A REPORT
ON

XGBOOST FOR BREAST CANCER PREDICTION

By

Name of the Student:

Vaibhav Chaudhari(2017B5A70834G)

Prepared under the fulfilment of Course CS F320

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Birla Institute of Technology & Science, Pilani
Goa Campus

Acknowledgments

The idea for this project was inspired by T. L. Octaviani and Z. Rustam (Department of Mathematics, Faculty of Mathematics and Natural Sciences (FMIPA), Universitas Indonesia), and I am thankful to Mr. Sandeep Vidyapu (Faculty, Department of Computer Sciences and Information Systems, BITS Goa) for giving me the assignment which has led me to a wider understanding of Data Sciences.

I also thank him for helping me and giving advice on this project without which this project could not have been completed.

Last but not least, I would extend my gratitude towards my family, for continuously motivating me to not give up on the project even if the results were not as expected.

Abstract

Breast Cancer is the most frequent cancer among women. According to the WHO, 627,000 women died of breast cancer in 2018, which is 15% of all cancer deaths among women. Delay in knowing the condition of breast cancer in women leads to increased mortality rates. In this project,we use the XGBoost Machine learning algorithm for Breast Cancer Classification to obtain the results. Random Forest and XGBoost are two popular decision tree algorithms for machine learning. In this paper, we compare the accuracy of both the algorithms to achieve a more accurate and reliable classification performance. One of the key reasons we use the XGBoost algorithm is that it provides a better accuracy as compared to Random Forest Classifier for large datasets. The results of both the algorithms are then visually analyzed and we come to the conclusion that XGBoost algorithm gives better accuracy than Random Forest Classifier for a larger dataset.

Table of Contents

Acknowledgments 1
Abstract 2
Table of Contents 3
1) Introduction 4
2) Algorithms 5
2.1) Random Forest Classifier 5
2.2) XGBoost 6
3) Data Description 7
3.1) Attributes/Features 7
3.2) Data Cleaning 9
3.3) Label Encoding 9
3.4) Correlational Matrix and Correlational Heatmap 10
3.5) Visualization of the Updated Dataset 12
4) Methodology 15
4.1) Aim 15
4.2) Implementation 15
4.2.1) Splitting Data into Training and Testing 15
4.2.2) Random Forest 16
4.2.3) XGBoost 17
4.2.4) Accuracy 17
5) Results 18
6) References 21

1) Introduction

Breast cancer is a type of cancer that develops in the cells of the breasts. Cancer occurs when changes called mutations take place in genes that regulate cell growth. The mutations let the cells divide and multiply in an uncontrolled way which leads to the start of Cancer. Breast cancer is the most common invasive cancer in women impacting 2.1 million women each year and the second leading cause of cancer death in women after lung cancer. In 2018, it is estimated that 627,000 women died from breast cancer – that is approximately 15% of all cancer deaths among women.

Breast Cancer tumors can we widely classified into two types viz. Benign and Malignant. Benign tumors are non-cancerous breast tumors which have abnormal growths, but they do not spread outside of the breast. Malignant tumors are cancerous breast tumors which are aggressive as they invade and damage the surrounding tissues. In order to improve breast cancer outcomes and survival, early detection is critical which helps in combating the cancer efficiently.

XGBoost (eXtreme Gradient Boosting) is basically an implementation of the Gradient Boosting Machines created by Tianqi Chen. It is focused on computational speed and model performance. XGBoost dominates the structured or tabular databases on classification problems and also gives us accuracies better than the normal classifier algorithms in machine learning.

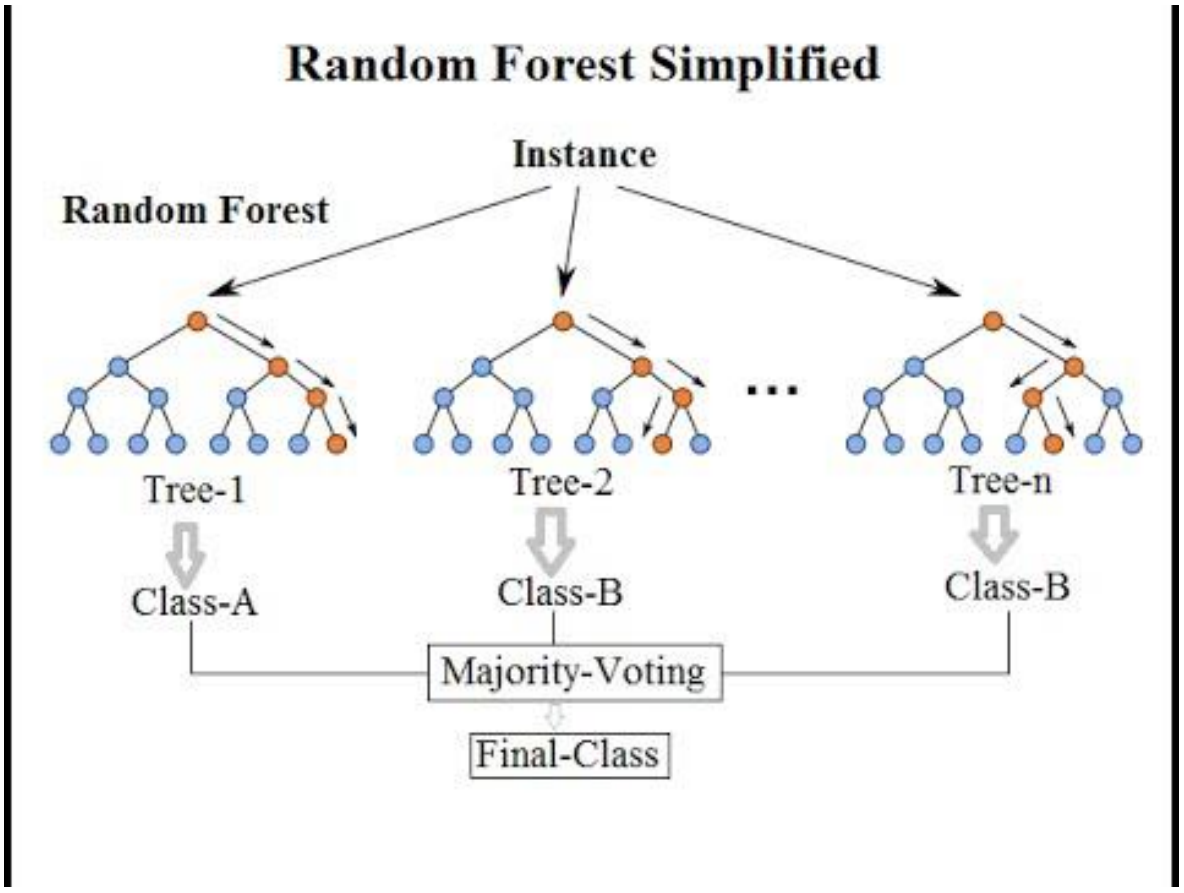
In this paper, we use XGBoost and Random Forest Classifier for predicting the breast cancer data. We use the Wisconsin Breast Cancer Database (WBCD) data (Diagnostic) from the UCI Repository which is usually used by researchers for diagnosis of breast cancer. The data consists of two classes, malignant and benign. We use XGBoost and Random Forest Classifier to analyze the WBCD data (Diagnostic) to compare the accuracies of the two models.

2) Algorithms

In the paper, the WBCD data (Original) was used for predicting the type of tumor using the Random Forest Classifier but in this paper, I have used the WBCD data (Diagnostic) for predicting the type of tumor using the XGBoost and the Random Forest Classifier to compare the accuracies of both the models and hence using the model with better accuracy to display the results visually for better understanding (NOVELTY) .

2.1) Random Forest Classifier

A brief explanation of the Random Forest Classifier algorithm comes from the name. Rather than utilize the predictions of a single decision tree, the algorithm will take the ensemble result of a large number of decision trees (a forest of them). The "Random" part of the name comes from the term "bootstrap aggregating", or "bagging". What this means is that each tree within the forest only gets to train on some subset of the full training dataset (the subset is determined by sampling with replacement). The elements of the training data for each tree that are left unseen are held "out-of-bag" for estimation of accuracy. Randomness also helps decide which feature input variables are seen at each node in each decision tree. Once all individual trees are fit to a random subset of the training data, using a random set of feature variable at each node, the ensemble of them all is used to give the final prediction.

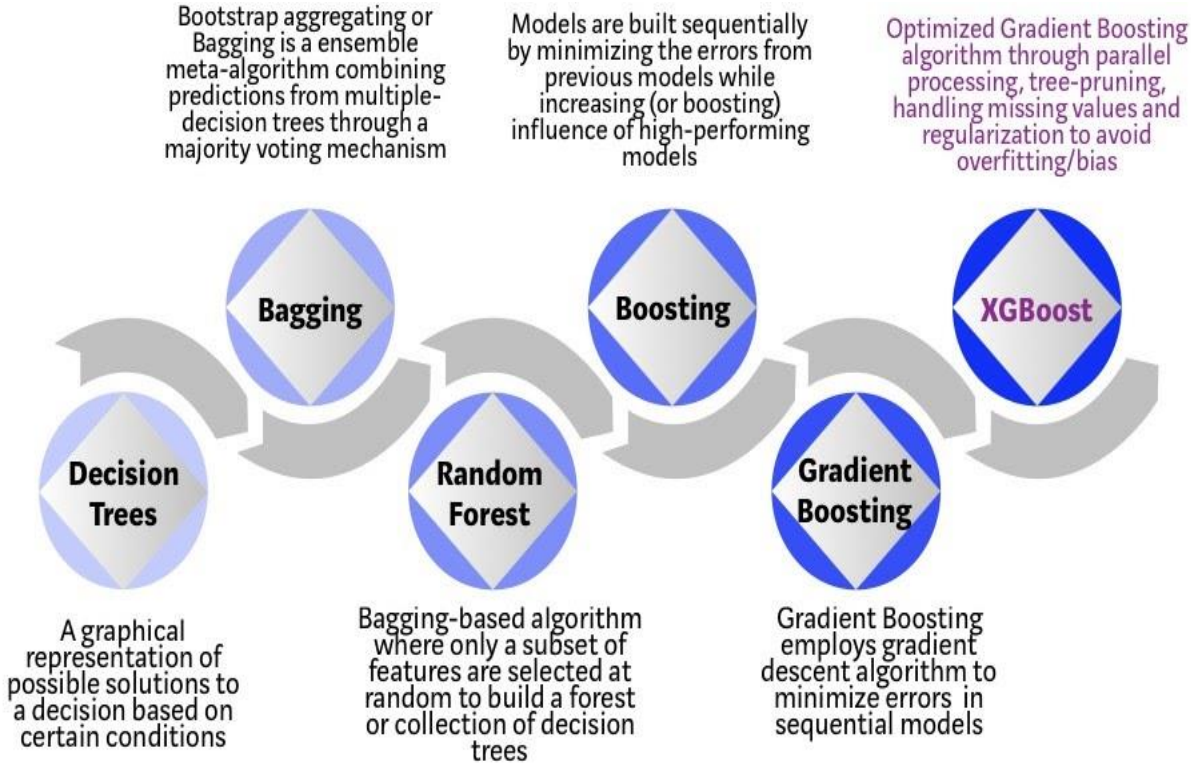


We use the Gini Index as our cost function used to evaluate splits in the dataset. We minimize it. A split in the dataset involves one input attribute and one value for that attribute. It can be used to divide training patterns into two groups of rows. A Gini score gives an idea of how good a split is by how mixed the classes are in the two groups created by the split.

2.2) XGBoost

XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework. When it comes to small-to-medium structured data, decision tree-based algorithms are considered best. Boosting is an ensemble method. The idea of boosting is to train weak learners sequentially, each trying to correct its predecessor. The term 'Boosting' refers to a family of algorithms which converts weak learners to strong learners. Gradient Boosting is a special case of boosting where errors are minimized by gradient descent algorithm. XGBoost or Extreme Gradient Boosting is a more powerful version of Gradient Boosting. However, XGBoost improves upon the base Gradient Boosting framework through systems optimization and algorithmic enhancements.

The system optimization consists of Parallelization, Tree Pruning and Hardware Optimization whereas the algorithmic enhancement consists of Regularization, Sparsity Awareness, Weighted Quantile Sketch and Cross-validation.



3) Data Description

Th dataset used in this project is the Wisconsin Breast Cancer Database (WBCD) data (Diagnostic).

Dataset	No. of Instances	No. of Attributes	No. of Classes	No. of Missing / NaN Values
Wisconsin Breast Cancer Database data (Diagnostic).	569	32	2 (Benign and Malignant)	None

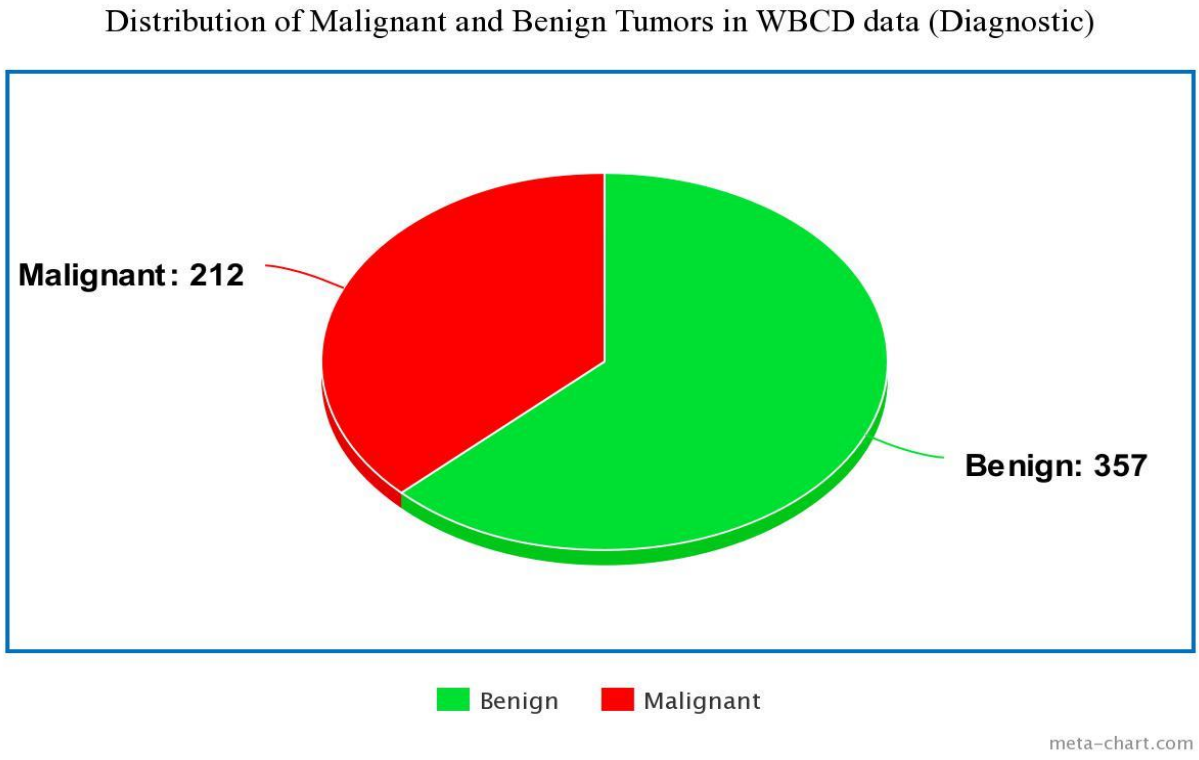
3.1) Attributes/Features

The dataset contains 32 attributes which include the 2 nominal attributes id and diagnosis and ten real-valued features which are computed for each cell nucleus. 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 instance, attribute 3 is Mean Radius, attribute 13 is Radius SE, attribute 23 is Worst Radius.

S. No.	Attribute Name	Attribute Type
1	id	Nominal
2	diagnosis	Nominal
3	radius_mean	Ratio
4	texture_mean	Ratio
5	perimeter_mean	Ratio
6	area_mean	Ratio
7	smoothness_mean	Ratio
8	compactness_mean	Ratio
9	concavity_mean	Ratio
10	concave points_mean	Ratio
11	symmetry_mean	Ratio
12	fractal_dimension_mean	Ratio
13	radius_se	Ratio
14	texture_se	Ratio
15	perimeter_se	Ratio
16	area_se	Ratio
17	smoothness_se	Ratio
18	compactness_se	Ratio
19	concavity_se	Ratio

20	concave points_se	Ratio
21	symmetry_se	Ratio
22	fractal_dimension_se	Ratio
23	radius_worst	Ratio
24	texture_worst	Ratio
25	perimeter_worst	Ratio
26	area_worst	Ratio
27	smoothness_worst	Ratio
28	compactness_worst	Ratio
29	concavity_worst	Ratio
30	concave points_worst	Ratio
31	symmetry_worst	Ratio
32	fractal_dimension_worst	Ratio

The diagnosis of the tumors is done for 2 classes namely, Malignant and Benign as shown in the diagram below.




```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        import pandas as pd
        import os
```

```
In [ ]: df = pd.read_csv ('data.csv')
        print (df)
        X = df.iloc[:,2:32]
        y = df.iloc[:, 1]
```

3.2) Data Cleaning

We then check the number of missing / NaN (Not a Number) values in the dataset using the following code which gives us the conclusion that our dataset is complete and has no missing values which need to be replaced. This means that our dataset can be used without making any changes in it.

```
In [1117]: df.isna().sum()
```

```
Out[1117]: 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
           dtype: int64
```

3.3) Label Encoding

Since the diagnosis attribute of the dataset is not in numerical format, we need to convert it to a number in order to use the data to train our models. Label encoding is simply converting each value in a column to a number. It converts distinct values in a column to different numbers. The code to use Label Encoding can be seen below.

```
In [1153]: from sklearn.preprocessing import LabelEncoder
           labelencoder_y = LabelEncoder()
           y = labelencoder_y.fit_transform(y)
```

3.4) Correlational Matrix and Correlational Heatmap

In order to use the dataset more efficiently, we find the correlation matrix between the different features of the dataset and select those features which have a correlation >= Threshold or correlation <= -Threshold as the main features which will be used to predict whether the tumor is malignant or benign.

```
In [ ]: X.corr()
```

The value of the threshold can be set anywhere between 0 and 1. After repeated trial and error, I found out that we get the best accuracy when the threshold is set at 0.9 as we get most of the features to work with (21 features out of 30 features) which ensures that not most of the data is lost and it also gives us better accuracy for this case as we have more data from different features to work with.

```
def correlation(dataset, threshold):
    col_corr = set() # Set of all the names of deleted columns
    corr_matrix = dataset.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if (corr_matrix.iloc[i, j] >= threshold or corr_matrix.iloc[i, j] <= threshold*-1) and (corr_matrix.columns[j] not in col_corr):
                colname = corr_matrix.columns[i] # getting the name of column
                col_corr.add(colname)
                if colname in dataset.columns:
                    del dataset[colname] # deleting the column from the dataset
    return dataset
X=correlation(df.iloc[:,2:32],0.7)
```

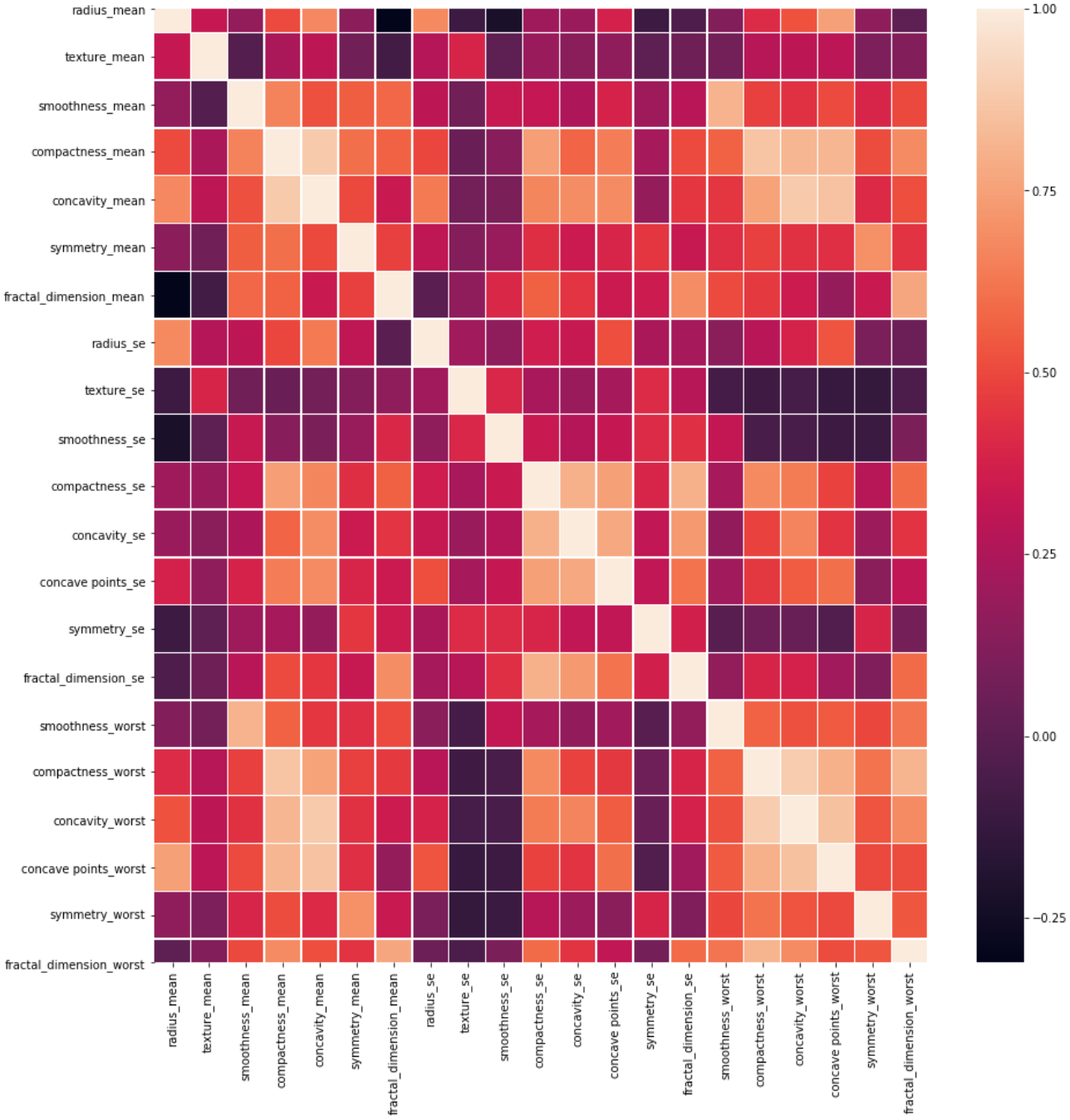
Now we can easily find the correlation heat map and the new correlation matrix for the new set of reduced features. The new and the original correlation matrix are given as .csv files in the zip folder.

NOTE: Although the reduced set of attributes speeds up the process of training the model and testing it on the testing data, it comes at the cost of reduction of the accuracy of the models (only by at max 5%) since our dataset has only 569 instances which are less as compared to traditional datasets used for training the models. If we want to have better accuracy, we can use the original 30 attributes to train our model which will increase our accuracy but it will not give a considerable change so we ignore it and reduce the attributes of the dataset by finding correlation between its features.

We use the following code to make the correlation heatmap which gives us a pretty good visualization of the correlation between the features.

```
In [22]: import matplotlib.pyplot as plt
         import seaborn as sns

         fig, ax = plt.subplots(figsize=(15,15))
         sns.heatmap(X.corr(), annot=False, linewidths=.5, ax=ax)
```

Correlation Heatmap for the new set of Features(21 features out of 30 features)

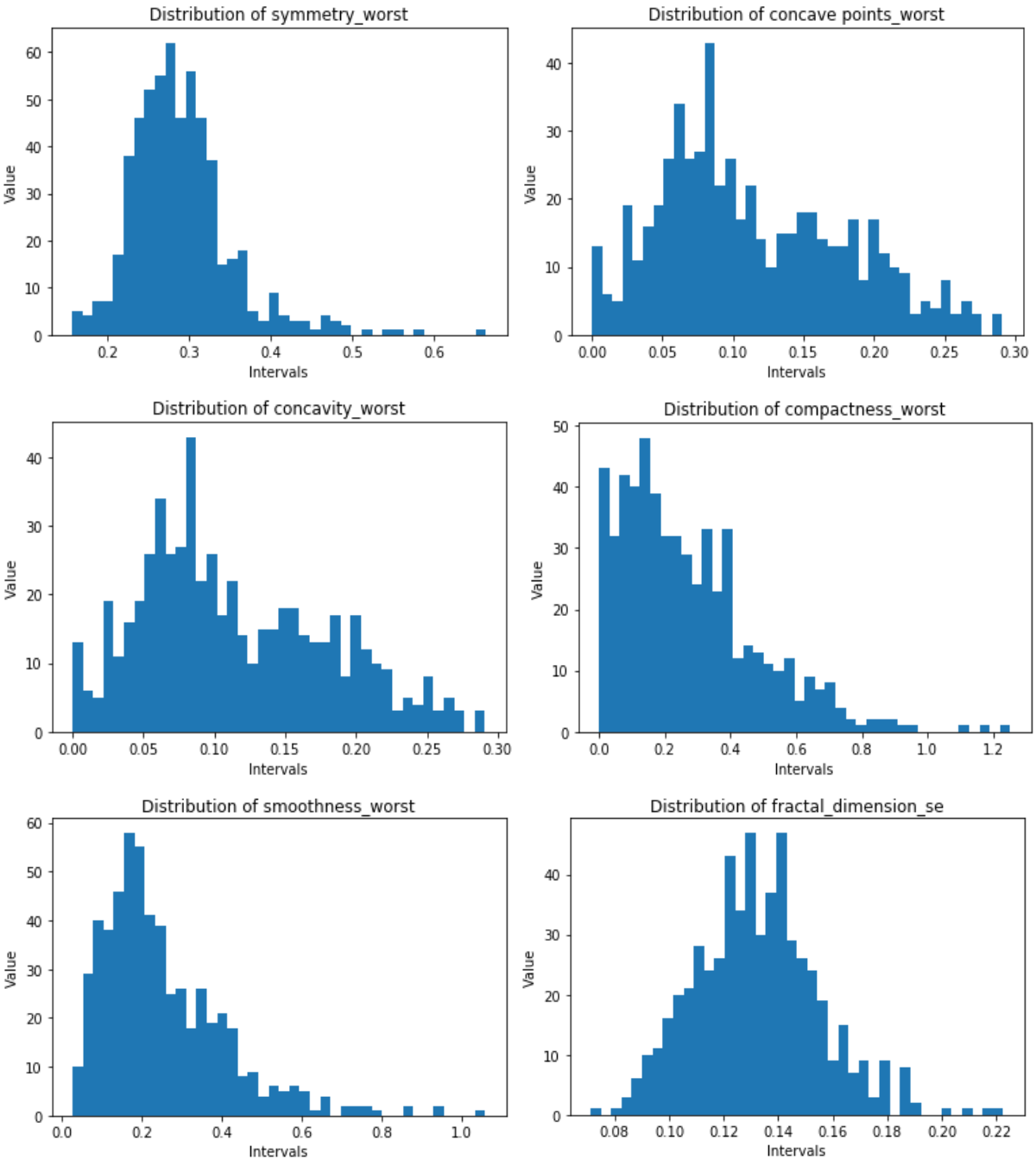


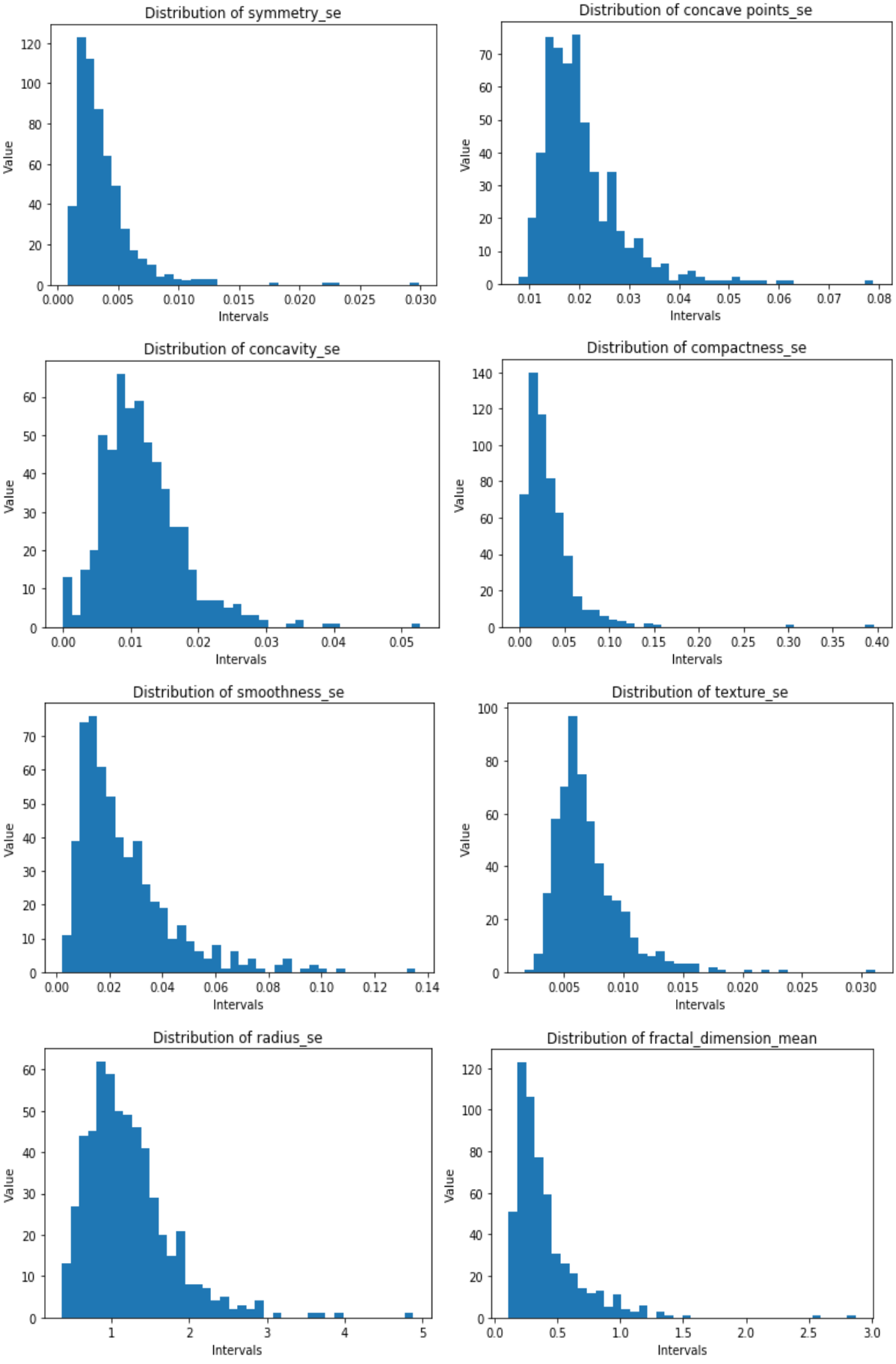
3.5) Visualization of the Updated Dataset

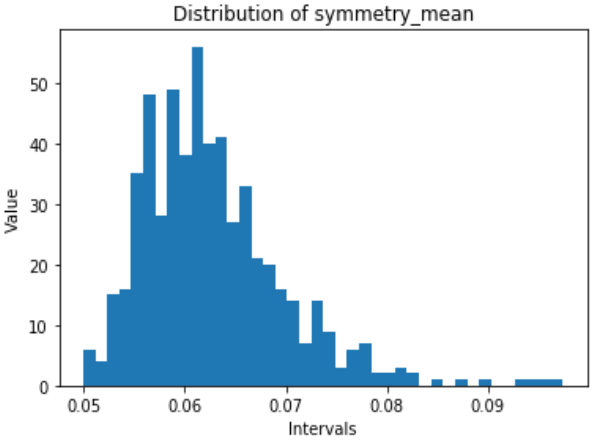
We can easily visualize the values of the different features/attributes by plotting the vales of different features independently using histograms as it gives us a clear idea of the values in a particular attribute. The code used for this is:

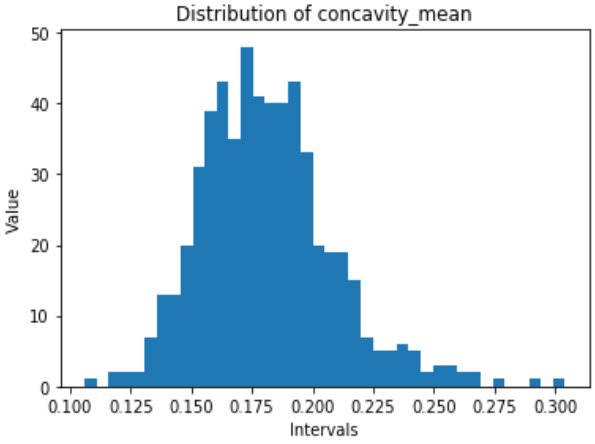
```
In [82]: x=X.iloc[:,5]
         plt.hist(x,bins=40)
         plt.xlabel('Intervals')
         plt.ylabel('Value')
         plt.title('Distribution of compactness_mean')
         plt.show()
```

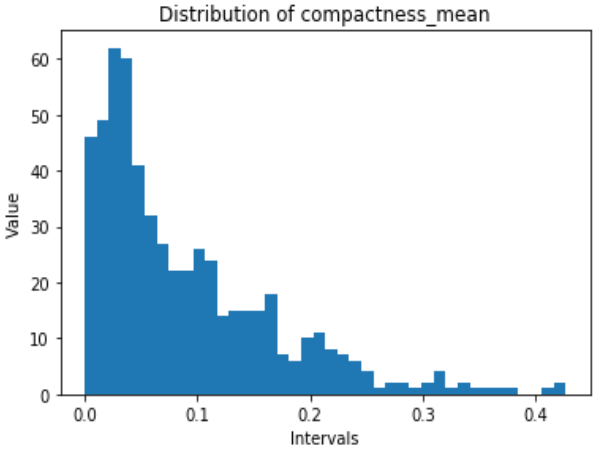
Histograms for updated WBCD dataset (Diagnostic):

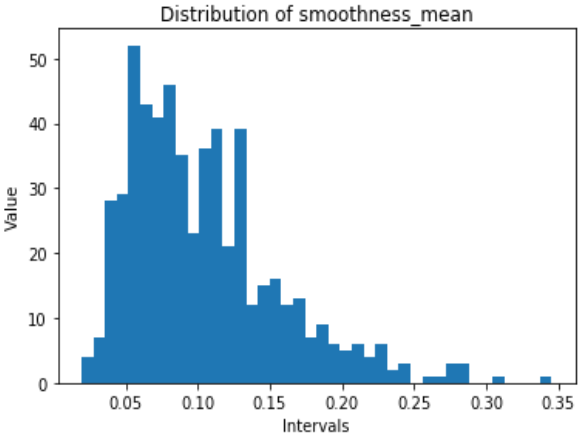


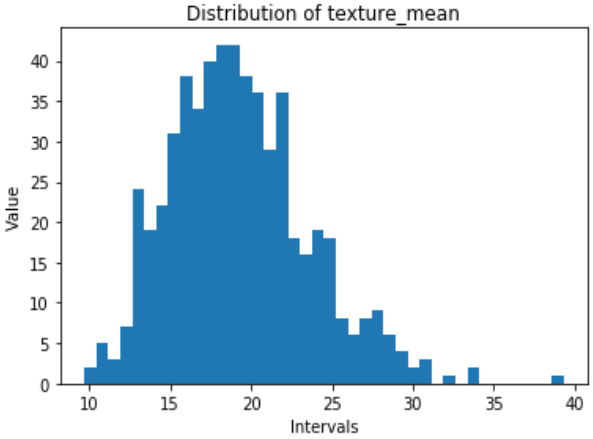

Distribution of symmetry_se
Value
Intervals
Distribution of concave points_se
Value
Intervals
Distribution of concavity_se
Value
Intervals
Distribution of compactness_se
Value
Intervals
Distribution of smoothness_se
Value
Intervals
Distribution of texture_se
Value
Intervals
Distribution of radius_se
Value
Intervals
Distribution of fractal_dimension_mean
Value
Intervals

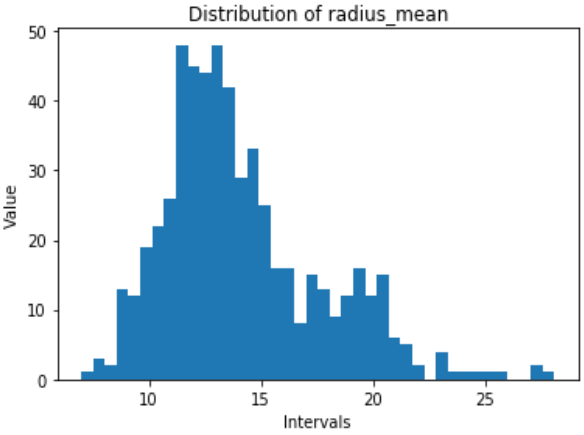

Distribution of symmetry_mean
Value
Intervals
0.05
0.06
0.07
0.08
0.09

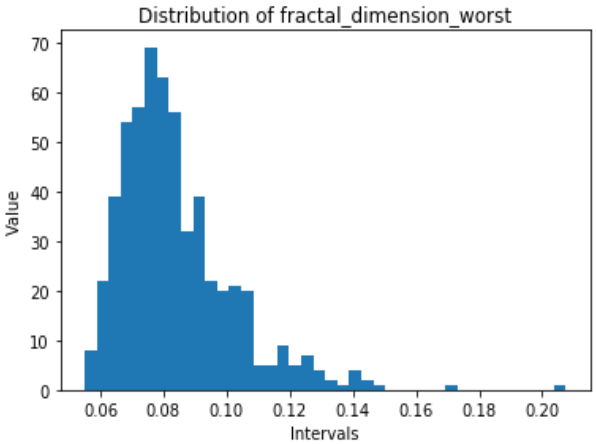

Distribution of concavity_mean
Value
Intervals
0.100
0.125
0.150
0.175
0.200
0.225
0.250
0.275
0.300


Distribution of compactness_mean
Value
Intervals
0.0
0.1
0.2
0.3
0.4


Distribution of smoothness_mean
Value
Intervals
0.05
0.10
0.15
0.20
0.25
0.30
0.35


Distribution of texture_mean
Value
Intervals
10
15
20
25
30
35
40


Distribution of radius_mean
Value
Intervals
10
15
20
25


Distribution of fractal_dimension_worst
Value
Intervals
0.06
0.08
0.10
0.12
0.14
0.16
0.18
0.20

4) Methodology

4.1) Aim

Our aim in this project is to implement both the model, Random Forest Classifier and XGBoost Classifier. After implementing both the models, we have to choose one of the above two model based on the accuracies as the better model for predicting the class of the Breast Cancer Data Diagnosis. We calculate the accuracies of both the models for different percentages of training dataset.

4.2) Implementation

4.2.1) Splitting Data into Training and Testing

The first step in the implementation of the models is the splitting of data into the training and the testing data. We split the arrays or matrices into random train and test subsets using the sklearn.model_selection library and import the train_test_split function. We feed these arrays or matrices to the models as the input to get the desired results. We generally split the training and the testing data as 70% of the dataset for the former and the remaining 30% of the dataset for the latter. This ratio can be varied and depending on this we get different accuracies. In general, the more the size of training data, the better is the accuracy of the model as the model gets to train itself on a wider variety of instances if more data is present.
We can split the data into the two parts using the following code:

```
In [ ]: from sklearn.model_selection import train_test_split
        X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size = 0.1, random_state = 0)
```

Example where data split in ratio: 90% training and 10% test

The parameter random_state controls the shuffling applied to the data before applying the split. The function returns 4 arrays or matrices containing train-test split of input arrays or matrices.

In this project, we have split the data into following ways:

Cases	Training Data	Testing Data
1	90%	10%
2	80%	20%
3	70%	30%
4	60%	40%
5	50%	50%
6	40%	60%
7	30%	70%
8	20%	80%
9	10%	90%

4.2.2) Random Forest

I'm using this model from sklearn.ensemble library. The function used is RandomForestClassifier. 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. The sub-sample size is controlled with the max_sapmles parameter if bootstrap=True (default), otherwise the whole dataset is used to build each tree. I have used the following parameters to improve the accuracy of the model:

- **n_estimators** - The number of trees in the forest. (default=100)
- **max_depth** - The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples. (default -None)
- **criterion** - The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain. (default-gini).

```
In [590]: from sklearn.ensemble import RandomForestClassifier
          classifier = RandomForestClassifier(n_estimators = 50, criterion = 'gini', random_state = 0)
          classifier.fit(X_train, Y_train)

Out[590]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                                 max_depth=None, max_features='auto', max_leaf_nodes=None,
                                 min_impurity_decrease=0.0, min_impurity_split=None,
                                 min_samples_leaf=1, min_samples_split=2,
                                 min_weight_fraction_leaf=0.0, n_estimators=50,
                                 n_jobs=None, oob_score=False, random_state=0, verbose=0,
                                 warm_start=False)

In [591]: Y_pred = classifier.predict(X_test)

In [592]: from sklearn.metrics import accuracy_score
          accuracy_rf = accuracy_score(Y_test,Y_pred)*100
```

4.2.3) XGBoost

I'm using this model from xgboost library. The function used is XGBClassifier. I have used the following parameters to improve the accuracy of the model:

- **max_depth** (*int*) – Maximum tree depth for base learners.
- **learning_rate** (*float with range 0 to 1*) – Boosting learning rate
- **n_estimators** - The number of trees in the forest. (*default=100)*

For both of these models there are two methods fit and predict. For the model.fit function we give X_train and Y_train as input. The model makes a boosting framework such that we get the particular y form the given x if we go by this framework. Then we use the model.predict function. This function uses the model we obtained before. We pass the X_test and get Y predicted or Y_pred as output. Then we compare this with Y_test to find accuracy of data.

```
In [594]: from xgboost import XGBClassifier
          classifier = XGBClassifier( n_estimators=1000,learning_rate=0.2,max_depth=30)
          classifier.fit(X_train, Y_train)

Out[594]: XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                        colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
                        importance_type='gain', interaction_constraints='',
                        learning_rate=0.2, max_delta_step=0, max_depth=30,
                        min_child_weight=1, missing=nan, monotone_constraints='()',
                        n_estimators=1000, n_jobs=0, num_parallel_tree=1,
                        objective='binary:logistic', random_state=0, reg_alpha=0,
                        reg_lambda=1, scale_pos_weight=1, subsample=1,
                        tree_method='exact', validate_parameters=1, verbosity=None)

In [595]: Y_pred = classifier.predict(X_test)

In [596]: from sklearn.metrics import accuracy_score
          accuracy_xgb = accuracy_score(Y_test,Y_pred)*100
```

4.2.4) Accuracy

For accuracy we use accuracy_score function from sklearn.metrics. It has following parameters-

- **y_true** -Ground truth (correct) labels. It can be a label indicator array or sparse matrix.
- **y_pred**-Predicted labels, by a classifier. It can be a label indicator array or sparse matrix.
- **normalize** -If False, return the number of correctly classified samples. Otherwise, return the fraction of correctly classified samples. (default - True)

This function returns a float that is the fraction of labels correctly classified. This is essentially equal to (TP + TN) / (TP + TN + FP + FN).

Where TP - True Positive, TN - True Negative, FP = False Positive, FN - False Negative.

Accuracy = accuracy_score* 100 = (TP + TN)*100 / (TP + TN + FP + FN)

5) Results

We implement the models on the datasets based on the splitting of testing and training data and obtain the following results:

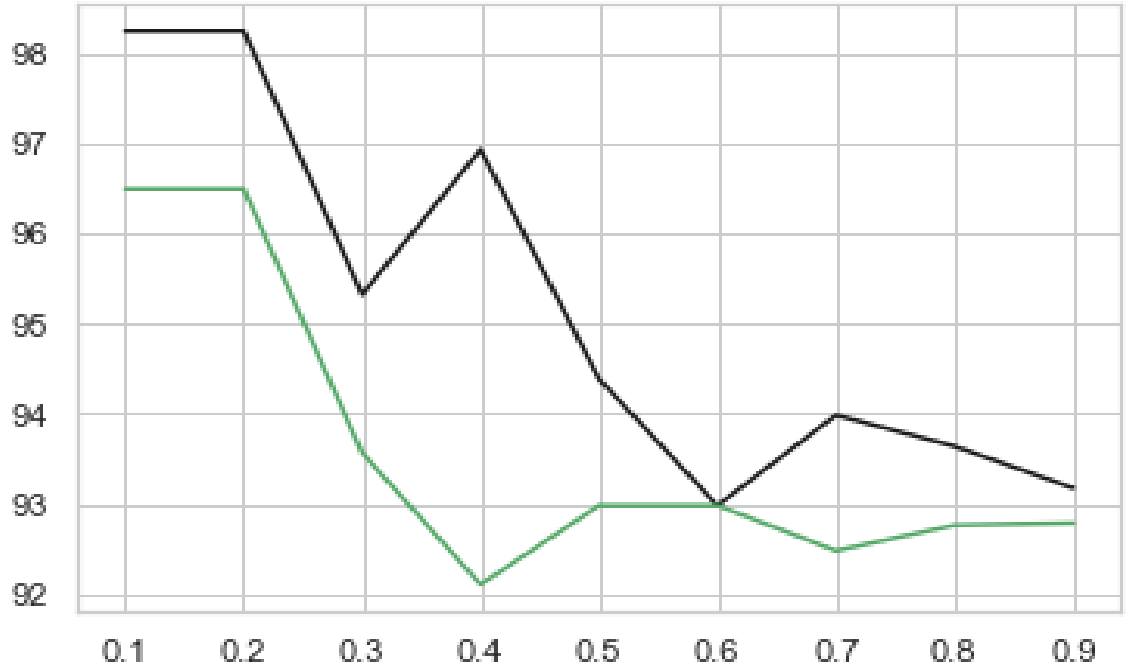
Training Data	Testing Data	XGBoost Accuracy	Random Forest Accuracy
10%	90%	93.17738791423001%	92.78752436647173%
20%	80%	93.64035087719299%	92.76315789473685%
30%	70%	93.98496240601504%	92.4812030075188%
40%	60%	92.98245614035088%	92.98245614035088%
50%	50%	94.38596491228071%	92.98245614035088%
60%	40%	96.9298245614035%	92.10526315789474%
70%	30%	95.32163742690058%	93.56725146198829%
80%	20%	98.24561403508771%	96.49122807017544%
90%	10%	98.24561403508771%	96.49122807017544%

We can clearly see that the accuracy of prediction of class of the Brest Cancer Data is always higher for the XGBoost model as compared to the Random Forest Model. Thus we can conclude that the XGBoost model is better at classification as it has better performance than the Random Forest model. We will be able to better visualize the performance of both the models and compare their results using the following code:

```
import pylab as plt
```

```
xgboost_result=[98.24561403508771,98.24561403508771,95.32163742690058, 96.9298245614035,94.38596491228071,92.98245614035088,
            93.98496240601504, 93.64035087719299, 93.17738791423001]
rf_result=[96.49122807017544,96.49122807017544,93.56725146198829,92.10526315789474,92.98245614035088,92.98245614035088,
            92.4812030075188,92.76315789473685,92.78752436647173]
```

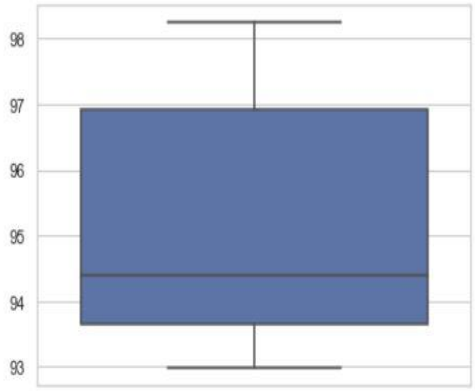
```
fig,ax = plt.subplots()
x_dat = [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]

plt.plot(x_dat,xgboost_result,color='k')
plt.plot(x_dat,rf_result,color='g')

plt.show()
```



Here the black line represents the accuracy of the XGBoost model for various ratio of testing data and green line represents the accuracy of the Random Forest model for various ratio of testing data.

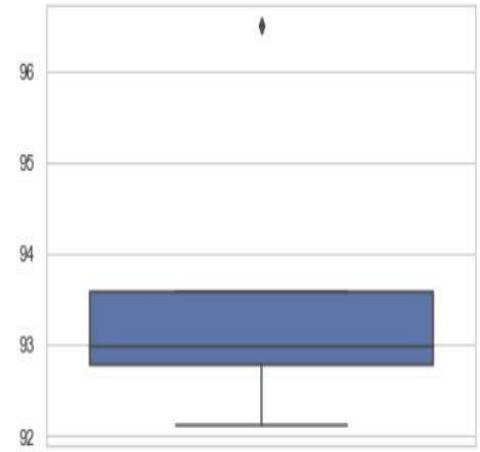
We can also visualize the results of the accuracy using the boxplot diagrams of both the models.

```
import seaborn as sns
sns.set(style="whitegrid")
ax = sns.boxplot(y=xgboost_result)
```



This box-plot is for XGBoost Model.

```
ay=sns.boxplot(y=rf_result)
```



This box-plot is for Random Forest Model.

From all these we can conclude that XGBoost is a better model when it comes to classification as it gives more accuracy.

6) References

1. UCI Repository, Wisconsin Breast Cancer Database (WBCD) (2018), available at http://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/breast-cancer-wisconsin.data
2. Random forest for breast cancer prediction, T. L. Octaviani and Z. Rustam (Department of Mathematics, Faculty of Mathematics and Natural Sciences (FMIPA), Universitas Indonesia).
3. World Health Organisation, https://www.who.int/cancer/prevention/diagnosis-screening/breast-cancer/
4. xgboost: eXtreme Gradient Boosting
https://cran.r-project.org/web/packages/xgboost/vignettes/xgboost.pdf
5. L Breiman, Machine Learning 45, 5-32 (2001)