**Titanic Disaster Dataset Report**

* **Task**

The task which we allotted was to process and trained 10 Machine Learning Models from the scikit\_learn library of Python using different evaluation metrics and methodologies on the famous dataset ‘Titanic Disaster Dataset’.

* **Data Preprocessing**

I had used different methodologies in data preprocessing includes which includes

* Removing the missing rows from dataset
* Using Encoding measures
* Using principal component Analysis
* Standardization

We had to delete our missing rows in the dataset as later on models can become over fit or expose to noise (error) in the dataset very quickly. Other option we had is to replace those missing values by taking a majority value or take a mean of values and fill those empty slots with them.

I’ve use a One Hot Encoding (since the data was not nominal) to transform the categorical attributes to numeric attributes as Machine Learning models we’ve trained only accept this kind of inputs

Using Principal Component Analysis (Dimensionality Reduction Technique) was tough decision as there was no Curse of Dimensionality in our dataset but ID’s of passenger wasn’t helping in deciding the target Class to a great extent and the accuracy with PCA was a lot better than without applying PCA.

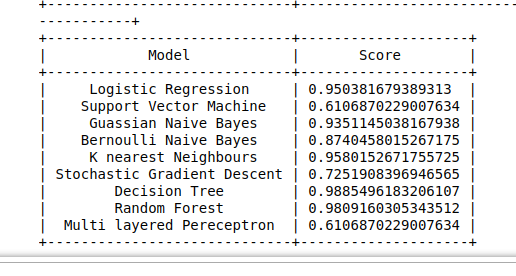
We need to scale our Data (Normalization or Standardization) for algorithms like KNN all the Deep learning ones as if the data is not in the normalize form the attributes with the values of greater scale might be dominant over the attributes with the values of lower scale. This will have led to worst generalization of our model which is opposite to our primary goal.

* **Evaluation Methodologies**

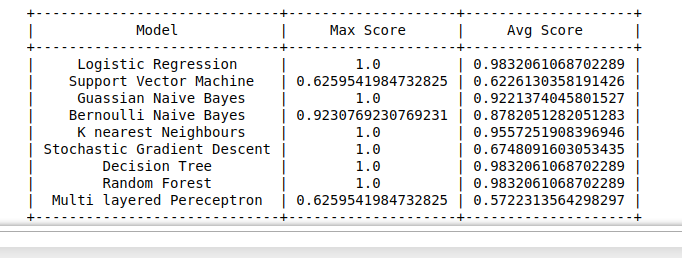
The two methodologies I have used are

* Test split
* K fold Cross Validation

In the first Approach I have used the standard split Ratio of 80-20 to split our data into two sub parts which includes the Training Data and Testing Data respectively. Now I have trained all my models purely on this methodology first and the result I got are shared below



Now moving onto our second Approach which is to use K Fold Cross Validation with a standard value of K=10. So what K fold does is it creates k folds in each iteration depending upon the value of k which we have given to our machine and then it trains our model on k-1 folds and test our model on 1 fold. This approach helps us highly in generalizing our model better as it is training on all the dataset with replacements instead of splitting the data and then training on that dataset in single iteration which we had done in previous approach. It outputs the Array of K values as each value represents result of single fold. Result which were given are as followed



* **Evaluation Metrics**

Evaluation metrics which are used in this project are

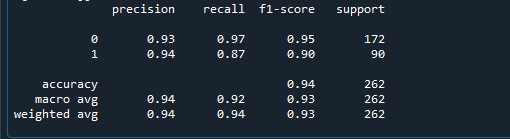
* Recall (Sensitivity)
* Precision (False Positive Rate)
* F1
* Accuracy

It’s not the best practice to use Accuracy in each case as sometimes there is ratio of unbalancing in the dataset which Accuracy is unable to define and its suffers to great deal.

Recall tells us about true positives which were total in dataset and out of these how many does our model was able to predict.

As the name suggest Precision tells us about the True Positives out of All positives (true positive or false positive) our model predicted.

F1 score tells us the weighted mean of both of these by using the Harmonic mean which penalizes the F1 score if one of them Accuracy or Recall is decreasing and the other one is increasing and vice versa. We can Classification Report to explore these.



* **Machine Learning Algorithms**

Machine Learning Algorithms used in this Project is

* **Logistic Regression**

Don’t get confused by its name! It is a classification not a regression algorithm. It is used to estimate discrete values (Binary values like 0/1, yes/no, true/false) based on given set of independent variable(s). In simple words, it predicts the probability of occurrence of an event by fitting data to a [logit function](https://en.wikipedia.org/wiki/Logistic_function). Hence, it is also known as **logit regression**. Since, it predicts the probability, its output values lie between 0 and 1 (as expected).

* **Decision Tree**

This is one of my favorite algorithm and I use it quite frequently. It is a type of supervised learning algorithm that is mostly used for classification problems. Surprisingly, it works for both categorical and continuous dependent variables. In this algorithm, we split the population into two or more homogeneous sets. This is done based on most significant attributes/ independent variables to make as distinct groups as possible.

* **Support Vector Machine**

It is a classification method. In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate.

For 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**)

* **Naïve Bayes**

It is a classification technique based on [Bayes’ theorem](https://en.wikipedia.org/wiki/Bayes%27_theorem) with an assumption of independence between predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, a naive Bayes classifier would consider all of these properties to independently contribute to the probability that this fruit is an apple.

* **KNN**

It can be used for both classification and regression problems. However, it is more widely used in classification problems in the industry. K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases by a majority vote of its k neighbors. The case being assigned to the class is most common amongst its K nearest neighbors measured by a distance function.

These distance functions can be Euclidean, Manhattan, Minkowski and Hamming distance. First three functions are used for continuous function and fourth one (Hamming) for categorical variables. If K = 1, then the case is simply assigned to the class of its nearest neighbor. At times, choosing K turns out to be a challenge while performing kNN modeling.

* **Random Forest**

Random Forest 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).

* **Artificial Neural Networks**

[Deep Learning](http://www.dataversity.net/deep-learning-updates-machine-learning-deep-reinforcement-learning-limitations/) uses Neural Networks to imitate how the human brain works. Thousands of interconnected artificial neurons are arranged in multiple processing layers. (Two layers are common with other Machine Learning systems.) The additional processing layers provide higher-level abstractions, offering better classifications and more accurate predictions. [Deep Learning](http://www.dataversity.net/brief-history-deep-learning/) is ideal for working with Big Data, voice recognition, and conversational skills.

Artificial neurons often [have a weight](https://singularityhub.com/2018/02/26/putting-ai-in-your-pocket-mit-chip-cuts-neural-network-power-consumption-by-95/#sm.001khz6c114xid9cx5d25wg0wdjqx) which adjusts as the learning process proceeds. The weight increases or decreases the strength of the signal at a connection. Artificial neurons may have a threshold such that only if the aggregate signal crosses that threshold is the signal sent.

* **Stochastic Gradient Descent**

Stochastic Gradient Descent (SGD) is a simple yet very efficient approach to fitting linear classifiers and regressors under convex loss functions such as (linear) [Support Vector Machines](https://en.wikipedia.org/wiki/Support_vector_machine) and [Logistic Regression](https://en.wikipedia.org/wiki/Logistic_regression). Even though SGD has been around in the machine learning community for a long time, it has received a considerable amount of attention just recently in the context of large-scale learning.

SGD has been successfully applied to large-scale and sparse machine learning problems often encountered in text classification and natural language processing. Given that the data is sparse, the classifiers in this module easily scale to problems with more than 10^5 training examples and more than 10^5 features.