1. **DATASET**

We have used the NASA Metrics Data Program (MDP) datasets for prediction of software faults. Faults can occur in software even when the development is finished due to various human errors and shortcomings of the system used for software development. These faults can cause failure in the actual implementation of the software at the user’s end. The existing software fault prediction systems have done a helpful job in making error-free software. The systems are still unable to achieve high accuracy and robust quality check. Due to the ever-expanding scope of software, a system which can predict the faultiness of software with maximum accuracy and minimum manual interaction is mandatorily required.

The MDP dataset is a standard dataset prepared by NASA through collection of software data from various sources. It is a binary dataset which describes software based on various numerical measures. In other words, it contains the attributes or features which describe software, and a binary class which labels any instance of software as faulty or not faulty. This dataset is widely used all over the world for making software fault prediction systems and conducting research in this field. Before applying prediction models on the dataset, it is important to do a quality check. As the original dataset, which was made available by the organization, struggled with its quality due to a number of issues, such as duplicate data points, missing values, etc, it was necessary for researchers to build preprocessing algorithms for cleaning of data. Due to its heavy use and demand of the researchers, the MDP dataset evolved through time and it is now available in a preprocessed and cleaned state, which we have used in our experiments. This cleaned data is completely suitable to use with any prediction technique without any preprocessing. The cleaned MDP dataset is described in Table 1.

The validity of the preprocessing algorithm used to clean the dataset is highly important. It helps us in deciding if the data is suitable for using with various prediction models or not. We have used the data based on the quality issues described in the paper “Data Quality: Some Comments on the NASA Software Defect Datasets.” It removes the duplicate data points from the dataset and adds the missing data values.

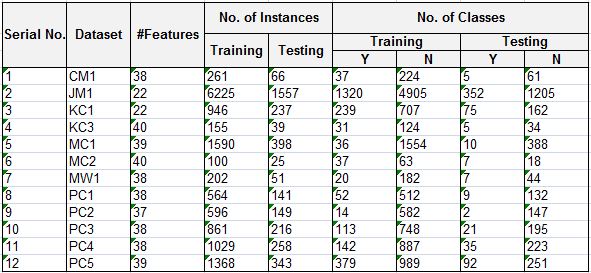


Table : *The table shows the description of the cleaned and preprocessed dataset from the NASA Metrics Data Program. The dataset have been divided into two parts- for training and testing.*

As the above table shows, we have divided the data into two parts – training data and testing data. The ratio we have used is 4:1. Since the dataset is a binary dataset, it labels the instances with two classes, i.e. either ‘Y’ or ‘N’. The number of each class corresponding to the training and testing data is also helpful if we know it in advance. These formulations of data in a table format are necessary for choosing the prediction methods and also implement them. After knowing our dataset, we should choose the prediction methods to apply from existing machine learning techniques.

1. **LITERATURE REVIEW**

A classifier can be used with the MDP data, which will identify some pattern to label the new data. There are several classifiers which can be applied. The selection of a classifier depends on the efficiency and total training and testing time.

**Random Forest**

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. Basic parameters to Random Forest Classifier can be total number of trees to be generated and decision tree related parameters like minimum split, split criteria etc.

Referenced from [*https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html*](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html).

**K-Nearest Neighbours**

The principle behind nearest neighbor methods is to find a predefined number of training samples closest in distance to the new point, and predict the label from these. The number of samples can be a user-defined constant (k-nearest neighbor learning), or vary based on the local density of points (radius-based neighbor learning). The distance can, in general, be any metric measure: standard Euclidean distance is the most common choice.

Referenced from [*https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html*](https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html)*.*

**AdaBoost:**

An AdaBoost classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.

Referenced from [*https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html*](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html)*.*

**Gaussian Naïve Bayes:**

In Gaussian Naive Bayes, continuous values associated with each feature are assumed to be distributed according to a **Gaussian distribution**. Naive Bayes classifiers are a collection of classification algorithms based on **Bayes’ Theorem**. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.

Referenced from [*https://scikit-learn.org/stable/modules/generated/sklearn.naive\_bayes.GaussianNB.html*](https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html)*.*

**Linear Support Vector:**

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. Then, we perform classification by finding the hyper-plane that differentiate the two classes very well (look at the below snapshot). Support Vectors are simply the co-ordinates of individual observation. Support Vector Machine is a frontier which best segregates the two classes (hyper-plane/ line).

Referenced from [*https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html*](https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html)*.*

1. **EXPERIMENT**

We considered all of the above mentioned algorithms and implemented those using Python. The classes ‘Y’ and ‘N’ were converted to floating point values, i.e. 1.0 and 0.0 respectively, in order to make the classes suitable for use with the library based machine learning classifiers. We checked the accuracy of these algorithms on each dataset to find a relation between type of algorithm and the dataset. We also calculated the mean accuracy for each algorithm. The results of the experiment are shown in Table 2.

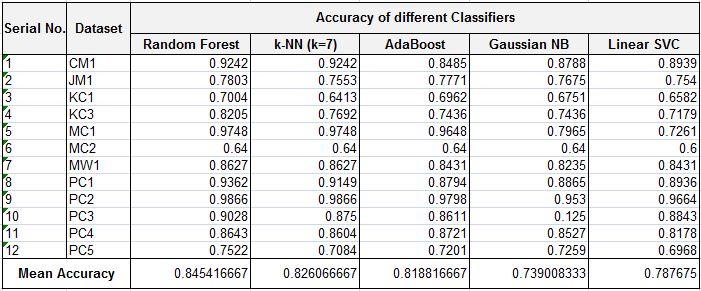


Table : The table shows acuuracy values for each algorithm on different dataset, along with the mean accuracies.

The random forest classifier gave the best accuracy value. While, the Gaussian naïve bayes classifier was low in accuracy, its computation time for training and testing was significantly higher.