# CAPSTONE REPORT

## 1. Definition

### 1.1. Project Overview:

In this project, machine learning will be used to build a model that can predict whether a person suffers from breast cancer or not. Breast Cancer is a cancer that forms in the cells of the breasts. Breast Cancer is highly predominant in women in today's world. It is caused by uncontrolled growth of abnormal cells in the breast. It can start in the breast and can spread to other areas of the body in the course of time.
Identifying correctly whether a tumor is benign or malignant is vital in deciding what is the best treatment, saving and improving the quality of life. In this project, we used Breast Cancer Wisconsin (Diagnostic) Data Set to create a model able to predict if a tumor is or not dangerous based on characteristics that were computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. For gaining a detailed knowledge on particular domain we can refer to some investigation done earlier [1][2].
Implementation of different supervised learning such as Stochastic Gradient Descent Classifier, Logistic Regression, Support Vector Machines, Decision Tree Classifier, and XGB classifier has been implemented to lead to high accuracy yielding true positive and true negative results.

### 1.2. Problem Statement

A tumor can be in two stages and they are benign and malignant. Benign stage tumors are not dangerous to health and are non cancerous whereas malignant tumors has the potentiality of being dangerous and are cancerous.

The physical characteristics of a malignant tumor differ from the benign tumor cells. Thus a measure on the different characteristics such as radius (mean of distances from center to points on the perimeter), texture_mean (standard deviation of gray-scale values), perimeter_mean, area_mean, smoothness_mean(mean of local variation in radius lengths), compactness, concavity, concave points, symmetry or fractal dimension helps us to understand that a new sample for classification belongs to which class of tumor i.e. Malignant or Benign.

The dataset is clearly a classification oriented problem as the column diagnosis consists of two values Malignant (M) and Benign (B). For obtaining a model we need to split the dataset into training set, validation set and testing set. A Testing set is used to predict how good the model performs on unseen data.

### 1.3. Metric

Before applying ML algorithms, it is important to decide how we will evaluate their performance. Depending on the task at hand, accuracy may or may not be a good metric by which to judge the ML model. As it is a classification problem so we can use F1 score for measuring the performances of the model.

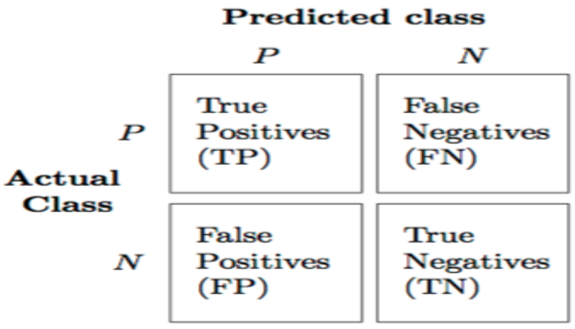
**F1 Score:** In tumor cells classification is important to avoid false negatives because if a malignant tumor is predict as benign the patient will not receive treatment. That's why F1 is a ideal metric to score our model.

recall = True positive / (True positive + False negative)

precision = True positive / (True positive + False positive)

$$\mathrm{F1} = 2 * \frac{\mathrm{precision} * \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}$$

F1 score is the harmonic average of the precision and recall. This score can range from 0 to 1, with 1 being the best possible F1 score i.e. perfect precision and recall. And 0 is the worst F1 score.



|  |  | Predicted class *P* | Predicted class *N* |
|---|---|---|---|
| Actual Class | *P* | True Positives (TP) | False Negatives (FN) |
| Actual Class | *N* | False Positives (FP) | True Negatives (TN) |

Using F1 Score formula.

In statistical analysis of binary classification, the F1 score is a measure of a test's accuracy. It considers both the precision p and the recall r of the test to compute the score: p is the number of correct positive results divided by the number of all positive results returned by the classifier, and r is the number of correct positive results divided by the number of all relevant samples (all samples that should have been identified as positive).

## 2. Analysis

## 2.1. Data Exploration

To create a model to predict whether a tumor cell is benign or malignant we will use a labelled dataset which stems from kaggle and is available at https://www.kaggle.com/uciml/breast-cancer-wisconsin-data .

Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image.

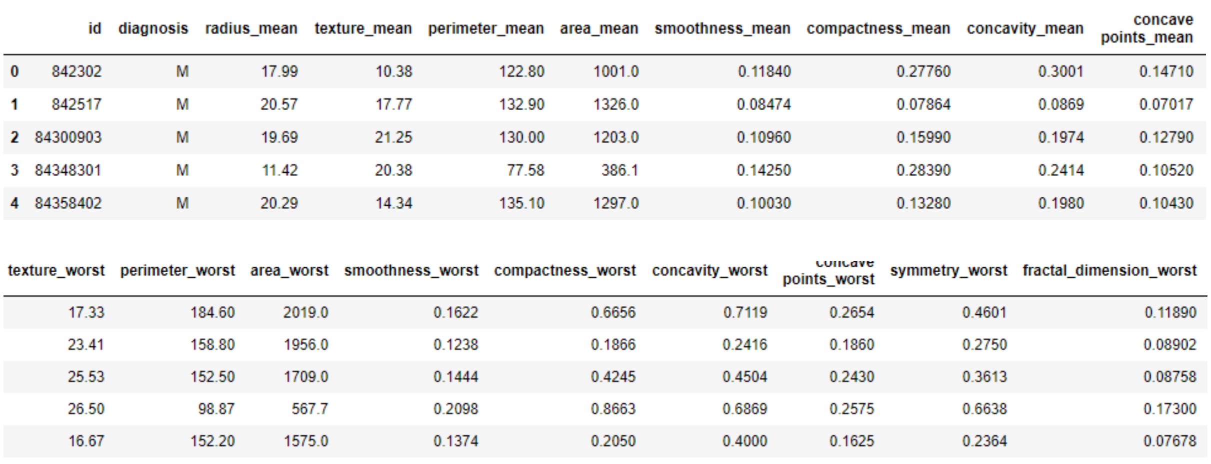
The dataset have the following structure:

- Number of instances: 569
- Number of attributes: 32 (ID, diagnosis, 30 real-valued input features)
- Diagnosis (M = malignant, B = benign)
- Missing attribute values: none
- Class distribution: 357 benign, 212 malignant
- All feature values are recoded with four significant digits.

- Ten real-valued features are computed for each cell nucleus:
  a) radius (mean of distances from center to points on the perimeter)
  b) texture (standard deviation of gray-scale values)
  c) perimeter
  d) area
  e) smoothness (local variation in radius lengths)
  f) compactness (perimeter^2 / area - 1.0)
  g) concavity (severity of concave portions of the contour)
  h) concave points (number of concave portions of the contour)
  i) symmetry
  j) 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 instance, field 3 is Mean Radius, field 13 is Radius SE, and field 23 is Worst Radius.

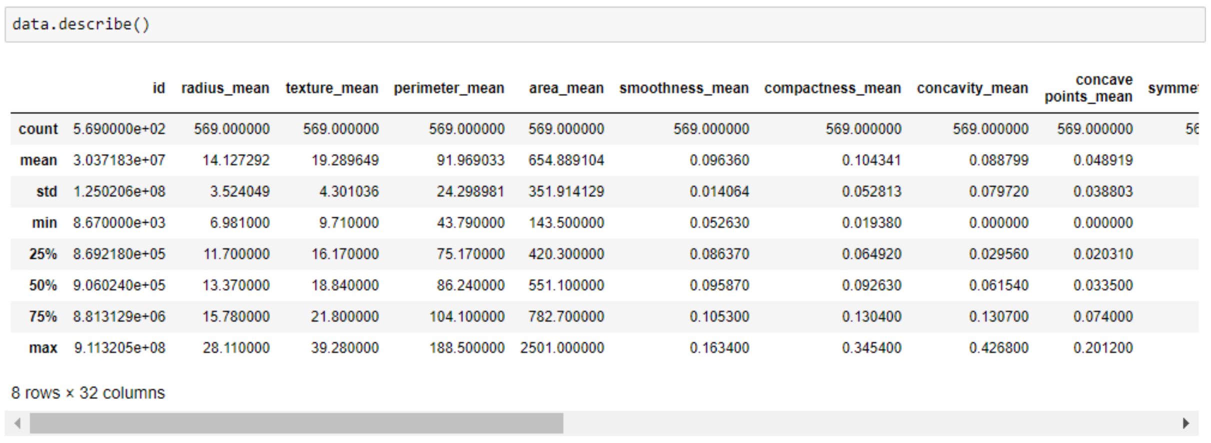
Before beginning to implement the solution, it is worthwhile to explore the data. Below lies some instances of the dataset.



| | id | diagnosis | radius_mean | texture_mean | perimeter_mean | area_mean | smoothness_mean | compactness_mean | concavity_mean | concave points_mean |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 842302 | M | 17.99 | 10.38 | 122.80 | 1001.0 | 0.11840 | 0.27760 | 0.3001 | 0.14710 |
| 1 | 842517 | M | 20.57 | 17.77 | 132.90 | 1326.0 | 0.08474 | 0.07864 | 0.0869 | 0.07017 |
| 2 | 84300903 | M | 19.69 | 21.25 | 130.00 | 1203.0 | 0.10960 | 0.15990 | 0.1974 | 0.12790 |
| 3 | 84348301 | M | 11.42 | 20.38 | 77.58 | 386.1 | 0.14250 | 0.28390 | 0.2414 | 0.10520 |
| 4 | 84358402 | M | 20.29 | 14.34 | 135.10 | 1297.0 | 0.10030 | 0.13280 | 0.1980 | 0.10430 |

| texture_worst | perimeter_worst | area_worst | smoothness_worst | compactness_worst | concavity_worst | concave points_worst | symmetry_worst | fractal_dimension_worst |
|---|---|---|---|---|---|---|---|---|
| 17.33 | 184.60 | 2019.0 | 0.1622 | 0.6656 | 0.7119 | 0.2654 | 0.4601 | 0.11890 |
| 23.41 | 158.80 | 1956.0 | 0.1238 | 0.1866 | 0.2416 | 0.1860 | 0.2750 | 0.08902 |
| 25.53 | 152.50 | 1709.0 | 0.1444 | 0.4245 | 0.4504 | 0.2430 | 0.3613 | 0.08758 |
| 26.50 | 98.87 | 567.7 | 0.2098 | 0.8663 | 0.6869 | 0.2575 | 0.6638 | 0.17300 |
| 16.67 | 152.20 | 1575.0 | 0.1374 | 0.2050 | 0.4000 | 0.1625 | 0.2364 | 0.07678 |

For generating the descriptive statistics we have implemented describe() function that summarizes the central tendency and dispersion of the dataset.



```
data.describe()
```

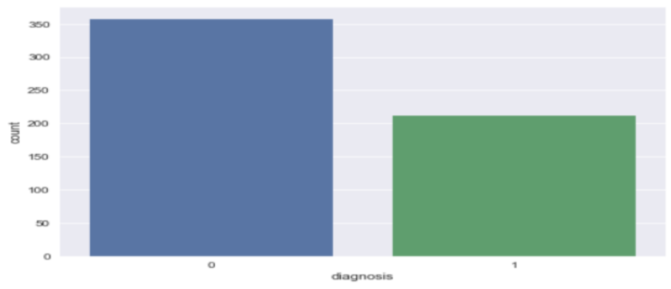
| | id | radius_mean | texture_mean | perimeter_mean | area_mean | smoothness_mean | compactness_mean | concavity_mean | concave points_mean | symme |
|---|---|---|---|---|---|---|---|---|---|---|
| count | 5.690000e+02 | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 56 |
| mean | 3.037183e+07 | 14.127292 | 19.289649 | 91.969033 | 654.889104 | 0.096360 | 0.104341 | 0.088799 | 0.048919 | |
| std | 1.250206e+08 | 3.524049 | 4.301036 | 24.298981 | 351.914129 | 0.014064 | 0.052813 | 0.079720 | 0.038803 | |
| min | 8.670000e+03 | 6.981000 | 9.710000 | 43.790000 | 143.500000 | 0.052630 | 0.019380 | 0.000000 | 0.000000 | |
| 25% | 8.692180e+05 | 11.700000 | 16.170000 | 75.170000 | 420.300000 | 0.086370 | 0.064920 | 0.029560 | 0.020310 | |
| 50% | 9.060240e+05 | 13.370000 | 18.840000 | 86.240000 | 551.100000 | 0.095870 | 0.092630 | 0.061540 | 0.033500 | |
| 75% | 8.813129e+06 | 15.780000 | 21.800000 | 104.100000 | 782.700000 | 0.105300 | 0.130400 | 0.130700 | 0.074000 | |
| max | 9.113205e+08 | 28.110000 | 39.280000 | 188.500000 | 2501.000000 | 0.163400 | 0.345400 | 0.426800 | 0.201200 | |

8 rows × 32 columns

To get a better understanding of the dataset, the first thing is to note the class distribution which can be visualized from the graph below. It shows that the number of Benign cancer cells is 357 and number of Malignant cancer cells is 212.

```
# Values of 'Benign' and 'Malignant' cancer cells
sns.countplot(x="diagnosis", data=data)
data.loc[:,'diagnosis'].value_counts()
```

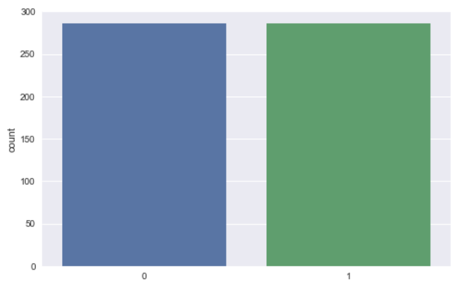
```
0    357
1    212
Name: diagnosis, dtype: int64
```



As we can see above that the distribution of both classes is not uniform for which Feature sampling such as SMOTE and ADASYNE is implemented to resample the classes. And the result obtained can be seen below.

```
#Passing the scaled data for sampling using Synthetic Minority Oversampling Technique (SMOTE)
sm = SMOTE(random_state=12, ratio = 1.0)
X_train_res, y_train_res = sm.fit_sample(X_train_std, y_train)
print(sorted(Counter(y_train_res).items()))
sns.set(rc={'figure.figsize':(8,5)})
sns.countplot(y_train_res)
```

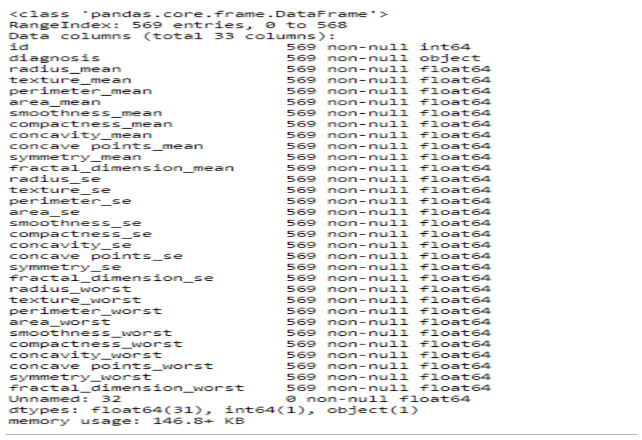
```
[(0, 286), (1, 286)]

<matplotlib.axes._subplots.AxesSubplot at 0x256aea56e10>
```



## 2.2. Exploratory Visualization

First we need to have a look at the dataset to find out the number of entries, the columns or attributes, their types and having any null values or not.

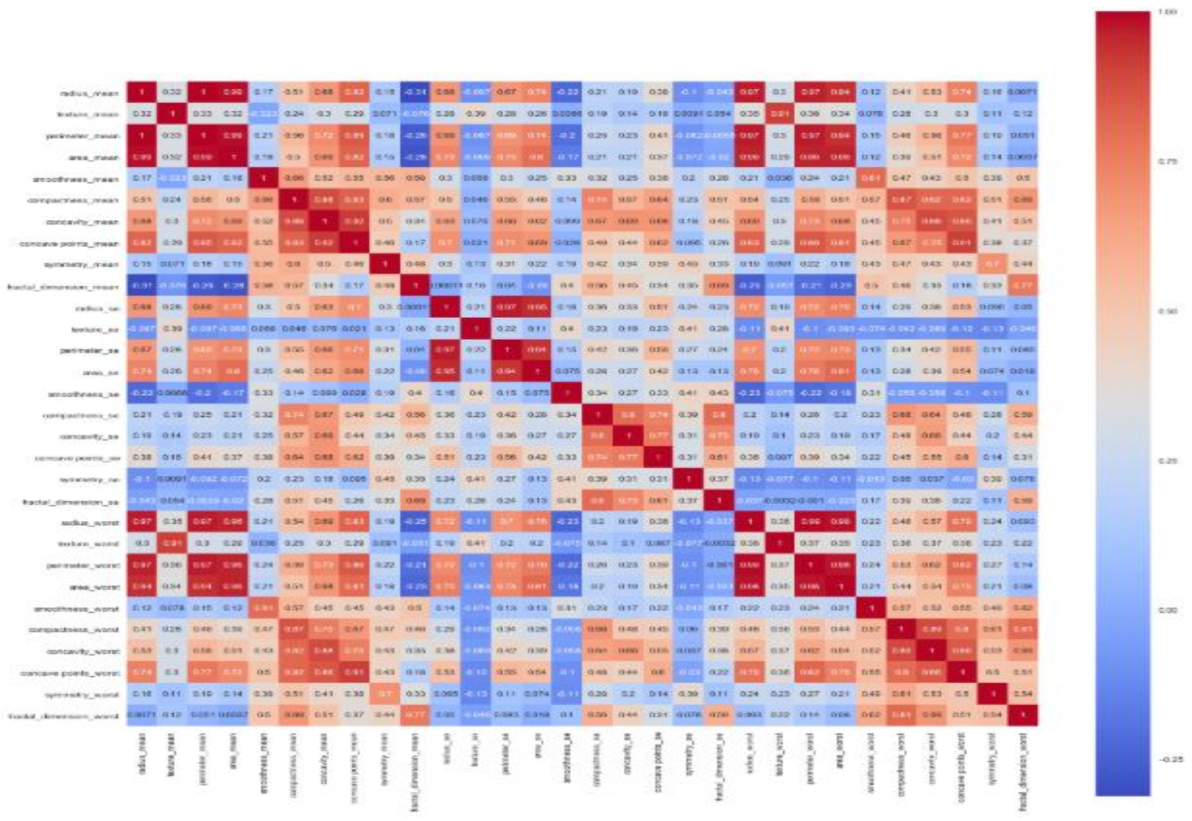
```
# Analyzing the data
data.describe()
data.info()
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
id                         569 non-null int64
diagnosis                  569 non-null object
radius_mean                569 non-null float64
texture_mean               569 non-null float64
perimeter_mean             569 non-null float64
area_mean                  569 non-null float64
smoothness_mean            569 non-null float64
compactness_mean           569 non-null float64
concavity_mean             569 non-null float64
concave points_mean        569 non-null float64
symmetry_mean              569 non-null float64
fractal_dimension_mean     569 non-null float64
radius_se                  569 non-null float64
texture_se                 569 non-null float64
perimeter_se               569 non-null float64
area_se                    569 non-null float64
smoothness_se              569 non-null float64
compactness_se             569 non-null float64
concavity_se               569 non-null float64
concave points_se          569 non-null float64
symmetry_se                569 non-null float64
fractal_dimension_se       569 non-null float64
radius_worst               569 non-null float64
texture_worst              569 non-null float64
perimeter_worst            569 non-null float64
area_worst                 569 non-null float64
smoothness_worst           569 non-null float64
compactness_worst          569 non-null float64
concavity_worst            569 non-null float64
concave points_worst       569 non-null float64
symmetry_worst             569 non-null float64
fractal_dimension_worst    569 non-null float64
Unnamed: 32                0 non-null float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

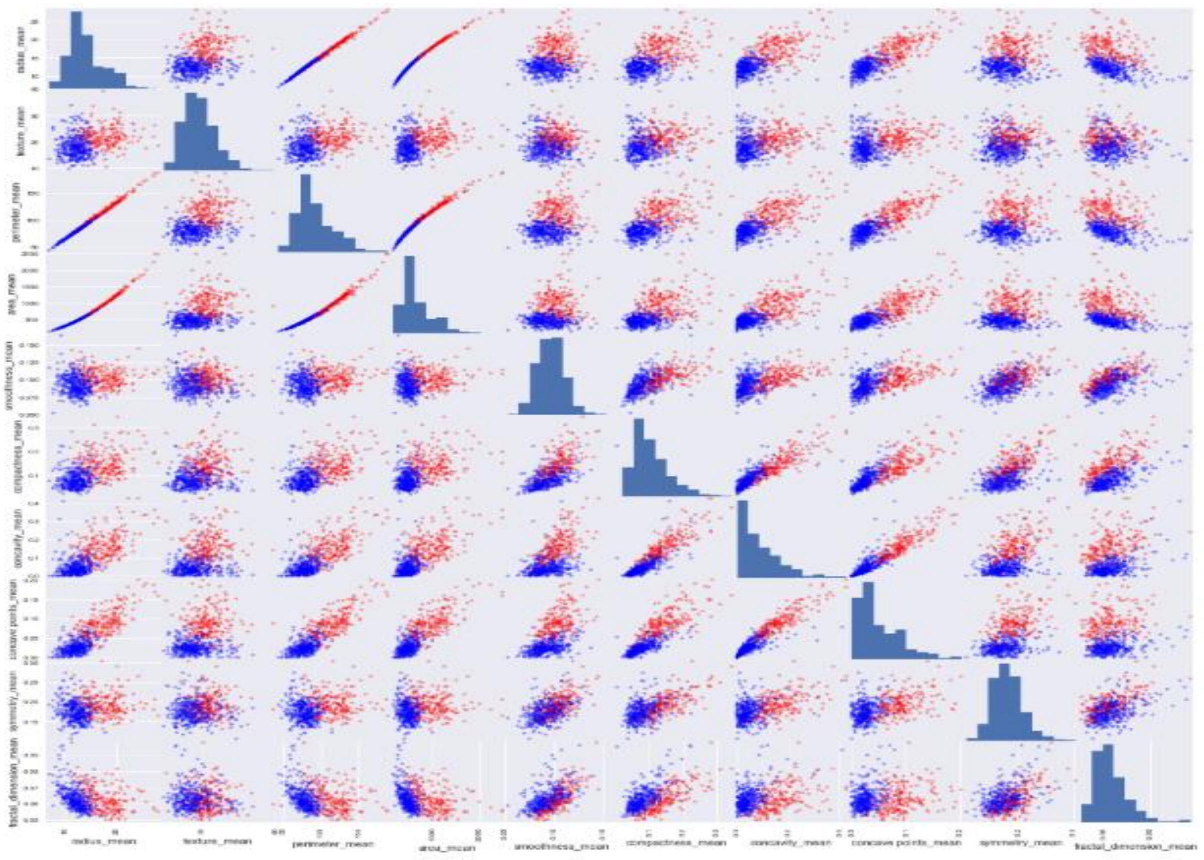
For finding out the correlation between the features, heat map is used. The heat map below shows the numerical correlation between different features, where dark blue indicates negative correlation, neutral indicates no correlation, and red indicates positive correlation. Radius, Perimeter and Area have strong positive correlation.

```
#Below we will use Seaborn to create a heat map of the correlations between the features.
plt.figure(figsize=(20,20))
sns.heatmap(data.corr(), annot=True, square=True, cmap='coolwarm');
```



We can also use scatter plot matrix to find pair wise relationships to check the correlations between the mean features. The plot shows that Radius, Perimeter and Area have strong correlation.

```
features_mean= list(data.columns[1:11])
color_dic = {'M':'red', 'B':'blue'}
colors = data['diagnosis'].map(lambda x: color_dic.get(x))

sm = pd.plotting.scatter_matrix(data[features_mean], c=colors, alpha=0.4, figsize=((20,20)));

plt.show()
```



## 2.3. Algorithms and Techniques

### Machine Learning Algorithms

In this project, seven different classifiers have been used to analyze the results and get a higher F1 score. A brief explanation on each of the classifier is given below.

### Stochastic Gradient Descent (SGD) Classifier

SGD Classifier implements a plain stochastic gradient descent learning routine which supports different loss functions and penalties for classification. The gradient of the loss is estimated for each sample at a time and the model is updated along the way with a decreasing strength schedule (aka learning rate). SGD allows minibatch (online/out-of-core) learning. For best results using the default learning rate schedule, the data should have zero mean and unit variance.

## Logistic Regression(LR) Classifier

It was developed by statistician David Cox in 1958. The binary logistic model is used to estimate the probability of a binary response based on one or more predictor (or independent) variables (features). It allows one to say that the presence of a risk factor increases the odds of a given outcome by a specific factor. The model itself simply models probability of output in terms of input, and does not perform statistical classification, though it can be used to make a classifier, for instance by choosing a cutoff value and classifying inputs with probability greater than the cutoff as one class and below the cutoff as the other.

## Decisions Trees Classifier(DTC)

Decision tree is one of the most popular machine learning algorithms used all along. Decision trees are used for both classification and regression problems. Decision trees are commonly used in operations research, specifically in decision analysis, to help identify a strategy most likely to reach a goal, but are also a popular tool in machine learning.

A decision tree is a tree where each node represents a feature (attribute), each link (branch) represents a decision (rule) and each leaf represents an outcome (categorical or continues value).

## XGBoost (Extreme Gradient Boosting) Classifier

XGBoost is an implementation of gradient boosted decision trees designed for speed and performance. XGBoost dominates structured or tabular datasets on classification and regression predictive modeling problems. XGBoost (Extreme Gradient Boosting) belongs to a family of boosting algorithms and uses the gradient boosting (GBM) framework at its core. Boosting is an ensemble technique where new models are added to correct the errors made by existing models. Models are added sequentially until no further improvements can be made. It is called gradient boosting because it uses a gradient descent algorithm to minimize the loss when adding new models.

## Support Vector Machines(SVM)

Support Vectors are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier. An SVM model is a representation of the examples as points in space mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

## Random Forest (RF) Classifier

Random Forest or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

### Gradient Boosting(GB) Classifier

Gradient Boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

## 2.4. Benchmark

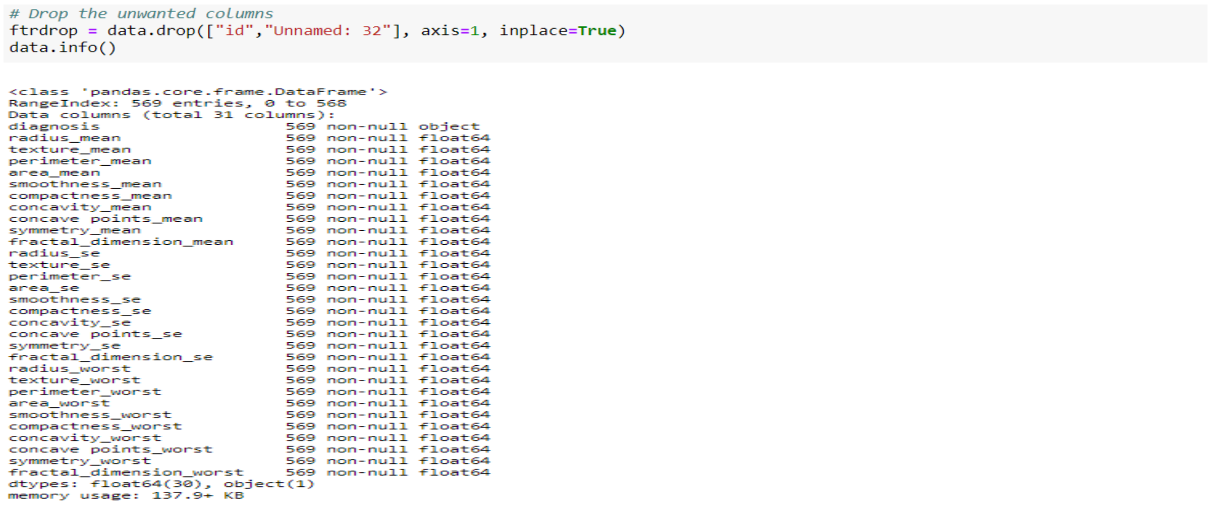
Logistic Regression is used as a benchmark model for the dataset. The result obtained by our model will be compared with the result obtained by the benchmark model [3].

The judgement on accuracy has been performed using F1 score since they are mostly used for binary classification problems. Accuracy of Logistic Regression in the referred paper [3] is 96.79% on training set. To realize how much effective our method is, after optimizing our model we will compare the obtained results with results extracted from the benchmark model.

## 3. Methodology

## 3.1. Data Preprocessing

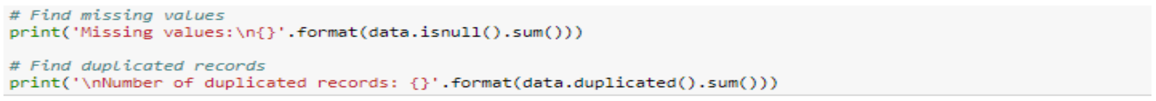
After describing the information on data, we found some unwanted columns. The "Unannamed: 32" feature has only NaN (not a number) values. In the dataset, the attribute ‘id’ won’t help in prediction that whether a tumor is Benign or Malignant. So, both the columns has to be removed.



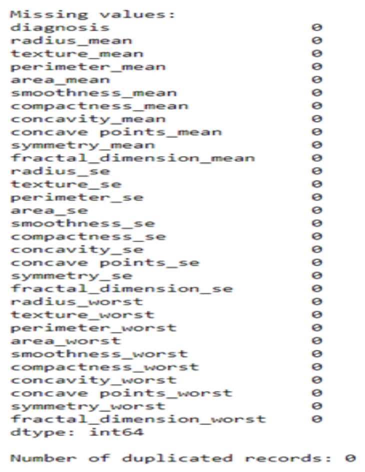
```
# Drop the unwanted columns
ftrdrop = data.drop(["id","Unnamed: 32"], axis=1, inplace=True)
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
diagnosis                  569 non-null object
radius_mean                569 non-null float64
texture_mean               569 non-null float64
perimeter_mean             569 non-null float64
area_mean                  569 non-null float64
smoothness_mean            569 non-null float64
compactness_mean           569 non-null float64
concavity_mean             569 non-null float64
concave points_mean        569 non-null float64
symmetry_mean              569 non-null float64
fractal_dimension_mean     569 non-null float64
radius_se                  569 non-null float64
texture_se                 569 non-null float64
perimeter_se               569 non-null float64
area_se                    569 non-null float64
smoothness_se              569 non-null float64
compactness_se             569 non-null float64
concavity_se               569 non-null float64
concave points_se          569 non-null float64
symmetry_se                569 non-null float64
fractal_dimension_se       569 non-null float64
radius_worst               569 non-null float64
texture_worst              569 non-null float64
perimeter_worst            569 non-null float64
area_worst                 569 non-null float64
smoothness_worst           569 non-null float64
compactness_worst          569 non-null float64
concavity_worst            569 non-null float64
concave points_worst       569 non-null float64
symmetry_worst             569 non-null float64
fractal_dimension_worst    569 non-null float64
dtypes: float64(30), object(1)
memory usage: 137.9+ KB
```

We need to verify whether any row has missing values or if there is any duplicated records.

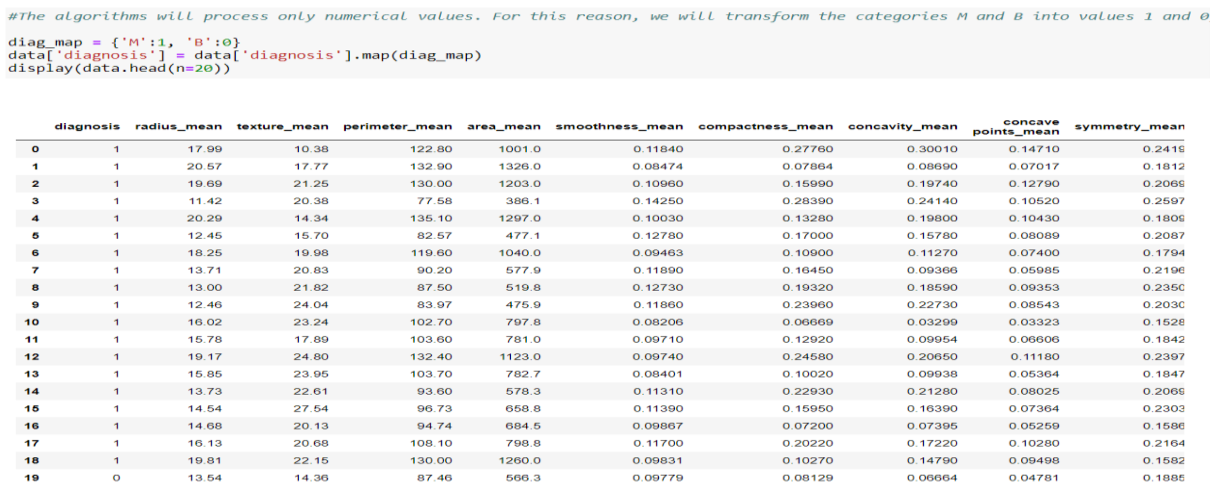


```
# Find missing values
print('Missing values:\n{}'.format(data.isnull().sum()))

# Find duplicated records
print('\nNumber of duplicated records: {}'.format(data.duplicated().sum()))
```



```
Missing values:
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

Number of duplicated records: 0
```

As it is a classification dataset, the algorithms will process numerical values 0 for Benign and 1 for Malignant present in ‘diagnosis’ column.

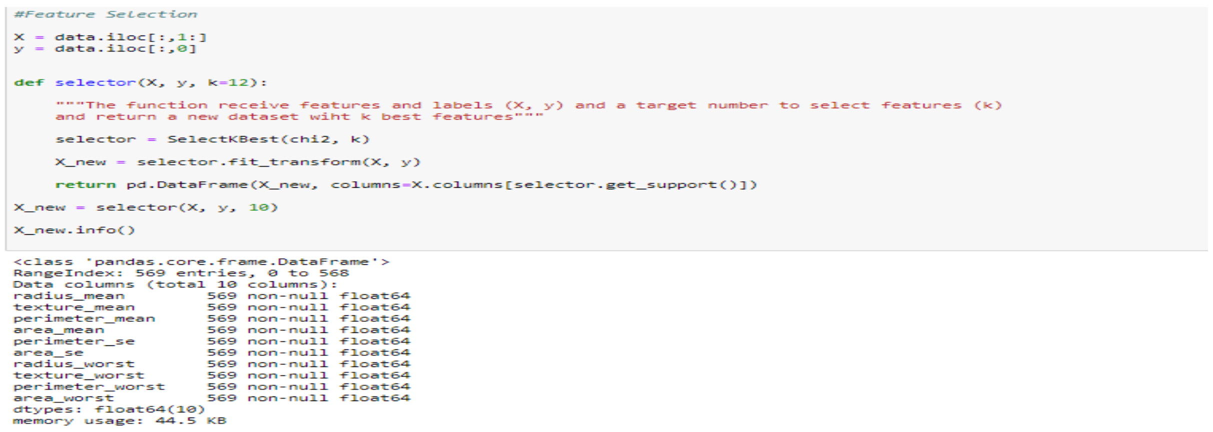


```
#The algorithms will process only numerical values. For this reason, we will transform the categories M and B into values 1 and 0

diag_map = {'M':1, 'B':0}
data['diagnosis'] = data['diagnosis'].map(diag_map)
display(data.head(n=20))
```

| | diagnosis | radius_mean | texture_mean | perimeter_mean | area_mean | smoothness_mean | compactness_mean | concavity_mean | concave points_mean | symmetry_mean |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 17.99 | 10.38 | 122.80 | 1001.0 | 0.11840 | 0.27760 | 0.30010 | 0.14710 | 0.2419 |
| 1 | 1 | 20.57 | 17.77 | 132.90 | 1326.0 | 0.08474 | 0.07864 | 0.08690 | 0.07017 | 0.1812 |
| 2 | 1 | 19.69 | 21.25 | 130.00 | 1203.0 | 0.10960 | 0.15990 | 0.19740 | 0.12790 | 0.2069 |
| 3 | 1 | 11.42 | 20.38 | 77.58 | 386.1 | 0.14250 | 0.28390 | 0.24140 | 0.10520 | 0.2597 |
| 4 | 1 | 20.29 | 14.34 | 135.10 | 1297.0 | 0.10030 | 0.13280 | 0.19800 | 0.10430 | 0.1809 |
| 5 | 1 | 12.45 | 15.70 | 82.57 | 477.1 | 0.12780 | 0.17000 | 0.15780 | 0.08089 | 0.2087 |
| 6 | 1 | 18.25 | 19.98 | 119.60 | 1040.0 | 0.09463 | 0.10900 | 0.11270 | 0.07400 | 0.1794 |
| 7 | 1 | 13.71 | 20.83 | 90.20 | 577.9 | 0.11890 | 0.16450 | 0.09366 | 0.05985 | 0.2196 |
| 8 | 1 | 13.00 | 21.82 | 87.50 | 519.8 | 0.12730 | 0.19320 | 0.18590 | 0.09353 | 0.2350 |
| 9 | 1 | 12.46 | 24.04 | 83.97 | 475.9 | 0.11860 | 0.23960 | 0.22730 | 0.08543 | 0.2030 |
| 10 | 1 | 16.02 | 23.24 | 102.70 | 797.8 | 0.08206 | 0.06669 | 0.03299 | 0.03323 | 0.1528 |
| 11 | 1 | 15.78 | 17.89 | 103.60 | 781.0 | 0.09710 | 0.12920 | 0.09954 | 0.06606 | 0.1842 |
| 12 | 1 | 19.17 | 24.80 | 132.40 | 1123.0 | 0.09740 | 0.24580 | 0.20650 | 0.11180 | 0.2397 |
| 13 | 1 | 15.85 | 23.95 | 103.70 | 782.7 | 0.08401 | 0.10020 | 0.09938 | 0.05364 | 0.1847 |
| 14 | 1 | 13.73 | 22.61 | 93.60 | 578.3 | 0.11310 | 0.22930 | 0.21280 | 0.08025 | 0.2069 |
| 15 | 1 | 14.54 | 27.54 | 96.73 | 658.8 | 0.11390 | 0.15950 | 0.16390 | 0.07364 | 0.2303 |
| 16 | 1 | 14.68 | 20.13 | 94.74 | 684.5 | 0.09867 | 0.07200 | 0.07395 | 0.05259 | 0.1586 |
| 17 | 1 | 16.13 | 20.68 | 108.10 | 798.8 | 0.11700 | 0.20220 | 0.17220 | 0.10280 | 0.2164 |
| 18 | 1 | 19.81 | 22.15 | 130.00 | 1260.0 | 0.09831 | 0.10270 | 0.14790 | 0.09498 | 0.1582 |
| 19 | 0 | 13.54 | 14.36 | 87.46 | 566.3 | 0.09779 | 0.08129 | 0.06664 | 0.04781 | 0.1885 |

## Feature Selection

This preprocessing step is used to select the best features in order to reduce data dimension while doing predictive analysis and we are going to perform Feature Selection using SelectKBest() module in sklearn combination of features. SelectKBest() removes all but the k highest scoring features, The value K has been chosen as 10 based on the analysis of heatmap.



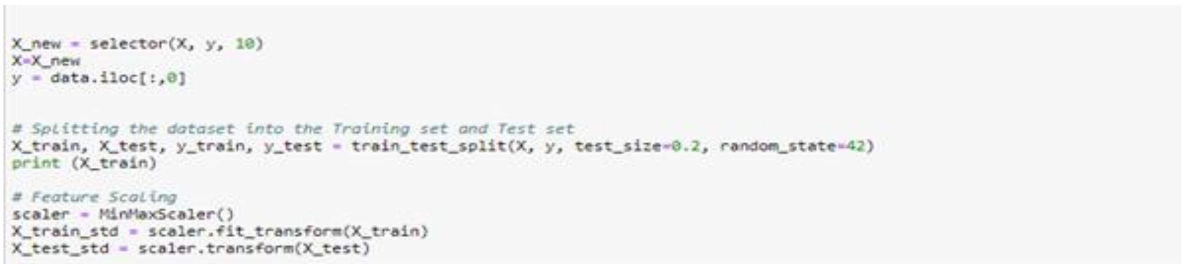
```
#Feature Selection

X = data.iloc[:,1:]
y = data.iloc[:,0]

def selector(X, y, k=12):

    """The function receive features and labels (X, y) and a target number to select features (k)
    and return a new dataset wiht k best features"""

    selector = SelectKBest(chi2, k)

    X_new = selector.fit_transform(X, y)

    return pd.DataFrame(X_new, columns=X.columns[selector.get_support()])

X_new = selector(X, y, 10)

X_new.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 10 columns):
radius_mean          569 non-null float64
texture_mean         569 non-null float64
perimeter_mean       569 non-null float64
area_mean            569 non-null float64
perimeter_se         569 non-null float64
area_se              569 non-null float64
radius_worst         569 non-null float64
texture_worst        569 non-null float64
perimeter_worst      569 non-null float64
area_worst           569 non-null float64
dtypes: float64(10)
memory usage: 44.5 KB
```

## Feature Scaling and Standardization

Since the range of values of raw data varies widely, in some machine learning algorithms, objective functions will not work properly without normalization. For example, the majority of classifiers calculate the distance between two points by the Euclidean distance. If one of the features has a broad range of values, the distance will be governed by this particular feature. Therefore, the range of all features should be normalized so that each feature contributes approximately proportionately to the final distance.

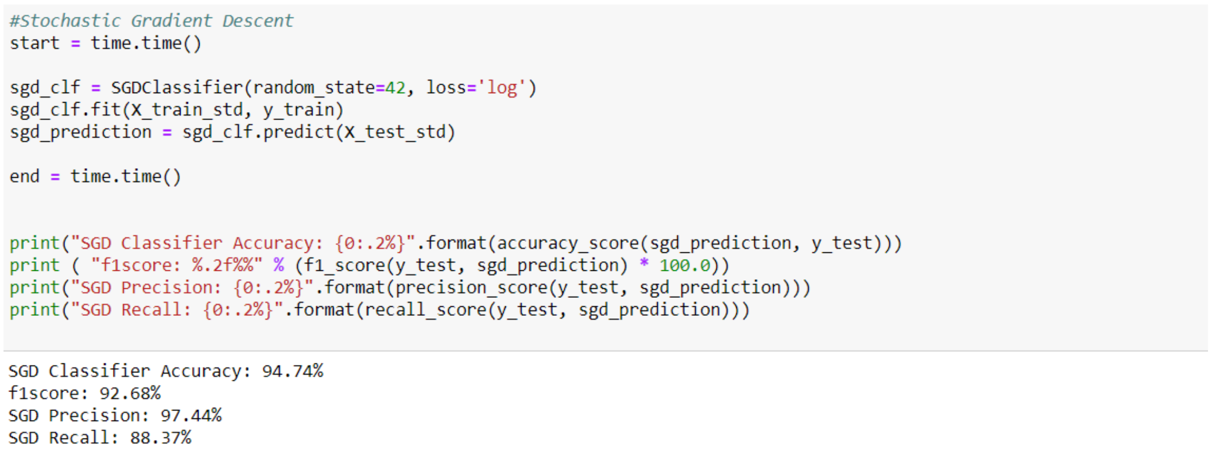
It is almost always a good idea to split and standardize the data before we start training any ML algorithms. In sklearn we can do this with train_test_split() and MinMaxScalar(). MinMaxScalar () is implemented on data without labels.



```
X_new = selector(X, y, 10)
X=X_new
y = data.iloc[:,0]


# Splitting the dataset into the Training set and Test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print (X_train)

# Feature Scaling
scaler = MinMaxScaler()
X_train_std = scaler.fit_transform(X_train)
X_test_std = scaler.transform(X_test)
```

## 3.2. Implementation

To predict a tumor is Benign or Malignant we have chosen some of the best classifier algorithms to perform the task. Initially dimension reduction was performed using SelectKBest(). And then the data set was split into training and testing set followed by scaling. These scaled data was tested on different classifier without parameter tuning. To implement it, I used Scikit Learn that is natively installed from Anaconda. And also installed the XGBoost Classifier. The performance of each classifier is taken into account on the basis of Accuracy, F1 score, precision and recall values.



```
#Stochastic Gradient Descent
start = time.time()

sgd_clf = SGDClassifier(random_state=42, loss='log')
sgd_clf.fit(X_train_std, y_train)
sgd_prediction = sgd_clf.predict(X_test_std)

end = time.time()


print("SGD Classifier Accuracy: {0:.2%}".format(accuracy_score(sgd_prediction, y_test)))
print ( "f1score: %.2f%%" % (f1_score(y_test, sgd_prediction) * 100.0))
print("SGD Precision: {0:.2%}".format(precision_score(y_test, sgd_prediction)))
print("SGD Recall: {0:.2%}".format(recall_score(y_test, sgd_prediction)))
```

```
SGD Classifier Accuracy: 94.74%
f1score: 92.68%
SGD Precision: 97.44%
SGD Recall: 88.37%
```

```python
#Logistic Regression

start = time.time()

lr_clf =LogisticRegression(random_state=42)
lr_clf.fit(X_train_std, y_train)
lr_prediction = lr_clf.predict(X_test_std)
end = time.time()

print("Logistic Regression Accuracy: {0:.2%}".format(accuracy_score(lr_prediction, y_test)))
print ( "f1score: %.2f%%" % (f1_score(y_test, lr_prediction) * 100.0))
print("Logistic Regression Precision: {0:.2%}".format(precision_score(y_test, lr_prediction)))
print("Logistic Regression Recall: {0:.2%}".format(recall_score(y_test, lr_prediction)))
```

```
Logistic Regression Accuracy: 94.74%
f1score: 92.68%
Logistic Regression Precision: 97.44%
Logistic Regression Recall: 88.37%
```

```python
# Decision Tree Classifier

start = time.time()

dct_clf =DecisionTreeClassifier(random_state=42)
dct_clf.fit(X_train_std, y_train)
dct_prediction = dct_clf.predict(X_test_std)

end = time.time()


print("DCT Classifier Accuracy: {0:.2%}".format(accuracy_score(dct_prediction, y_test)))
print ( "f1score: %.2f%%" % (f1_score(y_test, dct_prediction) * 100.0))
print("Decision Tree Classifier Precision: {0:.2%}".format(precision_score(y_test, dct_prediction)))
print("Decision Tree Classifier Recall: {0:.2%}".format(recall_score(y_test, dct_prediction)))
```

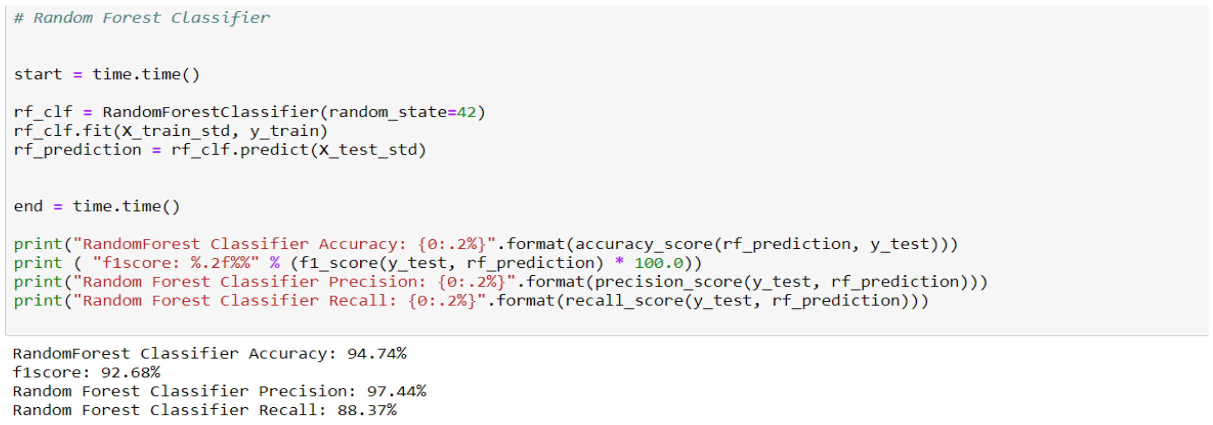
```
DCT Classifier Accuracy: 92.98%
f1score: 90.70%
Decision Tree Classifier Precision: 90.70%
Decision Tree Classifier Recall: 90.70%
```

```python
# XGB Classifier

start = time.time()

X_train = np.array(X_train_std)
X_test = np.array(X_test_std)

xgb_clf =XGBClassifier(random_state=42)
xgb_clf.fit(X_train_std, y_train.ravel())
xgb_prediction = xgb_clf.predict(X_test_std)

end = time.time()


print("XGB Classifier Accuracy: {0:.2%}".format(accuracy_score(xgb_prediction, y_test)))
print ( "f1score: %.2f%%" % (f1_score(y_test, xgb_prediction) * 100.0))
print("XGB Classifier Precision: {0:.2%}".format(precision_score(y_test, xgb_prediction)))
print("XGB Classifier Recall: {0:.2%}".format(recall_score(y_test, xgb_prediction)))
```

```
XGB Classifier Accuracy: 94.74%
f1score: 92.86%
XGB Classifier Precision: 95.12%
XGB Classifier Recall: 90.70%
```

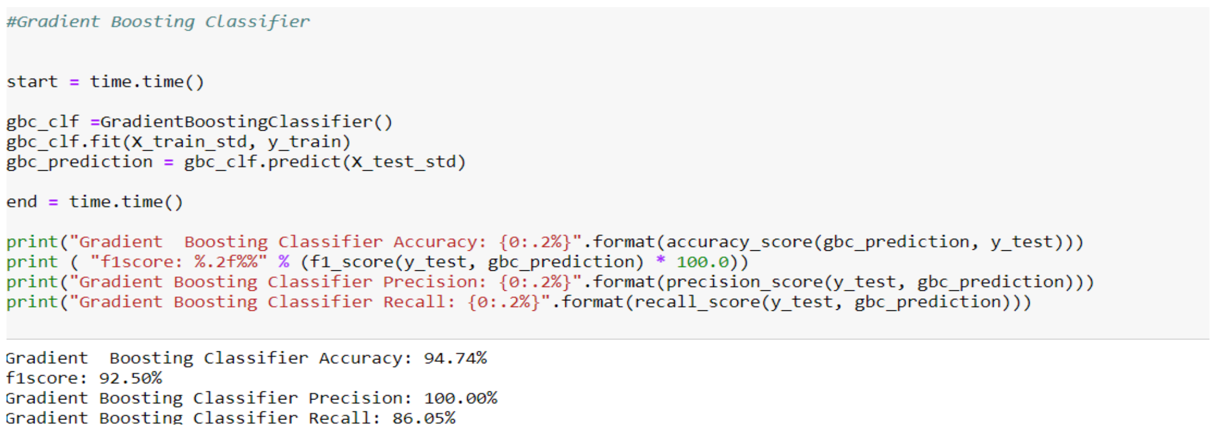
```python
# SVM Classifier


start = time.time()

svm_clf = svm.SVC()
svm_clf.fit(X_train_std, y_train)
svm_prediction = svm_clf.predict(X_test_std)

end = time.time()


print("SVM Classifier Accuracy: {0:.2%}".format(accuracy_score(svm_prediction, y_test)))
print ( "f1score: %.2f%%" % (f1_score(y_test, svm_prediction) * 100.0))
print("SVM Classifier Precision: {0:.2%}".format(precision_score(y_test, svm_prediction)))
print("SVM Classifier Recall: {0:.2%}".format(recall_score(y_test, svm_prediction)))
```

```
SVM Classifier Accuracy: 92.98%
f1score: 90.00%
SVM Classifier Precision: 97.30%
SVM Classifier Recall: 83.72%
```



```
# Random Forest Classifier


start = time.time()

rf_clf = RandomForestClassifier(random_state=42)
rf_clf.fit(X_train_std, y_train)
rf_prediction = rf_clf.predict(X_test_std)


end = time.time()

print("RandomForest Classifier Accuracy: {0:.2%}".format(accuracy_score(rf_prediction, y_test)))
print ( "f1score: %.2f%%" % (f1_score(y_test, rf_prediction) * 100.0))
print("Random Forest Classifier Precision: {0:.2%}".format(precision_score(y_test, rf_prediction)))
print("Random Forest Classifier Recall: {0:.2%}".format(recall_score(y_test, rf_prediction)))
```

```
RandomForest Classifier Accuracy: 94.74%
f1score: 92.68%
Random Forest Classifier Precision: 97.44%
Random Forest Classifier Recall: 88.37%
```



```
#Gradient Boosting Classifier


start = time.time()

gbc_clf =GradientBoostingClassifier()
gbc_clf.fit(X_train_std, y_train)
gbc_prediction = gbc_clf.predict(X_test_std)

end = time.time()

print("Gradient  Boosting Classifier Accuracy: {0:.2%}".format(accuracy_score(gbc_prediction, y_test)))
print ( "f1score: %.2f%%" % (f1_score(y_test, gbc_prediction) * 100.0))
print("Gradient Boosting Classifier Precision: {0:.2%}".format(precision_score(y_test, gbc_prediction)))
print("Gradient Boosting Classifier Recall: {0:.2%}".format(recall_score(y_test, gbc_prediction)))
```

```
Gradient  Boosting Classifier Accuracy: 94.74%
f1score: 92.50%
Gradient Boosting Classifier Precision: 100.00%
Gradient Boosting Classifier Recall: 86.05%
```

Thus, among all the classifier without parameter tuning, F1 score of SGD, Logistic Regression, XGBclassifier and Random Forest is same i.e. 92.68%.

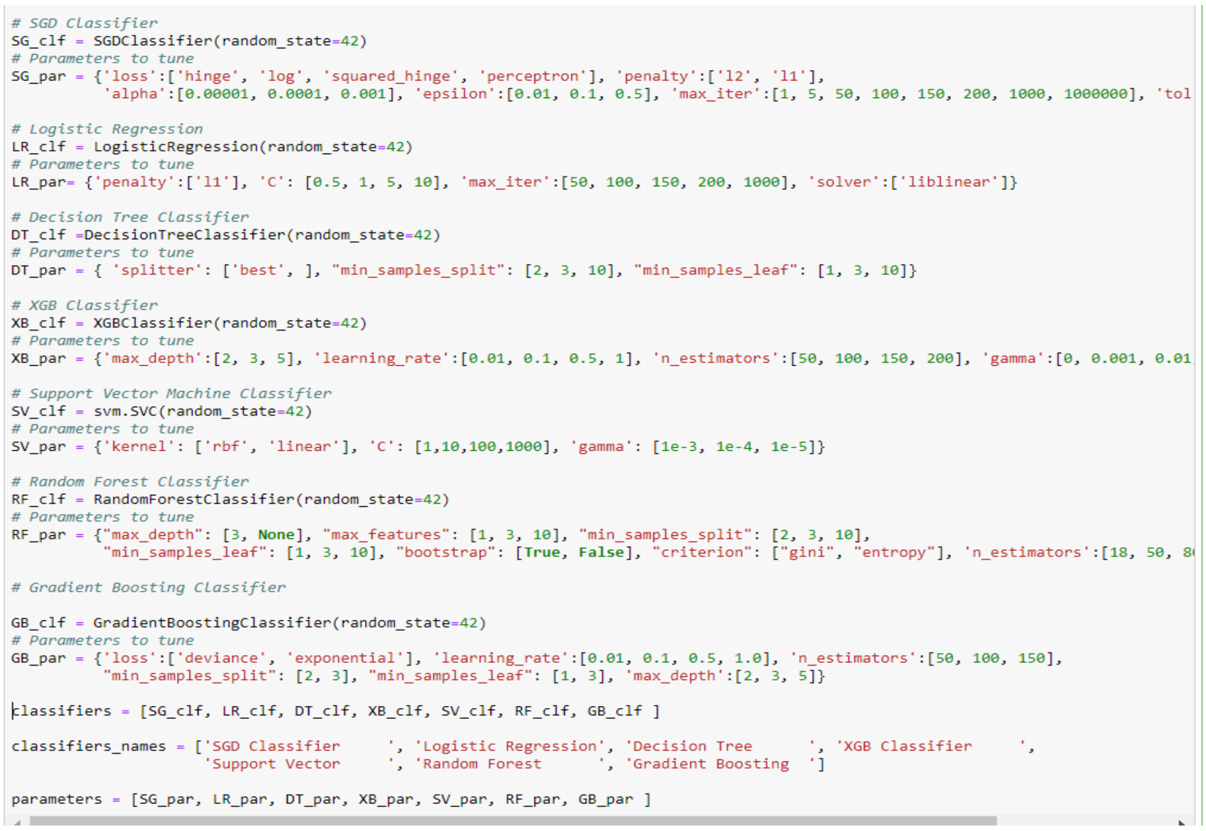
Our solution was implemented using python 3.5 as well as a few common python libraries for machine learning and data science. Pandas was used to load and handle the raw data, Numpy to handle the math efficiently, matplotlib and seaborn for visualizations. “Model_selection” module was used to cross validate and split data into a training and testing set as well as to tune the hyper-parameters of the model. “ensemble”, “linear_model”, “tree” module was used for implementing different classifiers. For scaling “preprocessing” module was implemented. Feature selection was done using “feature_selection” module. For performing resampling “over_sampling” module was implemented. “Metrics” module was implemented for the performance evaluation of the models. Implementing the above classifiers was not a complex issue because a fairly nice documentation for implementation of each classifier with small examples has been provided in the website of scikit-learn.

## 3.3. Refinement

To optimize the parameters of different classifiers, GridSearchCV() has been implemented. It is almost always a good idea to optimize the parameters when implementing a Machine Learning algorithm.

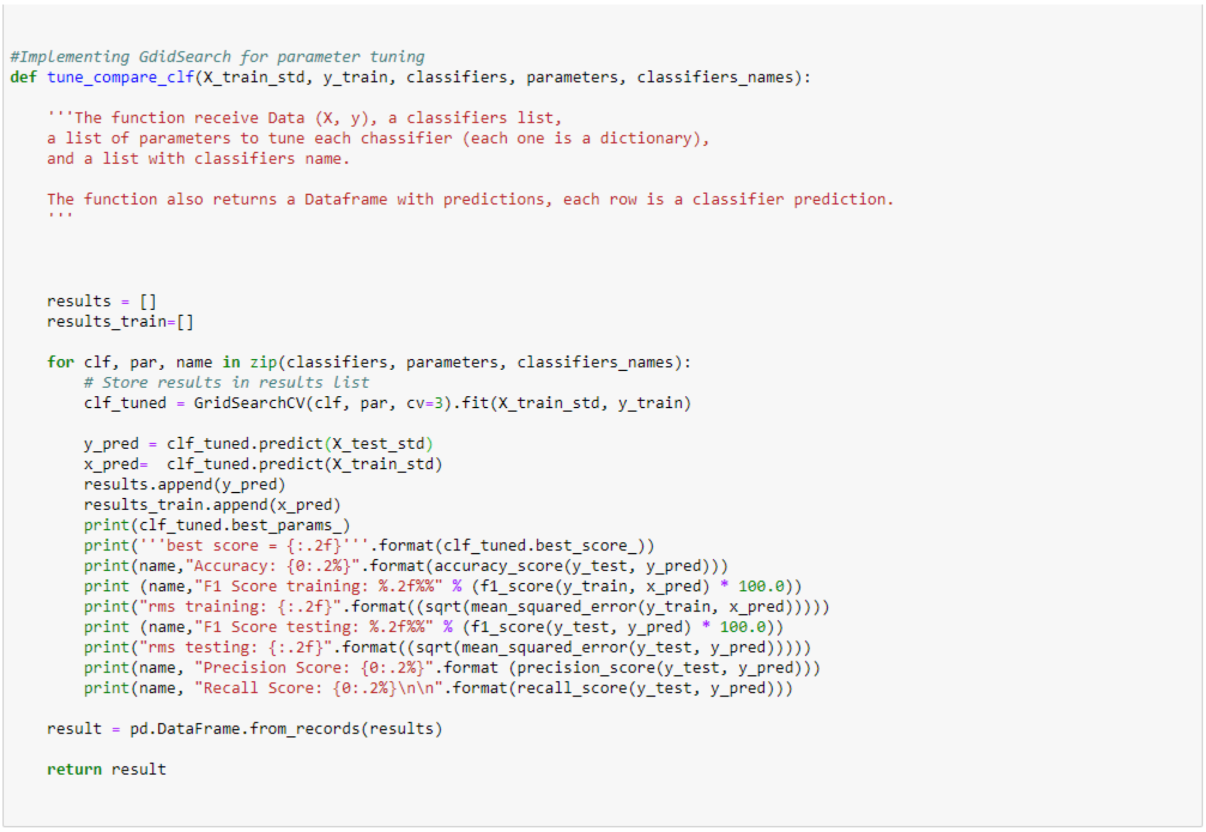
Using GridSearchCV() we did an Exhaustive search over specified parameter values for an estimator. The parameters of the estimator used to apply these methods are optimized by cross-validated grid-search over a parameter grid.

The parameters considered for tuning for different classifiers is mentioned below.



```python
# SGD Classifier
SG_clf = SGDClassifier(random_state=42)
# Parameters to tune
SG_par = {'loss':['hinge', 'log', 'squared_hinge', 'perceptron'], 'penalty':['l2', 'l1'],
          'alpha':[0.00001, 0.0001, 0.001], 'epsilon':[0.01, 0.1, 0.5], 'max_iter':[1, 5, 50, 100, 150, 200, 1000, 1000000], 'tol

# Logistic Regression
LR_clf = LogisticRegression(random_state=42)
# Parameters to tune
LR_par= {'penalty':['l1'], 'C': [0.5, 1, 5, 10], 'max_iter':[50, 100, 150, 200, 1000], 'solver':['liblinear']}

# Decision Tree Classifier
DT_clf =DecisionTreeClassifier(random_state=42)
# Parameters to tune
DT_par = { 'splitter': ['best', ], "min_samples_split": [2, 3, 10], "min_samples_leaf": [1, 3, 10]}

# XGB Classifier
XB_clf = XGBClassifier(random_state=42)
# Parameters to tune
XB_par = {'max_depth':[2, 3, 5], 'learning_rate':[0.01, 0.1, 0.5, 1], 'n_estimators':[50, 100, 150, 200], 'gamma':[0, 0.001, 0.01

# Support Vector Machine Classifier
SV_clf = svm.SVC(random_state=42)
# Parameters to tune
SV_par = {'kernel': ['rbf', 'linear'], 'C': [1,10,100,1000], 'gamma': [1e-3, 1e-4, 1e-5]}

# Random Forest Classifier
RF_clf = RandomForestClassifier(random_state=42)
# Parameters to tune
RF_par = {"max_depth": [3, None], "max_features": [1, 3, 10], "min_samples_split": [2, 3, 10],
          "min_samples_leaf": [1, 3, 10], "bootstrap": [True, False], "criterion": ["gini", "entropy"], 'n_estimators':[18, 50, 8

# Gradient Boosting Classifier

GB_clf = GradientBoostingClassifier(random_state=42)
# Parameters to tune
GB_par = {'loss':['deviance', 'exponential'], 'learning_rate':[0.01, 0.1, 0.5, 1.0], 'n_estimators':[50, 100, 150],
          "min_samples_split": [2, 3], "min_samples_leaf": [1, 3], 'max_depth':[2, 3, 5]}

classifiers = [SG_clf, LR_clf, DT_clf, XB_clf, SV_clf, RF_clf, GB_clf ]

classifiers_names = ['SGD Classifier     ', 'Logistic Regression', 'Decision Tree      ', 'XGB Classifier     ',
                     'Support Vector     ', 'Random Forest      ', 'Gradient Boosting  ']

parameters = [SG_par, LR_par, DT_par, XB_par, SV_par, RF_par, GB_par ]
```

Function tune_compare_clf() implements GridSearchCV() for parameter tuning and displays the Accuracy, F1 Score, Precision and Recall of all classifiers. Root Mean Square Error was also found out to find the residual error of each classifier.



```python
#Implementing GdidSearch for parameter tuning
def tune_compare_clf(X_train_std, y_train, classifiers, parameters, classifiers_names):

    '''The function receive Data (X, y), a classifiers list,
    a list of parameters to tune each chassifier (each one is a dictionary),
    and a list with classifiers name.

    The function also returns a Dataframe with predictions, each row is a classifier prediction.
    '''



    results = []
    results_train=[]

    for clf, par, name in zip(classifiers, parameters, classifiers_names):
        # Store results in results list
        clf_tuned = GridSearchCV(clf, par, cv=3).fit(X_train_std, y_train)

        y_pred = clf_tuned.predict(X_test_std)
        x_pred=  clf_tuned.predict(X_train_std)
        results.append(y_pred)
        results_train.append(x_pred)
        print(clf_tuned.best_params_)
        print('''best score = {:.2f}'''.format(clf_tuned.best_score_))
        print(name,"Accuracy: {0:.2%}".format(accuracy_score(y_test, y_pred)))
        print (name,"F1 Score training: %.2f%%" % (f1_score(y_train, x_pred) * 100.0))
        print("rms training: {:.2f}".format((sqrt(mean_squared_error(y_train, x_pred)))))
        print (name,"F1 Score testing: %.2f%%" % (f1_score(y_test, y_pred) * 100.0))
        print("rms testing: {:.2f}".format((sqrt(mean_squared_error(y_test, y_pred)))))
        print(name, "Precision Score: {0:.2%}".format (precision_score(y_test, y_pred)))
        print(name, "Recall Score: {0:.2%}\n\n".format(recall_score(y_test, y_pred)))

    result = pd.DataFrame.from_records(results)

    return result
```

```
result = tune_compare_clf(X_train_std, y_train, classifiers, parameters, classifiers_names)
```

To find the best classifier, we performed some data manipulation techniques such as Feature selection, scaling, and resampling the data as discussed below

For feature selection, **selector(**X_train_std, y_train**, k)** function receive features and labels and a target number to select features (k) based on analysis of heatmap and returns a new dataset with k best features.

After feature selection, the dataset has been divided into testing and training set using function **train_test_split()** and Feature scaling has been done only on dataset training set without labels using function **MinMaxScaler().**

As we have seen from the graph predicted above in section 3.1, the number of Benign samples is 357 and Malignant is 212. Due to this imbalance in data, balancing is required which has been achieved by using methods **ADASYN and SMOTE**. Both the function takes the scaled data and returns the resampled dataset which is again sent for parameter tuned classifiers to find the F1 score, Accuracy, Precision and Recall.

## 4. Results

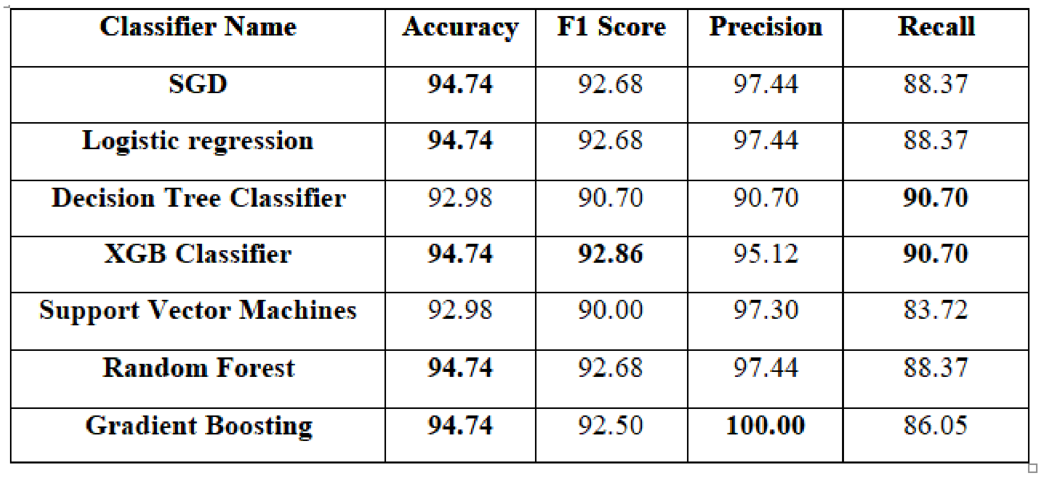
### 4.1. Model Evaluation and Validation

Before applying ML algorithms, it is important to decide how we need to evaluate the classifiers performance.To evaluate results we have used unseen data (20% of samples randomly chosen from original dataset) to compare results with true result (true label) and calculate precision, recall, F1 score and Accuracy.

- Accuracy, i.e. the fraction of correct predictions is typically not enough information to evaluate a model. Although it is a starting point, it can lead to invalid decisions. Models with high accuracy may have inadequate precision or recall scores. For this reason some other evaluation metrics were also assessed.
- Precision or the ability of the classifier not to label as positive a sample that is negative. The best value is 1 and the worst value is 0. In our study case, precision is when the algorithm guesses that a cell is malignant and actually measures how certain we are that this cell is a true malignant. For example, a precision of 0.9 means that if the model predicts 100 malignant cells, the 90 of them are malignant and the rest 10 are benign (false).
- Recall or the ability of the classifier to find all the positive samples. The best value is 1 and the worst value is 0. In context to the study, recall shows how well our identifier can find the malignant cells. For example, a low recall score of 0.8 indicates that our identifier finds only 80% of all the real malignant cells in the prediction. The rest 20% of real malignant cells will not be found by the diagnosis based on this algorithm, something that is unacceptable.
- F1 score, a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal. The formula for the F1 score is: F1 = 2 x (precision x recall) / (precision + recall).

Thus, in overall what we really want to know is how well this model generalizes to unseen data. To investigate this, their performance classifying the other 20% of the dataset which was held out during training was measured and the performance is reported below. The accuracy, F1 Score , precision and Recall is measured on the test dataset.

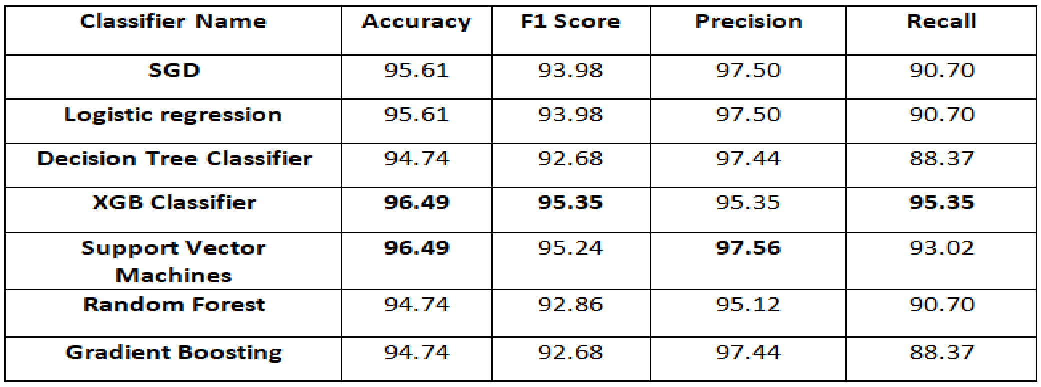
**Results after using feature selection and scaling and without tuning the parameters of the classifier**



| Classifier Name | Accuracy | F1 Score | Precision | Recall |
|---|---|---|---|---|
| SGD | **94.74** | 92.68 | 97.44 | 88.37 |
| Logistic regression | **94.74** | 92.68 | 97.44 | 88.37 |
| Decision Tree Classifier | 92.98 | 90.70 | 90.70 | **90.70** |
| XGB Classifier | **94.74** | **92.86** | 95.12 | **90.70** |
| Support Vector Machines | 92.98 | 90.00 | 97.30 | 83.72 |
| Random Forest | **94.74** | 92.68 | 97.44 | 88.37 |
| Gradient Boosting | **94.74** | 92.50 | **100.00** | 86.05 |

From the above table it can be observed that XGB classifier has performed well than other classifiers with its F1 score as 92.86 %.

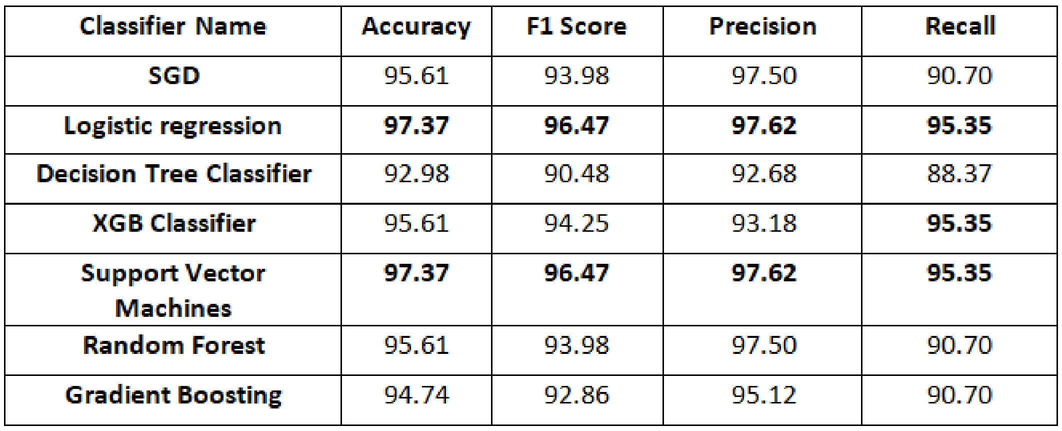
**Results after using feature selection, scaling and tuning the parameters of the classifier**



| Classifier Name | Accuracy | F1 Score | Precision | Recall |
|---|---|---|---|---|
| SGD | 95.61 | 93.98 | 97.50 | 90.70 |
| Logistic regression | 95.61 | 93.98 | 97.50 | 90.70 |
| Decision Tree Classifier | 94.74 | 92.68 | 97.44 | 88.37 |
| XGB Classifier | **96.49** | **95.35** | 95.35 | **95.35** |
| Support Vector Machines | **96.49** | 95.24 | **97.56** | 93.02 |
| Random Forest | 94.74 | 92.86 | 95.12 | 90.70 |
| Gradient Boosting | 94.74 | 92.68 | 97.44 | 88.37 |

After Tuning the parameter the performance of XGB has increased. It’s F1 score raised to 95.35

**Result after using feature selection, Scaling, Resampling using SMOTE and then implementing tune_compare_clf() for Parameter Tuning**



| Classifier Name | Accuracy | F1 Score | Precision | Recall |
|---|---|---|---|---|
| SGD | 95.61 | 93.98 | 97.50 | 90.70 |
| Logistic regression | **97.37** | **96.47** | **97.62** | **95.35** |
| Decision Tree Classifier | 92.98 | 90.48 | 92.68 | 88.37 |
| XGB Classifier | 95.61 | 94.25 | 93.18 | **95.35** |
| Support Vector Machines | **97.37** | **96.47** | **97.62** | **95.35** |
| Random Forest | 95.61 | 93.98 | 97.50 | 90.70 |
| Gradient Boosting | 94.74 | 92.86 | 95.12 | 90.70 |

In the above table we can observe that Logistic regression and Support Vector Machines have both performed well having the F1 Score as 96.47% and Decision Tree Classifier has worst performance with F1 Score as 90.48%.

**Result after using feature selection, Scaling, Resampling using ADASYN and then implementing tune_compare_clf() for Parameter Tuning**

| Classifier Name | Accuracy | F1 Score | Precision | Recall |
|---|---|---|---|---|
| **SGD** | 95.61 | 93.98 | 97.50 | 90.70 |
| **Logistic regression** | **99.12** | **98.85** | **97.73** | **100** |
| **Decision Tree Classifier** | 92.11 | 89.66 | 88.64 | 90.70 |
| **XGB Classifier** | 95.61 | 94.12 | 95.24 | 93.02 |
| **Support Vector Machines** | 96.49 | 95.45 | 93.33 | 97.67 |
| **Random Forest** | 95.61 | 94.12 | 95.24 | 93.02 |
| **Gradient Boosting** | 94.74 | 92.86 | 95.12 | 90.70 |

In the above table we can observe that Logistic regression has highest F1 Score as 98.85 with 97.73% precision and 100% Recall.

To evaluate results, we found out the comparison between classifiers on the basis of F1 score using training set and testing set. As discussed earlier unseen data (20% of samples randomly chosen from original dataset) was kept for testing i.e. the data was divided into 80:20 ratio to compare results with true result (true label). Below we can see the F1 score of each classifier on training and testing set

F1 Score for training set using data manipulation techniques

| Classifier Name | Feature Selection +Scaling+Parameter Tuning | Feature Selection +Scaling+SMOTE+Parameter Tuning | Feature Selection +Scaling+ADASYN+Parameter Tuning |
|---|---|---|---|
| SGD | 91.84 | 92.28 | 82.75 |
| Logistic regression | 92.45 | 95.96 | 93.84 |
| Decision Tree Classifier | 96.70 | 97.88 | **100** |
| XGB Classifier | **100** | **100** | **100** |
| Support Vector Machines | 95.21 | 96.50 | 94.65 |
| Random Forest | 99.40 | **100** | **100** |
| Gradient Boosting | 95.09 | 93.30 | **100** |

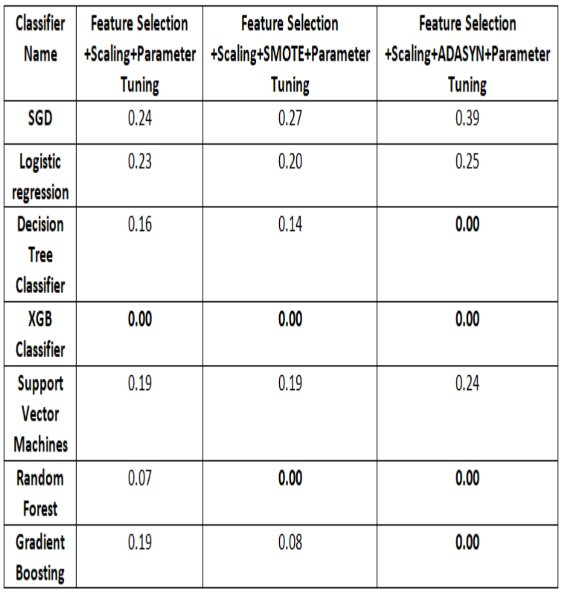
F1 Score for testing set using data manipulation techniques

| Classifier Name | Feature Selection +Scaling+Parameter Tuning | Feature Selection +Scaling+SMOTE+Parameter Tuning | Feature Selection +Scaling+ADASYN+Parameter Tuning |
|---|---|---|---|
| SGD | 93.98 | 93.98 | 93.98 |
| Logistic regression | 92.68 | **96.47** | **98.85** |
| Decision Tree Classifier | 92.68 | 90.48 | 89.66 |
| XGB Classifier | **95.35** | 94.25 | 94.12 |
| Support Vector Machines | 95.24 | **96.47** | 95.45 |
| Random Forest | 92.86 | 93.98 | 94.12 |
| Gradient Boosting | 92.68 | 92.86 | 92.86 |

Thus, from the above table we can conclude that using different manipulation techniques, even though DTC, XGB, RF, GB Classifier performed with 100% F1 Score on training set. But their performance decreased on unseen data and Logistic Regression outperformed among all the classifiers with 98.85 as the F1 score.

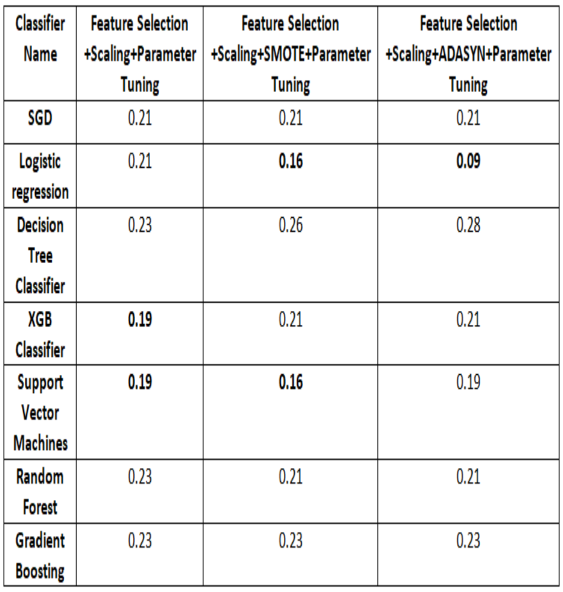
We also calculated Root Mean Square Error (RMSE) on both training and testing dataset to find out the residual error. Training RMSE is calculated on the training dataset, testing RMSE on the test dataset. The test dataset RMSE gives us an idea of how well the method will perform on new data.

RMSE for training set using data manipulation techniques



| Classifier Name | Feature Selection +Scaling+Parameter Tuning | Feature Selection +Scaling+SMOTE+Parameter Tuning | Feature Selection +Scaling+ADASYN+Parameter Tuning |
|---|---|---|---|
| SGD | 0.24 | 0.27 | 0.39 |
| Logistic regression | 0.23 | 0.20 | 0.25 |
| Decision Tree Classifier | 0.16 | 0.14 | **0.00** |
| XGB Classifier | **0.00** | **0.00** | **0.00** |
| Support Vector Machines | 0.19 | 0.19 | 0.24 |
| Random Forest | 0.07 | **0.00** | **0.00** |
| Gradient Boosting | 0.19 | 0.08 | **0.00** |

RMSE for testing set using data manipulation techniques



| Classifier Name | Feature Selection +Scaling+Parameter Tuning | Feature Selection +Scaling+SMOTE+Parameter Tuning | Feature Selection +Scaling+ADASYN+Parameter Tuning |
|---|---|---|---|
| SGD | 0.21 | 0.21 | 0.21 |
| Logistic regression | 0.21 | **0.16** | **0.09** |
| Decision Tree Classifier | 0.23 | 0.26 | 0.28 |
| XGB Classifier | **0.19** | 0.21 | 0.21 |
| Support Vector Machines | **0.19** | **0.16** | 0.19 |
| Random Forest | 0.23 | 0.21 | 0.21 |
| Gradient Boosting | 0.23 | 0.23 | 0.23 |

The smaller the RMSE value, the better the performance of classifier. So we observe that Logistic regression has the lower RMSE value as 0.09 on testing data set.

Even though RMSE is sensitive to noise and we have not performed any technique on outlier removal but still our model is Robust because we have implemented Cross-validation technique while training the data. So the model can be trusted which yields such a small residual error.

Thus we can arrive at the judgement that the model generalizes well on unseen data. A combination of Feature selection, Scaling, Resampling using ADASYN and then Parameter tuning was the best approach. It gave us the highest score for the classifier and more reliable result.

## 4.2. Justification

Logistic Regression is used as the benchmark model [3] for this project. The bench mark model is evaluated using Accuracy. The accuracy of logistic regression on training data set is 96.79% and using testing data, accuracy is 92.5%. The classification accuracy is predicted in terms of Sensitivity and Specificity. And in the reported paper [3] the highest accuracy of

Classification model is SVM having 97.59% using training data and 94.5%using testing data. Our implementation of SVM resulted in an Accuracy of 96.47% using test data set which is more than the benchmark model.

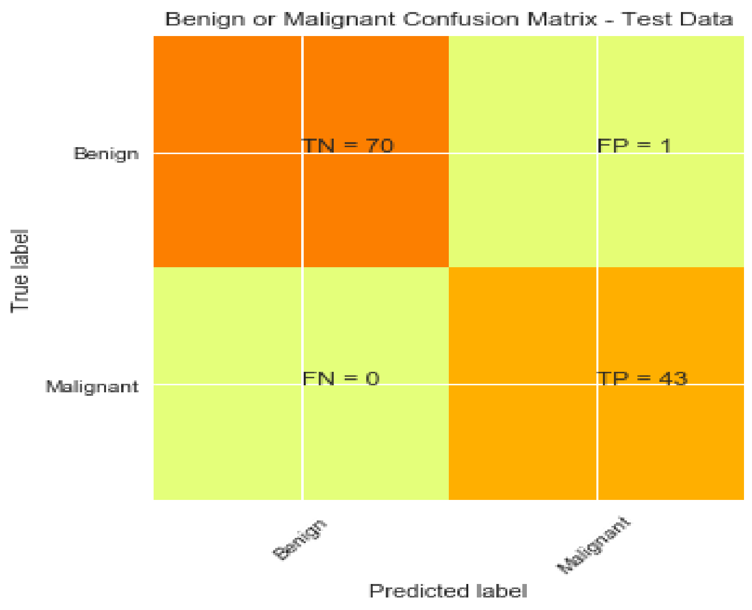
Depending on the task at hand, accuracy may or may not be a good metric by which to judge the ML model. Models with high accuracy may have inadequate precision or recall scores. So we opted for F1 score, which has been predicted in terms of Precision and Recall.

To realize how much effective our method is, after optimizing our model we compared the obtained results with results extracted from the benchmark model. Using the data manipulation techniques, the F1 Score of Logistic Regression is 98.58% on test data set. Thus, our results are significantly more accurate that our benchmark model and it proves the power of modern machine learning algorithms against similar methods used before.

## 5. Conclusion

### 5.1. Free- form visualization

In our final result, after using feature selection, scaling, re-sampling and then parameter tuning, Logistic Regression has the highest F1 score on testing data set which is 98.58%. Logistic regression model predicted wrongly only "one" case as we can see in the confusion matrix below:



- 70 data points was correctly predicted as Benign tumors
- 43 data points was correctly predicted as Malignant tumors
- 01 data point was wrongly predicted as Malignant tumor
- 00 data point was not wrongly predicted as Benign tumor

In disease classification is very important have low false negative occurrences because if a patience have a disease and a model predict that they doesn't, the patient will not receive treatment and this may worsen the clinical picture.

## 5.2 Reflection

The Wisconsin Breast Cancer dataset has 30 features and is difficult to realize what is and what isn't important to correctly classify a tumor cell. Choosing an algorithm to get optimal result is also a difficult task. One challenge encountered during implementation was to manually create the parameter grid which was to be exhaustively searched to deliver the best tuned model. With a number of potential hyper-parameters available to fine-tune, it was not exactly intuitive which ones to focus on, and subsequently, which values to include. Furthermore, the GridSearcchCV module is rather computationally intensive for large parameter grid, however, after some experimentation, a useful and succinct parameter grid was eventually settled upon.

Many data manipulation techniques were implemented such as feature selection, scaling, re-sampling. There was class imbalance which needs to be taken care of. Thus oversampling was performed to handle them. Precision and Recall was found out for all classifier using different manipulation techniques. Again to check whether the model generalizes on unseen data well we also found out the RMSE.

## 5.2. Improvement

The approach I adopted in this project was to try with different algorithm, parameters, methods and combinations to find the best configuration.

In disease classification, it is very important to have low false negative occurrences because if a patient  have a disease and a model predict that they doesn't, the patient will not receive treatment and this may worsen the clinical picture. Thus, Precision and Recall value can be taken care of by giving emphasis on Feature engineering, different cross validation values, combining new data manipulation techniques, more parameter tuning of algorithms, outliers removal. But this can increase exponentially the number of train sets which may cause time consumption issues. And also for better result more data are required and other algorithms can also be used.

## 6. References

1. https://www.insideradiology.com.au/breast-fna/

2. https://www.breastcancer.org/

3. G. Ravi Kumar, Dr. G. A. Ramachandra, K.Nagamani, "An Efficient Prediction of Breast Cancer Data", International Journal of Innovations in Engineering and Technology (IJIET), Vol. 2 Issue 4 August 2013, p-139-144.

4. https://medium.com/@kathrynklarich/exploring-and-evaluating-ml-algorithms-with-the-wisconsin-breast-cancer-dataset-506194ed5a6a

**5.** https://github.com/CesarTrevisan/Breast-Cancer-Detection