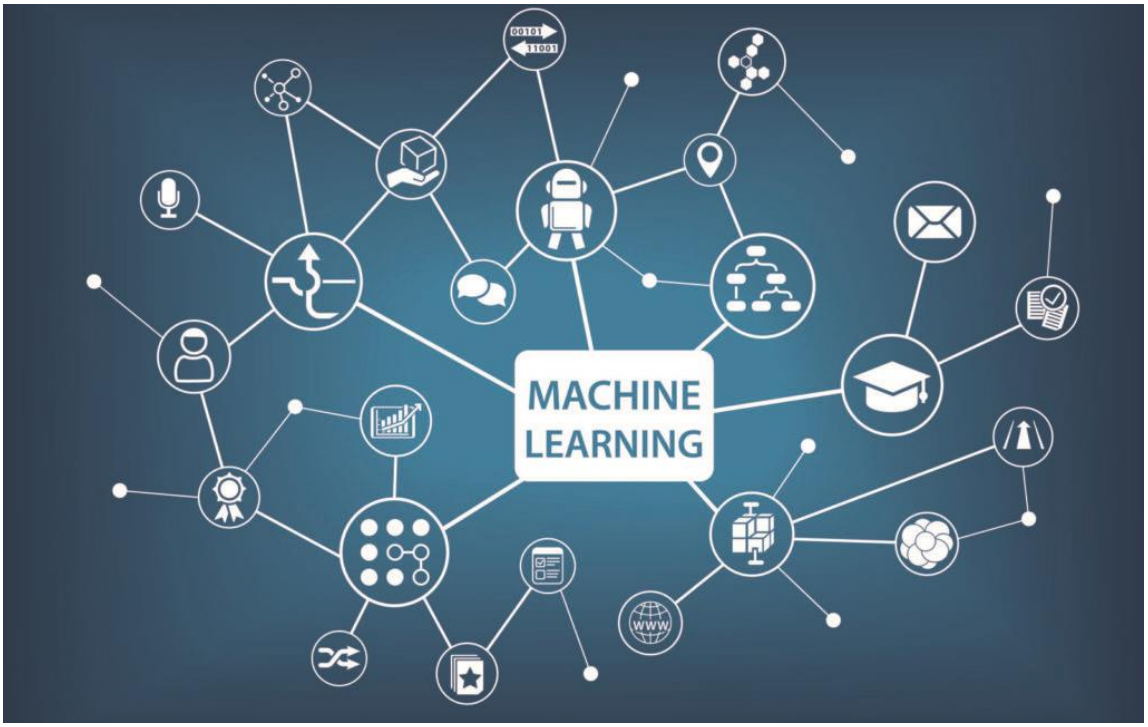
VRIJE UNIVERSITEIT BRUSSEL

Option-: 2

Select two Machine Learning Algorithms and analyze their performance on an interesting problem or dataset.

COURSE – Techniques of Artificial Intelligence.



Name: Vaibav Paliwal

VUB Number: 0551609

Email: Vaibav.Paliwal@vub.be

DECLARATION

This is to declare that the work done and displayed via this report is performed by **Vaibav Paliwal**. This report is a part of the course Techniques of Artificial Intelligence which comes under the program of Master of Science: Applied Computer Science at the Vrije Universiteit Brussel.

Acknowledgement

This is to thank all my course Professors and teaching assistant Isel Del Carmen Grau Garcia for the structured curriculum of the course because of which I could gain knowledge to understand and perform this project in a better way. With their support and guidance, I completed this project on time.

Table of Contents

INTRODUCTION

Project Description-:

In this project, I used the Breast Cancer dataset from UCI Repository, which is a classification type of dataset. I checked the performance of two famous supervised type of machine learning algorithm on the dataset. The algorithm that I used in my project are Random Forest algorithm and Support Vector Machine algorithm. I implemented the whole project in Python programming language using Jupyter Notebook. I used various libraries of Python for completing this project.

About Dataset -:

I used the Breast Cancer Wisconsin original dataset which is donated on 15th July 1992. I took this dataset from UCI Machine Learning Repository. This is open source repository which is consisting of many datasets. The dataset consists of 699 instances and 10 attributes. It is a very interesting dataset and already have lot of web hits and published in many pages. I converted the dataset in csv file and then used in Python for project. The link for dataset is:

https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Original%29

Problem Description -:

The dataset is a classification type of dataset, so as problem is also classification type of problem, where we need to determine whether a person is having cancerous cell (Malignant) or non-cancerous cell (Benign). The final attribute of dataset (Class) consisting of two values:
2 for Benign and 4 for Malignant.

About Machine Learning -:

It is a subset of artificial intelligence which focuses mainly on machine learning from their experience and making predictions based on its experience. It enables the machines to make data-driven decisions rather than being explicitly programmed for carrying out a certain task. These programs or algorithms are designed in a way that they learn and improve over time when are exposed to new data.

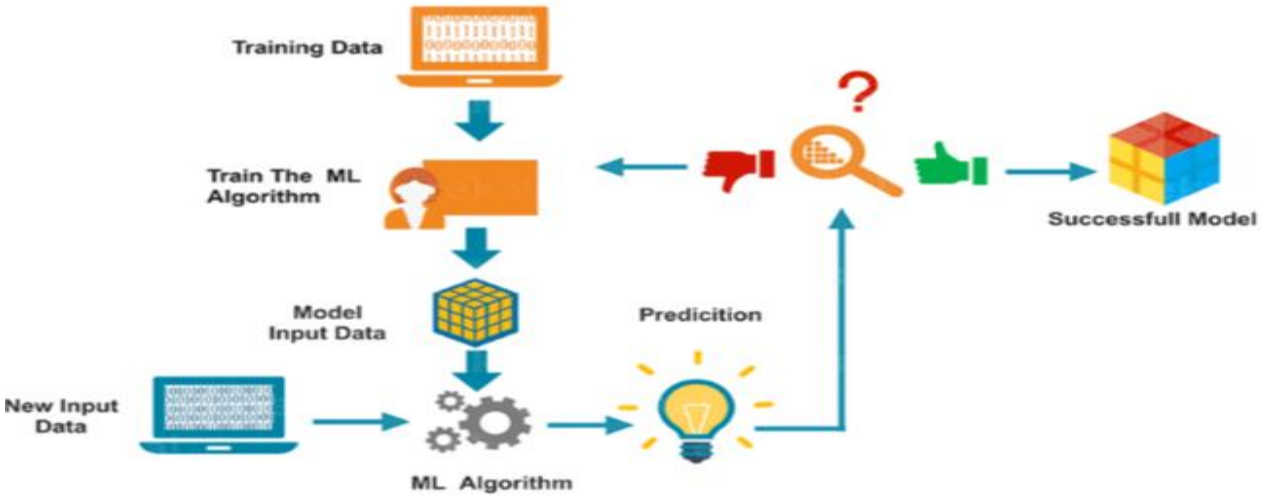
There are three types of Machine Learning:

1. **Supervised Machine Learning-:** It is considered as the learning guided by teachers. In this type of machine learning we have a dataset which acts as a teacher and its role is to train the model or the machine. Once the model gets trained it can start making a prediction or decision when new data is given to it.
   Examples: Regression, Decision trees, Random Forest, Support Vector Machine, KNN.

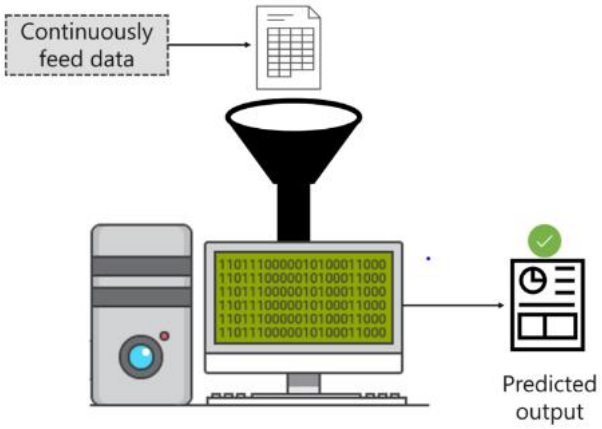
2. **Unsupervised Machine Learning -:** In this machine learning, we do not have any target or outcome variable to predict / estimate. It is used for clustering population in different groups, which is widely used for segmenting customers in different groups for specific intervention.
   Examples of Unsupervised Learning: Apriori algorithm, K-means.

3. **Reinforcement Machine Learning -:** It is the ability of an agent to interact with the environment and find out what is the best outcome. It follows the concept of hit and trial method. This machine learns from experience and tries to capture the best possible knowledge to make accurate business decisions.
   Example of Reinforcement Learning: Markov Decision Process.

How Does Machine Learning works?
The picture will give an idea of working model of Machine Learning.



Another Picture that will help in getting idea that how ML work:



Algorithm Used in Project

There are many algorithms present like SVM, KNN, Naïve Bayes, Random Forest, Decision Trees which can be used to solve the classification type of problems.
In my project I used two best supervised type of machine learning algorithms (Support Vector Machine and Random Forest), both the algorithms work very well in finding accuracy for classification problems.

About Support Vector Machine -:

It is a supervised machine learning algorithm which is mainly used to classify data into different classes. Unlike most algorithms, SVM makes use of a hyperplane which acts like a decision boundary between the various classes. SVM can be used to generate multiple separating hyperplanes such that the data is divided into segments and each segment contains only one kind of data.

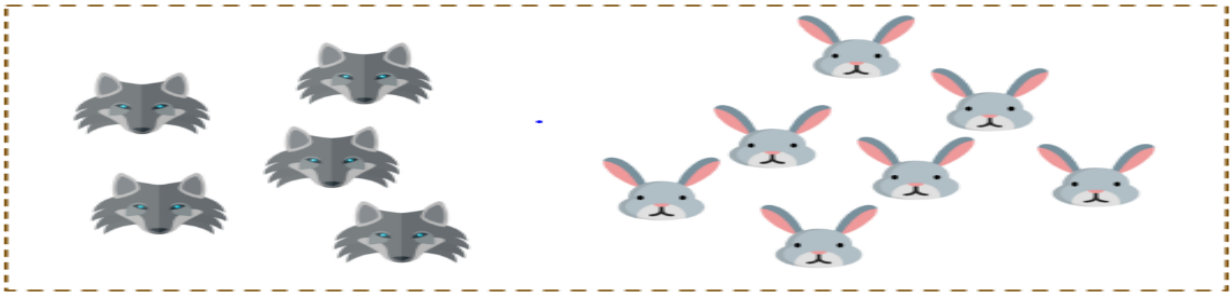
Example: if we only had two features like Height and Hair length of an individual, we'd first plot these two variables in two-dimensional space where each point has two co-ordinates (these co-ordinates are known as **Support Vectors**).

Features of SVM:

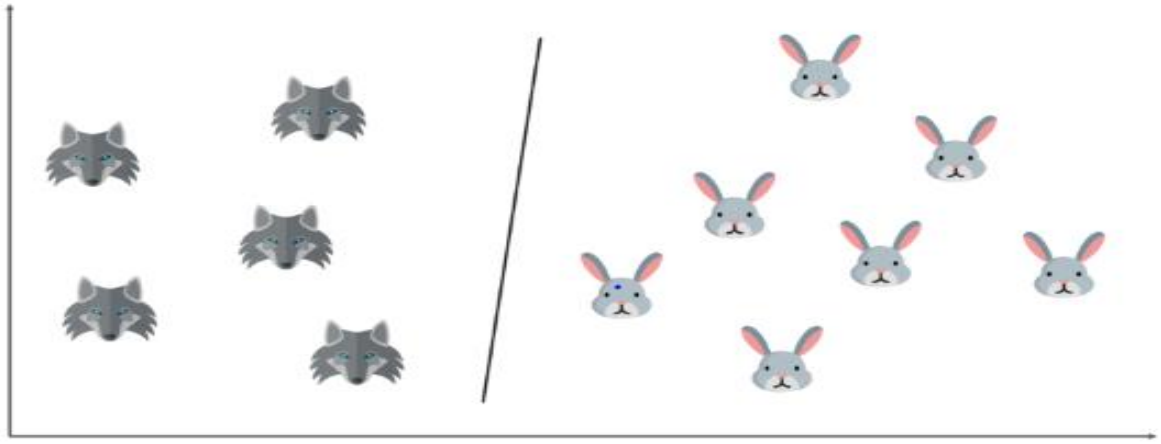
1. As mentioned before, it is a supervised learning algorithm, which means it trains itself on a set of labeled data. It studies from the labeled training data and then classify new input data depending on what it learned in training phase.
2. This algorithm can be used for both classification and regression type of problems and this is one of the main advantage of this algorithm. The SVM is mainly known for classification and SVR (Support Vector Regressor) is used for regression problems.
3. It can also be used for classifying the non-linear type of data using kernels. In SVM there are different kernels (Linear, RBF, Sigmoid, Poly). These kernels are used in transforming the non-linear data into high dimensional, so that a clear hyperplane will be create between various classes of data.

How does SVM work?

To understand this let's consider an example of a farm, where we have rabbits and wolves and over problem is, we need to set up a fence to protect the rabbits from wolves. So, how we build fence?



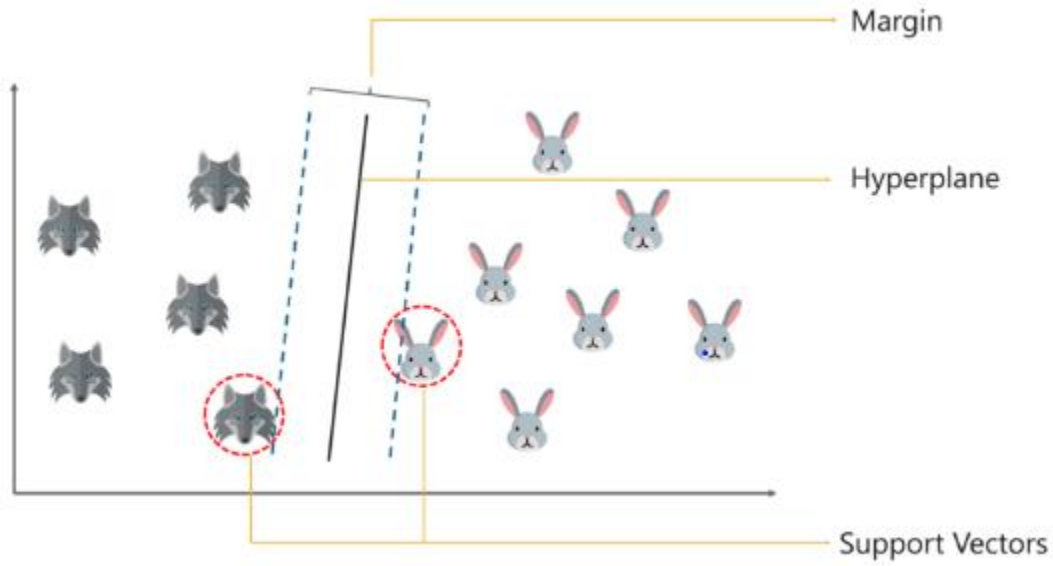
On of the solution comes in mind is to build a decision boundary based on the position of rabbits and wolves. So, it will look like:



Now you can see a clear fence along this line.

SVM works in same way, it draws a decision boundary which is known as hyperplane between any two classes to separate or classify them. The basic principle of SVM is to draw a hyperplane that best separates the two classes. Here in our example the two classes are rabbits and wolves.

To understand SVM in more details lets consider the below picture:

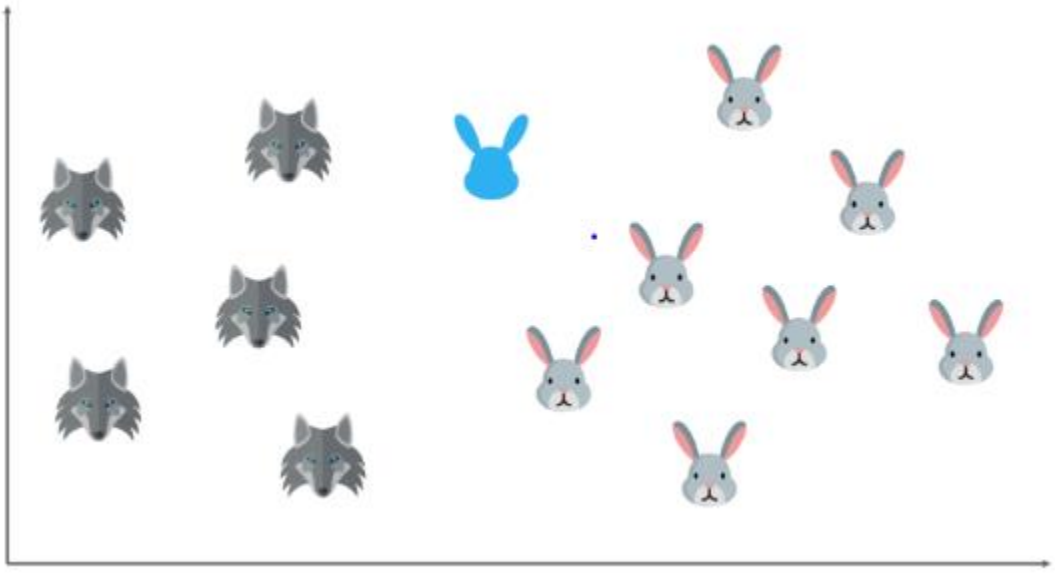


So, the first task is to draw the hyperplane that can be random and then you check the distance between the hyperplane and closest data prints from each class. The closest data points from the hyperplane are known as Support Vectors, as you can see in the picture. And, that's the reason behind this algorithm (Support Vector Machine).

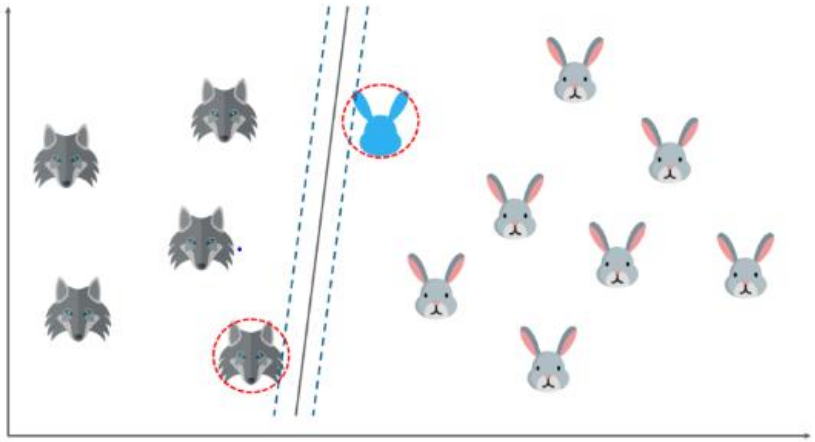
The hyperplane is drawn based on the support vectors and an optimum hyperplane is one that will have maximum distance from each of the support vectors. This distance between hyperplane and support vectors is known as Margin.

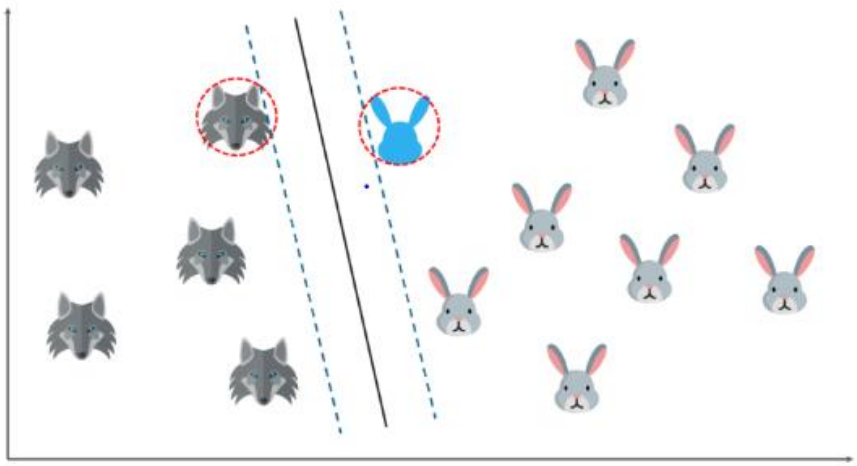
SVM used to classify the data by using a hyperplane, considering that the distance between hyperplane and support vectors is maximum.

Let's consider a situation, where you input a new data point and you want to draw a hyperplane that it best separates these two classes.



The blue one is the new input data. So, consider the two hyperplane pictures and see which one is having optimal hyperplane.



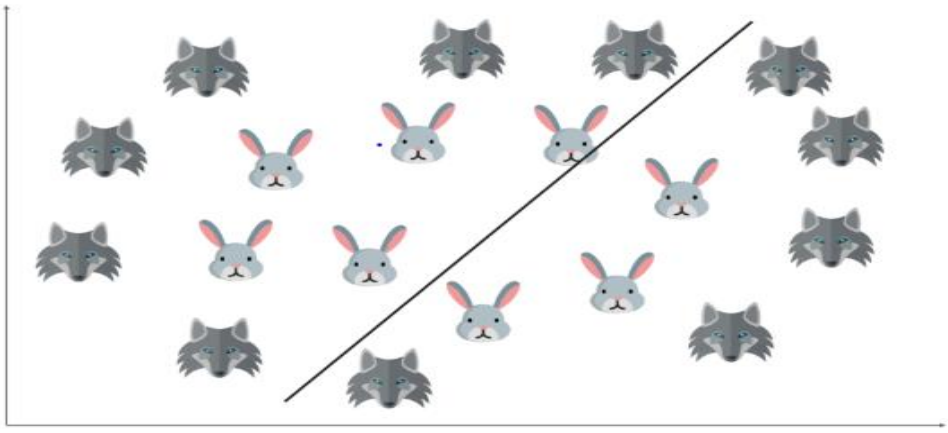


From definition we know an optimal hyperplane is one which is having maximum distance from the support vectors. So, from both the pictures we can see the Margin of second picture is clearly more than the first one. The second hyperplane is our optimal hyperplane.

Non-Linear Support Vector Machine -:

Consider the below picture and try to draw the hyperplane.

As we can see the above data is totally different from the previous data we considered. Now if you draw a hyperplane for such data, it will be like:

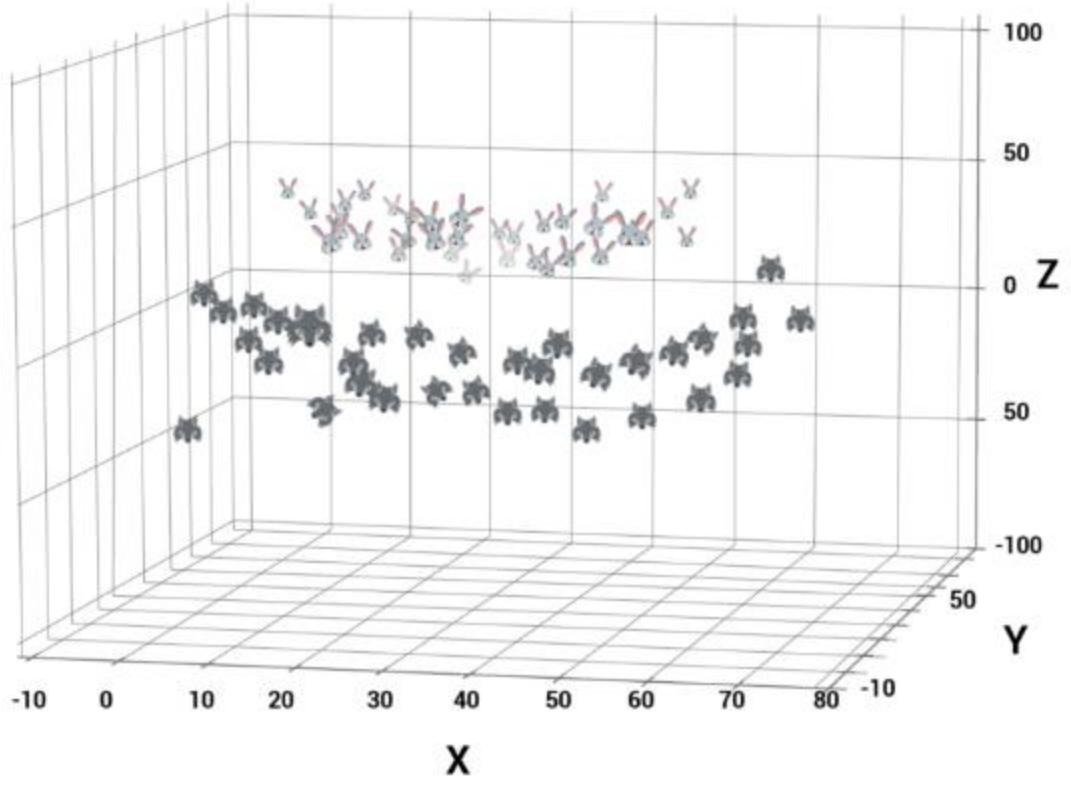


So, the above hyperplane is totally incorrect, and it doesn't separate the two classes.

To solve such problems Non-Linear SVM comes into to the picture. As I mentioned before that how can we use kernel to transfer data into another dimension that has clear dividing margin between classes of data.

Kernel function provides the option of transforming non-linear spaces into linear spaces. Till now, we plotted data based on two variables (x and y) on 2D space. We will transform the two-variable data into three variable data with (z) as third variable. Means, we are visualizing the data in 3D space.

When you transform your data from 2D to 3D space you will be able to see a clear dividing margin between the two classes of data. And then you can separate the two classes by drawing an optimal hyperplane between them.



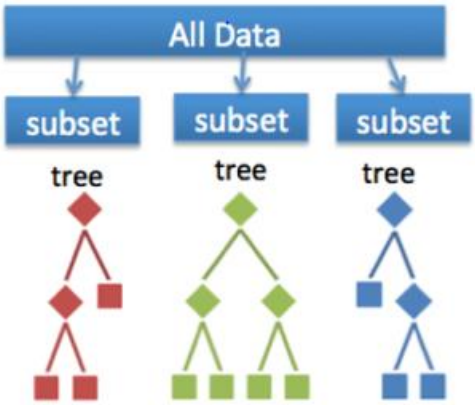
END Notes:

Support Vector Machines are very powerful classification algorithm. When used in conjunction with random forest and other machine learning tools, they give a very different dimension to ensemble models. Hence, they become very crucial for cases where very high predictive power is required. The performance of the SVM classifier was very accurate for even a small data set and its performance was compared to other classification algorithms like Naïve Bayes and in each case, the SVM outperformed Naive Bayes.

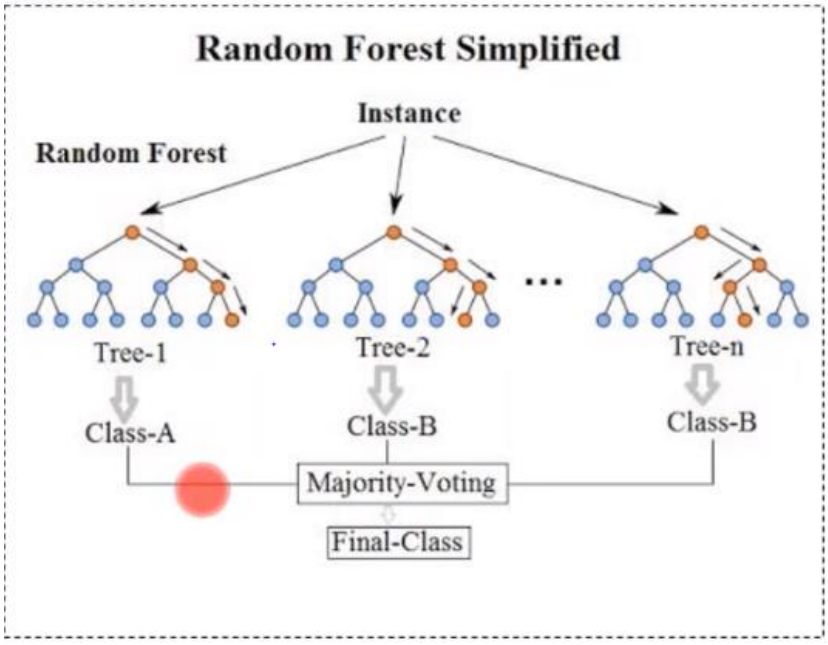
About Random Forest -:

Random Forest is a supervised type of machine learning algorithm which is used for both classification and regression problem. It is a trademark term for an ensemble of decision trees. In Random Forest, we've collection of decision trees (so known as "Forest"). To classify a new object based on attributes, each tree gives a classification and we say the tree "votes" for that class. The forest chooses the classification having the most votes (over all the trees in the forest).

Generally, the more trees in the forest the more robust the forest looks like. Similarly, in the random forest classifier, the higher the number of trees in the forest, greater is the accuracy of the results.



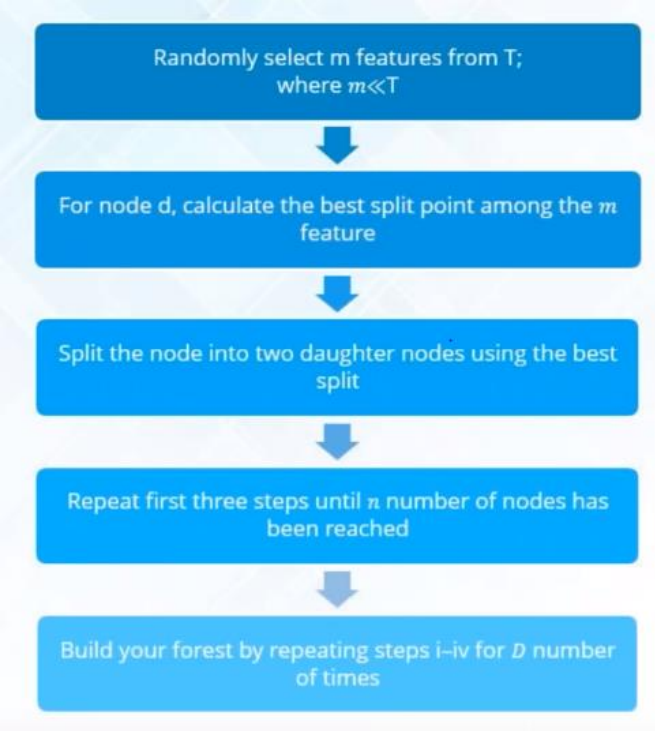
Random Forest basically compile the outcome of all the decision trees and based on majority voting it gives the final result. To get a better idea, consider the below picture.



How does Random Forest work?

The first step in Random Forest is that it will divide the data into smaller subsets and every subset need not to be distinct, some subsets may be overlapped.

Consider the picture for understanding the working of Random forest, it consists of several steps which are explained in the picture.



Here, T = Number of Features, D = number of trees to be constructed, V = output, the class with the highest vote.

Features of Random Forest -:

1. Most accurate learning algorithm.
2. It works well for both classification and regression problems.
3. It runs efficiently on large datasets.
4. It does not require input preparation because of its implicit features.
5. It does not take much time and can be easily grown in parallel.

End Notes -:

Random forest gives much more accurate predictions when compared to simple CART/CHAID or regression models in many scenarios. These cases generally have high number of predictive variables and huge sample size. This is because it captures the variance of several input variables at the same time and enables high number of observations to participate in the prediction.

Project Code and Description

I have completed this project in Python using Jupyter Notebook. In the description I will show the step wise code with the description.

Python libraries used in the code-:

1. Pandas.
2. Sklearn (Scipy).
3. Numpy.
4. Matplot.
5. Seaborn.

Let's look at the code step wise:

Step1-: The first step is to import the necessary libraries for the code.

```
In [77]: import numpy as np
         import sklearn
         from sklearn import model_selection
         from sklearn.model_selection import cross_validate
         from sklearn.svm import SVC
         from sklearn.tree import DecisionTreeClassifier
         from sklearn.neighbors import KNeighborsClassifier
         from sklearn.metrics import classification_report
         from sklearn.metrics import accuracy_score
         from pandas.plotting import scatter_matrix
         import matplotlib.pyplot as plt
         import pandas as pd
         import seaborn as sns
```

Step2-: Now as we imported the necessary libraries let's import the dataset in the form of csv file using pandas library.

```
In [78]: data = pd.read_csv("Breast_cancer.csv")
```

Step3-: Check for the missing data and preprocess it, we will also look at the data axes and attributes.

```
In [79]: # Preprocess the data
         data.replace('?',-99999, inplace=True)
         print(data.axes)

         print(data.columns)
```

```
[RangeIndex(start=0, stop=699, step=1), Index(['ID', 'Clump Thickness', 'Uniformity of Cell Size',
       'Uniformity of Cell Shape', 'Marginal Adhesion',
       'Single Epithelial Cell Size', 'Bare Nuclei', 'Bland Chromatin',
       'Normal Nucleoli', 'Mitoses', 'Class'],
      dtype='object')]
Index(['ID', 'Clump Thickness', 'Uniformity of Cell Size',
       'Uniformity of Cell Shape', 'Marginal Adhesion',
       'Single Epithelial Cell Size', 'Bare Nuclei', 'Bland Chromatin',
       'Normal Nucleoli', 'Mitoses', 'Class'],
      dtype='object')
```

Step4-: In this step we will randomly select one row and visualize its data, we will also look for the shape of data, means total number of instances and attributes. The highlighted output is the shape of dataset.

```
In [80]: #visualize and explore the data
         print(data.loc[20])

         # Print the shape of the dataset
         print(data.shape)
```

```
ID                             1054590
Clump Thickness                      7
Uniformity of Cell Size              3
Uniformity of Cell Shape             2
Marginal Adhesion                   10
Single Epithelial Cell Size          5
Bare Nuclei                         10
Bland Chromatin                      5
Normal Nucleoli                      4
Mitoses                              4
Class                                4
Name: 20, dtype: object
(699, 11)
```

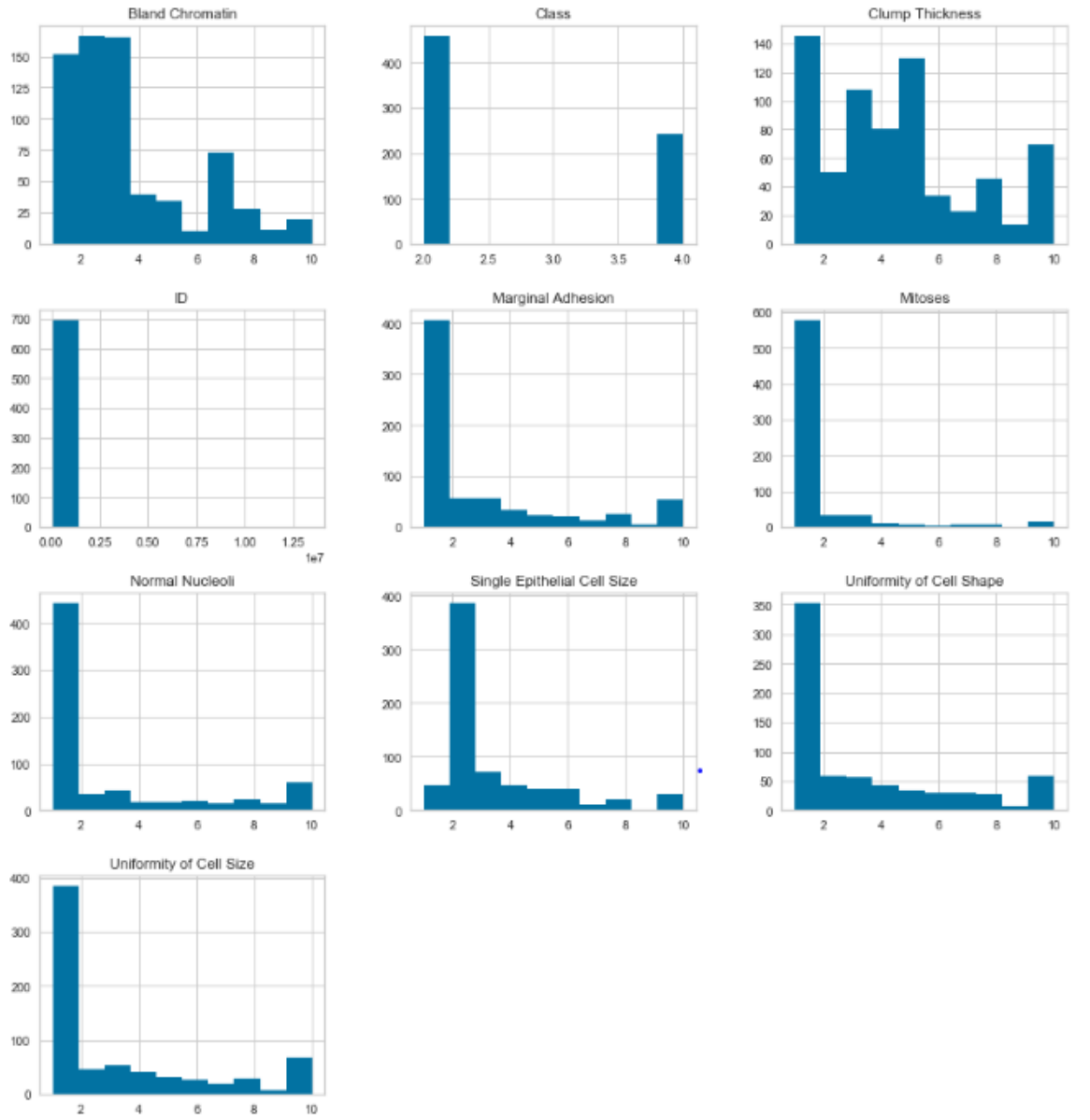
Step5-: Now we will describe our data, it means we will look at the statistics value for each attribute. The (describe) function of pandas lib Generate descriptive statistics that summarize the central tendency, dispersion and shape of a dataset's distribution, excluding NaN values.

```
In [81]: #describing the data
         print(data.describe())
```

```
                 ID  Clump Thickness  Uniformity of Cell Size  \
count  6.990000e+02       699.000000               699.000000
mean   1.071704e+06         4.417740                 3.134478
std    6.170957e+05         2.815741                 3.051459
min    6.163400e+04         1.000000                 1.000000
25%    8.706885e+05         2.000000                 1.000000
50%    1.171710e+06         4.000000                 1.000000
75%    1.238298e+06         6.000000                 5.000000
max    1.345435e+07        10.000000                10.000000

       Uniformity of Cell Shape  Marginal Adhesion  \
count                699.000000         699.000000
mean                   3.207439           2.806867
std                    2.971913           2.855379
min                    1.000000           1.000000
25%                    1.000000           1.000000
50%                    1.000000           1.000000
75%                    5.000000           4.000000
max                   10.000000          10.000000

       Single Epithelial Cell Size  Bland Chromatin  Normal Nucleoli  \
count                   699.000000       699.000000       699.000000
mean                      3.216023         3.437768         2.866953
std                       2.214300         2.438364         3.053634
min                       1.000000         1.000000         1.000000
25%                       2.000000         2.000000         1.000000
50%                       2.000000         3.000000         1.000000
75%                       4.000000         5.000000         4.000000
max                      10.000000        10.000000        10.000000

          Mitoses       Class
count  699.000000  699.000000
mean     1.589413    2.689557
std      1.715078    0.951273
min      1.000000    2.000000
25%      1.000000    2.000000
50%      1.000000    2.000000
75%      1.000000    4.000000
max     10.000000    4.000000
```

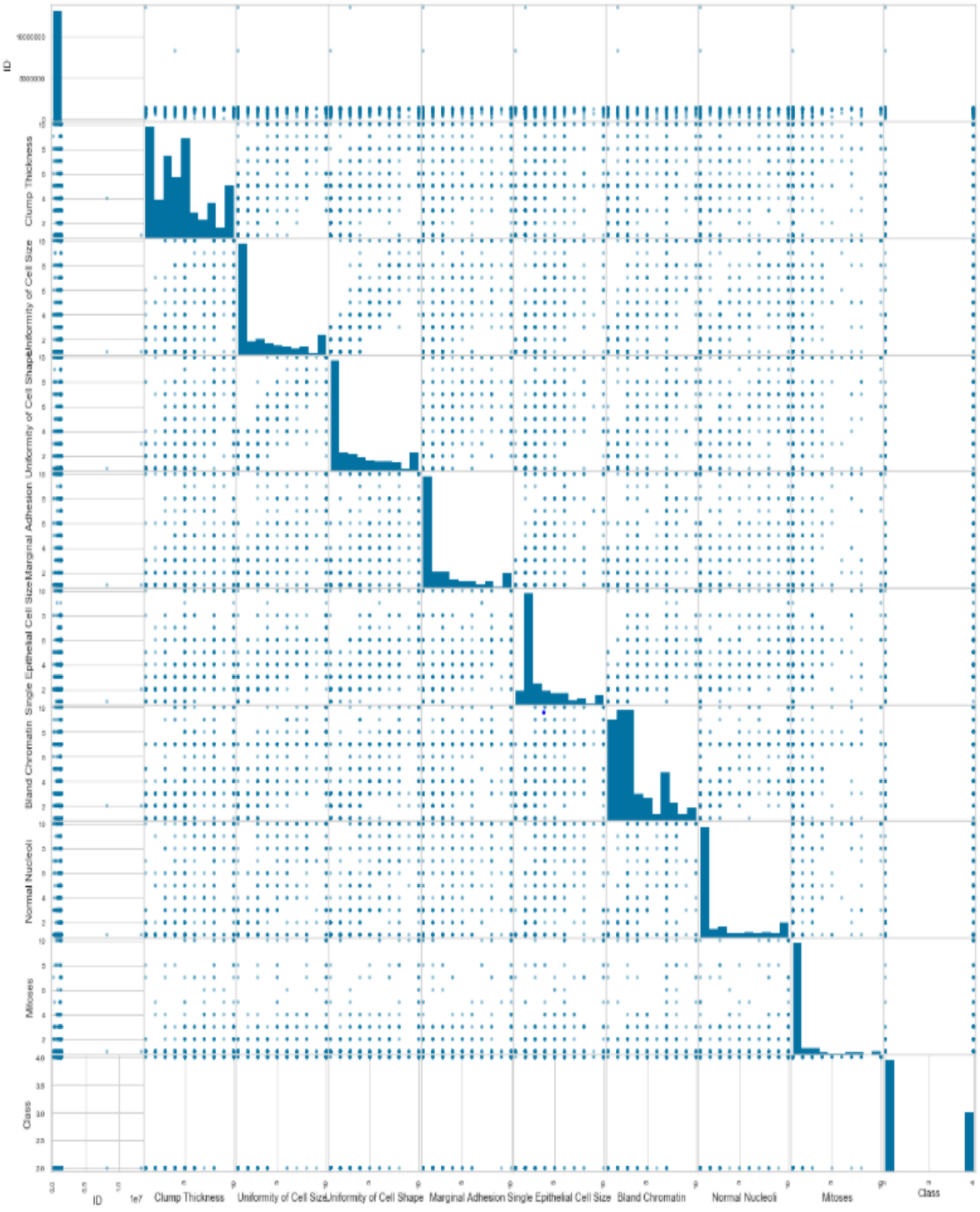
Step6-: Now we will do graphical representation of our dataset, in which we will use (histogram) feature to visualize graph of each attribute. A histogram is a representation of the distribution of data. This function calls matplotlib.pyplot.hist(), on each series in the Data Frame, resulting in one histogram per column.

```
In [82]: #Plotting the data
         data.hist(figsize=(15,15))
         plt.show()
```



Step7-: We will plot the scatter matrix for our dataset, which is broadly used for the understanding correlation between attributes. A scatter plot matrix can be formed for a collection of variables where each of the variables will be plotted against each other.

```
In [83]: #scattering the plot
         scatter_matrix(data, figsize=(20,20))
         plt.show()
```



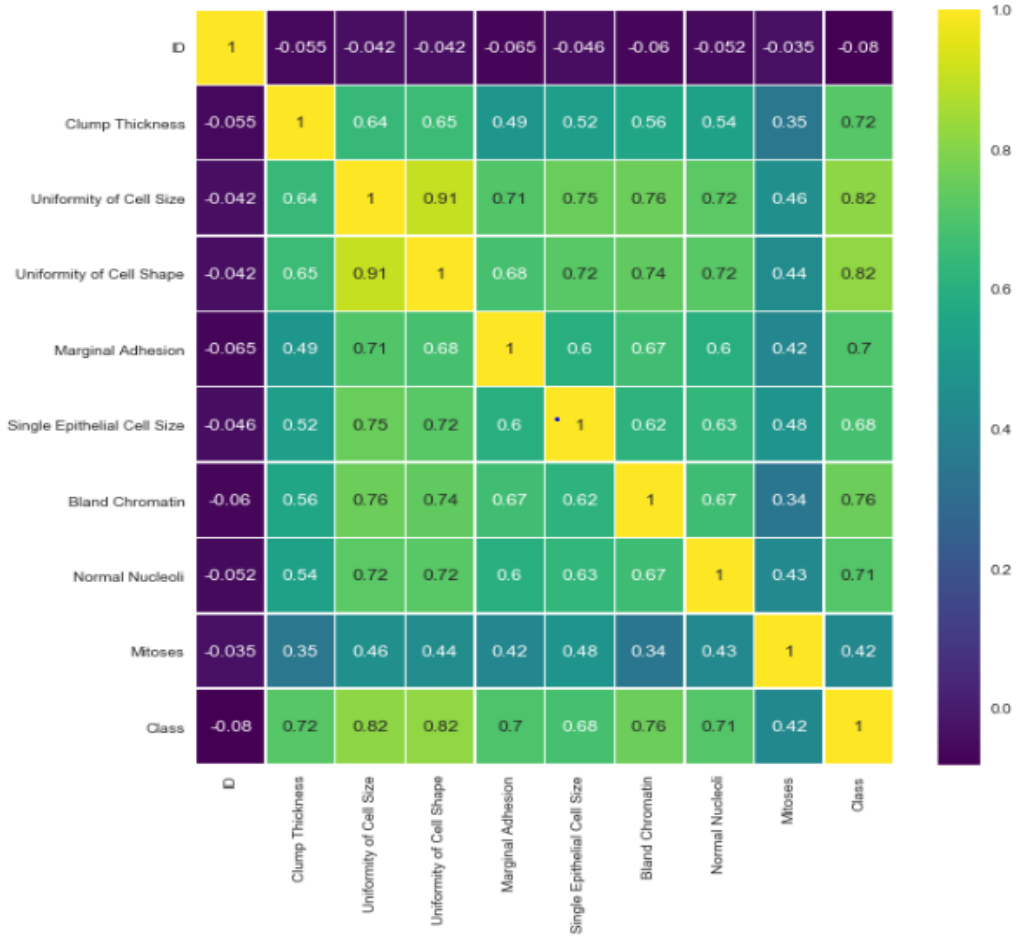
Step8-: In this step we will plot the correlation matrix to see the correlation between attributes. This also help us in determining that which attributes have high correlation and then we can decide which attribute is important for us. In Python the correlation values lie between (-1 and 1).

There are two key components of a correlation value:

1. magnitude – The larger the magnitude (closer to 1 or -1), the stronger the correlation.
2. sign – If negative, there is an inverse correlation. If positive, there is a regular correlation.

```
In [84]: # Correlation matrix
         corrmat = data.corr()
         plt.figure(figsize=(10,10))
         sns.heatmap(corrmat,cmap='viridis',annot=True,linewidths=0.5,)
```

```
Out[84]: <matplotlib.axes._subplots.AxesSubplot at 0x1b79123e518>
```



As, we can see (ID) column is not having any correlation with our main target (class), and its also known as well. Now if you look carefully, you will notice high correlation between Uniformity of cell size and uniformity of cell shape (0.91), so will check for the situation in one we will consider all attributes for our model and in another one, we will consider only one out of two attributes.

Step9-: In this step, we will convert the columns in a list and then divide our data into two variables (X and y), where X is consisting of all attributes except (class and ID). In y variable we will put target value which is our "class" attribute and then look for the shape of both variables.

```
In [85]: # Get all the columns from the dataFrame
         columns = data.columns.tolist()

         # Filter the columns to remove data we do not want
         columns = [c for c in columns if c not in ["Class", "ID"]]

         # Store the variable we'll be predicting on
         target = "Class"

         X = data[columns]
         y = data[target]

         # Print shapes
         print(X.shape)
         print(y.shape)

            (699, 9)
            (699,)
```

Step10-: Now look at the any random row of X and y to check we are going well.

```
In [86]: print(X.loc[20])
         print(y.loc[20])

            Clump Thickness               7
            Uniformity of Cell Size       3
            Uniformity of Cell Shape      2
            Marginal Adhesion            10
            Single Epithelial Cell Size   5
            Bare Nuclei                  10
            Bland Chromatin               5
            Normal Nucleoli               4
            Mitoses                       4
            Name: 20, dtype: object
            4
```

Step11-: This step is very important as we will spit our data into the training and testing to check the accuracy and for this we will use (model selection) library. When you're working on a model and want to train it, you obviously have a dataset. But after training, we have to test the model on some test dataset. To do this we will split the dataset into two sets, one for training and the other for testing; and you do this before you start training your model.

```
In [96]: #Creating X and y datasets for training
         #from sklearn.model_selection import train_test_split

         X_train, X_test, y_train, y_test = model_selection.train_test_split(X,y, test_size = 0.2)

In [97]: #Specify the testing option
         seed= 5
         scoring = 'accuracy'

         print(X_train.shape, X_test.shape)
         print(y_train.shape, y_test.shape)

            (559, 9) (140, 9)
            (559,) (140,)
```

Let's look at this in more detail, so here we split out data into (X_train, X_test & y_train, y_test). As you can see the (test_size = 0.2) it means we split our data where testing data is 20% and training data is 80% and to split data in to 80-20 ratio is a common practice in data science and recommended. For seed and scoring we will discuss later.

Step12 -: Sometimes we get the future warning in our code, so to ignore them we will use below command.

```
In [98]: # import warnings filter
         from warnings import simplefilter
         # ignore all future warnings
         simplefilter(action='ignore', category=FutureWarning)
```

Step13 -: This is the most important step of our code, where we will import our both algorithms (SVM and Random Forest) and then we will train model and test it using 10-fold cross validation. Firstly, look at the code and output then we will discuss the features and parameters.

```
In [29]: from sklearn.ensemble import RandomForestClassifier
         #Define models to train
         models = []
         #models.append(('KNN', KNeighborsClassifier(n_neighbors = 5)))
         models.append(('SVM', SVC(gamma = 'auto')))
         #models.append(('CART', DecisionTreeClassifier()))
         models.append(('RFC', RandomForestClassifier(max_depth=5, n_estimators = 40)))
         # evaluate each model in turn
         results = []
         names = []

         for name, model in models:
             kfold = model_selection.KFold(n_splits=10, random_state = seed)
             cv_results = model_selection.cross_val_score(model, X_train, y_train, cv=kfold, scoring=scoring)
             print(cv_results)
             results.append(cv_results)
             names.append(name)
             msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
             print(msg)
```

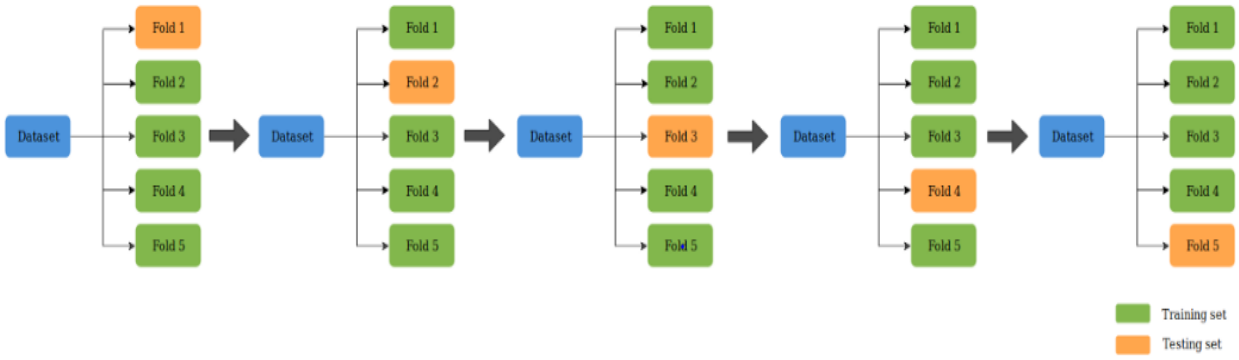
```
[0.96428571 0.96428571 1.         0.94642857 0.96428571 0.98214286
 0.89285714 0.92857143 0.91071429 0.94545455]
SVM: 0.949903 (0.030735)
[0.98214286 0.96428571 1.         0.96428571 0.98214286 1.
 0.94642857 0.98214286 0.89285714 0.92727273]
RFC: 0.964156 (0.032091)
```

As we can see we created an empty list name as models and then we append both the classifiers in our list (SVC and Random forest classifiers). In RF we have (n_estimators = 40), which means the number of trees we want to build.

Here we use KFold cross validation.

What is KFold cross validation?

It is a process where a given data set is split into a *K* number of sections/folds where each fold is used as a testing set at some point. Let's take the scenario of 5-Fold cross validation(K=5). Here, the data set is split into 5 folds. In the first iteration, the first fold is used to test the model and the rest are used to train the model. In the second iteration, 2nd fold is used as the testing set while the rest serve as the training set. This process is repeated until each fold of the 5 folds have been used as the testing set.



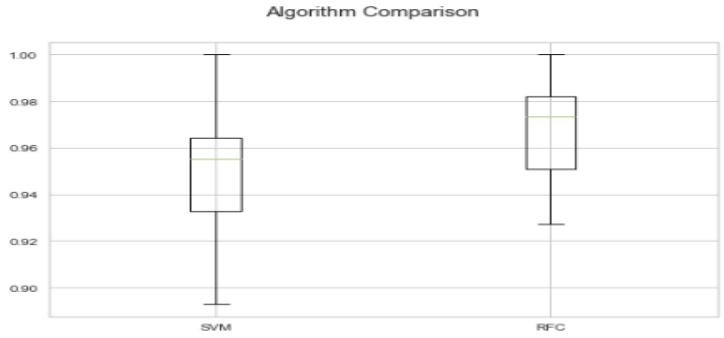
5-Fold Cross Validation

No look at some parameters (n_splits = 10) means value of K=10 and (random_state = seed) which we consider as random state and we have already defined (seed =5) previously. In cross_val_score, the scoring is the accuracy score.

A last we found the accuracy of all 10 folds score and calculated the mean and standard deviation.

Step14 -: Now we will plot an algorithm comparison box plot to compare the accuracy of both algorithms and as we can see the accuracy calculated by Random Forest is more than accuracy of SVM. It means RF is more accurate than SVM.

```
In [30]: # Compare Algorithms
         fig = plt.figure()
         fig.suptitle('Algorithm Comparison')
         ax = fig.add_subplot(111)
         plt.boxplot(results)
         ax.set_xticklabels(names)
         plt.show()
```



Step15 -: Let's visualize the result of all 10 folds graphically and look at the mean of all the score.

```
In [56]: from sklearn.model_selection import StratifiedKFold
         from yellowbrick.model_selection import CVScores

         _, ax = plt.subplots()

         cv = StratifiedKFold(10)

         oz = CVScores(RandomForestClassifier(max_depth=5, n_estimators = 40), ax=ax, cv=cv, scoring= 'accuracy')
         oz.fit(X,y)
         oz.poof()
```

Step16 -: Now we will make predictions on validation sheet, we will look at the accuracy score and classification report which is consisting of many important parameters.

```
In [57]: # Make predictions on validation dataset

         for name, model in models:
             model.fit(X_train, y_train)
             predictions = model.predict(X_test)
             print(name)
             print(accuracy_score(y_test, predictions))
             print(classification_report(y_test, predictions))
```

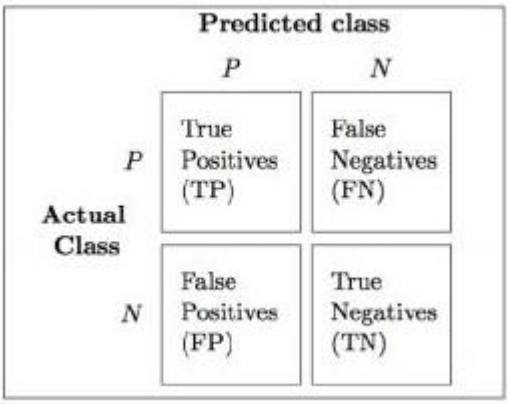
```
SVM
0.9642857142857143
              precision    recall  f1-score   support

           2       1.00      0.94      0.97        86
           4       0.92      1.00      0.96        54

   micro avg       0.96      0.96      0.96       140
   macro avg       0.96      0.97      0.96       140
weighted avg       0.97      0.96      0.96       140

RFC
0.9714285714285714
              precision    recall  f1-score   support

           2       1.00      0.95      0.98        86
           4       0.93      1.00      0.96        54

   micro avg       0.97      0.97      0.97       140
   macro avg       0.97      0.98      0.97       140
weighted avg       0.97      0.97      0.97       140
```

Accuracy Score -: In multilabel classification, this function computes subset accuracy: the set of labels predicted for a sample must *exactly* match the corresponding set of labels in y_true.

Classification Report -: In this report we can see the Precision, Recall, F1-score for each class. The reported averages include macro average (averaging the unweighted mean per label), weighted average (averaging the support-weighted mean per label), sample average (only for multilabel classification) and micro average (averaging the total true positives, false negatives and false positives) it is only shown for multi-label or multi-class with a subset of classes because it is accuracy otherwise.

We will discuss more about precision, recall, F1-score in next step.

Step17 -: Now we will look at the confusion matrix to evaluate the accuracy of a classification.
By definition a confusion matrix C is such that Ci,j is equal to the number of observations known to be in group i but predicted to be in group j.
Thus, in binary classification, the count of true negatives is C0,0, false negatives is C1,0, true positives is C1,1 and false positives is C0,1.
Normally a confusion matrix looks like:



Let's take about precision, F1-score, accuracy, recall, error rate.

- Precision refers to the accuracy of positive predictions.

  Precision = TP/TP+FP

- Accuracy - ratio of correctly predicted observation to the total observations.

  Accuracy = TP+TN/Total

- Recall (Sensitivity) - (false negatives) ratio of correctly predicted positive observations to the all observations in actual class - yes.

Recall = TP/Actual yes.

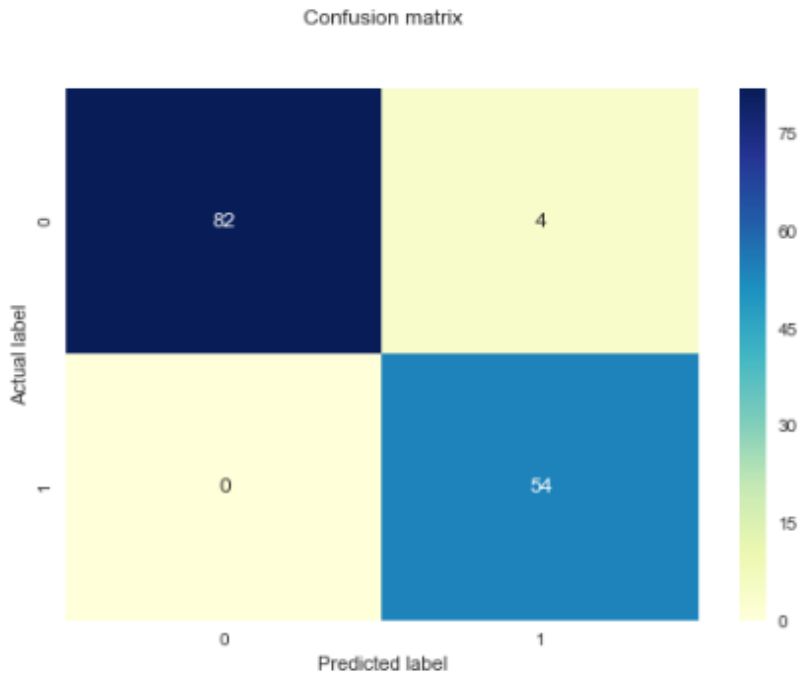
- F1 score - F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false.
- Error Rate = 1- Accuracy Score.

Now, look at our confusion matrix.

```
In [58]: from sklearn.metrics import  confusion_matrix
         predict = model.predict(X_test)
         print("=== Confusion Matrix ===")
         print(confusion_matrix(y_test, predict))
         print('\n')

         from sklearn import metrics
         cnf_matrix = metrics.confusion_matrix(y_test, predict)
         p = sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
         plt.title('Confusion matrix', y=1.1)
         plt.ylabel('Actual label')
         plt.xlabel('Predicted label')
```

```
=== Confusion Matrix ===
[[82  4]
 [ 0 54]]
```

```
Out[58]: Text(0.5,29.5,'Predicted label')
```



As, we can see for our 140 instances in testing data, we got only (4) wrong predictions, rest are correctly prediction.

Step18 -: In this step we will calculate the Cohen Kappa score and Matthews Correlation Coefficient (MCC).

```
In [59]: from sklearn.metrics import cohen_kappa_score
         cohen_score = cohen_kappa_score(y_test, predictions)
         print("Kappa Score: ", cohen_score)

         from sklearn.metrics import matthews_corrcoef

         MCC = matthews_corrcoef(y_test, predictions)

         print("MCC Score: ", MCC)
```

```
Kappa Score:  0.9405267629566695
MCC Score:  0.9421945411261875
```

Kappa Score -: A statistic that measures inter-annotator agreement. This function computes Cohen's kappa, a score that expresses the level of agreement between two annotators on a classification problem. It is defined as

$$\kappa=(po-pe)/(1-pe)$$

where po is the empirical probability of agreement on the label assigned to any sample (the observed agreement ratio), and pe is the expected agreement when both annotators assign labels randomly. pe is estimated using a per-annotator empirical prior over the class labels.
As a result, the kappa statistic, which is a number between -1 and 1. The maximum value means complete agreement; zero or lower means chance agreement.

MCC Score -: The Matthews correlation coefficient is used in machine learning as a measure of the quality of binary and multiclass classifications. It considers true and false positives and negatives and is generally regarded as a balanced measure which can be used even if the classes are of very different sizes. The MCC is in essence a correlation coefficient value between -1 and +1. A coefficient of +1 represents a perfect prediction, 0 an average random prediction and -1 an inverse prediction. The statistic is also known as the phi coefficient.

Let's play with the Dataset -:

1. Now we will play with the dataset to check the different scenarios and compare the ML algorithms. So, till not we split our data into 80-20 ratio and checked all the outputs.

Let's change the ratio to 70-30% and see the outputs.

In [72]:
```
#Creating X and y datasets for training
#from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = model_selection.train_test_split(X,y, test_size = 0.3)
```

In [73]:
```
#Specify the testing option
seed= 5
scoring = 'accuracy'

print(X_train.shape, X_test.shape)
print(y_train.shape, y_test.shape)
```

```
(489, 9) (210, 9)
(489,) (210,)
```

In [74]:
```
# import warnings filter
from warnings import simplefilter
# ignore all future warnings
simplefilter(action='ignore', category=FutureWarning)
```

In [75]:
```
from sklearn.ensemble import RandomForestClassifier
#Define models to train
models = []
#models.append(('KNN', KNeighborsClassifier(n_neighbors = 5)))
models.append(('SVM', SVC(gamma = 'auto')))
#models.append(('CART', DecisionTreeClassifier()))
models.append(('RFC', RandomForestClassifier(max_depth=5, n_estimators = 40)))
# evaluate each model in turn
results = []
names = []

for name, model in models:
    kfold = model_selection.KFold(n_splits=10, random_state = seed)
    cv_results = model_selection.cross_val_score(model, X_train, y_train, cv=kfold, scoring=scc
    print(cv_results)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
[0.91836735 0.97959184 0.95918367 0.91836735 0.95918367 0.91836735
 0.95918367 0.95918367 0.93877551 1.        ]
SVM: 0.951020 (0.026135)
[0.93877551 0.95918367 0.97959184 0.89795918 0.97959184 0.97959184
 0.97959184 0.97959184 1.         1.        ]
RFC: 0.969388 (0.029220)
```

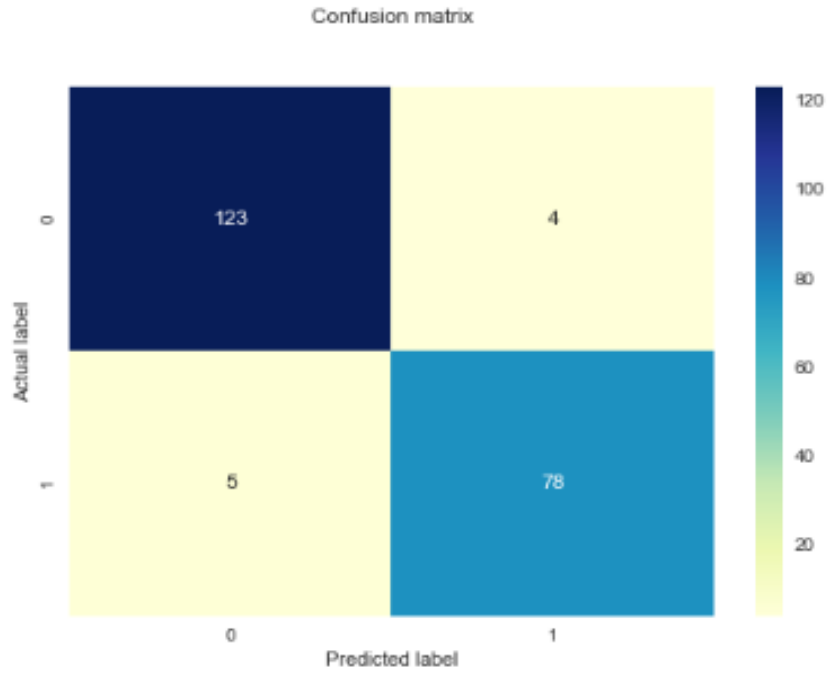
The accuracy of RF is still more than the SVM.

Let’s have a look on confusion matrix.

In [79]:
```
from sklearn.metrics import  confusion_matrix
predict = model.predict(X_test)
print("=== Confusion Matrix ===")
print(confusion_matrix(y_test, predict))
print('\n')

from sklearn import metrics
cnf_matrix = metrics.confusion_matrix(y_test, predict)
p = sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

```
=== Confusion Matrix ===
[[123   4]
 [  5  78]]
```

Out[79]: Text(0.5,29.5,'Predicted label')



Then look for 60-40% ratio -:

As you can see the accuracy is increased and RF again resulting in more accurate algorithm.

In [80]:
```
#Creating X and y datasets for training
#from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = model_selection.train_test_split(X,y, test_size = 0.4)
```

In [81]:
```
#Specify the testing option
seed= 5
scoring = 'accuracy'

print(X_train.shape, X_test.shape)
print(y_train.shape, y_test.shape)
```

```
(419, 9) (280, 9)
(419,) (280,)
```

In [82]:
```
# import warnings filter
from warnings import simplefilter
# ignore all future warnings
simplefilter(action='ignore', category=FutureWarning)
```

In [83]:
```
from sklearn.ensemble import RandomForestClassifier
#Define models to train
models = []
#models.append(('KNN', KNeighborsClassifier(n_neighbors = 5)))
models.append(('SVM', SVC(gamma = 'auto')))
#models.append(('CART', DecisionTreeClassifier()))
models.append(('RFC', RandomForestClassifier(max_depth=5, n_estimators = 40)))
# evaluate each model in turn
results = []
names = []

for name, model in models:
    kfold = model_selection.KFold(n_splits=10, random_state = seed)
    cv_results = model_selection.cross_val_score(model, X_train, y_train, cv=kfold, scoring=scoring)
    print(cv_results)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
[0.97619048 0.9047619  1.         0.92857143 0.92857143 0.97619048
 0.97619048 0.95238095 1.         0.92682927]
SVM: 0.956969 (0.031748)
[0.97619048 0.95238095 0.97619048 0.95238095 1.         1.
 1.         0.97619048 1.         0.97560976]
RFC: 0.980894 (0.017834)
```

In [84]:
```
# Compare Algorithms
```

2. As I mentioned in the correlation matrix step that we are getting high correlation among two attributes i.e. Uniformity of cell size and shape (0.91), so we try with only one attribute and remove the “uniformity of cell size” and look at the result.

In [130]:
```
# Get all the columns from the dataFrame
columns = data.columns.tolist()

# Filter the columns to remove data we do not want
columns = [c for c in columns if c not in ["Class", "ID", "Uniformity of Cell Size"]]

# Store the variable we'll be predicting on
target = "Class"

X = data[columns]
y = data[target]

# Print shapes
print(X.shape)
print(y.shape)
```

```
(699, 8)
(699,)
```

In [132]:
```
#Creating X and y datasets for training
#from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = model_selection.train_test_split(X,y, test_size = 0.2)
```

In [133]:
```
#Specify the testing option
seed= 5
scoring = 'accuracy'

print(X_train.shape, X_test.shape)
print(y_train.shape, y_test.shape)
```

```
(559, 8) (140, 8)
(559,) (140,)
```

In [134]:
```
# import warnings filter
from warnings import simplefilter
# ignore all future warnings
simplefilter(action='ignore', category=FutureWarning)
```

In [135]:
```
from sklearn.ensemble import RandomForestClassifier
#Define models to train
models = []
#models.append(('KNN', KNeighborsClassifier(n_neighbors = 5)))
models.append(('SVM', SVC(gamma = 'auto')))
#models.append(('CART', DecisionTreeClassifier()))
models.append(('RFC', RandomForestClassifier(max_depth=5, n_estimators = 40)))
# evaluate each model in turn
results = []
names = []

for name, model in models:
    kfold = model_selection.KFold(n_splits=10, random_state = seed)
    cv_results = model_selection.cross_val_score(model, X_train, y_train, cv=kfold, scoring=scoring)
    print(cv_results)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
[0.91071429 0.91071429 0.96428571 0.91071429 0.89285714 0.91071429
 1.         1.         1.         0.98181818]
SVM: 0.948182 (0.042570)
[1.         0.92857143 0.98214286 0.94642857 0.94642857 0.94642857
 0.98214286 1.         1.         1.        ]
RFC: 0.973214 (0.026786)
```

There is no big change in the results, means we can also find the output after choosing among the high correlated attributes. For, RF this shows the better results.

3. We will also compare the results, after changing some parameters of our algorithms. Firstly, we will check for SVM.

   Syntax:
   class sklearn.svm.SVC(*C=1.0*, *kernel='rbf'*, *degree=3*, *gamma='auto_deprecated'*, *coef0 =0.0*, *shrinking=True*, *probability=False*, *tol=0.001*, *cache_size=200*, *class_weight=None* , *verbose=False*, *max_iter=-1*, *decision_function_shape='ovr'*, *random_state=None*).

Let's try with kernel parameter, so It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable. By default, kernel is 'rbf'. (Check file: AI Project.ipynb)

1. Kernel = 'rbf'

```
In [36]: clf = SVC(kernel = 'rbf')

         clf.fit(X_train, y_train)
         accuracy = clf.score(X_test, y_test)
         print(accuracy)

         0.9714285714285714
```

2. Kernel = 'linear'

```
In [45]: clf = SVC(kernel = 'linear')

         clf.fit(X_train, y_train)
         accuracy = clf.score(X_test, y_test)
         print(accuracy)

         0.95
```

3. Kernel = 'sigmoid'

```
In [53]: clf = SVC(kernel = 'sigmoid')

         clf.fit(X_train, y_train)
         accuracy = clf.score(X_test, y_test)
         print(accuracy)

         0.37857142857142856
```

As we can see the best accuracy is coming, when kernel is in 'rbf' mode.

Secondly, we will check with the parameters of Random Forest.

Syntax:
class sklearn.ensemble.RandomForestClassifier(*n_estimators='warn', criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None*)

We will check for two parameters and rest we will keep default.

1. max_depth represents the depth of each tree in the forest. The deeper the tree, the more splits it has, and it captures more information about the data.
2. n_estimators represents the number of trees in the forest. Usually the higher the number of trees the better to learn the data. However, adding a lot of trees can slow down the training process considerably, therefore we do a parameter search to find the sweet spot. The default value is (10). (Check File: AI Project.ipynb)

Let's take (max_depth = 5, 10, 20) and (N_estimators = 10)

```
In [90]: clf = RandomForestClassifier(max_depth=5, n_estimators = 10)

         clf.fit(X_train, y_train)
         accuracy = clf.score(X_test, y_test)
         print(accuracy)

           0.9714285714285714
```

```
In [100]: clf = RandomForestClassifier(max_depth=10, n_estimators = 10)

          clf.fit(X_train, y_train)
          accuracy = clf.score(X_test, y_test)
          print(accuracy)

            0.9785714285714285
```

```
In [109]: clf = RandomForestClassifier(max_depth=20, n_estimators = 10)

          clf.fit(X_train, y_train)
          accuracy = clf.score(X_test, y_test)
          print(accuracy)

            0.9357142857142857
```

We got the better result keeping (max_depth = 10), now will check for estimators (10,20,40,50).

```
In [100]: clf = RandomForestClassifier(max_depth=10, n_estimators = 10)

          clf.fit(X_train, y_train)
          accuracy = clf.score(X_test, y_test)
          print(accuracy)
```

```
0.9785714285714285
```

```
In [118]: clf = RandomForestClassifier(max_depth=10, n_estimators = 20)

          clf.fit(X_train, y_train)
          accuracy = clf.score(X_test, y_test)
          print(accuracy)
```

```
0.9642857142857143
```

```
In [127]: clf = RandomForestClassifier(max_depth=10, n_estimators = 40)

          clf.fit(X_train, y_train)
          accuracy = clf.score(X_test, y_test)
          print(accuracy)
```

```
0.9857142857142858
```

```
In [136]: clf = RandomForestClassifier(max_depth=10, n_estimators = 50)

          clf.fit(X_train, y_train)
          accuracy = clf.score(X_test, y_test)
          print(accuracy)
```

```
0.9285714285714286
```

So, we are getting best accuracy with (max_depth = 10) and (n_estimators = 40).

Additional Features

I tried some additional features of Python libraries, and I found them interesting.

Let's have a look.

1. Voting Classifier -: The idea behind the VotingClassifier is to combine conceptually different machine learning classifiers and use a majority vote or the average predicted probabilities (soft vote) to predict the class labels. Such a classifier can be useful for a set of equally well performing model in order to balance out their individual weaknesses. (Check File: AI Project.ipynb)
   Firstly, lets compare different machine learning algorithms on our dataset.

```
In [169]: #Define models to train
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
models = []
models.append(('KNN', KNeighborsClassifier(n_neighbors = 3)))
models.append(('SVM', SVC(gamma = 'auto', kernel= 'rbf')))
models.append(('CART', DecisionTreeClassifier()))
models.append(('LR', LogisticRegression()))
models.append(('RFC', RandomForestClassifier(max_depth=5, n_estimators = 40)))

# evaluate each model in turn
results = []
names = []

for name, model in models:
    kfold = model_selection.KFold(n_splits=10, random_state = seed)
    cv_results = model_selection.cross_val_score(model, X_train, y_train, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean()*100, cv_results.std())
    print(msg)
```

```
KNN: 96.068182 (0.017457)
SVM: 95.711039 (0.025473)
CART: 93.918831 (0.019852)
LR: 95.532468 (0.014337)
RFC: 97.136364 (0.014319)
```

   Now look at the result of Voting Classifier.

```
In [170]: from sklearn.ensemble import VotingClassifier

ensemble = VotingClassifier(estimators = models, voting = 'hard', n_jobs = -1)

ensemble.fit(X_train, y_train)

predictions = ensemble.score(X_test, y_test)*100


print("The Voting Classifier Accuracy is: ", predictions)
```

```
The Voting Classifier Accuracy is:  98.57142857142858
```

   As we can see we are getting highest accuracy with the Voting Classifier.

2. Visualize a decision tree from the Random Forest trees using Python export_graphviz. The code for visualizing a tree is:

```
In [184]: from sklearn.ensemble import RandomForestClassifier
          model = RandomForestClassifier(max_depth=5, n_estimators = 50)
          model.fit(X_train, y_train)
          accuracy = model.score(X_test, y_test)
```

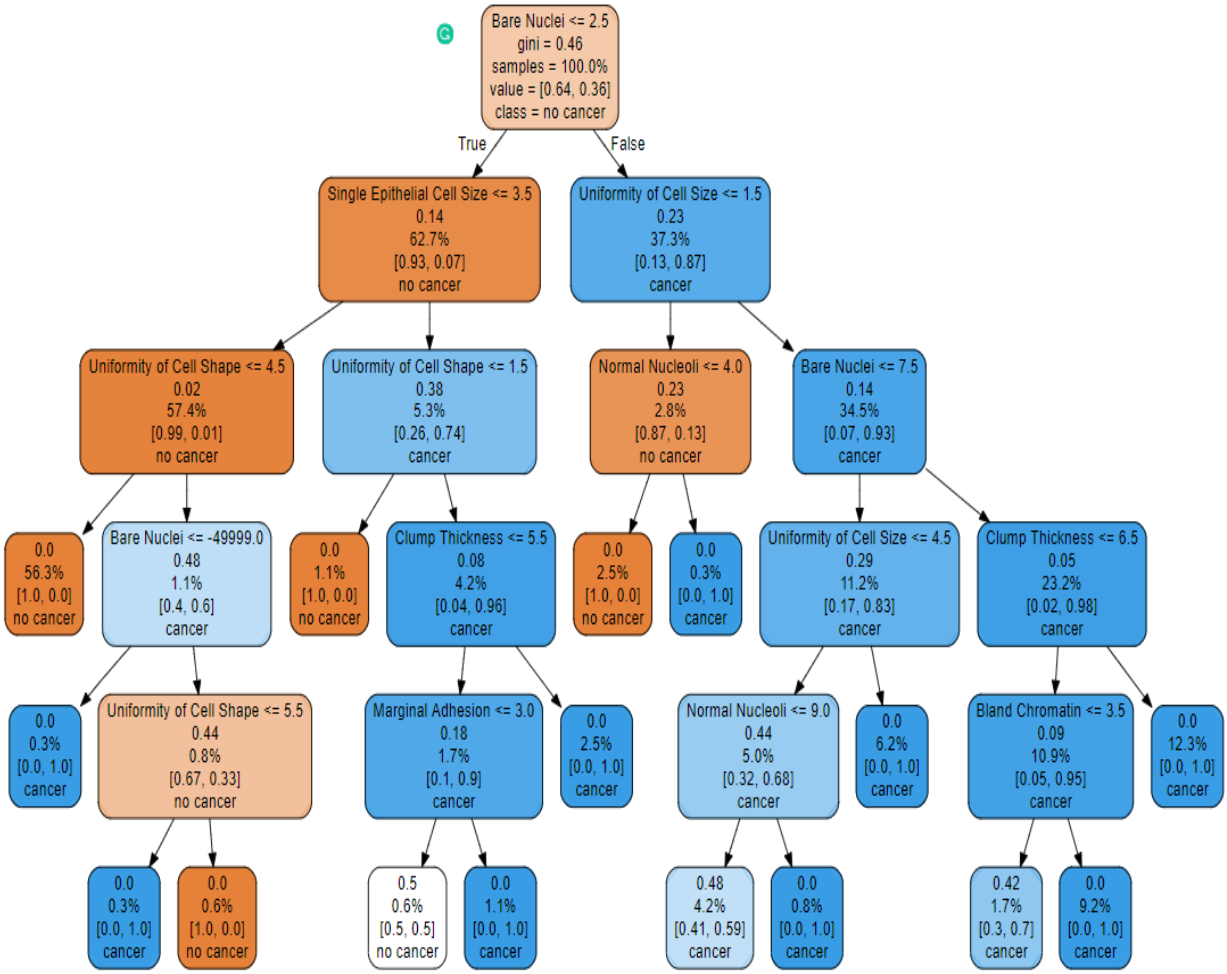
```
In [182]: estimator = model.estimators_[1]
          feature_names = [i for i in X_train.columns]

          y_train_str = y_train.astype('str')
          y_train_str[y_train_str == '2'] = 'no cancer'
          y_train_str[y_train_str == '4'] = 'cancer'
          y_train_str = y_train_str.values
```

```
In [183]: from sklearn.tree import export_graphviz #plot tree
          import graphviz
          export_graphviz(estimator, out_file='tree.dot',
                          feature_names = feature_names,
                          class_names = y_train_str,
                          rounded = True, proportion = True,
                          label='root',
                          precision = 2, filled = True)
          with open("tree.dot") as f:
              dot_graph = f.read()
          graphviz.Source(dot_graph)
```

Here we used export_graphviz and this function generates a GraphViz representation of the decision tree, which is then written into out_file.

Syntax:
sklearn.tree.export_graphviz(*decision_tree*, *out_file=None*, *max_depth=None*, *feature_names=None*, *class_names=None*, *label='all'*, *filled=False*, *leaves_parallel=False*, *impurity=True*, *node_ids=False*, *proportion=False*, *rotate=False*, *rounded=False*, *special_characters=False*, *precision=3).*

The tree will save in dot format and when you visualize, it will look like:



Visualizing a single decision tree can help give us an idea of how an entire random forest makes predictions.

In Random Forest every decision at a node is made by classification using single feature. Plotting a decision tree gives the idea of split value, number of datapoints at every node etc. Considering majority voting concept in random forest, data scientist usually prefers more no of trees (even up to 200) to build random forest, hence it is almost impracticable to conceive all the decision trees. But visualizing any 2–3 trees picked randomly will gives fairly a good intuition of model learning.

Conclusion

In this project we compared two machine learning algorithms in different scenarios for our classification type of dataset. In almost all cases Random Forest gives us better results than SVM in terms of accuracy. We also did parameter tuning of both the algorithms and found the best result of them. There are many additional features are present in Python that can help us in finding best result for our problem, we looked at some of them to have an idea. Support Vector Machine is considered as good algorithms for both small dataset and huge dataset. Random Forest works very well on huge datasets and because of its implicit features, it does not require much input preparation.

Reference

https://scikit-learn.org/stable/

https://www.edureka.co/blog

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

https://scikit-learn.org/stable/model_selection.html

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html

https://towardsdatascience.com/how-to-visualize-a-decision-tree-from-a-random-forest-in-python-using-scikit-learn-38ad2d75f21c

https://www.simplilearn.com/classification-machine-learning-tutorial

http://benalexkeen.com/correlation-in-python/

https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html#sphx-glr-auto-examples-model-selection-plot-confusion-matrix-py