Breast Cancer cell classification (Benign or Malignant)

Topics in Machine Learning, Fall 2019
CS-271
Machine Learning Project
Prof. Mark Stamp

By,
Rakesh Nagaraju (014279304)

Fall 2019
Final Project Report

1. Introduction

According to the National Cancer Institute "The **most common type of cancer** on the list is breast **cancer**, with 271,270 new cases expected in the United States in 2019". Also according to global statistics, breast cancer represents the majority of cases of cancer-related deaths, making it a significant public health problem in today's society. Thus we can say that Breast Cancer is a worldwide problem that needs to be solved. Diagnosing cancer at the early stages can improve the prognosis, chance of survival, significantly as it can, in turn, promote the timely treatment of patients. Additionally, accurate classification of Benign tumors can prevent patients from undergoing unnecessary treatments. Thus, the correct diagnosis of the disease and the right classification of Breast cancer cells as either Malignant or Benign is a subject of research.

Because of its unique advantages such as Identifying trends and patterns, continuous improvement, no human intervention, handling multidimensional data, we can use Machine Learning algorithms for classifying breast cancer cell and forecast modeling.

In this project, seven different Machine Learning algorithms such as Support Vector Machine(SVM), Logistic Regression, K-nn, Random Forest, Decision Tree, Naive Bayes and also a Neural Network are implemented on a breast cancer dataset, to train and predict the model for cell classification(Malignant and Benign).To avoid overfitting and underfitting, the dataset is divided into n-cross folds(default fold = 5).

Accuracy, Area under the curve(AUC) for PR curves and ROC curves are also calculated for every algorithm for each fold and the mean results are compared to see which algorithm performs better.

2. Dataset Information:

The dataset used in this Project is "Wisconsin Breast Cancer Dataset". This dataset is publicly available and was created by Dr. William H. Wolberg, a physician at the University Of Wisconsin Hospital at Madison, Wisconsin, USA. For this project, the dataset is a '.csv' file and it was downloaded from Kaggel[1].

This dataset was created by Dr. Wolberg who used fluid samples, taken from patients with solid breast masses and an easy-to-use graphical computer program called Xcyt.

This program uses a curve-fitting algorithm, to compute ten features from each one of the cells in the sample, then it calculates the mean value, extreme value and standard error of each feature for the image, returning a 30 real-valued vector.

2.1 Attribute Information:

The various attributes of the dataset are as follows : ID number, Diagnosis (M = malignant, B = benign), radius (mean of distances from center to points on the perimeter), texture (standard deviation of gray-scale values), Perimeter, Area, Smoothness (local variation in radius lengths), Compactness (perimeter² / area — 1.0), Concavity (severity of concave portions of the contour), Concave points (number of concave portions of the contour), Symmetry, Fractal dimension ("coastline approximation" — 1).

The mean, standard error and "worst" or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features. For example, field 3 is Mean Radius, field 13 is Radius SE, field 23 is Worst Radius.

2.2 Data Exploration:

Before applying Machine Learning concepts, we will explore and analyze the data to check for any junk values, null values, etc, and then change those to valid integers.

In this project, the python pandas head() method is used to view the data, as seen in figure1.

	id	diagnosis	...	fractal_dimension_worst	Unnamed: 32
0	842302	M	...	0.11890	NaN
1	842517	M	...	0.08902	NaN
2	84300903	M	...	0.08758	NaN
3	84348301	M	...	0.17300	NaN
4	84358402	M	...	0.07678	NaN

Figure 1

The data set contains 569 rows and 32 columns, out of the 569 persons, 357 are labeled as B (benign) and 212 as M (malignant). Except for the first two columns, the rest of the column contains the features. The first and second column holds the patient ID and the correct label of diagnosis.

Summary of the Dataset

Dataset	569 (rows), 32 (columns)
Column 1	Patient ID's (Total: 569)
Column 2	Diagnosis('1'- Malignant, '0'- Benign)
Column 3 - 32 (Features)	Numeric(1-10)
Class distribution	357 Benign (63%) and 212 Malignant (37%).

Next, we visualize the data to better understand the data distribution of the features by finding any missing or null data points of the data set, if there is any, using the pandas function "dataset.null().sum()" and "dataset.isna().sum()" as shown in below figure 2.

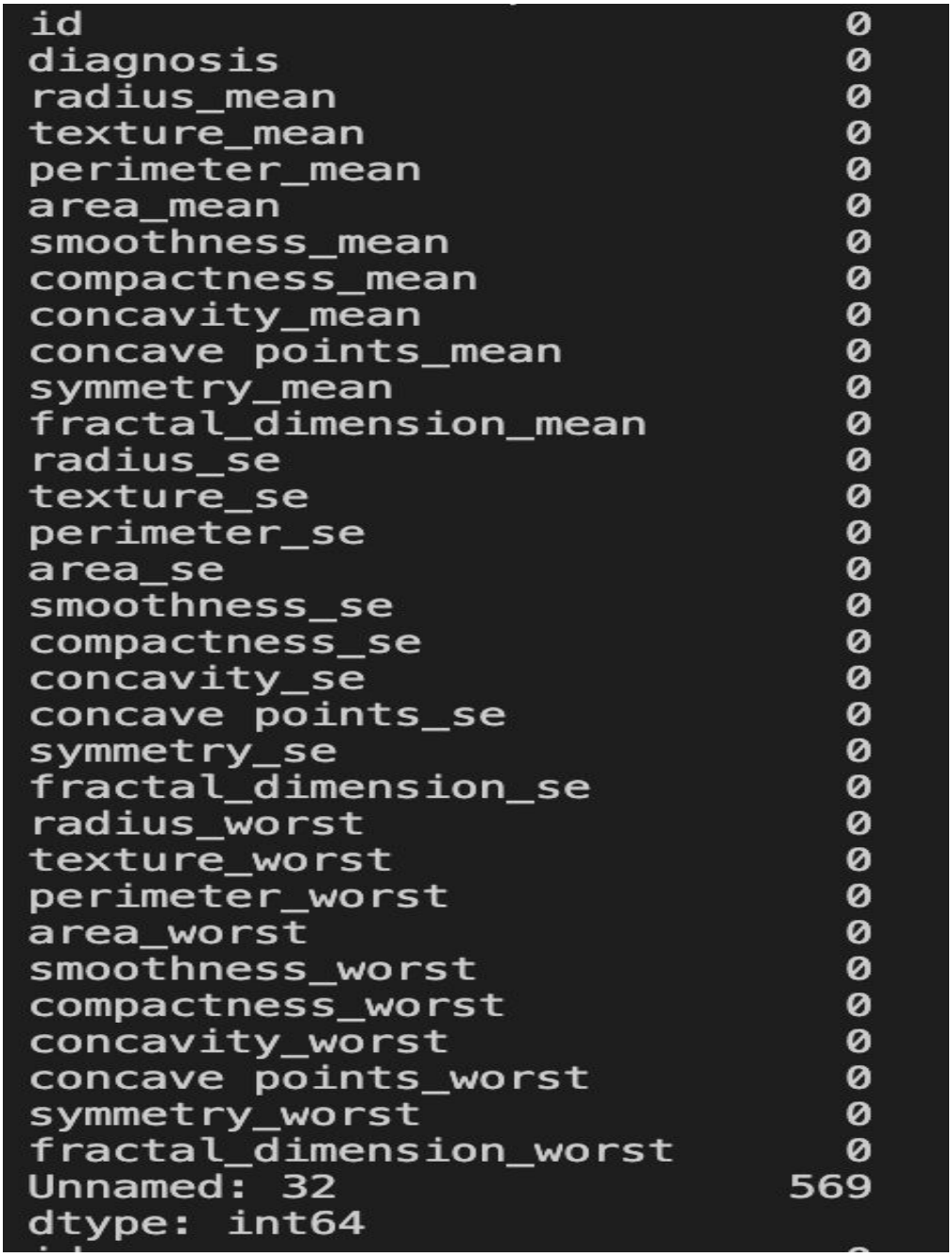

id 0
diagnosis 0
radius_mean 0
texture_mean 0
perimeter_mean 0
area_mean 0
smoothness_mean 0
compactness_mean 0
concavity_mean 0
concave points_mean 0
symmetry_mean 0
fractal_dimension_mean 0
radius_se 0
texture_se 0
perimeter_se 0
area_se 0
smoothness_se 0
compactness_se 0
concavity_se 0
concave points_se 0
symmetry_se 0
fractal_dimension_se 0
radius_worst 0
texture_worst 0
perimeter_worst 0
area_worst 0
smoothness_worst 0
compactness_worst 0
concavity_worst 0
concave points_worst 0
symmetry_worst 0
fractal_dimension_worst 0
Unnamed: 32 569
dtype: int64

Figure 2

Some data might contain Categorical data. Categorical data are variables that contain label values other than numeric values. We use Label Encoder, a python scikit-learn library to convert categorical data into numbers which are better understood by the model as seen in below figure 3.

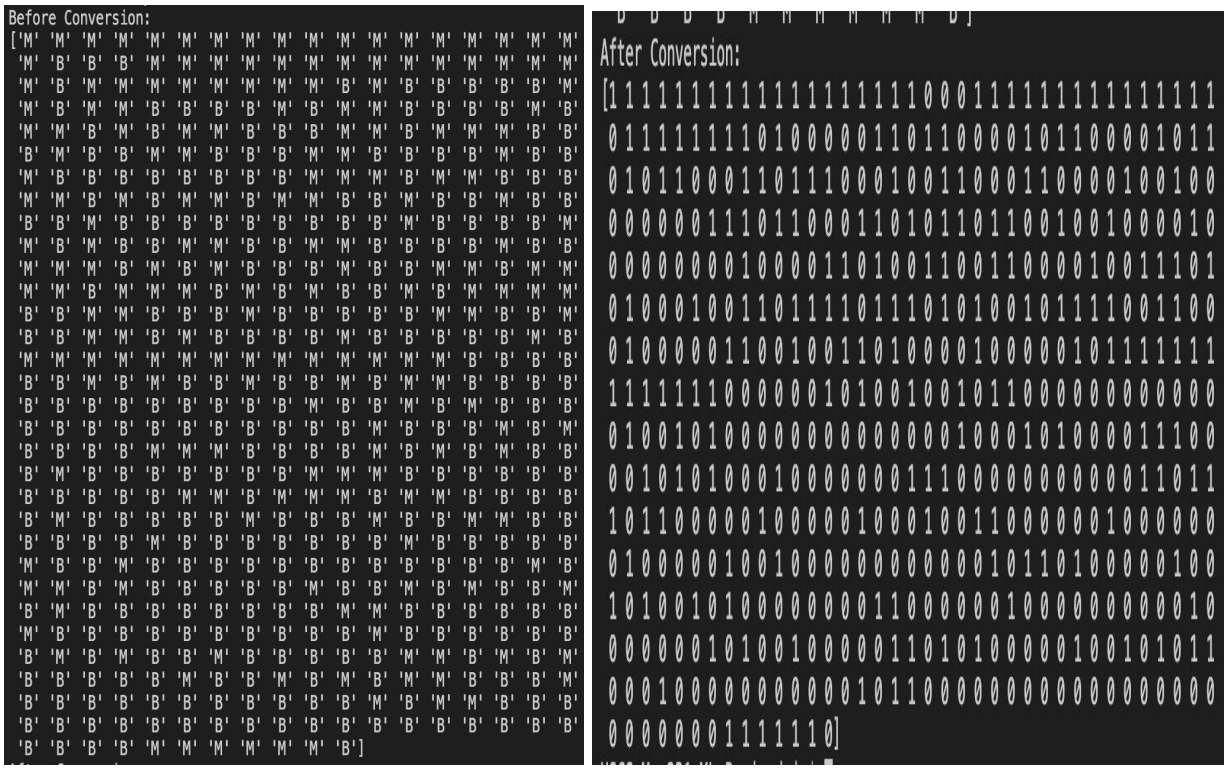


Figure 3

Feature Scaling: The dataset might also contain features varying in magnitudes, units, and range. To bring all features to the same level of magnitudes, scaling is done to fit a scale of 0-1.
StandardScaler method from the SciKit-Learn library is used to achieve this which can be seen in figure 4.

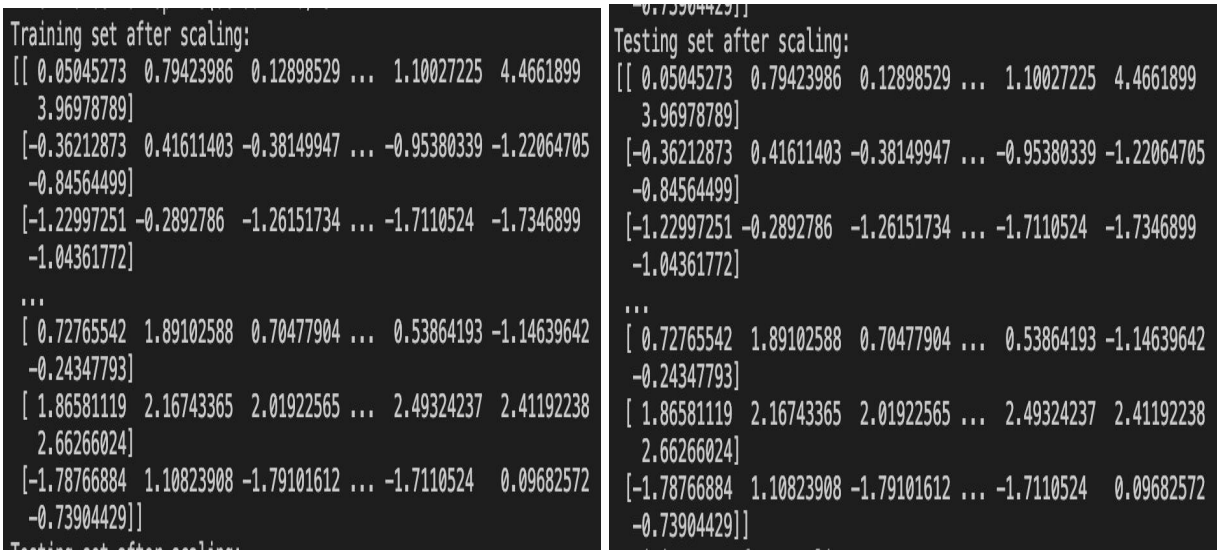
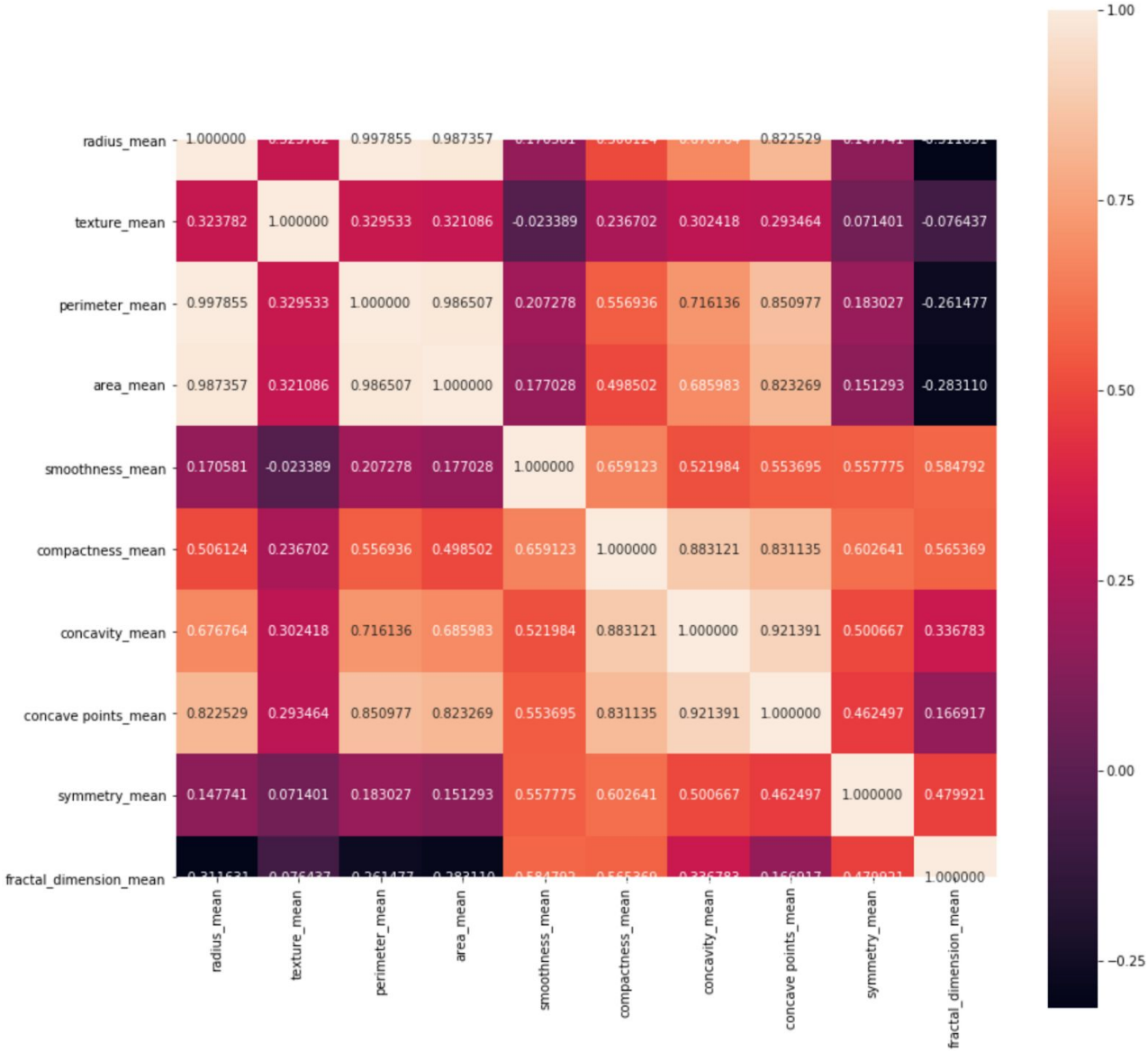
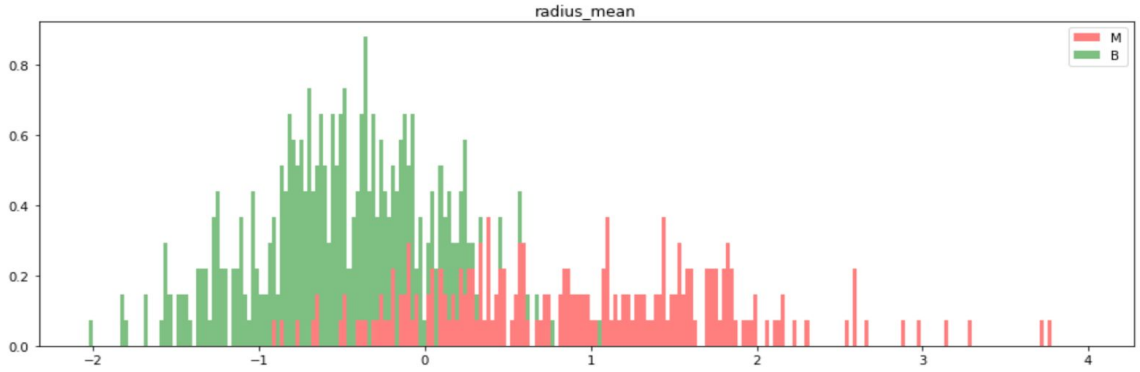
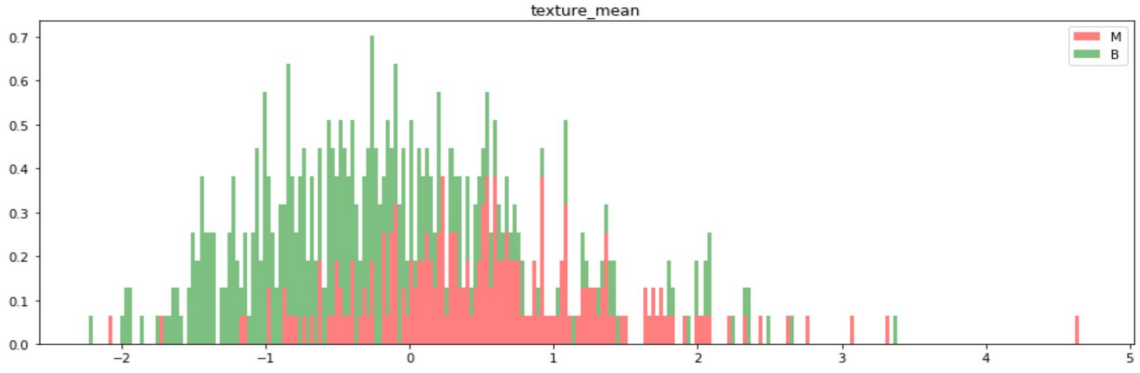


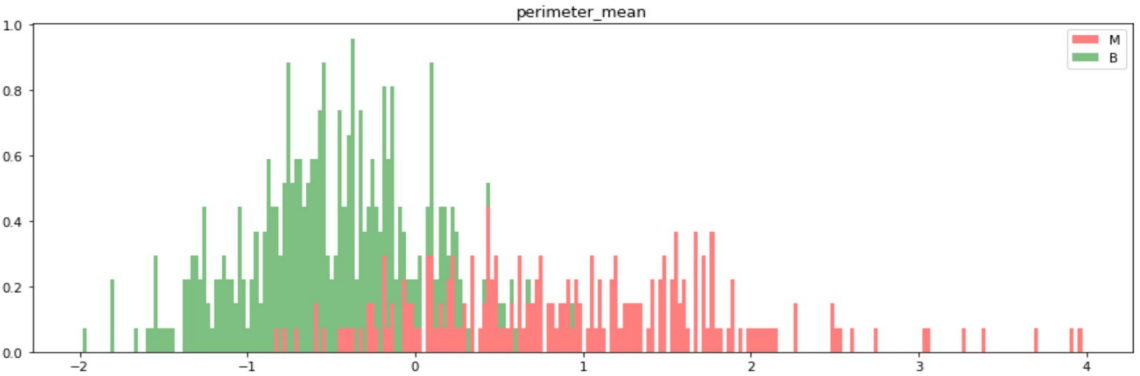
Figure 4

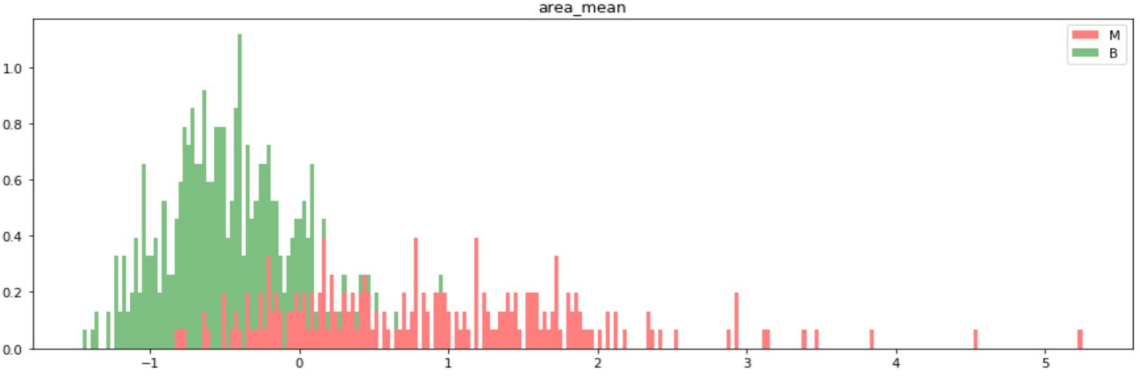
Additionally, we use pandas and seaborn libraries to construct a heatmap between mean features and diagnosis, where darker cells indicate Benign samples and lighter cells indicate Malware samples. Also, we split the dataset into malignant and benign and, then, we plot the histogram of these features which can be seen in below figure 5, where we can see the green bars as Benign and red bars as Malware..

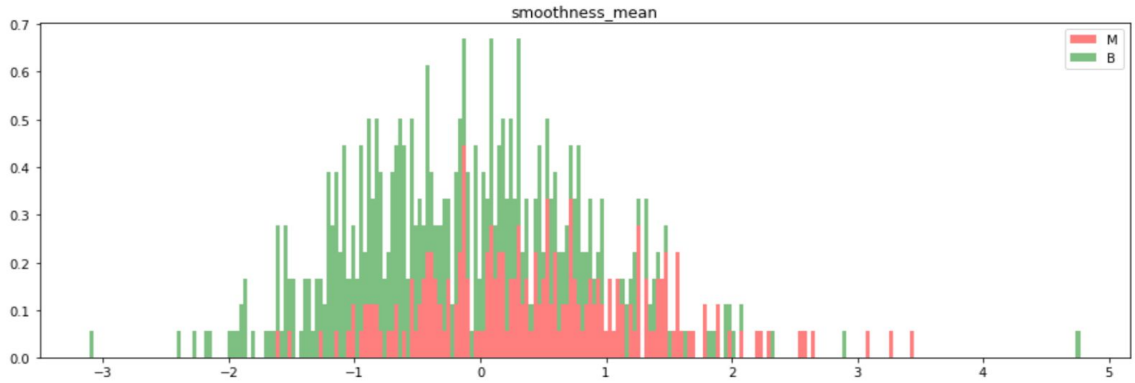


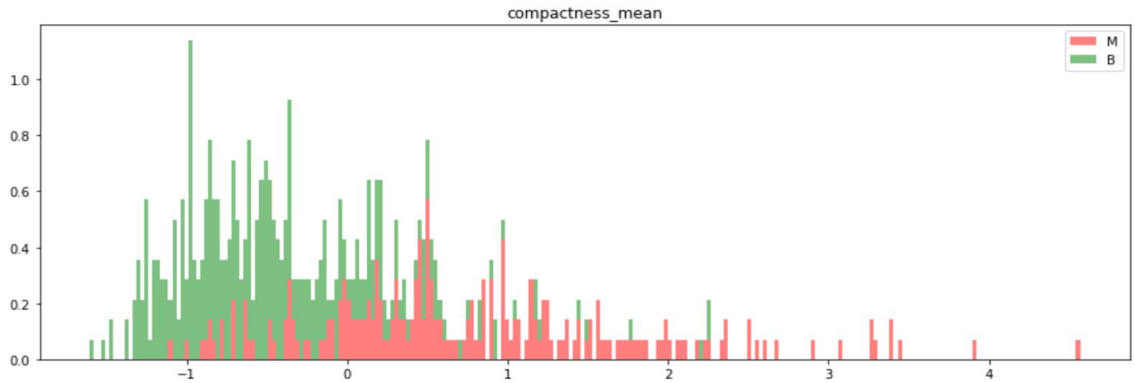

radius_mean
M
B

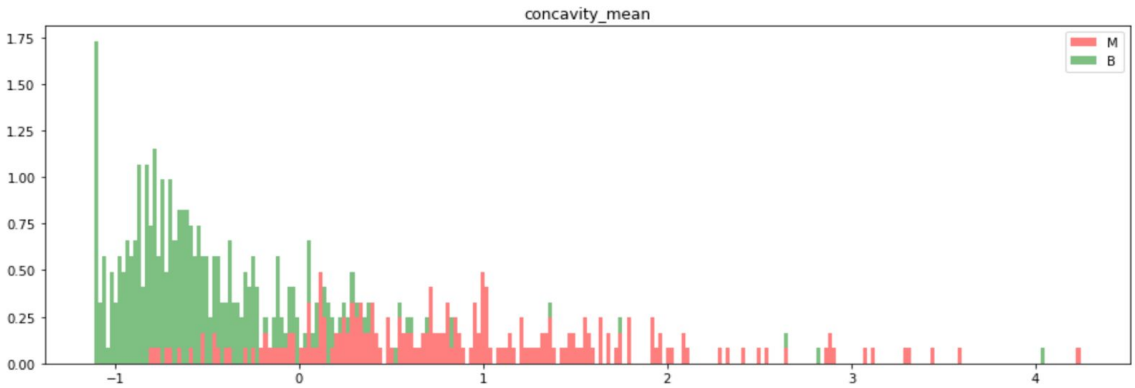

texture_mean
M
B

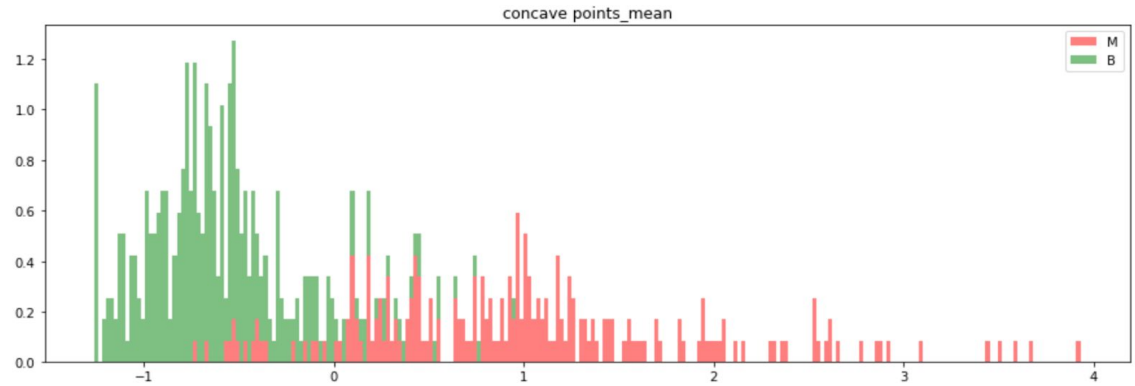

perimeter_mean
M
B


area_mean
M
B


smoothness_mean
M
B


compactness_mean
M
B


concavity_mean
M
B


concave points_mean
M
B

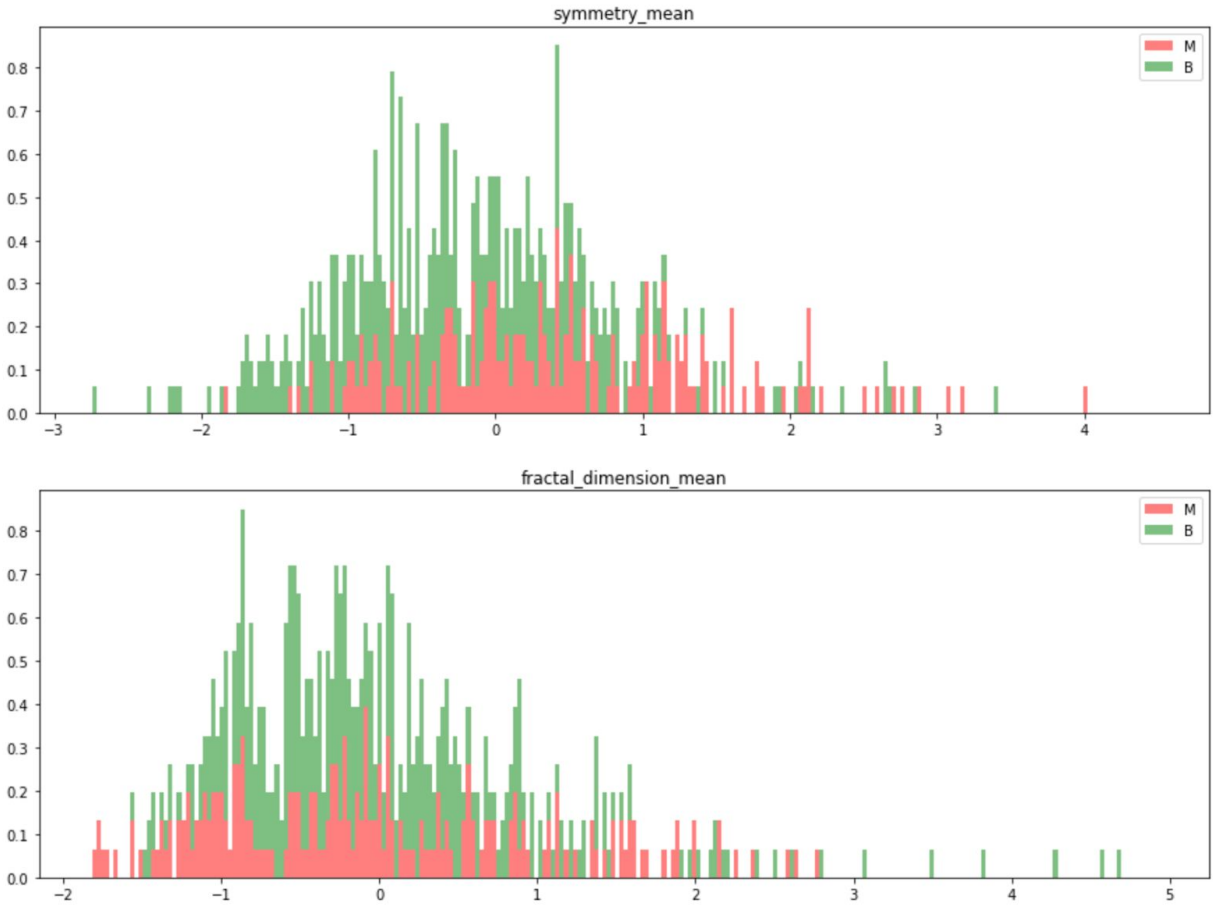


Figure 5

2.3 Splitting the dataset

Now, we have to split the data into training data and test data. In the training set, the output is known to the model and it learns on the data. Next, we have the test dataset where the output is not known to the model, and we test the trained model to obtain predictions.

Also, we have to take care of Overfitting (it means that the model trained, has trained "too well" and is now fit too closely to the training dataset) and Underfitting (it means that the model does not fit the training data and therefore misses the trends in the data). which usually happens when the data is not generalized.

In order to generalize the data, k-cross fold is implemented using the python sklearn KFOLD() method where we split our data into k subsets, and train on k-1 one of those subsets. In this Project, the default k is 5 and also the user is given an option to specify a valid number for k.

In the case of 5 fold validation, the dataset is divided into 5 sets and in each fold, one set is taken as a test set and the remaining sets are taken as a training set. The model is trained on the training set and evaluated on the test set.

3. **Implementation**

Once, the train and test set are obtained, we train a model on training data and test the model to obtain predictions on the test data for each fold.

3.1 Classification Techniques:

In this Project the following Machine Learning Algorithms models are implemented:

Support Vector Machine (SVM): Support Vector Machine is a supervised learning method and is used for binary classification and fits this project requirement of classifying cells as Malignant/Benign.
SVM runs on 4 main ideas: Separating hyperplane (to separate training data into two classes), Maximizing minimum separation between classes (choosing hyperplane such that to maximize the margin), working in a higher-dimensional space (project data to higher dimensional space) and Kernel trick (used to project data into higher dimension).
We are using python's skearn SVC packages to implement the support vector machine.

We use three different kernels:
Linear kernel: K(Xi,Xj) = Xi.Xj, where Xi and Xj are two input vectors.
Polynomial kernel: K(Xi,Xj) = (Xi.Xj + 1)^p, where Xi and Xj are two input vectors, p value is taken 3(default).
Gaussian radial-basis function (RBF) kernel: K(Xi,Xj) = exp(-(Xi – Xj).(Xi – Xj)/(2(σ^2))) , where Xi and Xj are two input vectors and gamma is set to auto. For all these the Regularisation parameter C is kept as constant. We have to choose C > 0, (because of softer the margin, the more errors we allow in training) and in this project, C is kept as a constant value of 3 and the random state as 0.

K-Nearest Neighbor (k-NN): The k-NN algorithm is a supervised learning technique and hence requires labeled training data and can be used for classification, which meets the project requirements. In k-NN, we classify a sample based on a majority vote of the k nearest samples in the training set. Another advantage of k-NN is that no training is required.
In this implementation, we keep k value as 5, p is set to 2(equivalent to using Euclidean distance), metric used is 'minkowski'.

Decision Tree: It creates a Tree that is used to make decisions, kind of like a flow chart. Each node is a test condition and each branch is an outcome of the test represented by the corresponding node.
Criterion is the function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.
In this implementation, we choose criterion as "entropy" and the random state as 0 and the rest of the parameters are kept as default.

Random Forest: Random Forest (RF) is a generalization of a decision tree. This training algorithm uses heuristic to do smart bagging.
In this implementation, we choose criterion as "entropy", random state as 0 and estimators as 10(default) with the rest of the parameters kept as default.

Naive Bayes: Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. In this implementation, we use the sklearn GaussianNB() package with all parameters kept as default.

Logistic Regression: Logistic regression uses an equation as the representation similar to linear regression. Input values X are combined linearly using weights to predict an output value Z. A key difference from linear regression

is that the output value being modeled is a binary value (0 or 1) rather than a numeric value. In this implementation, we keep penalty as 'l2', solver as 'lbfgs', random_state as 0 and max_iter = 1000.
Solver is taken as ‘lbfgs’ after comparing the accuracy of the models trained on other solvers: ‘sag’, ‘saga’ and ‘newton-cg’

Neural Network: Neural Network is something similar to a biological neuron. It consists of hidden layers. Forward Pass is used to determine weights and backpropagation is used to update the determined weights. Both Forward Pass and backpropagation are repeated through multiple iterations with loss calculated and the weights are updated for each epoch. Backpropagation relies on gradient descent. In our project, we are using a Multilayer Perceptron(MLP).
In our implementation, we are using the sklearn Keras package. In particular, a Sequential() model is implemented, which is a linear stack of layers. We are using 2 hidden layers, one input layer, and one output layer. We are calculating the Stochastic Gradient Descent SGD with a learning rate as 0.01, decay as 1*e-6, momentum as 0.9 and use it as an optimizer. For the first 3 layers, the activation function used is the Rectified Linear unit (ReLu) and for the output layer, “sigmoid” is used. The loss used here is the mean squared loss.

4. **Results:**

Once, the model is trained, the trained model is tested on the test dataset and the predictions are obtained. Next, we have to check how accurate the predictions are. For this, we have to find the True positives (correctly identified), False positives (incorrectly identified), True negatives (correctly rejected) and False negatives (incorrectly rejected). Sensitivity is the correctly identified classes and Specificity is the correctly rejected classes, as they should be. Then balance accuracy is calculated as the ratio between the sum of true positive rate, true negative rate and constant 2.
Accuracy = (TPR+TNR) / 2
To implement this, we are using the python sklearn balanced_accuracy_score, classification_report packages. Classification_report gives us true positives, false positives, true negatives, and false negatives. Balanced_accuracy_score gives us the accuracy score of the model.
For each fold, accuracy is calculated and in the end, the mean accuracy is calculated for that particular algorithm.

With the TPR and FPR know, we now plot the ROC curve for each fold and find the area under the curve. Then, we plot the mean ROC curve of all the folds combined and determine the mean area under the curve.

Next, we calculate recall (ratio of correctly classified as positive to the total number of actual positive cases) and precision(ratio of correctly classified positive to the total number that were classified as positive).
With the recall and precision known, we now plot the PR curve for each fold and also find the area under the curve. Then, we plot the mean PR curve of all the folds combined and determine the mean area under the curve.

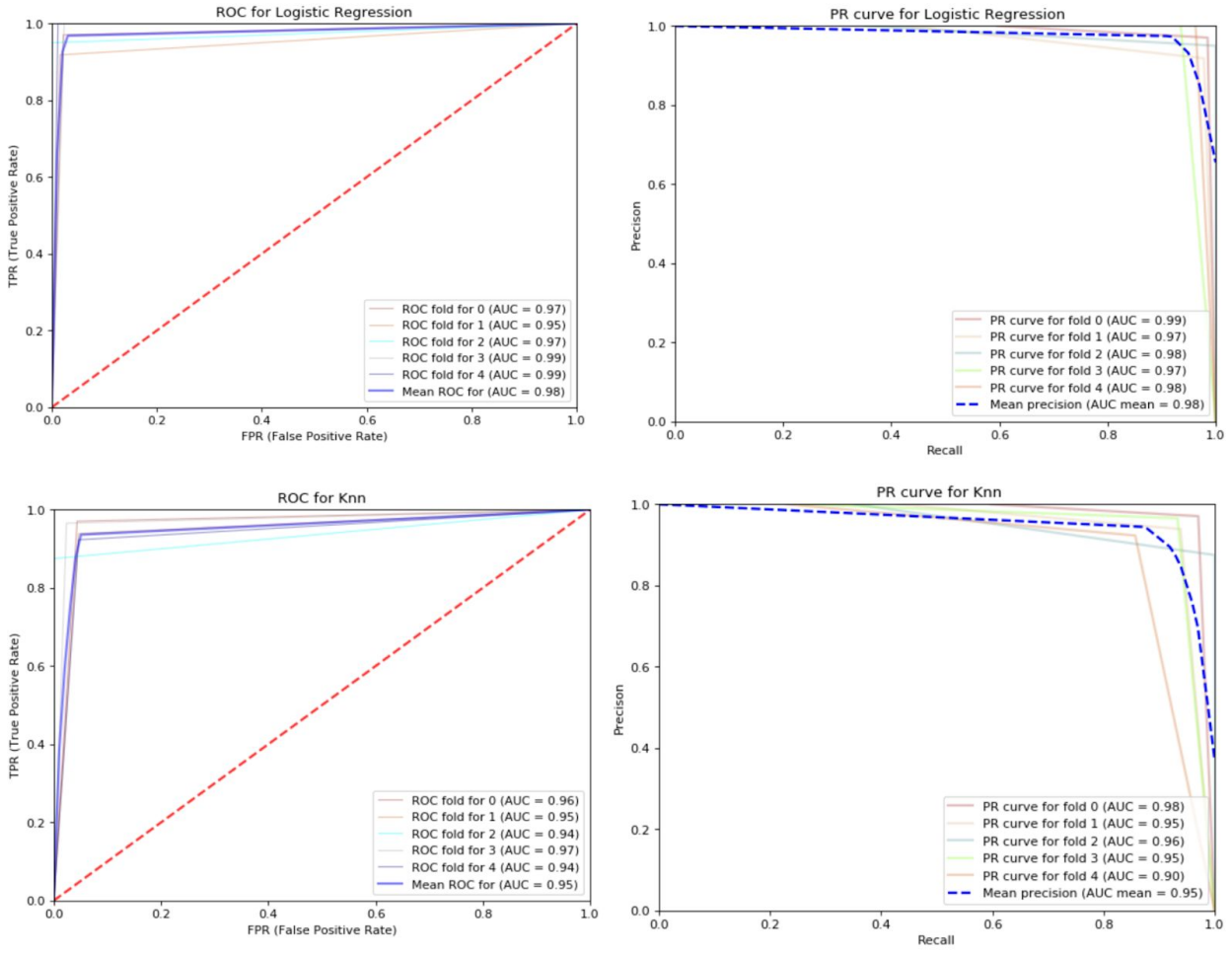
The Accuracy results of all the algorithms can be seen below in table-2: We see that Logistic Regression performs better than other algorithms.

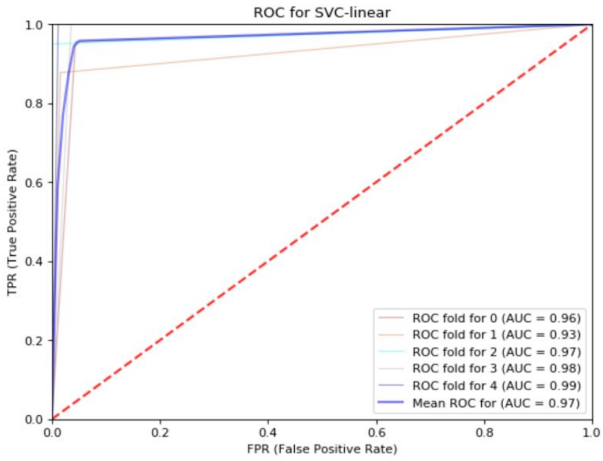
Algorithm	Mean Accuracy(%)
Logistic Regression	97.71
K-NN	95.78
SVC - linear	96.84
SVC - RBF	97.54
SVC - polynomial	93.15

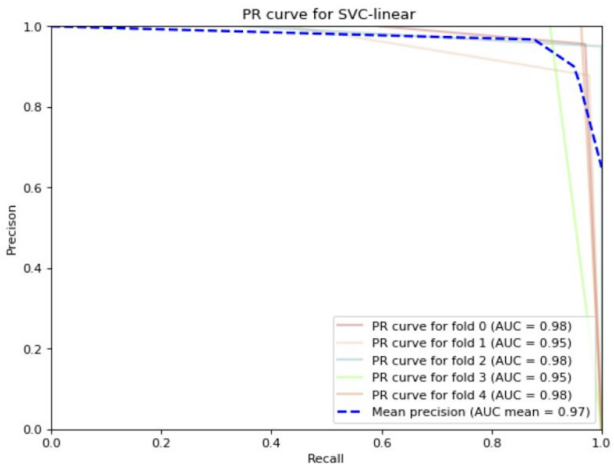
Naive Bayes	92.96
Decision Tree	94.19
Random Forest	94.55
Neural Network	96.68

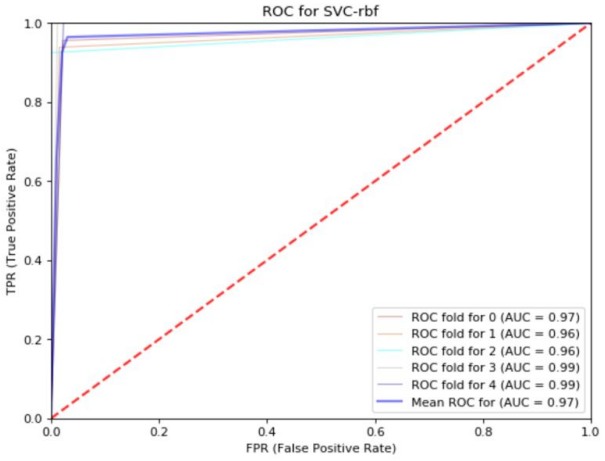
Table 2

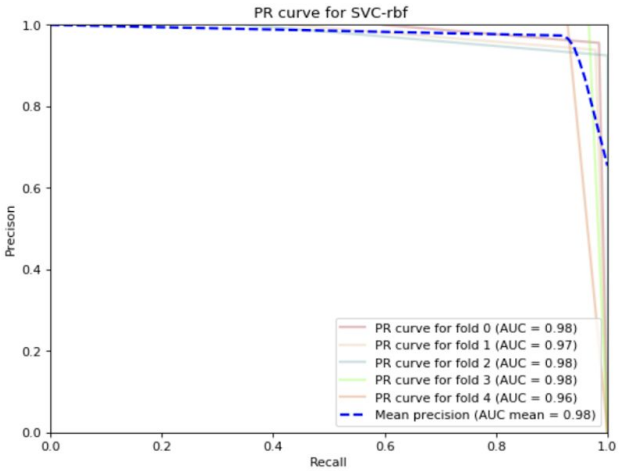
The ROC curves and PR curves for all the algorithms implemented are as shown below:

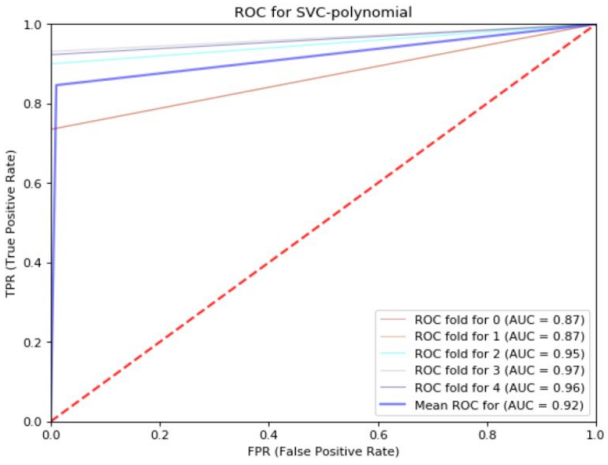


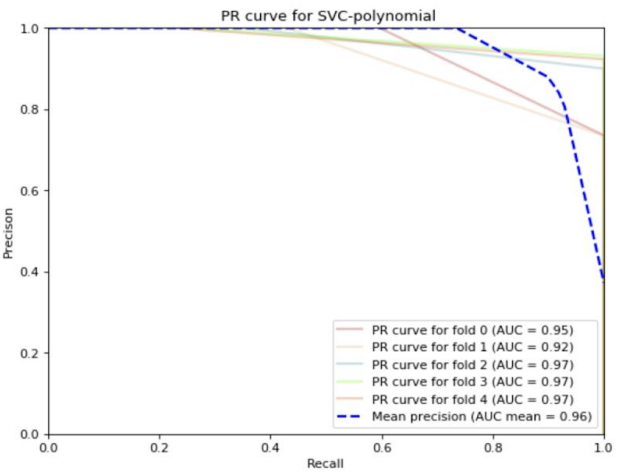

ROC for SVC-linear
TPR (True Positive Rate)
FPR (False Positive Rate)
ROC fold for 0 (AUC = 0.96)
ROC fold for 1 (AUC = 0.93)
ROC fold for 2 (AUC = 0.97)
ROC fold for 3 (AUC = 0.98)
ROC fold for 4 (AUC = 0.99)
Mean ROC for (AUC = 0.97)

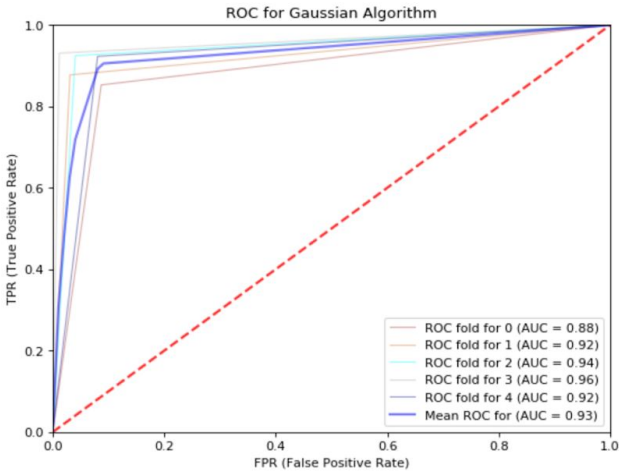

PR curve for SVC-linear
Precison
Recall
PR curve for fold 0 (AUC = 0.98)
PR curve for fold 1 (AUC = 0.95)
PR curve for fold 2 (AUC = 0.98)
PR curve for fold 3 (AUC = 0.95)
PR curve for fold 4 (AUC = 0.98)
Mean precision (AUC mean = 0.97)

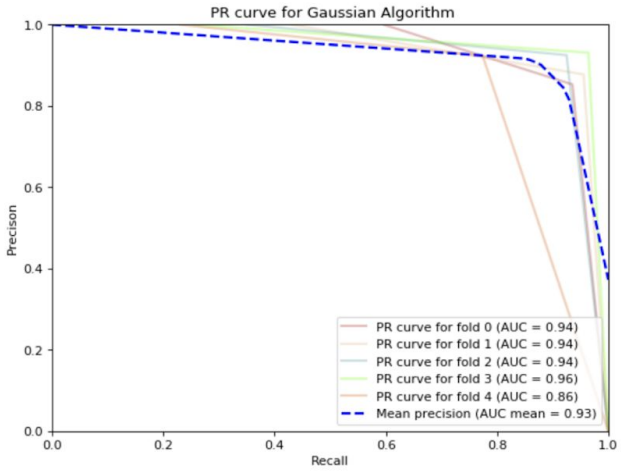

ROC for SVC-rbf
TPR (True Positive Rate)
FPR (False Positive Rate)
ROC fold for 0 (AUC = 0.97)
ROC fold for 1 (AUC = 0.96)
ROC fold for 2 (AUC = 0.96)
ROC fold for 3 (AUC = 0.99)
ROC fold for 4 (AUC = 0.99)
Mean ROC for (AUC = 0.97)

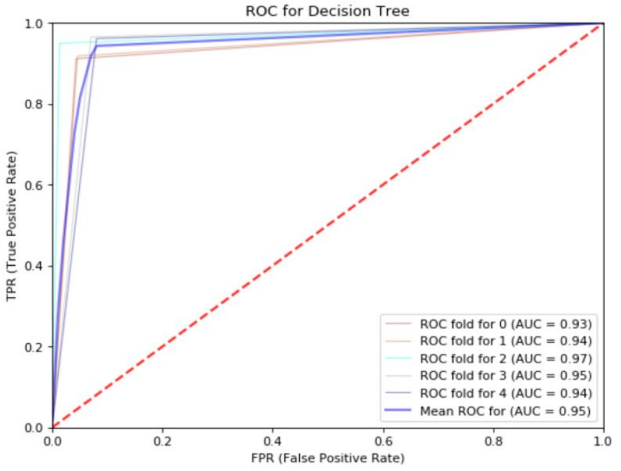

PR curve for SVC-rbf
Precison
Recall
PR curve for fold 0 (AUC = 0.98)
PR curve for fold 1 (AUC = 0.97)
PR curve for fold 2 (AUC = 0.98)
PR curve for fold 3 (AUC = 0.98)
PR curve for fold 4 (AUC = 0.96)
Mean precision (AUC mean = 0.98)

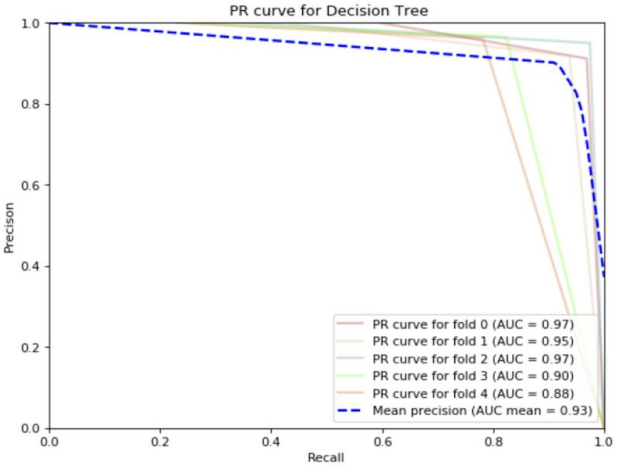

ROC for SVC-polynomial
TPR (True Positive Rate)
FPR (False Positive Rate)
ROC fold for 0 (AUC = 0.87)
ROC fold for 1 (AUC = 0.87)
ROC fold for 2 (AUC = 0.95)
ROC fold for 3 (AUC = 0.97)
ROC fold for 4 (AUC = 0.96)
Mean ROC for (AUC = 0.92)

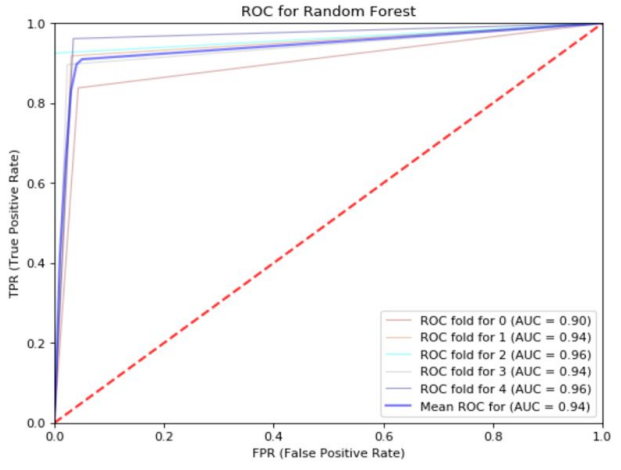

PR curve for SVC-polynomial
Precison
Recall
PR curve for fold 0 (AUC = 0.95)
PR curve for fold 1 (AUC = 0.92)
PR curve for fold 2 (AUC = 0.97)
PR curve for fold 3 (AUC = 0.97)
PR curve for fold 4 (AUC = 0.97)
Mean precision (AUC mean = 0.96)

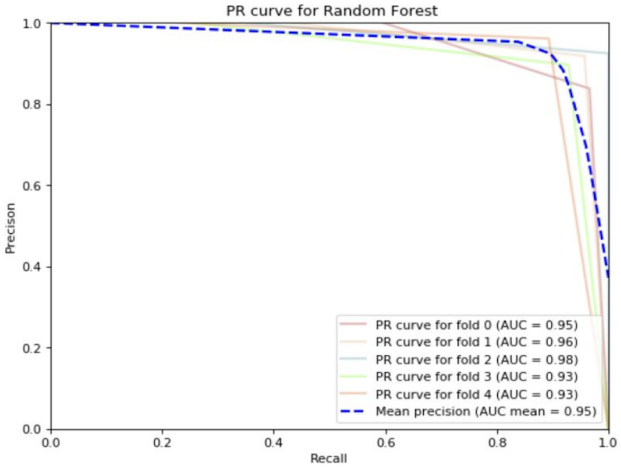

ROC for Gaussian Algorithm
TPR (True Positive Rate)
FPR (False Positive Rate)
ROC fold for 0 (AUC = 0.88)
ROC fold for 1 (AUC = 0.92)
ROC fold for 2 (AUC = 0.94)
ROC fold for 3 (AUC = 0.96)
ROC fold for 4 (AUC = 0.92)
Mean ROC for (AUC = 0.93)

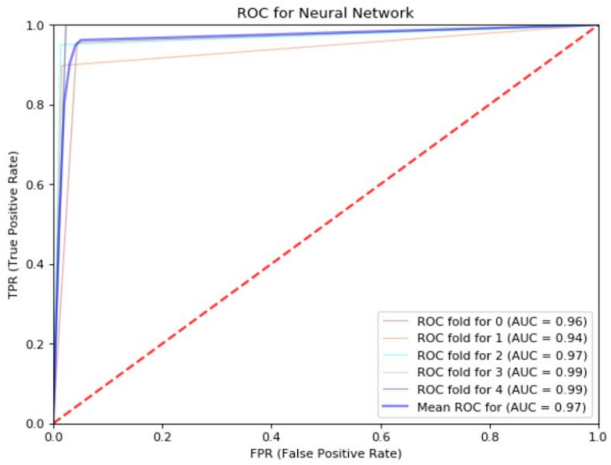

PR curve for Gaussian Algorithm
Precison
Recall
PR curve for fold 0 (AUC = 0.94)
PR curve for fold 1 (AUC = 0.94)
PR curve for fold 2 (AUC = 0.94)
PR curve for fold 3 (AUC = 0.96)
PR curve for fold 4 (AUC = 0.86)
Mean precision (AUC mean = 0.93)

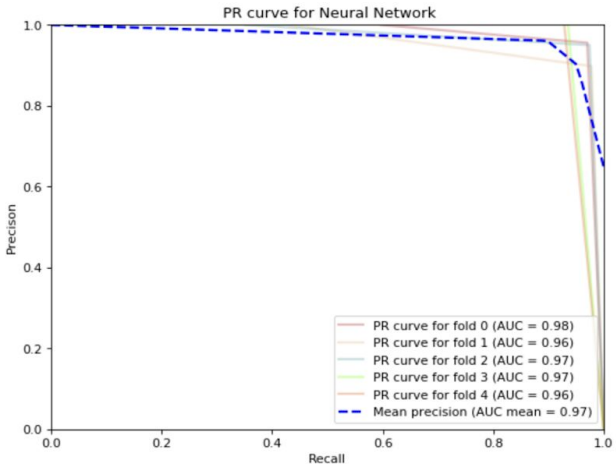

ROC for Decision Tree
TPR (True Positive Rate)
FPR (False Positive Rate)
ROC fold for 0 (AUC = 0.93)
ROC fold for 1 (AUC = 0.94)
ROC fold for 2 (AUC = 0.97)
ROC fold for 3 (AUC = 0.95)
ROC fold for 4 (AUC = 0.94)
Mean ROC for (AUC = 0.95)


PR curve for Decision Tree
Precison
Recall
PR curve for fold 0 (AUC = 0.97)
PR curve for fold 1 (AUC = 0.95)
PR curve for fold 2 (AUC = 0.97)
PR curve for fold 3 (AUC = 0.90)
PR curve for fold 4 (AUC = 0.88)
Mean precision (AUC mean = 0.93)


ROC for Random Forest
TPR (True Positive Rate)
FPR (False Positive Rate)
ROC fold for 0 (AUC = 0.90)
ROC fold for 1 (AUC = 0.94)
ROC fold for 2 (AUC = 0.96)
ROC fold for 3 (AUC = 0.94)
ROC fold for 4 (AUC = 0.96)
Mean ROC for (AUC = 0.94)


PR curve for Random Forest
Precison
Recall
PR curve for fold 0 (AUC = 0.95)
PR curve for fold 1 (AUC = 0.96)
PR curve for fold 2 (AUC = 0.98)
PR curve for fold 3 (AUC = 0.93)
PR curve for fold 4 (AUC = 0.93)
Mean precision (AUC mean = 0.95)





5. **Discussion:**

This project has been run on google Colab and also on visual studio code. This project is a user-friendly application. On running this python program, a Menu is prompted which contains individual choices of algorithm and a combined run of all the algorithms and asks the user to select and enter an algorithm number. Then, it prompts for a 'k' value for k splits for k-cross validation. Users can enter any number ranging from 3- 29. Other invalid values will be rejected and a default fold of 5 is considered. Then, the program runs the desired algorithm and displays the mean accuracy, ROC curve plots and PR curve plots for all the folds as well as their mean. There is still scope for improvement, additional algorithms can be tested for better accuracy and also by experimenting with existing models with various parameters we can increase the accuracy of the existing models.

6. **Conclusion:**

This project is intended to apply various different machine learning algorithms for breast cancer classification. Then, we choose a Wisconsin Breast Cancer dataset which contains breast cancer cell features such as compactness, radius. Next, we analyzed this dataset, checked for null values, replaced categorical data and we also scale the values to prepare the data.

Once data is prepared, we successfully implemented SVM, logistic regression, k-NN, naive Bayes, decision tree, random forest, Neural Network. For each of the algorithms, we used 5 fold validation. For each fold, we trained the model and obtained predictions on the test data, then we plot the ROC curve and PR curve, area under the curve, determine the balance accuracy. Once all 5 folds are completed, we also plot the mean ROC curve, mean PR curve, the mean area under the curve and mean accuracy. We also compare the accuracy of all the algorithms applied to determine the best fitting algorithm for our requirements.

7. **References:**

[1].https://www.Kaggle.com
[2].https://www.researchgate.net/publication/331674621_Prediction_of_Malignant_Benign_Breast_Cancer_A_Data_Mining_Approach_in_Healthcare_Applications
[3].https://towardsdatascience.com/building-a-simple-machine-learning-model-on-breast-cancer-data-eca4b3b99fa3
[4]https://towardsdatascience.com/predict-malignancy-in-breast-cancer-tumors-with-your-own-neural-network-and-the-wisconsin-dataset-76271a05e941
[5].https://scikit-learn.org