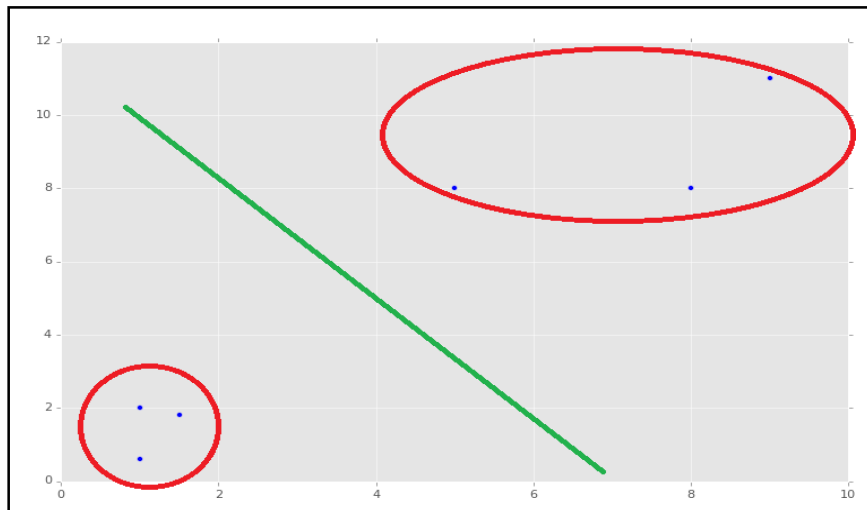
Next, let's consider that we have two features to consider. These features will be visualized as axis on our graph. So something like:

```
x = [1, 5, 1.5, 8, 1, 9]
y = [2, 8, 1.8, 8, 0.6, 11]
```

Then we can graph this data using:


```
plt.scatter(x,y)
plt.show()
```

Now, of course, we can see with our own eyes how these groups should be divided, though exactly where we might draw the dividing line might be debated:



So this is with two features, and we see we have a 2D graph. If we had three features, we could have a 3D graph. The 3D graph would be a little more challenging for us to visually group and divide, but still do-able. The problem occurs when we have four features, or four-thousand features. Now you can start to understand the power of machine learning, seeing and analyzing a number of dimensions imperceptible to us.

With that in mind, we're going to go ahead and continue with our two-featured example. Now, in order to feed data into our machine learning algorithm, we first need to compile an array of the features, rather than having them as x and y coordinate values.

Generally, you will see the feature list being stored in a capital X variable. Let's translate our above x and y coordinates into an array that is compiled of the x and y coordinates, where x is a feature and y is a feature.

```
X = np.array([[1,2],
              [5,8],
              [1.5,1.8],
              [8,8],
              [1,0.6],
              [9,11]])
```

Now that we have this array, we need to label it for training purposes. There are forms of machine learning called "unsupervised learning," where data labeling isn't used, as is the case with clustering, though this example is a form of supervised learning.

For our labels, sometimes referred to as "targets," we're going to use 0 or 1.

```
y = [0, 1, 0, 1, 0, 1]
```

Just by looking at our data set, we can see we have coordinate pairs that are "low" numbers and coordinate pairs that are "higher" numbers. We've then assigned 0 to the lower coordinate pairs and 1 to the higher feature pairs.

These are the labels. In the case of our project, we will wind up having a list of numerical features that are various statistics about stock companies, and then the "label" will be either a 0 or a 1, where 0 is under-perform the market and a 1 is out-perform the market.

Moving along, we are now going to define our classifier:

```
clf = svm.SVC(kernel='linear', C = 1.0)
```

We're going to be using the SVC (support vector classifier) SVM (support vector machine). Our kernel is going to be linear, and C is equal to 1.0. What is C you ask? Don't worry about it for now, but, if you must know, C is a valuation of "how badly" you want to properly classify, or fit, everything. The machine learning field is relatively new, and experimental. There exist many debates about the value of C, as well as how to calculate the value for C. We're going to just stick with 1.0 for now, which is a nice default parameter.

Next, we call:

```
clf.fit(X, y)
```

Note: this is an older tutorial, and Scikit-Learn has since deprecated this method. By version 0.19, this code will cause an error because it needs to be a numpy array, and re-shaped. From here, the learning is done. It should be nearly-instant, since we have such a small data set.

Next, we can predict and test. Let's print a prediction:

```
print(clf.predict([0.58, 0.76]))
```

We're hoping this predicts a 0, since this is a "lower" coordinate pair. Sure enough, the prediction is a classification of 0.

Next, what if we do:

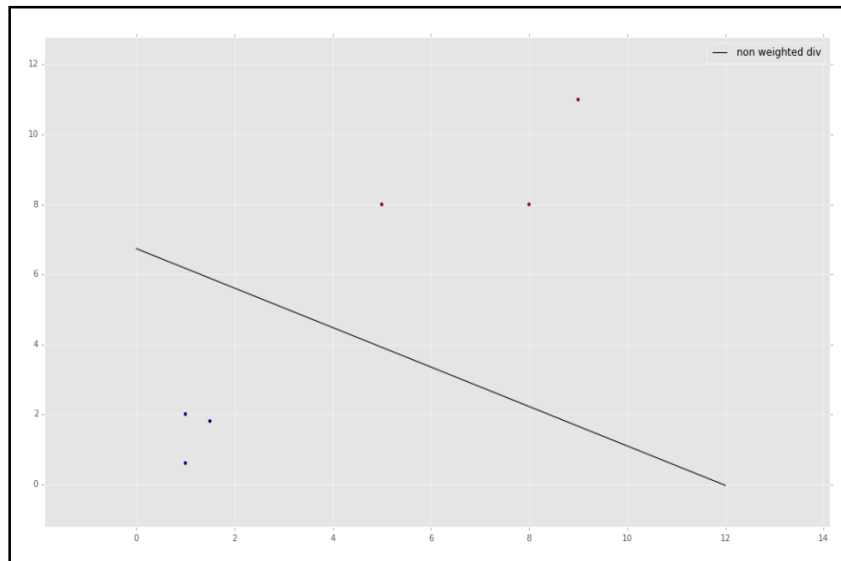
```
print(clf.predict([10.58, 10.76]))
```

And again, we have a theoretically correct answer of 1 as the classification. This was a blind prediction, though it was really a test as well, since we knew what the hopeful target was. Congratulations, you have 100% accuracy.

Now, to visualize your data:

```
w = clf.coef_[0]
print(w)
a = -w[0] / w[1]
xx = np.linspace(0,12)
yy = a * xx - clf.intercept_[0] / w[1]
h0 = plt.plot(xx, yy, 'k-', label="non weighted div")
plt.scatter(X[:, 0], X[:, 1], c = y)
plt.legend()
plt.show()
```

The result:



Visualizing the data is somewhat useful to see what the program is doing in the background, but is not really necessary to understand how to visualize it specifically at this point. You will likely find that the problems you are trying to solve simply cannot be visualized due to having too many features and thus too many dimensions to graph.

Code: Simple SVM using simple data set using python

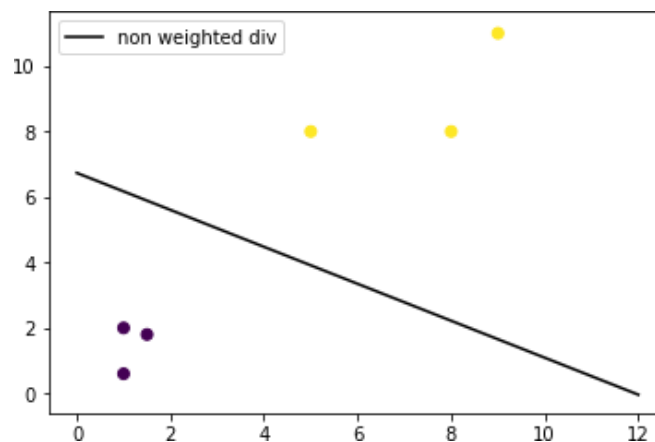
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm

#Hard code simple data set
#x = [1, 5, 1.5, 8, 1, 9]
#y = [2, 8, 1.8, 8, 0.6, 11]
#plt.scatter(x,y)
#plt.show()

X = np.array([[1,2],[5,8],[1.5,1.8],[8,8],[1,0.6],[9,11]])
y = [0,1,0,1,0,1]

clf = svm.SVC(kernel='linear', C = 1.0)
clf.fit(X,y)
w = clf.coef_[0]
print(w)
a = -w[0] / w[1]
xx = np.linspace(0,12)
yy = a * xx - clf.intercept_[0] / w[1]
h0 = plt.plot(xx, yy, 'k-', label="Non weighted div")
plt.scatter(X[:, 0], X[:, 1], c = y)
plt.legend()
plt.show()

print("Prediction of target for 0.58,0.76 values:")
t= clf.predict([[0.58,0.76]])
print(t)
print("Prediction of target for given value")
t= clf.predict([[10.58,10.76]])
print(t)
```



Prediction of target for 0.58,0.76 values: [0]

Prediction of target for 10.58,10.76 values: [1]

Class and Subject: BEIT Information Technology (2015)
Subject: Computing Laboratory VII (Part B)

MLA Lab 6 Artificial Neural Network for Machine Learning

Aim: Creating & Visualizing Neural Network for the given data. (Using Python)
(Use Churn_Modelling Dataset)

Introduction to neural networks (Intuition)

Ref: <https://www.analyticsvidhya.com> by Sunil Ray

If you have been a developer or seen one work – you know how it is to search for bugs in a code. You would fire various test cases by varying the inputs or circumstances and look for the output. The change in output provides you a hint on where to look for the bug – which module to check, which lines to read. Once you find it, you make the changes and the exercise continues until you have the right code / application.

Neural networks work in very similar manner. It takes several input, processes it through multiple neurons from multiple hidden layers and returns the result using an output layer. This result estimation process is technically known as “**Forward Propagation**”.

Next, we compare the result with actual output. The task is to make the output to neural network as close to actual (desired) output. Each of these neurons are contributing some error to final output. How do you reduce the error?

We try to minimize the value/ weight of neurons those are contributing more to the error and this happens while traveling back to the neurons of the neural network and finding where the error lies. This process is known as “**Backward Propagation**”.

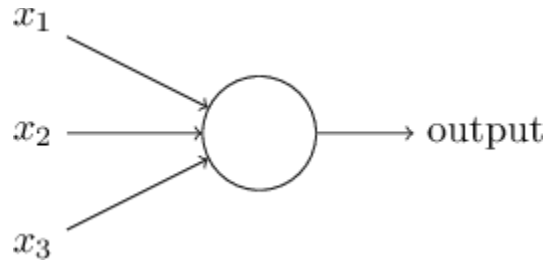
In order to reduce these number of iterations to minimize the error, the neural networks use a common algorithm known as “Gradient Descent”, which helps to optimize the task quickly and efficiently.

That’s it – this is how Neural network works! I know this is a very simple representation, but it would help you understand things in a simple manner.

Multi Layer Perceptron and its basics

Just like atoms form the basics of any material on earth – the basic forming unit of a neural network is a perceptron. So, what is a perceptron?

A perceptron can be understood as anything that takes multiple inputs and produces one output. For example, look at the image below.



The above structure takes three inputs and produces one output. The next logical question is what is the relationship between input and output? Let us start with basic ways and build on to find more complex ways.

Below, I have discussed three ways of creating input output relationships:

1. **By directly combining the input and computing the output** based on a threshold value. for eg: Take $x_1=0$, $x_2=1$, $x_3=1$ and setting a threshold $=0$. So, if $x_1+x_2+x_3>0$, the output is 1 otherwise 0. You can see that in this case, the perceptron calculates the output as 1.
2. **Next, let us add weights to the inputs.** Weights give importance to an input. For example, you assign $w_1=2$, $w_2=3$ and $w_3=4$ to x_1 , x_2 and x_3 respectively. To compute the output, we will multiply input with respective weights and compare with threshold value as $w_1*x_1 + w_2*x_2 + w_3*x_3 > \text{threshold}$. These weights assign more importance to x_3 in comparison to x_1 and x_2 .
3. **Next, let us add bias:** Each perceptron also has a bias which can be thought of as how much flexible the perceptron is. It is somehow similar to the constant b of a linear function $y = ax + b$. *It allows us to move the line up and down to fit the prediction with the data better. Without b the line will always goes through the origin $(0, 0)$ and you may get a poorer fit.* For example, a perceptron may have two inputs, in that case, it requires three weights. One for each input and one for the bias. Now linear representation of input will look like, $w_1*x_1 + w_2*x_2 + w_3*x_3 + 1*b$.

But, all of this is still linear which is what perceptrons used to be. But that was not as much fun. So, people thought of evolving a perceptron to what is now called as artificial neuron. A neuron applies non-linear transformations (activation function) to the inputs and biases.

Activation function

Activation Function takes the sum of weighted input ($w_1*x_1 + w_2*x_2 + w_3*x_3 + 1*b$) as an argument and return the output of the neuron.

$$a = f\left(\sum_{i=0}^N w_i x_i\right)$$

In above equation, we have represented 1 as x_0 and b as w_0 .

The activation function is mostly used to make a non-linear transformation which allows us to fit nonlinear hypotheses or to estimate the complex functions. There are multiple activation functions, like: “Sigmoid”, “Tanh”, ReLu and many other.

Forward Propagation, Back Propagation and Epochs

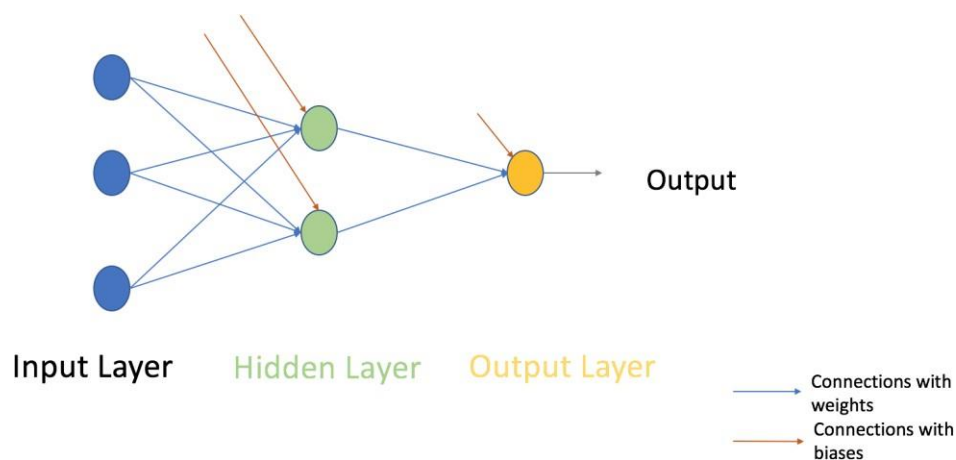
Till now, we have computed the output and this process is known as “**Forward Propagation**“. But what if the estimated output is far away from the actual output (high error). In the neural network what we do, we update the biases and weights based on the error. This weight and bias updating process is known as “**Back Propagation**“.

Back-propagation (BP) algorithms work by determining the loss (or error) at the output and then propagating it back into the network. The weights are updated to minimize the error resulting from each neuron. The first step in minimizing the error is to determine the gradient (Derivatives) of each node w.r.t. the final output. To get a mathematical perspective of the Backward propagation, refer below section.

This one round of forward and back propagation iteration is known as one training iteration aka “**Epoch**“.

Multi-layer perceptron

Now, let’s move on to next part of **Multi-Layer** Perceptron. So far, we have seen just a single layer consisting of 3 input nodes i.e x_1 , x_2 and x_3 and an output layer consisting of a single neuron. But, for practical purposes, the single-layer network can do only so much. An MLP consists of multiple layers called **Hidden Layers** stacked in between the **Input Layer** and the **Output Layer** as shown below.



The image above shows just a single hidden layer in green but in practice can contain multiple hidden layers. Another point to remember in case of an MLP is that all the layers are fully

connected i.e every node in a layer(except the input and the output layer) is connected to every node in the previous layer and the following layer.

Let's move on to the next topic which is training algorithm for a neural network (to minimize the error). Here, we will look at most common training algorithm known as Gradient descent.

Full Batch Gradient Descent and Stochastic Gradient Descent

Both variants of Gradient Descent perform the same work of updating the weights of the MLP by using the same updating algorithm but the difference lies in the number of training samples used to update the weights and biases.

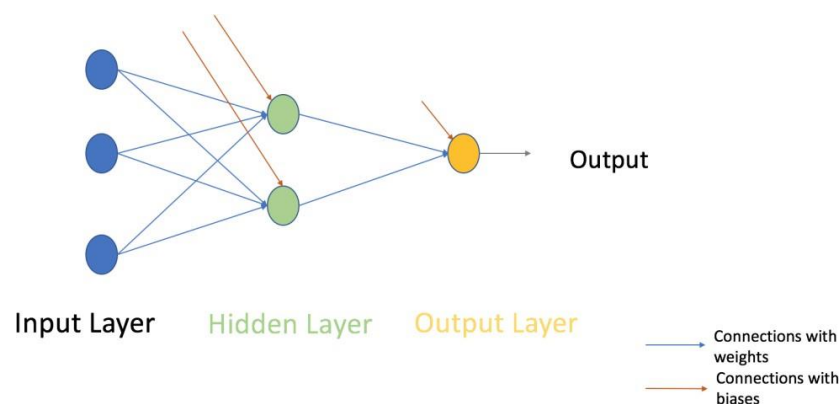
Full Batch Gradient Descent Algorithm as the name implies uses all the training data points to update each of the weights once whereas Stochastic Gradient uses 1 or more(sample) but never the entire training data to update the weights once.

Let us understand this with a simple example of a dataset of 10 data points with two weights **w1** and **w2**.

Full Batch: You use 10 data points (entire training data) and calculate the change in $w1$ ($\Delta w1$) and change in $w2$ ($\Delta w2$) and update $w1$ and $w2$.

SGD: You use 1st data point and calculate the change in $w1$ ($\Delta w1$) and change in $w2$ ($\Delta w2$) and update $w1$ and $w2$. Next, when you use 2nd data point, you will work on the updated weights

Steps involved in Neural Network methodology



Let's look at the step by step building methodology of Neural Network (MLP with one hidden layer, similar to above-shown architecture). At the output layer, we have only one neuron as we are solving a binary classification problem (predict 0 or 1). We could also have two neurons for predicting each of both classes.

First look at the broad steps:

0.) We take input and output

- X as an input matrix
- y as an output matrix

1.) We initialize weights and biases with random values (This is one time initiation. In the next iteration, we will use updated weights, and biases). Let us define:

- wh as weight matrix to the hidden layer
- bh as bias matrix to the hidden layer
- wout as weight matrix to the output layer
- bout as bias matrix to the output layer

2.) We take matrix dot product of input and weights assigned to edges between the input and hidden layer then add biases of the hidden layer neurons to respective inputs, this is known as linear transformation:

$$\text{hidden_layer_input} = \text{matrix_dot_product}(X, wh) + bh$$

3) Perform non-linear transformation using an activation function (Sigmoid). Sigmoid will return the output as $1/(1 + \exp(-x))$.

$$\text{hiddenlayer_activations} = \text{sigmoid}(\text{hidden_layer_input})$$

4.) Perform a linear transformation on hidden layer activation (take matrix dot product with weights and add a bias of the output layer neuron) then apply an activation function (again used sigmoid, but you can use any other activation function depending upon your task) to predict the output

$$\begin{aligned}\text{output_layer_input} &= \text{matrix_dot_product}(\text{hiddenlayer_activations} * wout) + bout \\ \text{output} &= \text{sigmoid}(\text{output_layer_input})\end{aligned}$$

All above steps are known as “**Forward Propagation**”

5.) Compare prediction with actual output and calculate the gradient of error (Actual – Predicted). Error is the mean square loss = $((Y-t)^2)/2$

$$E = y - \text{output}$$

6.) Compute the slope/ gradient of hidden and output layer neurons (To compute the slope, we calculate the derivatives of non-linear activations x at each layer for each neuron). Gradient of sigmoid can be returned as $x * (1 - x)$.

$\text{slope_output_layer} = \text{derivatives_sigmoid}(\text{output})$
 $\text{slope_hidden_layer} = \text{derivatives_sigmoid}(\text{hiddenlayer_activations})$

7.) Compute change factor(delta) at output layer, dependent on the gradient of error multiplied by the slope of output layer activation

$$d_output = E * \text{slope_output_layer}$$

8.) At this step, the error will propagate back into the network which means error at hidden layer. For this, we will take the dot product of output layer delta with weight parameters of edges between the hidden and output layer (wout.T).

$$\text{Error_at_hidden_layer} = \text{matrix_dot_product}(d_output, \text{wout.T})$$

9.) Compute change factor(delta) at hidden layer, multiply the error at hidden layer with slope of hidden layer activation

$$d_hiddenlayer = \text{Error_at_hidden_layer} * \text{slope_hidden_layer}$$

10.) Update weights at the output and hidden layer: The weights in the network can be updated from the errors calculated for training example(s).

$$\begin{aligned} \text{wout} &= \text{wout} + \text{matrix_dot_product}(\text{hiddenlayer_activations.T}, d_output) * \text{learning_rate} \\ \text{wh} &= \text{wh} + \text{matrix_dot_product}(X.T, d_hiddenlayer) * \text{learning_rate} \end{aligned}$$

learning_rate: The amount that weights are updated is controlled by a configuration parameter called the learning rate)

11.) Update biases at the output and hidden layer: The biases in the network can be updated from the aggregated errors at that neuron.

- bias at output_layer = bias at output_layer + sum of delta of output_layer at row-wise * learning_rate
- bias at hidden_layer = bias at hidden_layer + sum of delta of output_layer at row-wise * learning_rate

$$\begin{aligned} bh &= bh + \text{sum}(d_hiddenlayer, \text{axis}=0) * \text{learning_rate} \\ bout &= bout + \text{sum}(d_output, \text{axis}=0) * \text{learning_rate} \end{aligned}$$

Steps from 5 to 11 are known as “Backward Propagation”

One forward and backward propagation iteration is considered as one training cycle. As I mentioned earlier, When do we train second time then update weights and biases are used for forward propagation.

Above, we have updated the weight and biases for hidden and output layer and we have used full batch gradient descent algorithm.

Visualization of steps for Neural Network methodology

We will repeat the above steps and visualize the input, weights, biases, output, error matrix to understand working methodology of Neural Network (MLP).

Note:

- For good visualization images, I have rounded decimal positions at 2 or 3 positions.
- Yellow filled cells represent current active cell
- Orange cell represents the input used to populate values of current cell

Step 0: Read input and output

X				wh			bh			hidden_layer_input			hidden_layer_activations			wout	bout	output	y	E
1	0	1	0																1	
1	0	1	1																1	
0	1	0	1																0	

Step 1: Initialize weights and biases with random values (There are methods to initialize weights and biases but for now initialize with random values)

X				wh			bh			hidden_layer_input			hidden_layer_activations			wout	bout	output	y	E
1	0	1	0	0.42	0.88	0.55	0.46	0.72	0.08							0.30	0.69		1	
1	0	1	1	0.10	0.73	0.68										0.25			1	
0	1	0	1	0.60	0.18	0.47										0.23			0	
				0.92	0.11	0.52														

Step 2: Calculate hidden layer input:

$\text{hidden_layer_input} = \text{matrix_dot_product}(X, wh) + bh$

X				wh			bh			hidden_layer_input			hidden_layer_activations			wout	bout	output	y	E
1	0	1	0	0.42	0.88	0.55	0.46	0.72	0.08	1.48	1.78	1.10				0.30	0.69		1	
1	0	1	1	0.10	0.73	0.68				2.40	1.89	1.61				0.25			1	
0	1	0	1	0.60	0.18	0.47				1.48	1.56	1.27				0.23			0	
				0.92	0.11	0.52														

Step 3: Perform non-linear transformation on hidden linear input

hiddenlayer_activations = sigmoid(hidden_layer_input)

X				wh			bh			hidden_layer_input			hidden_layer_activations			wout	bout	output	y	E
1	0	1	0	0.42	0.88	0.55	0.46	0.72	0.08	1.48	1.78	1.10	0.81	0.86	0.75	0.30	0.69		1	
1	0	1	1	0.10	0.73	0.68				2.40	1.89	1.61	0.92	0.87	0.83	0.25			1	
0	1	0	1	0.60	0.18	0.47				1.48	1.56	1.27	0.81	0.83	0.78	0.23			0	
				0.92	0.11	0.52														

Step 4: Perform linear and non-linear transformation of hidden layer activation at output layer

*output_layer_input = matrix_dot_product (hiddenlayer_activations * wout) + bout*

output = sigmoid(output_layer_input)

X				wh			bh			hidden_layer_input			hidden_layer_activations			wout	bout	output	y	E
1	0	1	0	0.42	0.88	0.55	0.46	0.72	0.08	1.48	1.78	1.10	0.81	0.86	0.75	0.30	0.69	0.79	1	
1	0	1	1	0.10	0.73	0.68				2.40	1.89	1.61	0.92	0.87	0.83	0.25		0.80	1	
0	1	0	1	0.60	0.18	0.47				1.48	1.56	1.27	0.81	0.83	0.78	0.23		0.79	0	
				0.92	0.11	0.52														

Step 5: Calculate gradient of Error(E) at output layer

E = y-output

X				wh			bh			hidden_layer_input			hidden_layer_activations			wout	bout	output	y	E
1	0	1	0	0.42	0.88	0.55	0.46	0.72	0.08	1.48	1.78	1.10	0.81	0.86	0.75	0.30	0.69	0.79	1	0.21
1	0	1	1	0.10	0.73	0.68				2.40	1.89	1.61	0.92	0.87	0.83	0.25		0.80	1	0.20
0	1	0	1	0.60	0.18	0.47				1.48	1.56	1.27	0.81	0.83	0.78	0.23		0.79	0	-0.79
				0.92	0.11	0.52														

Step 6: Compute slope at output and hidden layer

Slope_output_layer= derivatives_sigmoid(output)

Slope_hidden_layer = derivatives_sigmoid(hiddenlayer_activations)

X				wh			bh			hidden_layer_input			hidden_layer_activations			wout	bout	output	y	E
1	0	1	0	0.42	0.88	0.55	0.46	0.72	0.08	1.48	1.78	1.10	0.81	0.86	0.75	0.30	0.69	0.79	1	0.21
1	0	1	1	0.10	0.73	0.68				2.40	1.89	1.61	0.92	0.87	0.83	0.25		0.80	1	0.20
0	1	0	1	0.60	0.18	0.47				1.48	1.56	1.27	0.81	0.83	0.78	0.23		0.79	0	-0.79
				0.92	0.11	0.52														

Slope hidden layer		
0.15	0.12	0.19
0.08	0.11	0.14
0.15	0.14	0.17

Slope Output
0.17
0.16
0.17

Step 7: Compute delta at output layer

$$d_output = E * slope_output_layer * lr$$

X				wh			bh			hidden_layer_input			hidden_layer_activations			wout	bout	output	y	E
1	0	1	0	0.42	0.88	0.55	0.46	0.72	0.08	1.48	1.78	1.10	0.81	0.86	0.75	0.30	0.69	0.79	1	0.21
1	0	1	1	0.10	0.73	0.68				2.40	1.89	1.61	0.92	0.87	0.83	0.25		0.80	1	0.20
0	1	0	1	0.60	0.18	0.47				1.48	1.56	1.27	0.81	0.83	0.78	0.23		0.79	0	-0.79
				0.92	0.11	0.52														

Slope hidden layer		
0.15	0.12	0.19
0.08	0.11	0.14
0.15	0.14	0.17

Slope Output	E
0.17	0.21
0.16	0.20
0.17	-0.79

delta output
0.04
0.03
-0.13

Step 8: Calculate Error at hidden layer

$\text{Error_at_hidden_layer} = \text{matrix_dot_product}(d_output, \text{wout.Transpose})$

X				wh			bh			hidden_layer_input			hidden_layer_activations			wout	bout	output	y	E
1	0	1	0	0.42	0.88	0.55	0.46	0.72	0.08	1.48	1.78	1.10	0.81	0.86	0.75	0.30	0.69	0.79	1	0.21
1	0	1	1	0.10	0.73	0.68				2.40	1.89	1.61	0.92	0.87	0.83	0.25		0.80	1	0.20
0	1	0	1	0.60	0.18	0.47				1.48	1.56	1.27	0.81	0.83	0.78	0.23		0.79	0	-0.79
				0.92	0.11	0.52														

Slope hidden layer			error at hidden layer		
0.15	0.12	0.19	0.010	0.009	0.008
0.08	0.11	0.14	0.010	0.008	0.008
0.15	0.14	0.17	-0.039	-0.033	-0.031

Slope Output	E
0.17	0.21
0.16	0.20
0.17	-0.79

delta output
0.04
0.03
-0.13

Step 9: Compute delta at hidden layer

$d_hiddenlayer = \text{Error_at_hidden_layer} * \text{slope_hidden_layer}$

X				wh			bh			hidden_layer_input			hidden_layer_activations			wout	bout	output	y	E
1	0	1	0	0.42	0.88	0.55	0.46	0.72	0.08	1.48	1.78	1.10	0.81	0.86	0.75	0.30	0.69	0.79	1	0.21
1	0	1	1	0.10	0.73	0.68				2.40	1.89	1.61	0.92	0.87	0.83	0.25		0.80	1	0.20
0	1	0	1	0.60	0.18	0.47				1.48	1.56	1.27	0.81	0.83	0.78	0.23		0.79	0	-0.79
				0.92	0.11	0.52														

Slope hidden layer			error at hidden layer		
0.15	0.12	0.19	0.010	0.009	0.008
0.08	0.11	0.14	0.010	0.008	0.008
0.15	0.14	0.17	-0.039	-0.033	-0.031

Slope Output	E
0.17	0.21
0.16	0.20
0.17	-0.79

delta hidden layer		
0.002	0.001	0.002
0.001	0.001	0.001
-0.006	-0.005	-0.005

delta output
0.04
0.03
-0.13

Step 10: Update weight at both output and hidden layer

$wout = wout + matrix_dot_product(hiddenlayer_activations.Transpose, d_output) * learning_rate$

$wh = wh + matrix_dot_product(X.Transpose, d_hiddenlayer) * learning_rate$

X				wh			bh			hidden_layer_input			hidden_layer_activations			wout	bout	output	y	E
1	0	1	0	0.42	0.88	0.55	0.46	0.72	0.08	1.48	1.78	1.10	0.81	0.86	0.75	0.29	0.69	0.79	1	0.21
1	0	1	1	0.10	0.73	0.68				2.40	1.89	1.61	0.92	0.87	0.83	0.25		0.80	1	0.20
0	1	0	1	0.60	0.18	0.47				1.48	1.56	1.27	0.81	0.83	0.78	0.23		0.79	0	-0.79
				0.92	0.11	0.51														

Slope hidden layer			error at hidden layer		
0.15	0.12	0.19	0.010	0.009	0.008
0.08	0.11	0.14	0.010	0.008	0.008
0.15	0.14	0.17	-0.039	-0.033	-0.031

Slope Output
0.17
0.16
0.17

E
0.21
0.20
-0.79

Learning Rate	0.1
---------------	-----

delta hidden layer		
0.002	0.001	0.002
0.001	0.001	0.001
-0.006	-0.005	-0.005

delta output
0.035
0.033
-0.131

Step 11: Update biases at both output and hidden layer

$bh = bh + sum(d_hiddenlayer, axis=0) * learning_rate$

$bout = bout + sum(d_output, axis=0) * learning_rate$

X				wh			bh			hidden_layer_input			hidden_layer_activations			wout	bout	output	y	E
1	0	1	0	0.42	0.88	0.55	0.46	0.72	0.08	1.48	1.78	1.10	0.81	0.86	0.75	0.29	0.68	0.79	1	0.21
1	0	1	1	0.10	0.73	0.68				2.40	1.89	1.61	0.92	0.87	0.83	0.25		0.80	1	0.20
0	1	0	1	0.60	0.18	0.47				1.48	1.56	1.27	0.81	0.83	0.78	0.23		0.79	0	-0.79
				0.92	0.11	0.51														

Slope hidden layer			error at hidden layer		
0.15	0.12	0.19	0.010	0.009	0.008
0.08	0.11	0.14	0.010	0.008	0.008
0.15	0.14	0.17	-0.039	-0.033	-0.031

Slope Output
0.17
0.16
0.17

E
0.21
0.20
-0.79

Learning Rate	0.1
---------------	-----

delta hidden layer		
0.002	0.001	0.002
0.001	0.001	0.001
-0.006	-0.005	-0.005

delta output
0.035
0.033
-0.131

Above, you can see that there is still a good error not close to actual target value because we have completed only one training iteration. If we will train model multiple times then it will be a very close actual outcome. I have completed thousands iteration and my result is close to actual target values ([[0.98032096] [0.96845624] [0.04532167]]).

```

#Source code for Artificial Neural Network using python
import numpy as np
#Input array
X=np.array([[1,0,1,0],[1,0,1,1],[0,1,0,1]])
#Output
y=np.array([[1],[1],[0]])
#Sigmoid Function
def sigmoid (x):
    return 1/(1 + np.exp(-x))
#Derivative of Sigmoid Function
def derivatives_sigmoid(x):
    return x * (1 - x)
#Variable initialization
epoch=5000 #Setting training iterations
lr=0.1 #Setting learning rate
inputlayer_neurons = X.shape[1] #number of features in data set
hiddenlayer_neurons = 3 #number of hidden layers neurons
output_neurons = 1 #number of neurons at output layer

#weight and bias initialization
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))

for i in range(epoch):
#Forward Propogation
    hidden_layer_input1=np.dot(X,wh)
    hidden_layer_input=hidden_layer_input1 + bh
    hiddenlayer_activations = sigmoid(hidden_layer_input)
    output_layer_input1=np.dot(hiddenlayer_activations,wout)
    output_layer_input= output_layer_input1+ bout
    output = sigmoid(output_layer_input)

#Backpropagation
    E = y-output
    slope_output_layer = derivatives_sigmoid(output)
    slope_hidden_layer = derivatives_sigmoid(hiddenlayer_activations)
    d_output = E * slope_output_layer
    Error_at_hidden_layer = d_output.dot(wout.T)
    d_hiddenlayer = Error_at_hidden_layer * slope_hidden_layer
    wout += hiddenlayer_activations.T.dot(d_output) *lr
    bout += np.sum(d_output, axis=0,keepdims=True) *lr
    wh += X.T.dot(d_hiddenlayer) *lr
    bh += np.sum(d_hiddenlayer, axis=0,keepdims=True) *lr

    print (output)

```

Ref: <https://www.analyticsvidhya.com> by Sunil Ray

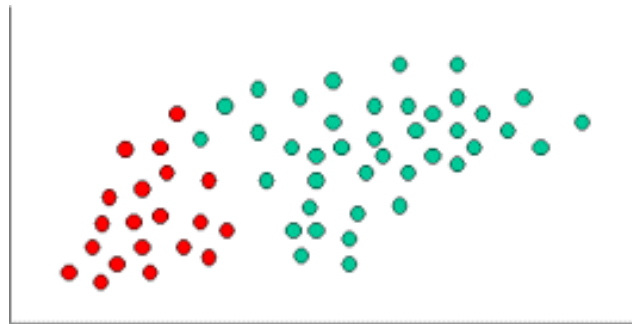
Class and Subject: BEIT Information Technology (2015)
Subject: Computing Laboratory VII (Part B)

MLA Lab 7 Naïve Bayes Classifier

Aim: On the given data perform the performance measurements using Simple Naïve Bayes algorithm such as Accuracy, Error rate, precision, Recall, TPR, FPR, TNR, FPR etc. (Using Weka API through JAVA) (Use weather Dataset)

Naive Bayes Classifier Introductory Overview [Ref: <http://www.statsoft.com>]

The Naive Bayes Classifier technique is based on the so-called Bayesian theorem and is particularly suited when the dimensionality of the inputs is high. Despite its simplicity, Naive Bayes can often outperform more sophisticated classification methods.



To demonstrate the concept of Naïve Bayes Classification, consider the example displayed in the illustration above. As indicated, the objects can be classified as either GREEN or RED. Our task is to classify new cases as they arrive, i.e., decide to which class label they belong, based on the currently existing objects.

Since there are twice as many GREEN objects as RED, it is reasonable to believe that a new case (which hasn't been observed yet) is twice as likely to have membership GREEN rather than RED. In the Bayesian analysis, this belief is known as the prior probability. Prior probabilities are based on previous experience, in this case the percentage of GREEN and RED objects, and often used to predict outcomes before they actually happen.

Thus, we can write:

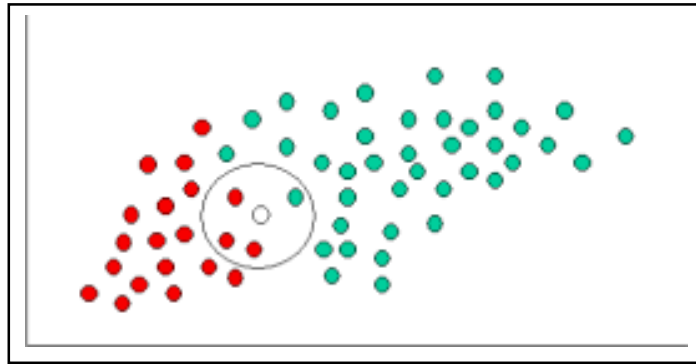
$$\text{Prior probability for GREEN} \propto \frac{\text{Number of GREEN objects}}{\text{Total number of objects}}$$

$$\text{Prior probability for RED} \propto \frac{\text{Number of RED objects}}{\text{Total number of objects}}$$

Since there is a total of 60 objects, 40 of which are GREEN and 20 RED, our prior probabilities for class membership are:

$$\text{Prior probability for GREEN} \propto \frac{40}{60}$$

$$\text{Prior probability for RED} \propto \frac{20}{60}$$



Having formulated our prior probability, we are now ready to classify a new object (WHITE circle). Since the objects are well clustered, it is reasonable to assume that the more GREEN (or RED) objects in the vicinity of X, the more likely that the new cases belong to that particular color. 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. Then we calculate the number of points in the circle belonging to each class label. From this we calculate the likelihood:

$$\text{Likelihood of } X \text{ given GREEN} \propto \frac{\text{Number of GREEN in the vicinity of } X}{\text{Total number of GREEN cases}}$$

$$\text{Likelihood of } X \text{ given RED} \propto \frac{\text{Number of RED in the vicinity of } X}{\text{Total number of RED cases}}$$

From the illustration above, it is clear that Likelihood of X given GREEN is smaller than Likelihood of X given RED, since the circle encompasses 1 GREEN object and 3 RED ones. Thus:

$$\text{Probability of } X \text{ given GREEN} \propto \frac{1}{40}$$

$$\text{Probability of } X \text{ given RED} \propto \frac{3}{20}$$

Although the prior probabilities indicate that X may belong to GREEN (given that there are twice as many GREEN compared to RED) the likelihood indicates otherwise; that the class membership of X is RED (given that there are more RED objects in the vicinity of X than GREEN). In the Bayesian analysis, the final classification is produced by combining both sources of information, i.e., the prior and the likelihood, to form a posterior probability using the so-called Bayes' rule (named after Rev. Thomas Bayes 1702-1761).

$$\begin{aligned}
 &\text{Posterior probability of } X \text{ being GREEN} \propto \\
 &\quad \text{Prior probability of GREEN} \times \text{Likelihood of } X \text{ given GREEN} \\
 &= \frac{4}{6} \times \frac{1}{40} = \frac{1}{60} \\
 &\text{Posterior probability of } X \text{ being RED} \propto \\
 &\quad \text{Prior probability of RED} \times \text{Likelihood of } X \text{ given RED} \\
 &= \frac{2}{6} \times \frac{3}{20} = \frac{1}{20}
 \end{aligned}$$

Finally, we classify X as RED since its class membership achieves the largest posterior probability.

Note. The above probabilities are not normalized. However, this does not affect the classification outcome since their normalizing constants are the same.

Technical Naïve Bayes

In the previous section, we provided an intuitive example for understanding classification using Naive Bayes. In this section are further details of the technical issues involved. Naive Bayes classifiers can handle an arbitrary number of independent variables whether continuous or categorical. Given a set of variables, $X = \{x_1, x_2, \dots, x_d\}$, we want to construct the posterior probability for the event C_j among a set of possible outcomes $C = \{c_1, c_2, \dots, c_d\}$. In a more familiar language, X is the predictors and C is the set of categorical levels present in the dependent variable. Using Bayes' rule:

$$p(C_j | x_1, x_2, \dots, x_d) \propto p(x_1, x_2, \dots, x_d | C_j) p(C_j)$$

where $p(C_j | x_1, x_2, \dots, x_d)$ is the posterior probability of class membership, i.e., the probability that X belongs to C_j . Since Naive Bayes assumes that the conditional probabilities of the independent variables are statistically independent we can decompose the likelihood to a product of terms:

$$p(X | C_j) \propto \prod_{k=1}^d p(x_k | C_j)$$

and rewrite the posterior as:

$$p(C_j | X) \propto p(C_j) \prod_{k=1}^d p(x_k | C_j)$$

Using Bayes' rule above, we label a new case X with a class level Cj that achieves the highest posterior probability.

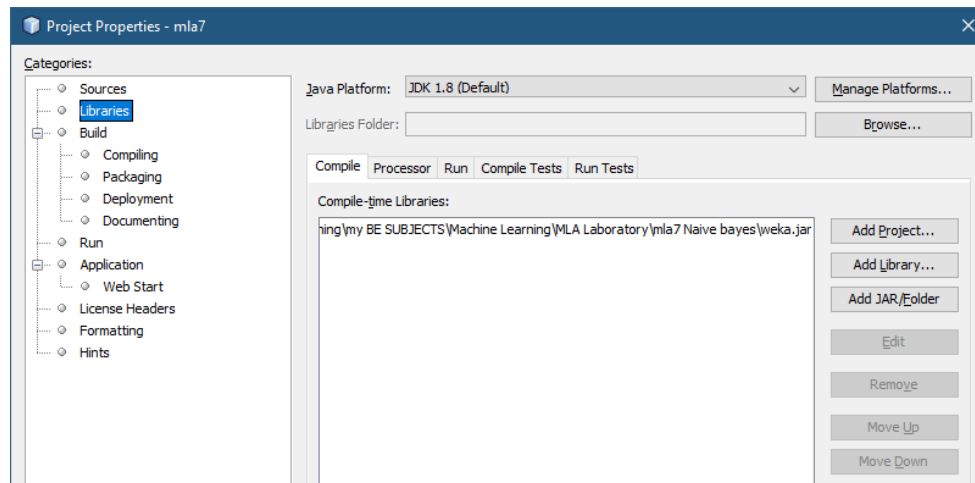
Although the assumption that the predictor (independent) variables are independent is not always accurate, it does simplify the classification task dramatically, since it allows the class conditional densities $p(x_k | C_j)$ to be calculated separately for each variable, i.e., it reduces a multidimensional task to a number of one-dimensional ones. In effect, Naive Bayes reduces a high-dimensional density estimation task to a one-dimensional kernel density estimation. Furthermore, the assumption does not seem to greatly affect the posterior probabilities, especially in regions near decision boundaries, thus, leaving the classification task unaffected.

Java Implementation Steps (Naïve Bayes for Intrusion detection classification)

1. Read the Training dataset (IDSTraining.arff)
 - a. Contains 2 classes: normal, anomaly
2. Read the test dataset (IDSTesting.arff)
 - a. We should predict if an instance belongs to normal or anomaly
3. Read the instances (from IDSTraining and IDSTesting)
4. Prepare the model using NB algorithm
5. Evaluate instances from IDSTest for the model built (i.e. predict instances)
6. Evaluate multiple parameters – precision, recall, accuracy, ...

In netbeans IDE

1. Create new java application project in netbeans or eclipse with name mla7
2. Adding weka jar api in project as shown below



3. Copy following source code in mla7.java

```
package mla7;
import weka.core.Instances;
//import weka.classifiers.*;
import weka.classifiers.Evaluation;
import weka.classifiers.bayes.NaiveBayes;
import weka.classifiers.evaluation.*; //Evaluation;

import java.io.BufferedReader;
import java.io.FileReader;
import java.lang.Exception;

//@author bnjagdale

public class Mla7
{
    public Mla7()
    {
        try
        {
            BufferedReader trainReader =
                new BufferedReader(new FileReader("E:/IDSTraining.arff"));
            //File with text examples
            BufferedReader classifyReader =
                new BufferedReader(new FileReader("E:/IDSTesting.arff"));
            //File with text to classify

            Instances trainInsts = new Instances(trainReader);
            Instances classifyInsts = new Instances(classifyReader);

            trainInsts.setClassIndex(trainInsts.numAttributes() - 1);
            classifyInsts.setClassIndex(classifyInsts.numAttributes() - 1);
            NaiveBayes model = new NaiveBayes();
            model.buildClassifier(trainInsts);
            //System.out.println(model);
            Evaluation eTest = new Evaluation(classifyInsts);
            eTest.evaluateModel(model, classifyInsts);
            String[] cmarray = {"normal", "anomaly"};
            ConfusionMatrix cm = new ConfusionMatrix(cmarray);
            //System.out.println(cm.correct());
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

```

        for (int i = 0; i < classifyInsts.numInstances(); i++)
        {
            classifyInsts.instance(i).setClassMissing();
            double cls =
            model.classifyInstance(classifyInsts.instance(i));
            classifyInsts.instance(i).setClassValue(cls);
        }
        System.out.println("Error Rate: "+eTest.errorRate()*100);
        System.out.println("Pct Correct: "+eTest.pctCorrect());
        for (int i=0; i<trainInsts.numClasses(); i++)
        {
            System.out.println("Class "+ i);
            System.out.println("Precision " +eTest.precision(i));
            System.out.println("Recall "+eTest.recall(i));
            System.out.println("Area under ROC "+eTest.areaUnderROC(i));
            System.out.println();
        }
        //System.out.println(classifyInsts);
    } //Try
    catch (Exception o)
    {
        System.err.println(o.getMessage());
    }
} //Mla7 constructor

public static void main(String[] args) {
    Mla7 nb = new Mla7();
}

} //Class Mla7

```

4. Clean and Build the project
5. Run the project and see the following output

Output Parameters (After you run the program)

run:

Error Rate: 11.538461538461538

Pct Correct: 88.46153846153847

Class 0

Precision 0.9

Recall 0.8181818181818182

Area under ROC 0.9757575757575757

Class 1

Precision 0.875

Recall 0.9333333333333333

Area under ROC 0.9757575757575757

BUILD SUCCESSFUL (total time: 1 second)

Class and Subject: BEIT Information Technology (2015)
Subject: Computing Laboratory VII (Part B)

MLA Lab 7 Principal Components Analysis

Aim: Principal Component Analysis-Finding Principal Components, Variance and Standard Deviation calculations of principal components.(Using R)(Iris Dataset)

Principal Component Analysis

In simple words, principal component analysis is a method of extracting important variables (in form of components) from a large set of variables available in a data set. It extracts low dimensional set of features from a high dimensional data set with a motive to capture as much information as possible. With fewer variables, visualization also becomes much more meaningful. PCA is more useful when dealing with 3 or higher dimensional data.

It is always performed on a symmetric correlation or covariance matrix. This means the matrix should be numeric and have standardized data.

Normalization:

The principal components are supplied with normalized version of original predictors. This is because, the original predictors may have different scales. For example: Imagine a data set with variables' measuring units as gallons, kilometers, light years etc. It is definite that the scale of variances in these variables will be large.

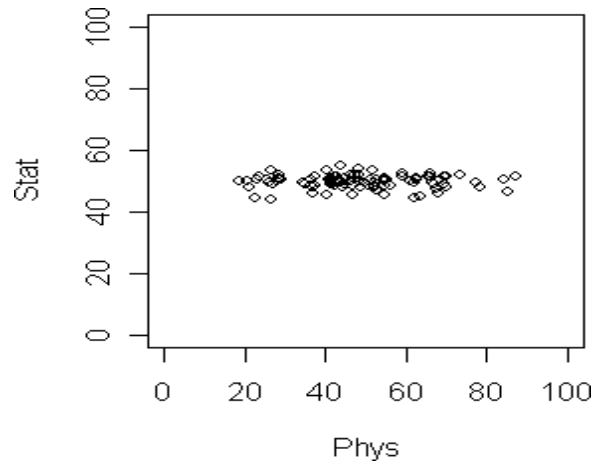
Performing PCA on un-normalized variables will lead to insanely large loadings for variables with high variance. In turn, this will lead to dependence of a principal component on the variable with high variance. This is undesirable.

PCA highlights [2]

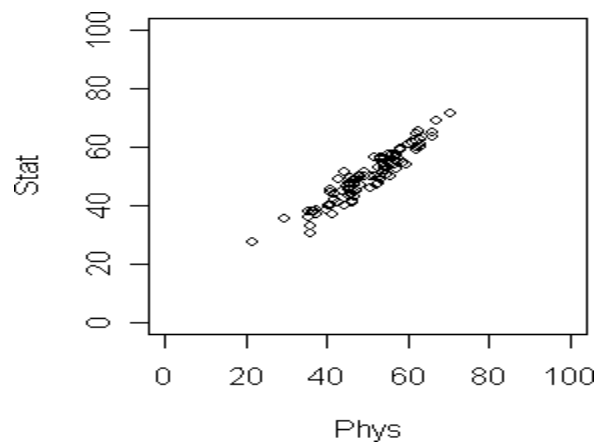
1. PCA is used to overcome features redundancy in a data set.
2. These features are low dimensional in nature.
3. These features a.k.a components are a resultant of normalized linear combination of original predictor variables.
4. These components aim to capture as much information as possible with high explained variance.
5. The first component has the highest variance followed by second, third and so on.
6. The components must be uncorrelated (remember orthogonal direction ?). See above.
7. Normalizing data becomes extremely important when the predictors are measured in different units.
8. PCA works best on data set having 3 or higher dimensions. Because, with higher dimensions, it becomes increasingly difficult to make interpretations from the resultant cloud of data.
9. PCA is applied on a data set with numeric variables.
10. PCA is a tool which helps to produce better visualizations of high dimensional data.

Example:

Consider 100 students with Physics and Statistics grades shown in the diagram below. The data set is in marks.dat.



If we want to compare among the students which grade should be a better discriminating factor? Physics or Statistics? Surely Physics, since the variation is larger there. This is a common situation in data analysis where the direction along which the data *varies the most* is of special importance. Now suppose that the plot looks like the following. What is the best way to compare the students now?



Here the direction of maximum variation is like a slanted straight line. This means we should take linear combination of the two grades to get the best result. In this simple data set the direction of

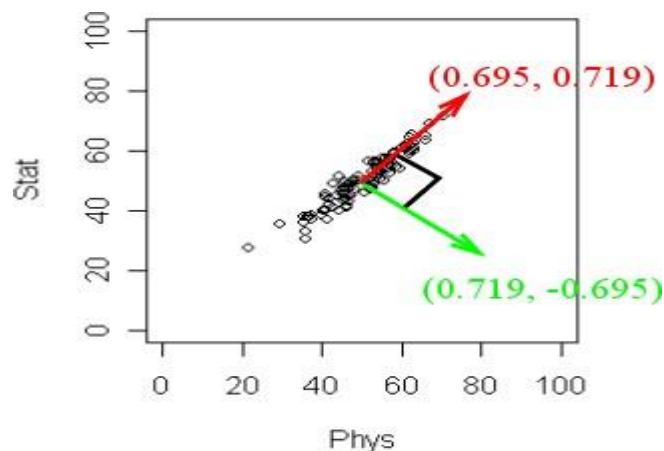
maximum variation is more or less clear. But for many data sets (especially high dimensional ones) such visual inspection is not adequate or even possible! So we need an objective method to find such a direction.

Principal Component Analysis (PCA) is one way to do this.

```
dat = read.table("marks.dat", head=T)
dim(dat)
names(dat)
pc = princomp(~Stat+Phys, dat)
pc$loading
```

Notice the somewhat non-intuitive syntax of the `princomp` function. The first argument is a so-called formula object in R (we have encountered this beast in the regression tutorial). In `princomp` the first argument must start with a `~` followed by a list of the variables (separated by plus signs).

The output may not be readily obvious. The next diagram will help.



R has returned two **principal components**. (Two because we have two variables). These are a unit vector at right angles to each other. You may think of PCA as choosing a new coordinate system for the data, the principal components being the unit vectors along the axes. The first principal component gives the direction of the maximum spread of the data. The second gives the direction of maximum spread perpendicular to the first direction. These two directions are packed inside the matrix `pc$loadings`. Each column gives a direction. The direction of maximum spread (the first principal component) is in the first column, the next principal component in the second and so on.

PCA Experiment in R studio [3]

```
#Data set Business Industry categories and their Progress
#The database is attached to the R search path. This means that
#the database is searched by R when evaluating a variable, so
#objects in the database can be accessed by simply giving their
#names.

mydata<-read.csv("E:/pca_gsp.csv")
attach(mydata)

# list the variables in mydata
names(mydata)

X <- cbind(Ag, Mining, Constr, Manuf, Manuf_nd, Transp, Comm,
Energy, TradeW, TradeR, RE, Services, Govt)

# mean,median,25th and 75th quartiles, min, max
summary(X)

# You can use the cor( ) function to produce correlations
cor(X)

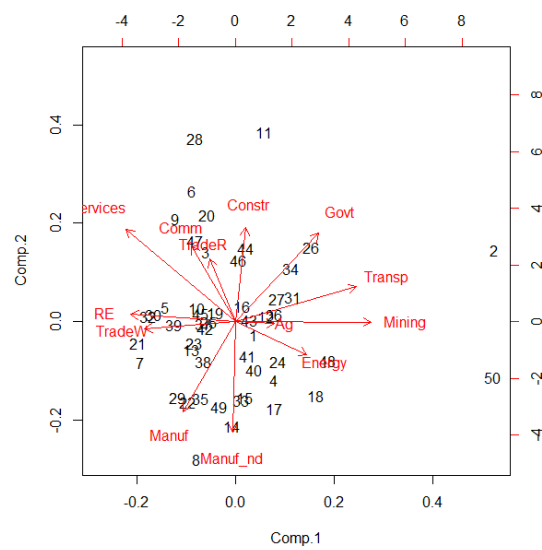
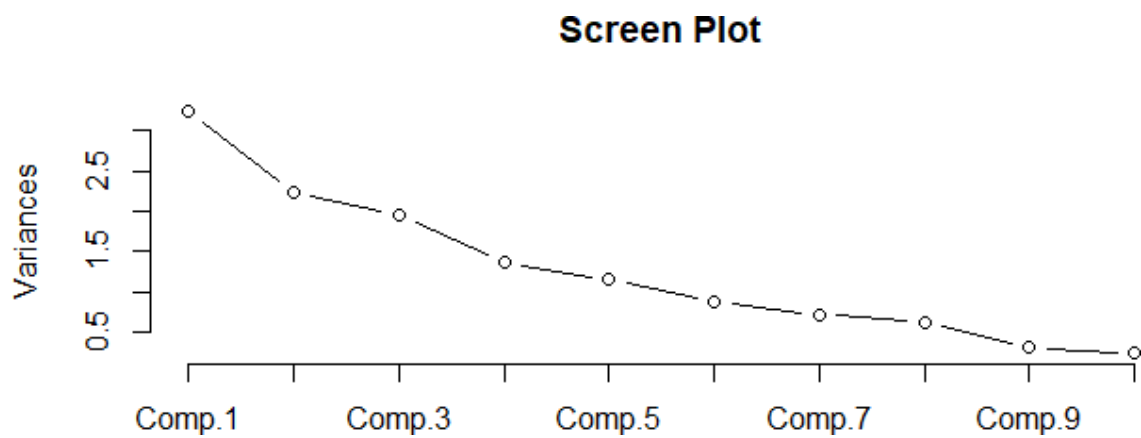
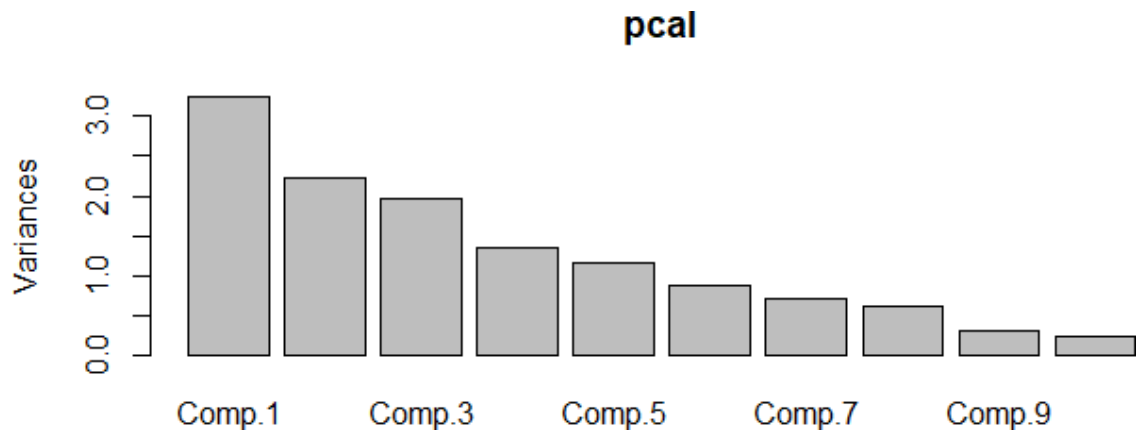
# princomp performs a principal components analysis on the given
numeric data matrix and #returns the results as an object of
class princomp.
pcal<-princomp(X, scores=TRUE, cor=TRUE)

#Summary. A very useful multipurpose function in R issummary(X),
where X can be one of #any number of objects, including datasets,
variables, and linear models, just to name a few. #When used,
the command provides summary data related to the individual
object that was fed #into it
summary(pcal)

# Extract or print loadings in factor analysis
loadings(pcal)

#Visualize the Principal Components
plot(pcal)
screeplot(pcal,type="line",main="Screen Plot")
biplot(pcal)
pcal$scores[1:10,]
```

Principal Components graph is shown below



References

- [1] <http://www.psu.edu>
- [2] <http://www.analyticsvidhya.com>
- [3] <https://github.com/dmitrijsc/dsc-pca-examples>

