Machine Learning Engineer Nanodegree

Capstone Project

Narreddy Aravind Reddy
January 24th, 2019

Breast Cancer Wisconsin (Diagnostic) Classification

I. Definition

Project Overview

In these days the Domain of Health Care is stepping to get engaged with Artificial Intelligence and Machine Learning. This project is based on the Domain of Health Care. This gives accurate results regarding the classification of Breast Cancer. It takes the features of cells in the body and predicts that the woman is affected with Breast Cancer of not. It's a good initiation to implement this project in Health Care Centers. The dataset is collected from UCI Machine Learning Respertories https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29and this project is comes under research.

The aim of this application is to avoid the *Human Errors* in the Domain of **Health Care**.

Problem Statement

The goal is to classify a new data point to either Malignant or Benign on basis of the features of the given data point. The tasks involved in it :

-_1.Download the data from https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29
-_2.Cleaning the data and removing the Null data points, duplicated data rows.
-_3.Visualising the data to know the characteristics of the features.
-_4.Test which classifier performs good on the data set using appropriate methods.
-_5.Test which parameters performs good on the classifier using appropriate methods.
-_6.Training the data with classifier.

-_7.Getting best Accuracy Scores.

The final application is regarding to classify either patient is affected with Breast Cancer or not, using the features of body cells.

Metrics

Accuracy Score= TP+TN/TP+FP+FN+TN(simply)

Reasons to use Accuracy Score:

$$Accuracy = \frac{Number\ of\ Correct\ predictions}{Total\ number\ of\ predictions\ made}$$

When the model is tested on a test set with 60% samples of class A and 40% samples of class B, then the ***test accuracy would drop down to 60%.****Classification Accuracy is great, but gives us the false sense of achieving high accuracy.*

The real problem arises, when the cost of misclassification of the minor class samples are very high. If we deal with a rare but fatal disease, the cost of failing to diagnose the disease of a sick person is much higher than the cost of sending a healthy person to more tests.

F1 Sore= 2*(Recall* Precision) / (Recall + Precision)

Reasons to use F1 Score:

F1 Score is the Harmonic Mean between precision and recall. The range for F1 Score is [0, 1]. It tells you how precise your classifier is (how many instances it classifies correctly), as well as how robust it is (it does not miss a significant number of instances).

High precision but lower recall, gives you an extremely accurate, but it then misses a large number of instances that are difficult to classify. The greater the F1 Score, the better is the performance of our model. Mathematically, it can be expressed as :

$$F1 = 2 * \frac{1}{\frac{1}{precision} + \frac{1}{recall}}$$

F1 Score

F1 Score tries to find the balance between precision and recall.

Precision = TP/TP+FP

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives}$$

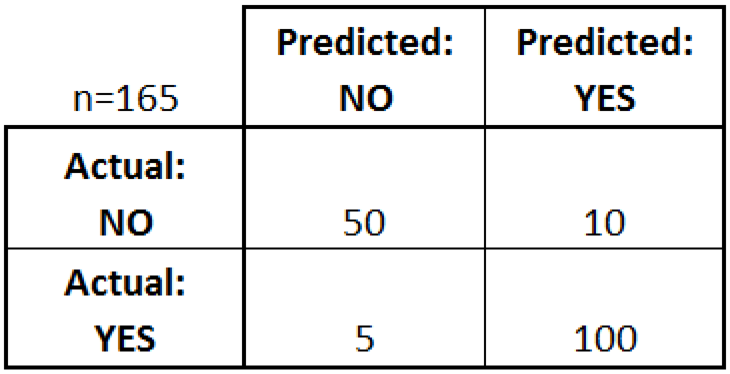
Recall = TP/TP+FN

$$Precision = \frac{TruePositives}{TruePositives + FalseNegatives}$$

Confusion Matrix = A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known. The confusion matrix itself is relatively simple to understand, but the related terminology can be confusing.

<u>Reasons to use Confusion Matrix:</u>

Lets assume we have a binary classification problem. We have some samples belonging to two classes : YES or NO. Also, we have our own classifier which predicts a class for a given input sample. On testing our model on 165 samples ,we get the following result.



n=165	**Predicted: NO**	**Predicted: YES**
Actual: NO	50	10
Actual: YES	5	100

True positives (TP): These are cases in which we predicted yes (they have the disease), and they do have the disease.

True negatives (TN): We predicted no, and they don't have the disease.

False positives (FP): We predicted yes, but they don't actually have the disease. (Also known as a "Type I error.")

False negatives (FN): We predicted no, but they actually do have the disease. (Also known as a "Type II error.")

referencelink:https://blog.exsilio.com/all/accuracy-precision-recall-f1-score-interpretation-of-performance-measures/

II. Analysis

Data Exploration

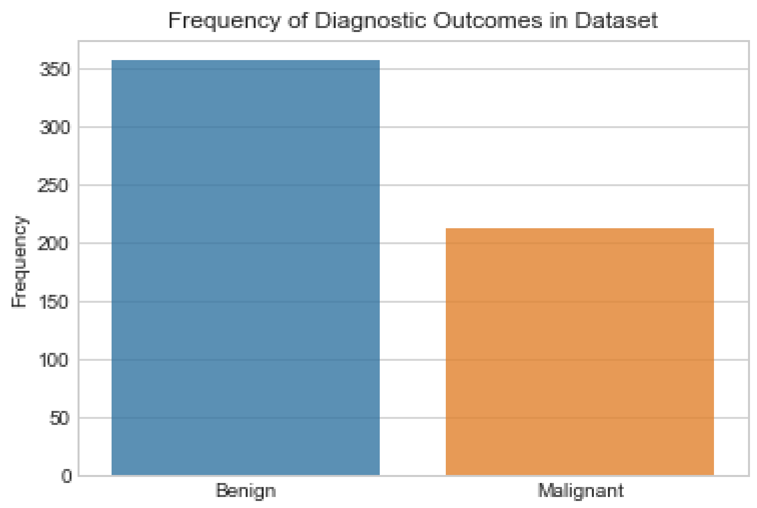
The toal number of instances contained in data is 569. The number of attributes in the data are 33 on total. The information of every attribute is given below Attribute Information:
1) ID number
2) Diagnosis (M = malignant, B = benign) 3-32)(**target value**)
Ten real-valued features are computed for 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)
From the data the columns of 'id','unnamed: 32' are removed as a part of Data Cleaning.

Statistical Information(as per features):

```
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)
```

The out feature statistics:



Null points and Duplicated instances:

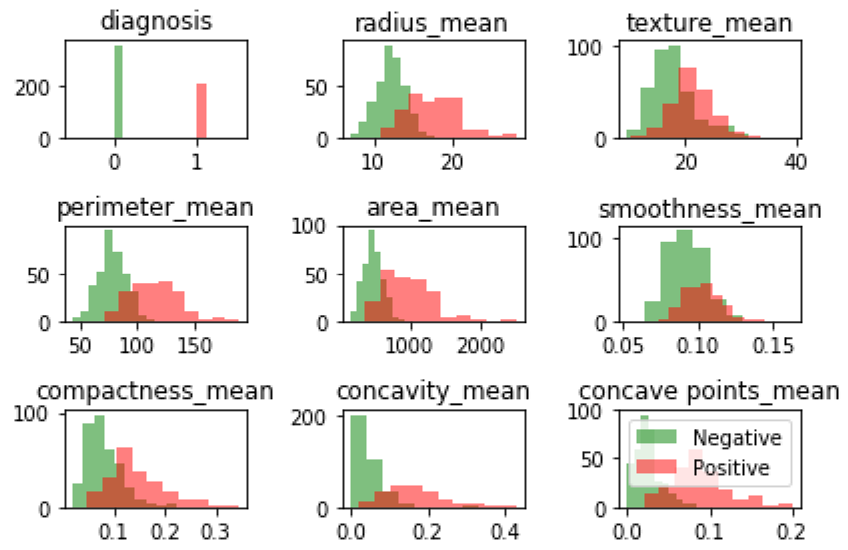
The data has no Null points and no instances are duplicated.

Exploratory Visualization

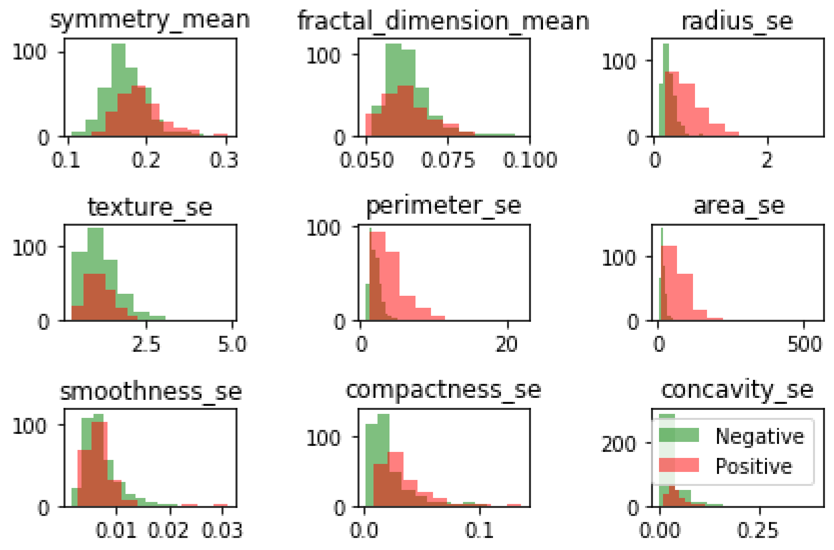
The green part of histogram represents the features with **diagnosis** = = **benign** and red part represents **diagnosis** = = **malignant**

The histograms of all the features of Data are given below:

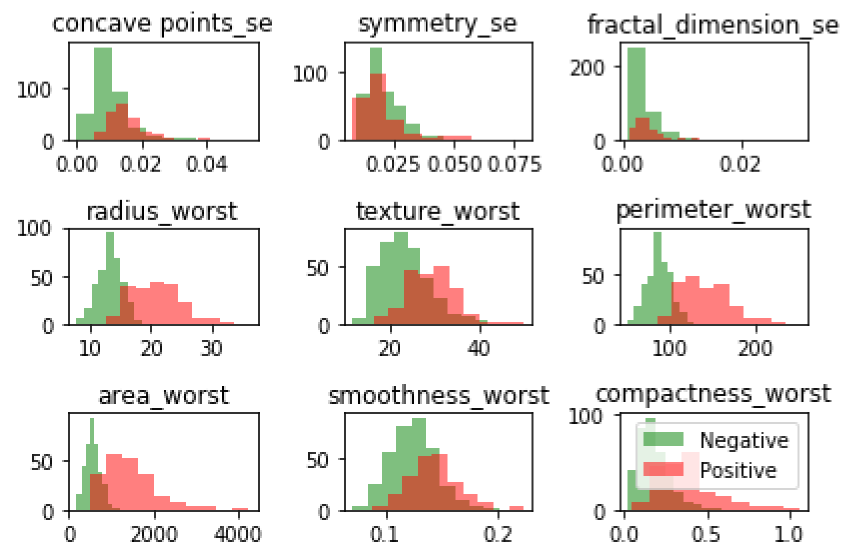
The columns of [0:9]



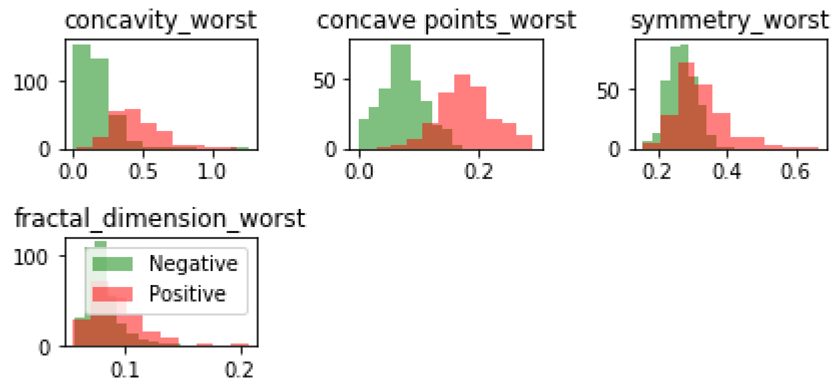
The columns of [9:18]



The columns of [18:27]



The columns of [27:31]

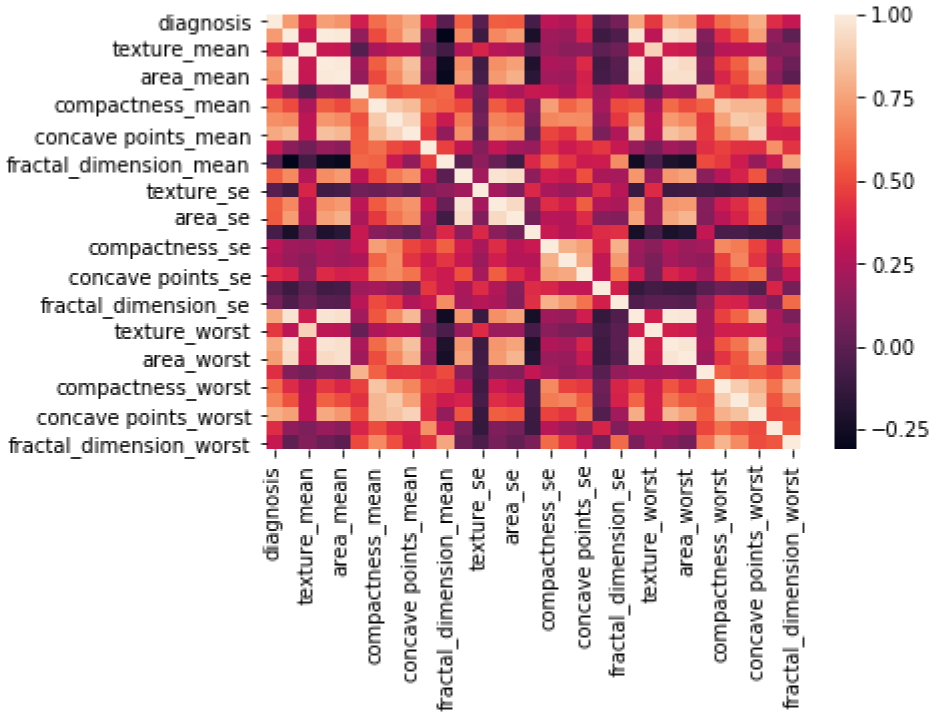


Histogram:

A histogram is a display of statistical information that uses rectangles to show the frequency of data items in successive numerical intervals of equal size. In the most common form of histogram, the independent variable is plotted along the horizontal axis and the dependent variable is plotted along the vertical axis. The data appears as colored or shaded rectangles of variable area.

By representing the features with Histograms we can know how the data is structured and if any preprocessing is required.

The below diagram represents (Heat Map) the correlation of features of the data.



Heat Map:

A heat map is a two-dimensional representation of data in which values are represented by colors. A simple heat map provides an immediate visual summary of information. More elaborate heat maps allow the viewer to understand complex data sets.¶

Algorithms and Techniques

Decision Trees: Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression.

Some advantages of decision trees are:

Simple to understand and to interpret. Trees can be visualized. Requires little data preparation. Other techniques often require data normalization, dummy variables need to be created and blank values to be removed. Note however that this module does not support missing values. The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree. Able to handle both numerical and categorical data. Other techniques are usually specialized in analyzing datasets that have only one type of variable. See algorithms for more information. Able to handle multi-output problems. Uses a white box model. If a given situation is observable in a model, the explanation for the condition is easily explained by Boolean logic. By contrast, in a black box model (e.g., in an artificial neural network), results may be more difficult to interpret. Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model. Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.

The disadvantages of decision trees include:

Decision-tree learners can create over-complex trees that do not generalize the data well. This is called over fitting. Mechanisms such as pruning (not currently supported), setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem. Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble. The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts. Consequently, practical decision-tree learning algorithms are based on heuristic algorithms such as the greedy algorithm where locally optimal decisions are made at each node. Such algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees in an ensemble learner, where the features and samples are randomly sampled with replacement. There are concepts that are hard to learn because decision trees do not express them easily, such as XOR, parity or multiplexer problems. Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the dataset prior to fitting with the decision tree.

Parameters:

class sklearn.tree.DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort=False

Random Forest:

Tree models are known to be high variance, low bias models. In consequence, they are prone to overfit the training data. This is catchy if we recapitulate what a tree model does if we do not prune it or introduce early stopping criteria like a minimum number of instances per leaf node. Well, it tries to split the data along the features until the instances are pure regarding the value of the target feature, there are no data left, or there are no features left to spit the dataset on. If one of the above holds true, we grow a leaf node. The consequence is that the tree model is grown to the maximal depth and therewith tries to reshape the training data as precise as possible which can easily lead to over fitting. Another drawback of classical tree models like the (ID3 or CART) is that they are relatively unstable. This instability can lead to the situation that a small change in the composition of the dataset leads to a completely different tree model.

Parameters:

Class sklearn.ensemble.RandomForestClassifier (n_estimators='warn', criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None

Advantages:

1.Reduction in over fitting: by averaging several trees, there is a significantly lower risk of over fitting.
2.Less variance: By using multiple trees, you reduce the chance of stumbling across a classifier that doesn't perform well because of the relationship between the train and test data.

Disadvantages:

1.It takes more time to train samples.

Logistic Regression:

Logistic Regression is a Machine Learning classification algorithm that is used to predict the probability of a categorical dependent variable. In logistic regression, the dependent variable is a binary variable that contains data coded as 1 (yes, success, etc.) or 0 (no, failure, etc.). In other words, the logistic regression model predicts P(Y=1) as a function of X.

Advantages :

Because of its efficient and straightforward nature, doesn't require high computation power, easy to implement, easily interpretable, used widely by data analyst and scientist. Also, it doesn't require scaling of features. Logistic regression provides a probability score for observations.

Disadvantages:

Logistic regression is not able to handle a large number of categorical features/variables. It is vulnerable to over fitting. Also, can't solve the non-linear problem with the logistic regression that is why it requires a transformation of non-linear features. Logistic regression will not perform well with independent variables that are not correlated to the target variable and are very similar or correlated to each other.

Parameters:

class sklearn. linear_model.LogisticRegression(penalty='l2', dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='warn', max_iter=100, multi_class='warn', verbose=0, warm_start=False, n_jobs=None)

The application is classification oriented . So, techniques that are used are taken from Classification techinques.

Benchmark

```
The Percentage of tumors classified as 'malignant' in this data
set is: 37.25834797891037.

 A good classifier should therefore outperform blind guessing
knowing the proportions i.e. > 62% accuracy.
```

To measure the performance of our predictions, we need a metric to score our predictions against the true outcomes. To determine the benchmark model the total percentage of tumors classified as 'miligant' in this data set is taken and based on that 'miligant' percentage the actual benchmark model is decided.

1. The Benchmark value is an accuracy score of 62%.

2. Because by having blind guessing we will get an accuracy score of 62%. So, if the application is able to give an accuracy score.

greater than 62% then the application is performing good.

III. Methodology

Data Preprocessing

In the step of Data Preprocessing Normalizer() is used to normalize the feature values of the data.
1. The whole data is divided into training and testing data using train_test_split from sklearn.model_selection
2. And then the training and testing data of feature values are preprocessed using sklearn.preprocessing.Normalizer ()
3. The normalization makes the curve of data look like Normal distribution curve.

Normalization: Normalization makes training less sensitive to the scale of features, so we can better solve for coefficients.

Implementation

The implementation process can be split into two main stages:
1. The classifier training stage
2. Tuning the parameters of best defined classifier

The Classifier training stage: The best Classifier for the data is got from applying Cross Validation on

1. Decision Trees

2. Random Forests

3. Logistic Regression

1. Among these the best classifier is brought by applying cross validation function.

The following shows the Cross validation values got for the taken testing Classifiers:

Cross Validation Accuracy DTC(Decision Tree Classifier): Accarcy: 0.922627 SD: 0.021320
Cross Validation Accuracy RFC(Random Forest Classifier): Accarcy: 0.943639 SD: 0.017351
Cross Validation Accuracy LR(Logistic Regression): Accarcy: 0.913187 SD: 0.018878

Among the three reports the Random Forest Classifier looks performing. So, it is the best Classifier for this data.

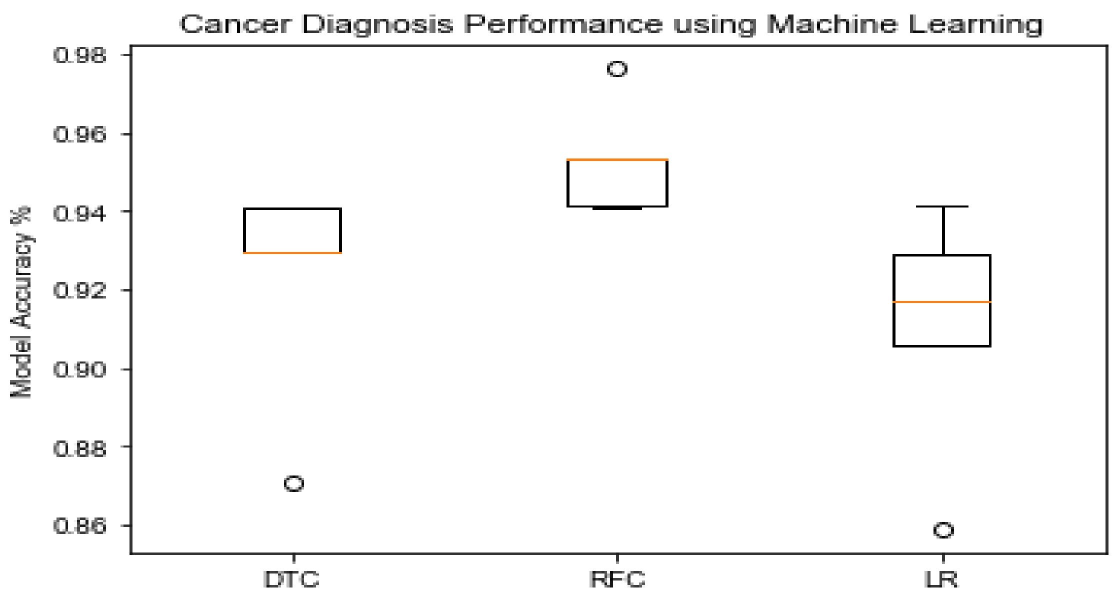
Tuning the parameters of best defined classifier : The parameter of the best defined classifier are tuned using appropriate methods to get best accuracy score.¶

1.After getting the necessary parameters of the best defined classifier the training data is fitted to the Classifier.
2.Now the scores like accuracy score, f1 score etc are obtained from the data using the Classifier.

Refinement:

1. In the previous section, in Defing best classifier a cross validation function is applied. The box representation is given below.

With this Box plot representation, the RFC (Random Forest Classifier) gives better score.



1. After getting the best classifier from cross validation. The accuracy score obtained when no tuning is applied on the classifier is

```
from sklearn.metrics import make_scorer, accuracy_score, fbeta_score

clf=RandomForestClassifier(n_estimators=10, random_state=42) clf.fit(X_train_normal,y_train)
pred=clf.predict(X_test_normal) accuracy_score(pred,y_test)
```

The final scores after applying the GridSearchCV function on the Random Forest Classifier is

Optimized Model

Final accuracy score on the testing data: 0.9720

Final F-score on the testing data: 0.9728

YES, it's a great accuracy score of 97.2% after Optimization. This is a good Refinement.

IV. Results

Model Evaluation and Validation

```
from sklearn.metrics import make_scorer, accuracy_score, fbeta_score

clf = RandomForestClassifier(random_state=42
parameters = {'n_estimators': [10,50,100,150,200], 'criterion': ['gini', 'entropy']}
grid_obj = GridSearchCV(clf,parameters,scoring='accuracy')
grid_fit = grid_obj.fit(X_train_normal,y_train)
best_clf = grid_fit.bestestimator
predictions = (clf.fit(X_train_normal, y_train)).predict(X_test_normal) best_predictions =
best_clf.predict(X_test_normal)
print("Unoptimized model\n------") print("Accuracy score on testing data:
{:.4f}".format(accuracy_score(y_test, predictions))) print("F-score on testing data:
{:.4f}".format(fbeta_score(y_test, predictions, beta = 0.5))) print("\nOptimized Model\n------")
print("Final accuracy score on the testing data: {:.4f}".format(accuracy_score(y_test,
best_predictions))) print("Final F-score on the testing data: {:.4f}".format(fbeta_score(y_test,
best_predictions, beta = 0.5)))
```

The accuracy and f1 score obtained are:

Unoptimized model

Accuracy score on testing data: 0.9577
F-score on testing data: 0.9434

Optimized Model

Final accuracy score on the testing data: 0.9718
Final F-score on the testing data: 0.9623

The Confusion matrix of the model:

```
                    Predicted Negative    Predicted Positive
```

```
Actual Negative        87                    6
Actual Positive         2                   51
```

The Classification Report :

```
             precision    recall  f1-score   support


       0       0.99      0.97      0.98        92


       1       0.94      0.98      0.96        51
```

```
avg / total       0.97       0.97       0.97       143
```

Justification

1. The benchmark model have an accuracy score of 62%, the optimized model has obtained an accuracy score of 97.2%. So, this model is good performing.

2. And also the Classification Report shows good results of recall, f1-score, and support.

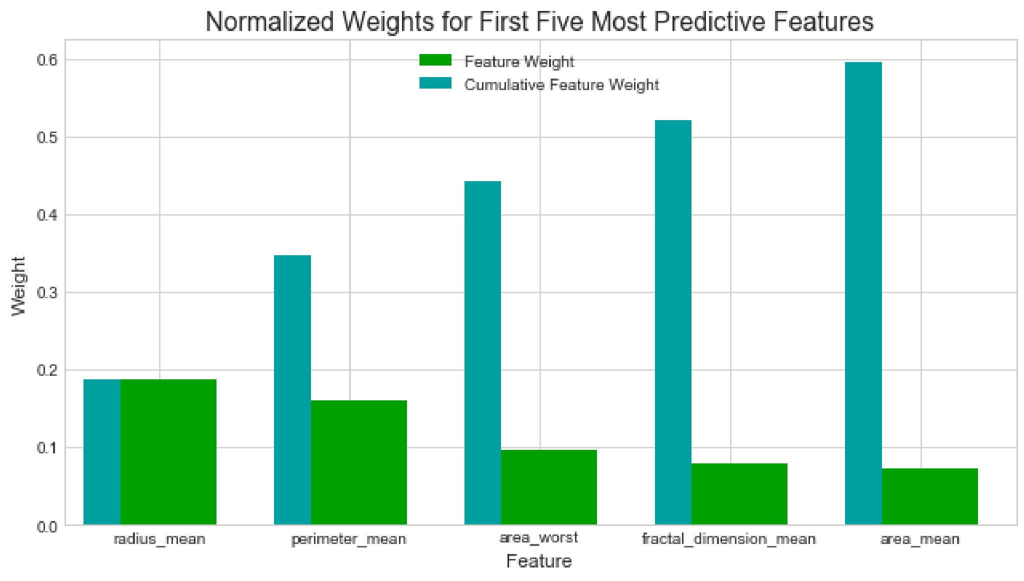
3. In Confusion matrix the number obtained between Actual Positive and Predicted Negative is very less, that means its better performing.

4 . There is a reasonable difference in between the unoptimized model and Optimized Model with a difference in accuracy score ad f1 score of approx. 3% and 3% respectively.

Conclusion

Free-Form Visualization:

After studying the data , the data features that most influences the target is given below:



From the above graph it is clear that radius mean, perimeter mean, primeter_worst, area_wrost, radius_wrost are First Five most Predictive Features.

Implementation - Extracting Feature Importance(Results)

Choose a `scikit-learn` supervised learning algorithm that has a `feature_importance_` attribute availble for it. This attribute is a function that ranks the importance of each feature when making predictions based on the chosen algorithm.

If the training time factor is a factor ,then the reduced dataset can be taken as a training set because the training time of reduced data is less eventhough the accuracy score and F score reduces.

```
Final Model trained on full data

 ------

Accuracy on testing data: 0.9718 F-score on testing data: 0.9623

Final Model trained on reduced data

 ------

 Accuracy on testing data: 0.9437 F-score on testing data: 0.9339
```

Reflection:

1. I have learnt how to visualize, and understand the data.
2.I have learnt that Data Cleaning plays crucial part in Data Analysis.
3. Removing the data features that are not necessary in evaluating an model is most important.
4.I got to know how to use best technique for the data using appropriate techniques.
5.I got to know how to tune the parameters of the technique using to get best score.
6.On total, I have learnt how to grab a data set and applying cleaning techniques on it and to fit to best techniques to get best scores.

Improvement:

This can be improved to classify an disease in Human Body by increasing the features and number of instances. The model is a part of my research in **Health Care**. This application can be taken to next level with the engagement of Internet Of Things and can be implemented in large scale in Health Care Centers which also reduces the *Human Errors* which is a Serious Problem in Health Care Domain since from past.