Politecnico di Torino

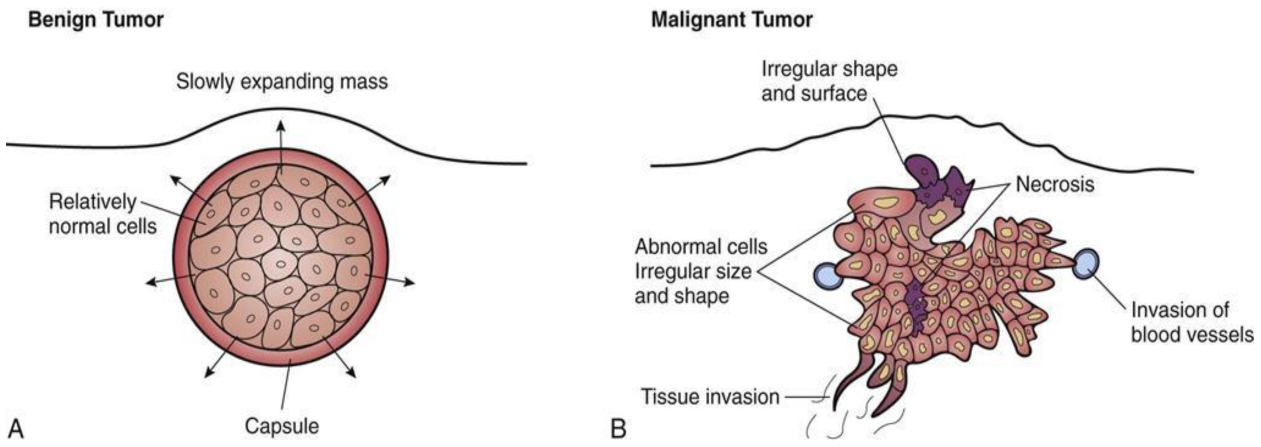
1859

Breast Cancer Wisconsin Diagnostic Dataset

Uditi Ojha -s270049

Introduction

Breast Cancer is currently the most common cancer among women in the world today. An early diagnosis of Breast Cancer can greatly improve the prognosis and chance of survival of the patients . Breast Cancer occurs as the result of abnormal growth of the cells in the breast tissue commonly referred to as tumors . A tumor does not mean cancer -tumors can be *benign (non cancerous)* , *pre-malignant (pre -cancerous)* or *malignant(cancerous)*. Tests such as MRI, ultrasound and biopsy are commonly used to diagnose breast cancer performed.
The code is available at https://github.com/Uditi131995/Breast-Cancer-Wisconsin



Identify Dataset

This is an analysis of the Breast Cancer Wisconsin (*Diagnostic*) DataSet, obtained from UCI. This data set was created by Dr. William H. Wolberg, physician at the University Of Wisconsin Hospital at Madison, Wisconsin,USA. To create the dataset Dr. Wolberg used fluid samples, taken from patients with solid breast masses and an easy-to-use graphical computer program called Xcyt, which is capable of performing the analysis of cytological features based on a digital scan. The program uses a curve-fitting algorithm, to compute ten features from each one of the cells in the sample, then it calculates the mean value, extreme value and standard error of each feature for the image, returning a 30 real-valued vector.

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)

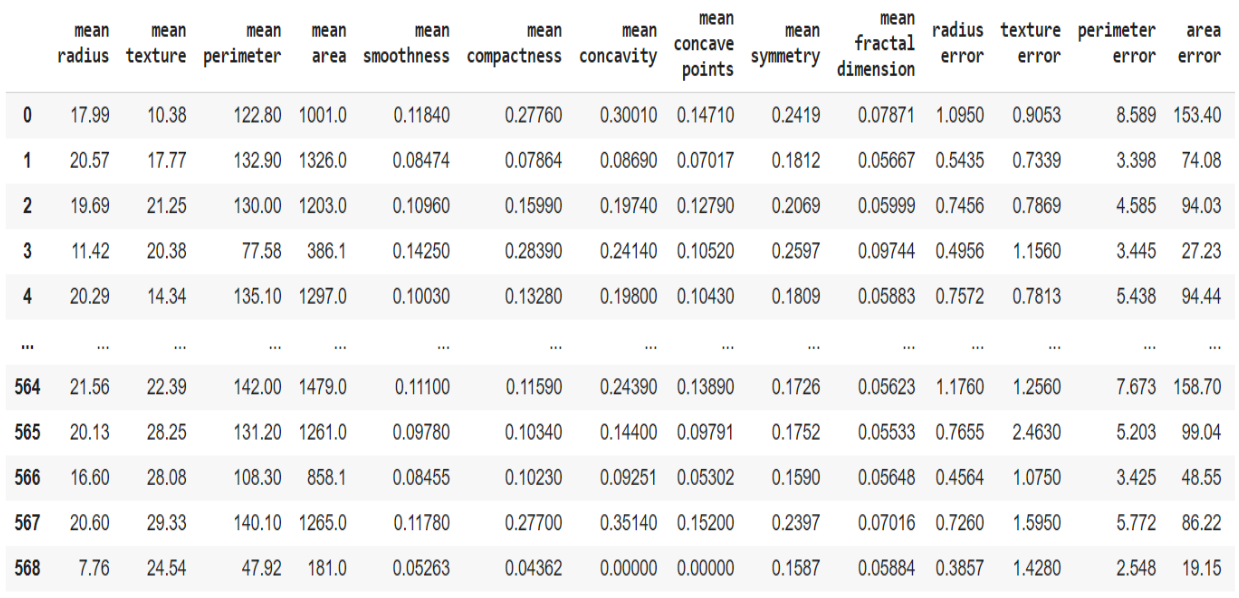
The **"mean"**, standard **"error"** and **"worst"** or largest (mean of the three largest values) of these features were **computed for each image, resulting in 30 features**.

The attribute information Diagnosis (M = Malignant , B=Benign) is categorical so mapped into 'target' attribute and the class into binary class , the correspondence of the class are summarized below:

Code	Attribute Equivalent
0	Malignant
1	Benign

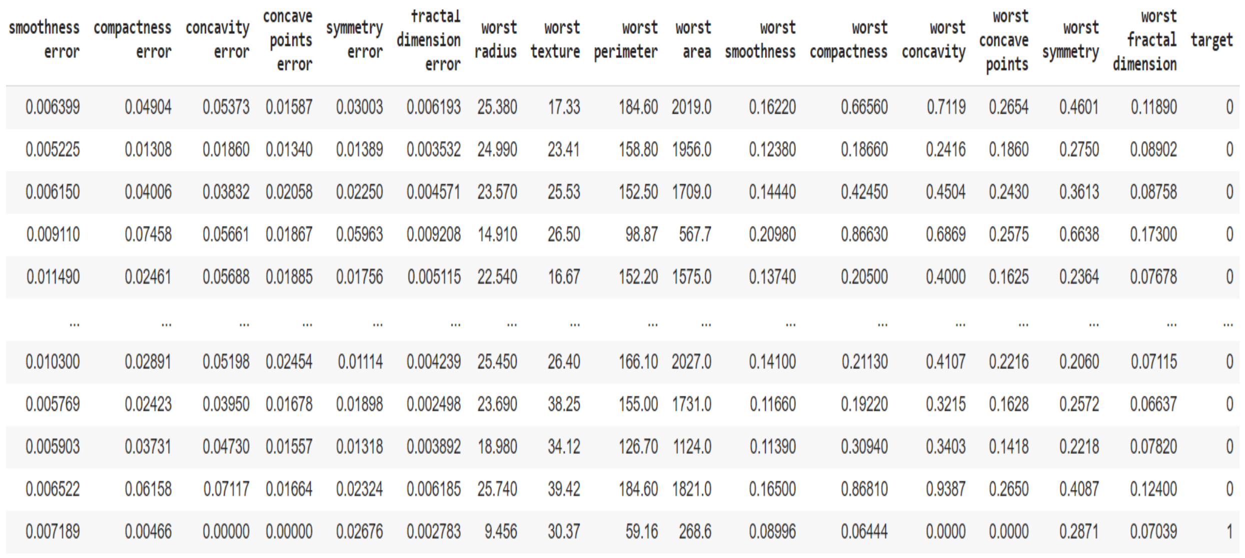
Descriptive Statistics

The first step is to visually inspect the new Dataset loaded directly from scikit learn.We see that there are a total of **569** instances out of which **212** are **Malignant** and **357** are **Benign**. The distribution of the data is **62.74 % Benign** and **37.25 % Malignant**. Although there are more cases which are negative and less positive, since the **data distribution** is near to **60%** and **40%** split it can be considered **not fully** an **unbalanced** dataset. Hence **no sampling** strategy is applied. Moreover we check for the presence of any missing values and there is **no missing** value in the dataset.



	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	radius error	texture error	perimeter error	area error
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871	1.0950	0.9053	8.589	153.40
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	0.5435	0.7339	3.398	74.08
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999	0.7456	0.7869	4.585	94.03
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744	0.4956	1.1560	3.445	27.23
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	0.7572	0.7813	5.438	94.44
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	1.1760	1.2560	7.673	158.70
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	0.7655	2.4630	5.203	99.04
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	0.4564	1.0750	3.425	48.55
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	0.7260	1.5950	5.772	86.22
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	0.3857	1.4280	2.548	19.15

Figure 1: Head of the Dataframe(Part 1)



smoothness error	compactness error	concavity error	concave points error	symmetry error	fractal dimension error	worst radius	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension	target
0.006399	0.04904	0.05373	0.01587	0.03003	0.006193	25.380	17.33	184.60	2019.0	0.16220	0.66560	0.7119	0.2654	0.4601	0.11890	0
0.005225	0.01308	0.01860	0.01340	0.01389	0.003532	24.990	23.41	158.80	1956.0	0.12380	0.18660	0.2416	0.1860	0.2750	0.08902	0
0.006150	0.04006	0.03832	0.02058	0.02250	0.004571	23.570	25.53	152.50	1709.0	0.14440	0.42450	0.4504	0.2430	0.3613	0.08758	0
0.009110	0.07458	0.05661	0.01867	0.05963	0.009208	14.910	26.50	98.87	567.7	0.20980	0.86630	0.6869	0.2575	0.6638	0.17300	0
0.011490	0.02461	0.05688	0.01885	0.01756	0.005115	22.540	16.67	152.20	1575.0	0.13740	0.20500	0.4000	0.1625	0.2364	0.07678	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
0.010300	0.02891	0.05198	0.02454	0.01114	0.004239	25.450	26.40	166.10	2027.0	0.14100	0.21130	0.4107	0.2216	0.2060	0.07115	0
0.005769	0.02423	0.03950	0.01678	0.01898	0.002498	23.690	38.25	155.00	1731.0	0.11660	0.19220	0.3215	0.1628	0.2572	0.06637	0
0.005903	0.03731	0.04730	0.01557	0.01318	0.003892	18.980	34.12	126.70	1124.0	0.11390	0.30940	0.3403	0.1418	0.2218	0.07820	0
0.006522	0.06158	0.07117	0.01664	0.02324	0.006185	25.740	39.42	184.60	1821.0	0.16500	0.86810	0.9387	0.2650	0.4087	0.12400	0
0.007189	0.00466	0.00000	0.00000	0.02676	0.002783	9.456	30.37	59.16	268.6	0.08996	0.06444	0.0000	0.0000	0.2871	0.07039	1

Figure 2: Head of the Dataframe(Part 2)

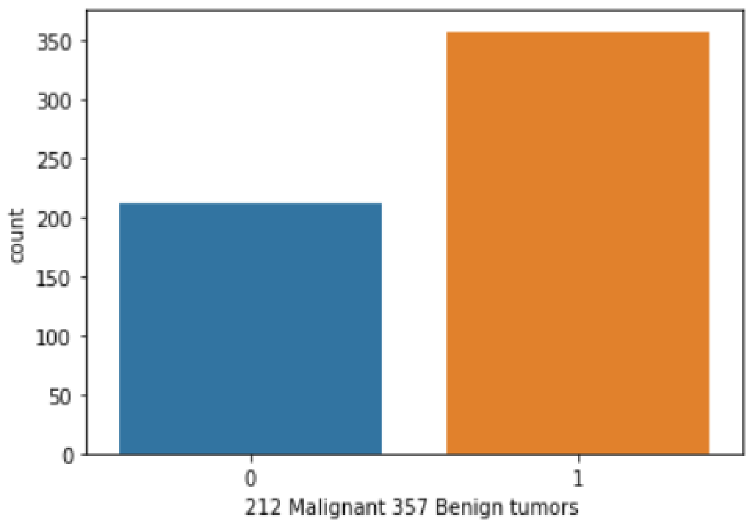
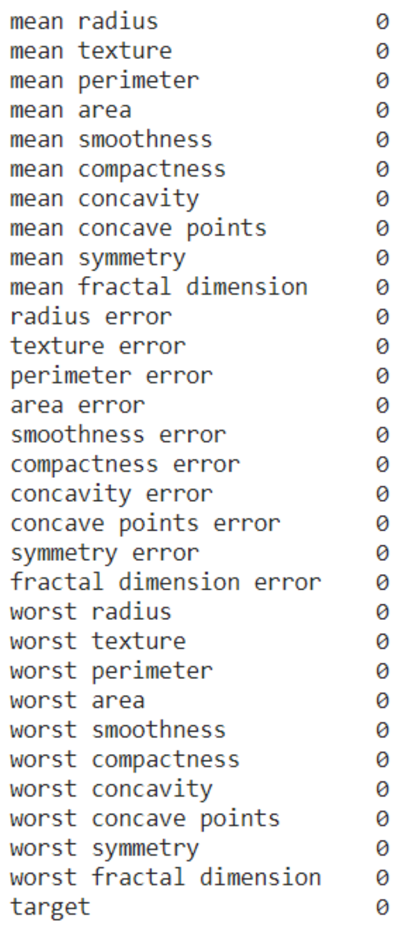


Figure : 3 Distribution of two classes



```
mean radius                0
mean texture               0
mean perimeter             0
mean area                  0
mean smoothness            0
mean compactness           0
mean concavity             0
mean concave points        0
mean symmetry              0
mean fractal dimension     0
radius error               0
texture error              0
perimeter error            0
area error                 0
smoothness error           0
compactness error          0
concavity error            0
concave points error       0
symmetry error             0
fractal dimension error    0
worst radius               0
worst texture              0
worst perimeter            0
worst area                 0
worst smoothness           0
worst compactness          0
worst concavity            0
worst concave points       0
worst symmetry             0
worst fractal dimension    0
target                     0
```

Figure 4 : Missing values from the dataset grouped by column

Data Visualization

There are a total of 30 features which we can visualize. I decided to plot 10 features at a time. This leads to 3 plots containing 10 features each. We begin by plotting the **Histogram** followed by the **Correlation Matrix** and **Boxplot**.

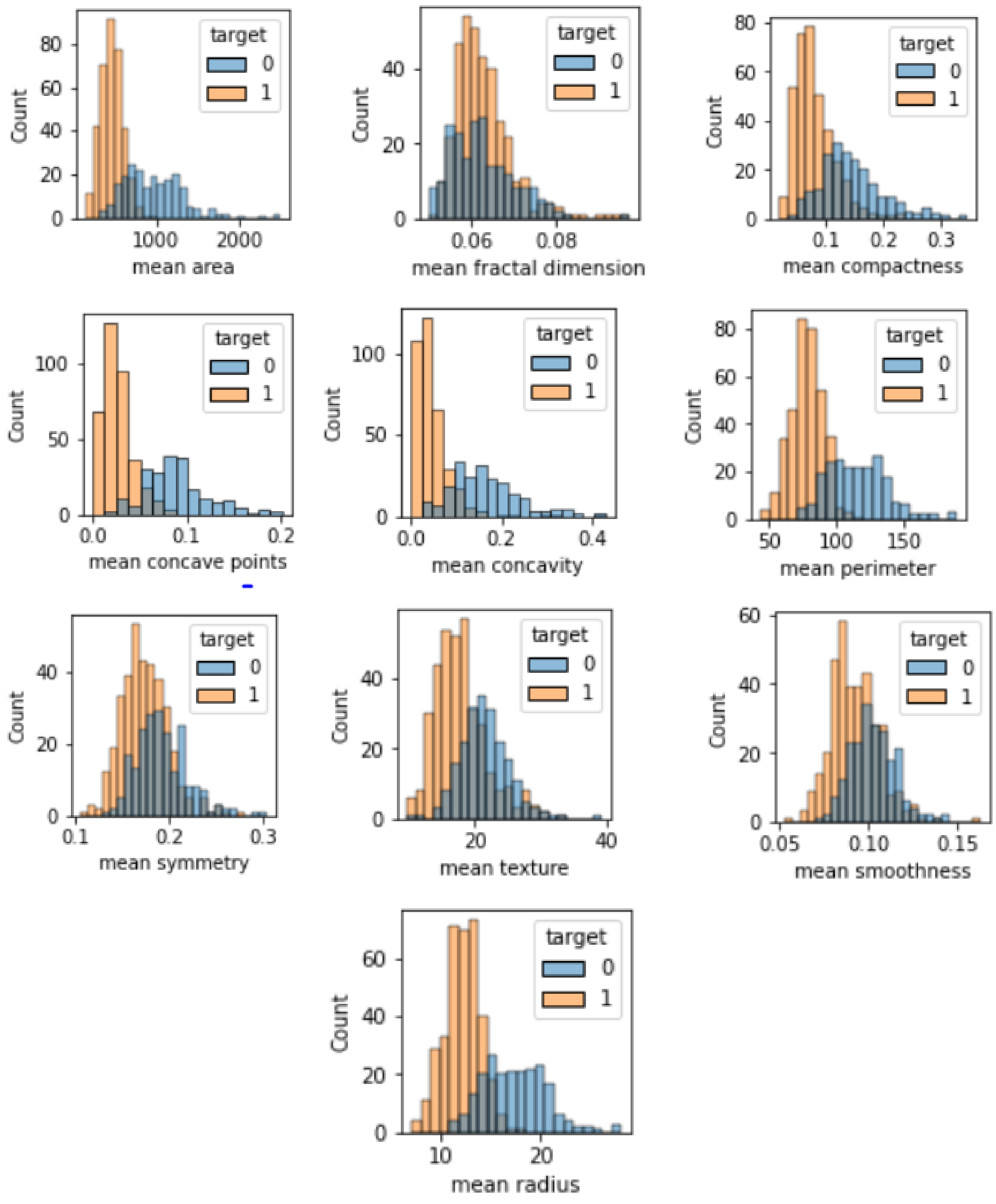


Figure 5 : Histograms of "mean" variable group by diagnosis.

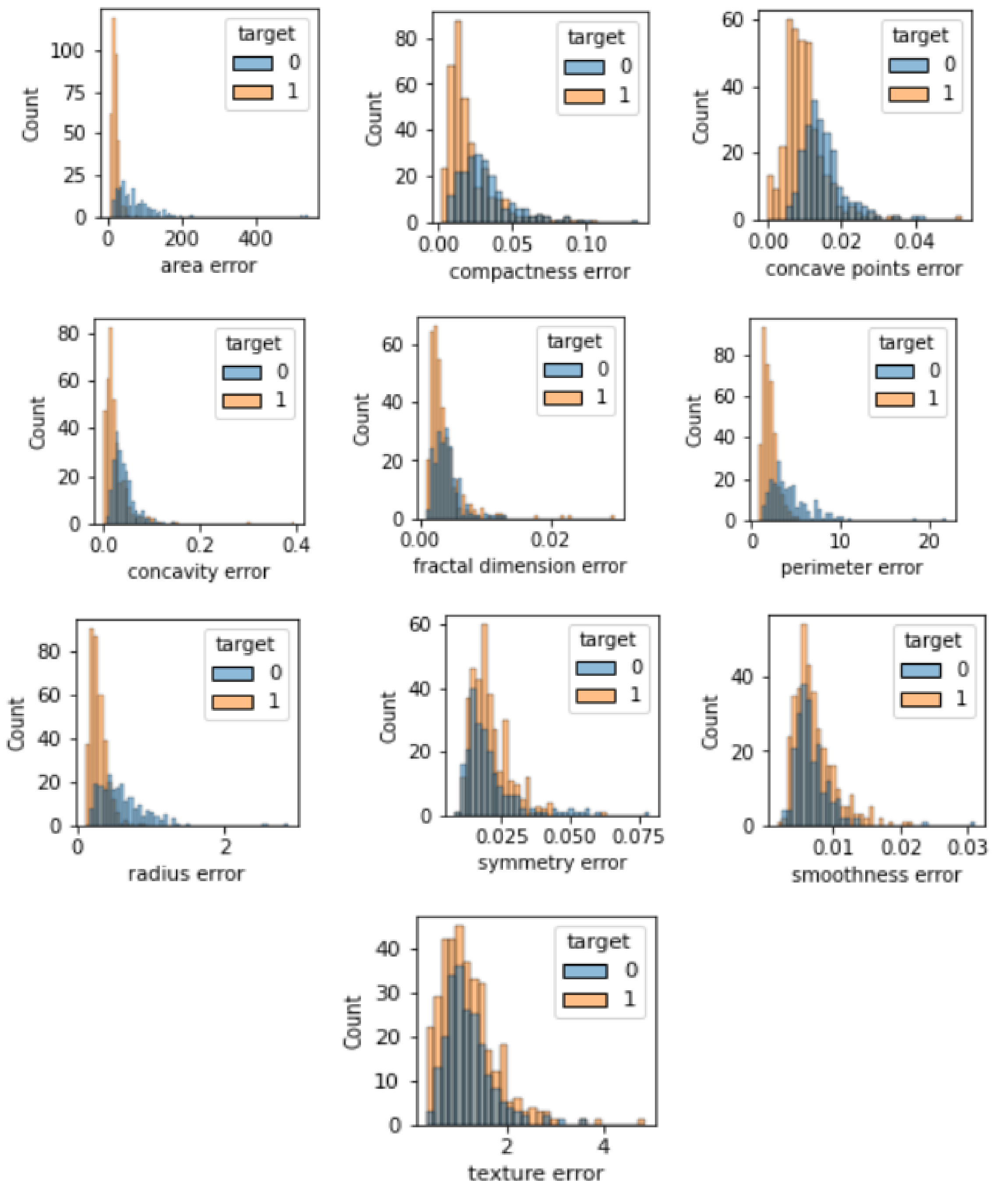


Figure 6 : Histograms of "error" variable group by diagnosis.

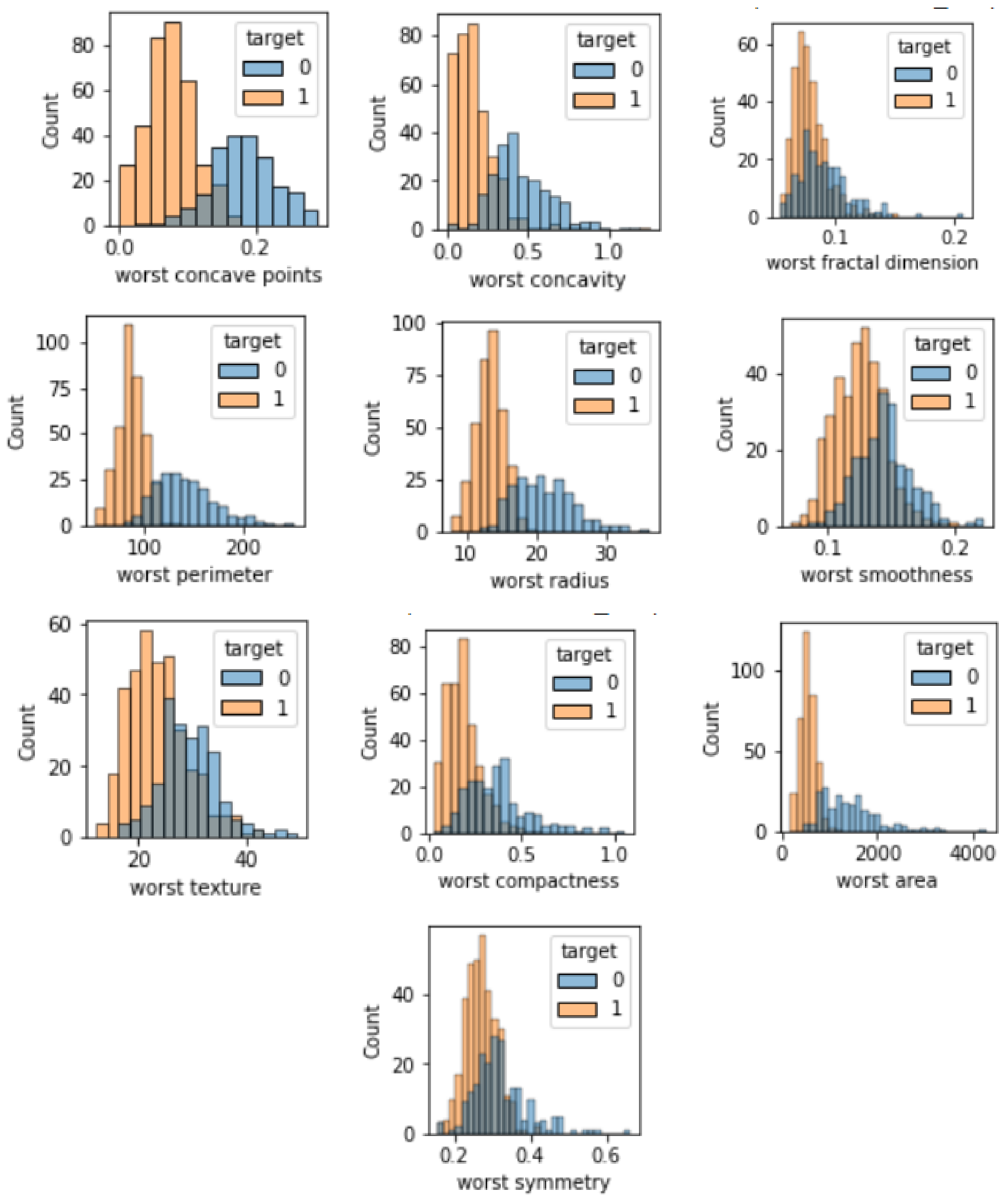


Figure 7 : Histograms of "worst" variable group by diagnosis.

Summary : Most of the features are normally distributed. We do have fairly **good** separations for **worst concave points**, **worst concavity**,**worst perimeter**, **mean area**, **mean perimeter**. We do have a tight superposition for some of the values, like **symmetry error**, **smoothness error** , **texture error**.

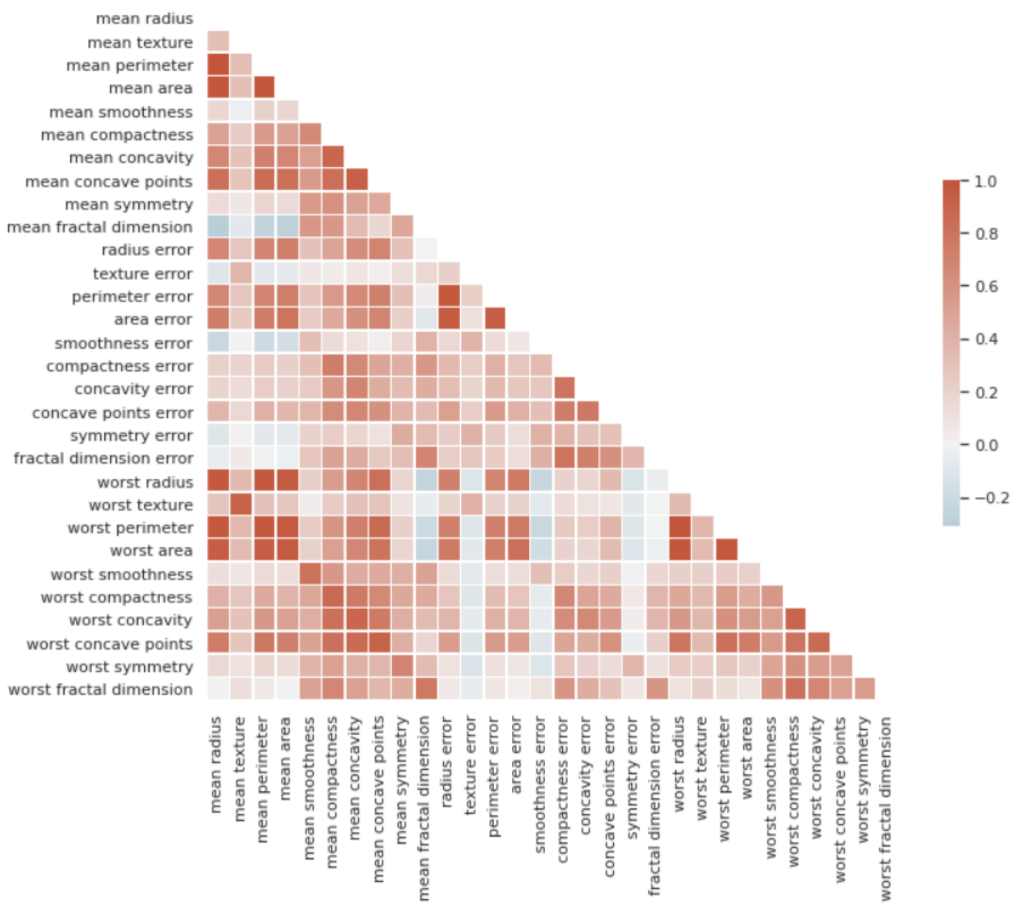
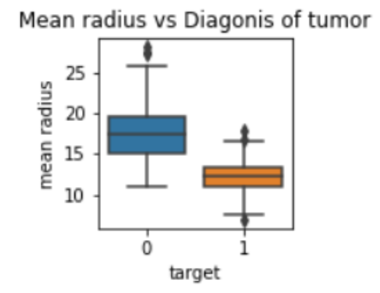
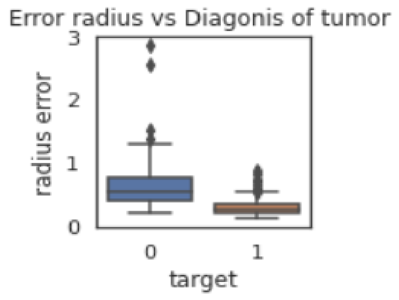


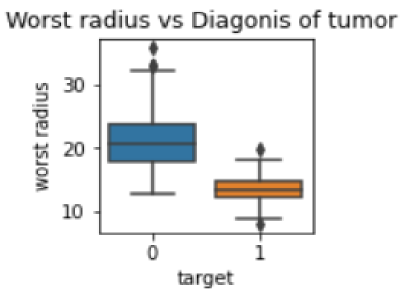
Figure 8: Correlation Matrix

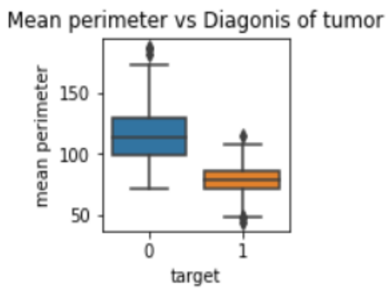
Summary : In order to understand the correlation between the features I plotted the correlation matrix. A correlation Matrix is a table showing the correlation coefficient between the variables. Each cell in the table shows the correlation between the two variables. The correlated features provide redundant information. By avoiding highly correlated features we can avoid positive bias for the information contained in these features.This also shows us, that when we want to make statements about the biological/ medical importance of specific features, we need to keep in mind that just because they are suitable to predicting an outcome they are not necessarily causal - they could simply be correlated with causal factors.

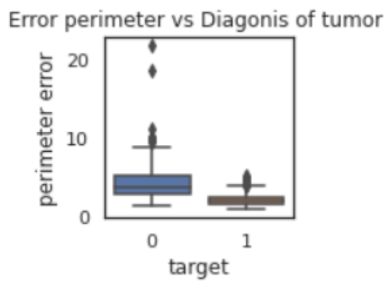
The **mean, error and worst** dimension lengths of **compactness, concavity and concave** points of tumors are highly correlated amongst each other (**correlation > 0.8**).The **mean texture and worst texture** have a (**correlation >0.9**) The **mean, errors,worst** dimensions of **radius, perimeter and area** of tumors have a (**correlation = 1**).

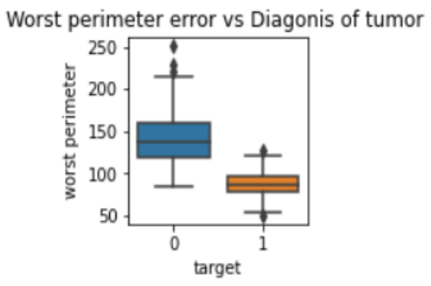

Mean radius vs Diagonis of tumor
mean radius
10
15
20
25
0
1
target

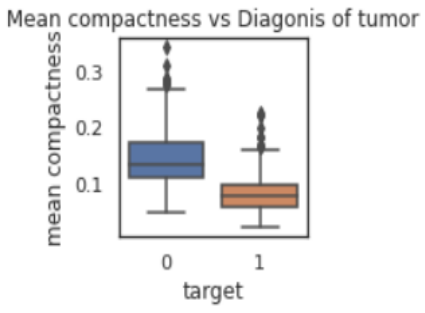

Error radius vs Diagonis of tumor
radius error
0
1
2
3
0
1
target

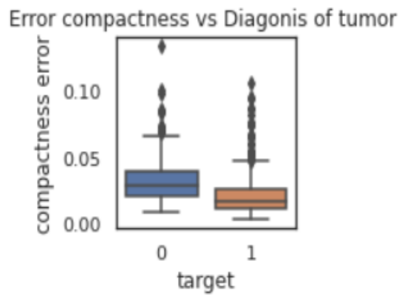

Worst radius vs Diagonis of tumor
worst radius
10
20
30
0
1
target

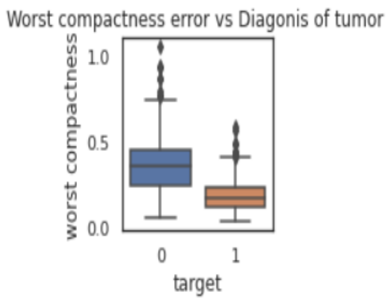

Mean perimeter vs Diagonis of tumor
mean perimeter
50
100
150
0
1
target


Error perimeter vs Diagonis of tumor
perimeter error
0
10
20
0
1
target


Worst perimeter error vs Diagonis of tumor
worst perimeter
50
100
150
200
250
0
1
target


Mean compactness vs Diagonis of tumor
mean compactness
0.1
0.2
0.3
0
1
target


Error compactness vs Diagonis of tumor
compactness error
0.00
0.05
0.10
0
1
target


Worst compactness error vs Diagonis of tumor
worst compactness
0.0
0.5
1.0
0
1
target

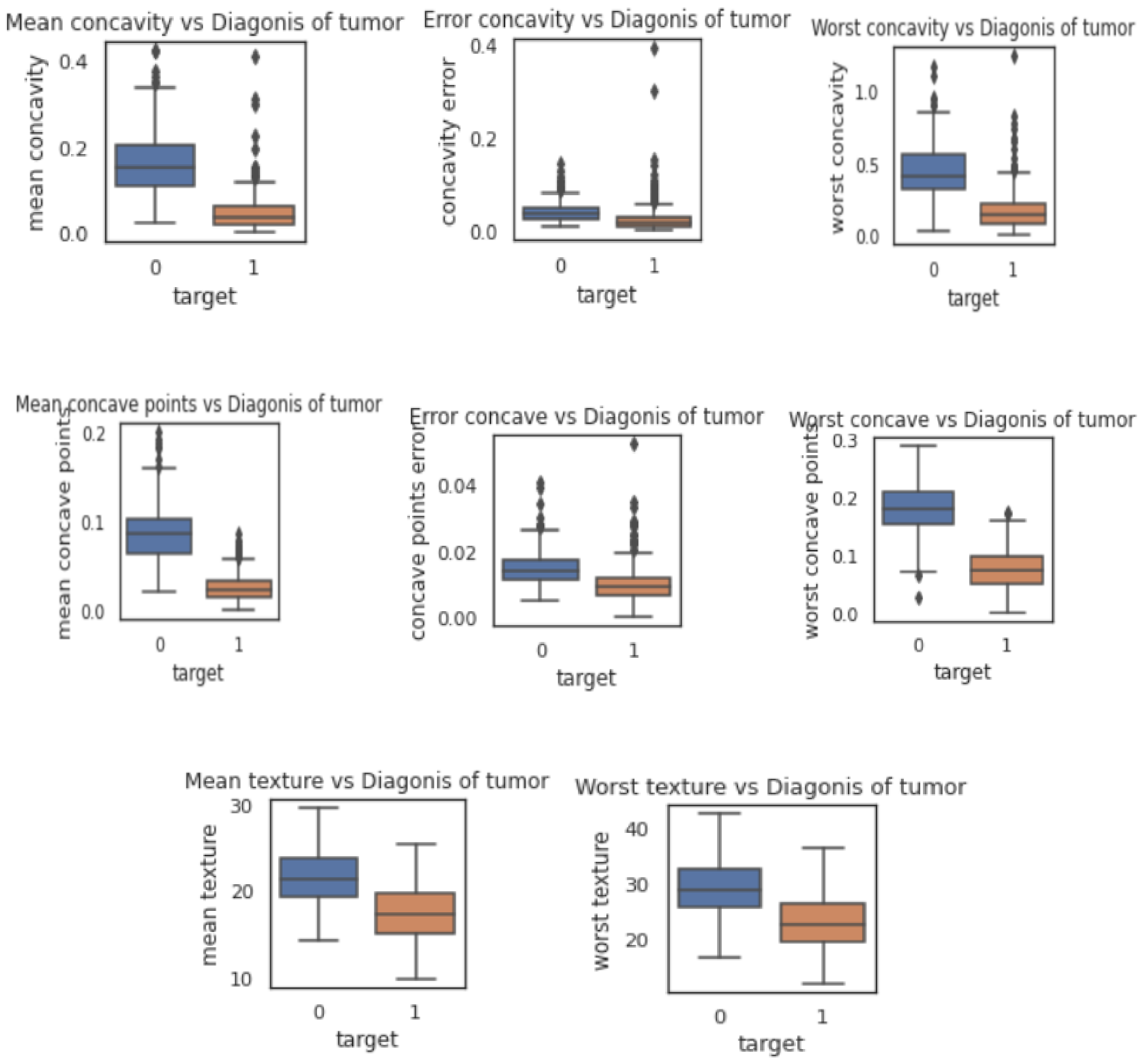


Figure 9: Highly correlated features Vs Class distribution

Summary: In order to understand if there is a **difference** between the **data distribution** between the **Malignant and Benign groups**, I visualized some highly correlated features via Boxplots. In most of the above cases we see , **Malignant** groups have a **wider range** of values compared to **Benign** groups.

Principal Component Analysis

Principal Component Analysis is a technique used for finding a low-dimensional representation of some data that captures as much information as possible. The idea is that each of a series of n observations lives in a p-dimensional space, but not all of these dimensions are equally interesting. PCA finds dimensions that are a linear combination of the original p features.
The first principal component of a set of features $X_1 \ldots\ldots X_2$ is the normalized linear combination of the features :

$$Z_1 = \phi_{11} X_1 + \phi_{21} X_2 + \ldots + \phi_{p1} X_p$$

that has the largest variance.Since it is normalized,it is also constrained to $\Sigma_{j=1}^{p} \phi_{j1}^2 = 1$

We refer to the elements $\phi_{11}, \ldots\ldots\ldots \phi_p$ loading of the first principal component together the loading make up the first principal loading vector $\phi_1 = (\phi_{11}, \ldots\ldots\ldots \phi_{12})^T$.
We constraints the loadings so that their sum of squares is equal to one, otherwise setting these elements to be arbitrarily large in absolute value could result in an arbitrarily large variance.

The PCA try to solve the following optimization problem:

$$\max imize \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{j=1}^{p} \phi_{j_1} x_{ij} \right)^2 \; subject \; to \; \sum_{j=1}^{p} \phi_{j_1=1}^2$$

The calculated vector of features defines the various directions where the data vary the most. The first principal component is a linear combination of feature columns that has the maximal variance. The second principal component has the same definition but the features must be uncorrelated with the first principal component and so on.

There are two benefits for applying PCA , firstly it reduces the curse of dimensionality and it makes the data uncorrelated and a lot of algorithms work better if the data is uncorrelated. We run the PCA algorithm on our datasets to visualize our dataset and better understand the variance . At first we check the variance achieved by *2 principal components* and the *3 principal components* and later perform a plot to better understand the number of principal components needed .

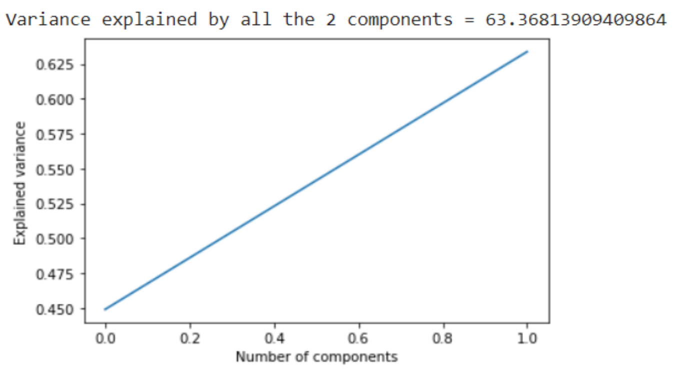


Figure 10: Above graph represents the variance obtained by 2 principal components

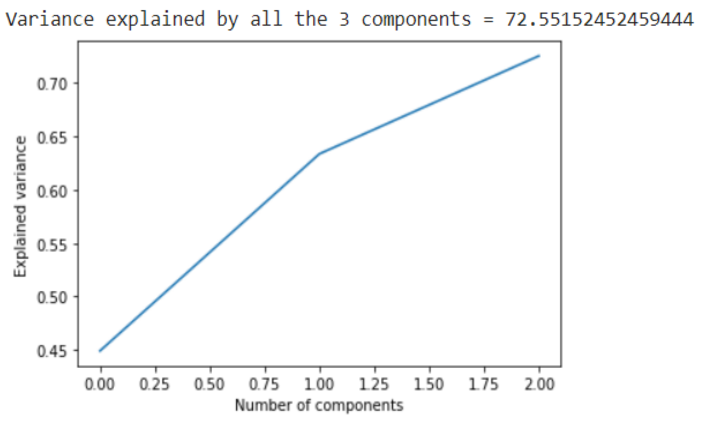


Figure 11: Above graph represents the variance obtained by 3 principal component

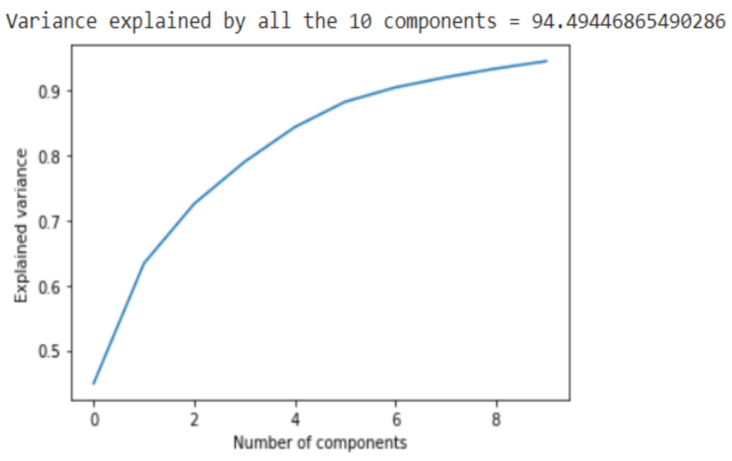


Figure 12: Above graph represents the variance obtained by 10 components

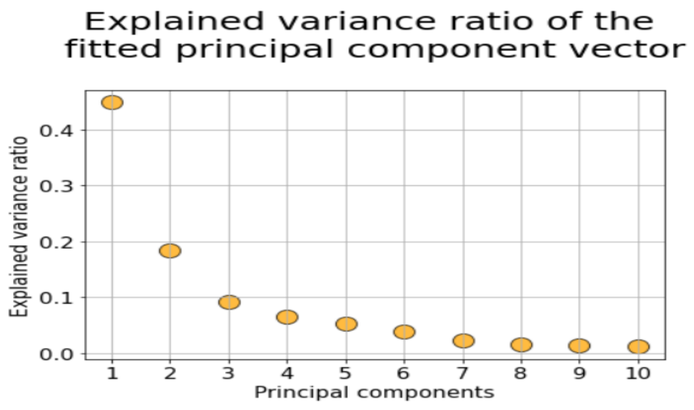


Figure 13: The first 6 PCs which together explain about 88.76% variability in the data

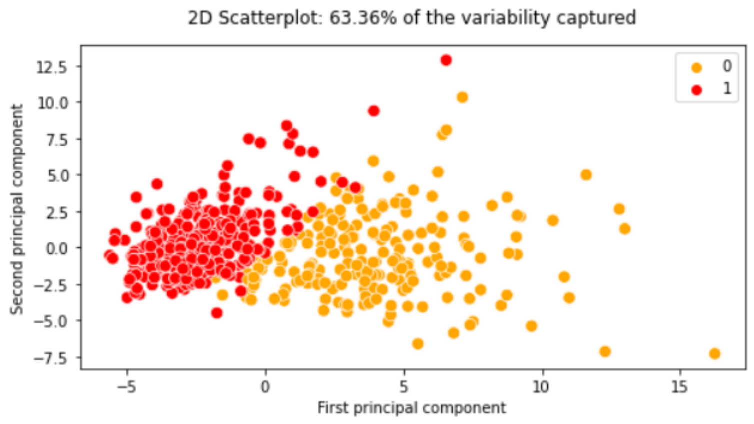


Figure 14: Scatterplot of breast cancer classes based on the first 2 principal components of the cancer features

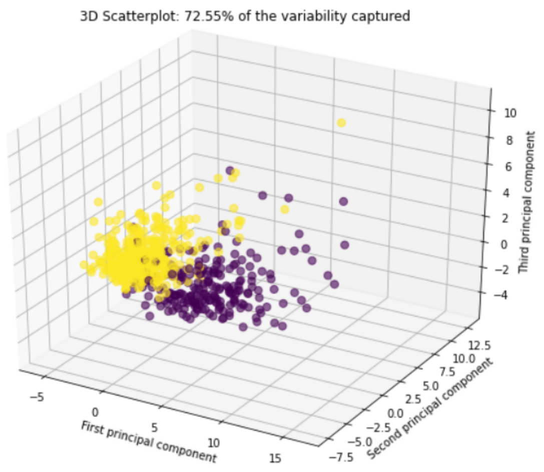


Figure 15: Scatterplot of breast cancer classes based on the first 3 principal components of the cancer features.

Classification

The main purpose of this report is to build a model which classifies tumors as Malignant and Benign. Hence I have used various classification models :

- K-Nearest Neighbours.
- Linear and RBF SVM.
- Logistic Regression.
- Decision Tree.
- Random Forest.

To measure the performance of the trained models were used some common metric:

- Classification Report
- Confusion Matrix
- Roc Curve.

The **classification report** visualizer displays the *precision, recall, F1*, and *support* scores for the model.

- **Precision** is the ability of the classifier not to label an instance positive that is actually negative. Precision is the accuracy of positive predictions.
- **Recall** is the ability of the classifier to find all the positive instances.
- **F1** score is the harmonic mean between precision and recall. The range for F1 Score is [0,1]. This measure means how precise the classifier is (how many instances it classifies correctly), as well as how robust it is.

$$\Pr ecision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1 = 2 \times \frac{(Recall \times \Pr ecision)}{(Recall + \Pr ecision)}$$

$$Accurracy = \frac{TP + TN}{(TP + TN + FP + FN)}$$

The **confusion Matrix** is a table that is used to describe the performance of a classifier on a set of test data for which the true values are known.
Some of the basic terms are :

- **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*.")

The **ROC(Receiver Operating Characteristics)** curve is one of the most important evaluation metrics for checking any classification model's performance. ROC is a probability curve that represents the degree or measure of separability. It tells how much a model is capable of distinguishing between classes. It's a plot of the false positive rate (x-axis) versus the true positive rate(y-axis) for a number of different candidate threshold values between 0 and 1.

$$TPR = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{FP + TN}$$

Smaller values on the x-axis of the plot indicate lower false positives and higher true negatives. Larger values on the y-axis of the plot indicate higher true positives and lower false negatives. A model with perfect skill is represented at a point(0,1). A model with perfect skills is represented by a line that travels from the bottom left of the plot to the top left and then across the top to the top right.

We have split our data in **Train,Validation and Test** set in the ratio of **5:2:3**.We scale our data using **MinMax Scaler**. The dataset was **stratified** in order to preserve the proportion of the target as in the original dataset as in the train and test dataset.The validation set is used to select the *best hyperparameters* for each model.

K-Nearest-Neighbors

The intuition behind the **KNN** classifier with parameter k is straightforward:
given a set of training examples $\{x_i \; y_i\}$ and a testing point x that we want to label, we compute the distance $D(x, \; x_i)$ for every point in the training set and the output class $y^{\cdot}$ of x according to the most frequent label among k closet distances $x_{i1} \cdots\cdots x_{ik}$.

The choice of the distance measure is crucial for the KNN classifier. It defines the similarity between 2 samples and can have a huge impact on the performance of the algorithm. The most general metric is so called the L_p norm or the *Minkowsky* distance:

$$L_p(x,y) = \left(\sum_{i=1}^{d} |x_i - y_i|^p \right)^{\frac{1}{p}}$$

The most used one is the $L2$ norm also known as *Euclidean distance*, which is nothing but Minkowsky distance a parameter p=2. Here I have decided to use *Euclidean Distance* as a distance measure.

The best choice of K depends upon the data , the generally large value of K reduces the effect of noise on the classification , but makes boundaries between the class less distinct. A good K can be selected by various heuristics techniques like K-fold cross validation and others. In case of binary class classification it's always helpful to choose K as an odd number as this avoids tied votes. The accuracy of K can be severely degraded by the presence of noise or irrelevant features, or if the features scales are not consistent with their importance.

To analyze better we have selected five different values of **K = [1,3, 5 ,7,9]** at start.
We train our model on the training set and evaluate our model on the validation set. Then select the optimal value of the K from the validation set and use it to test our model.

```
The optimal number of neighbors is  3
```

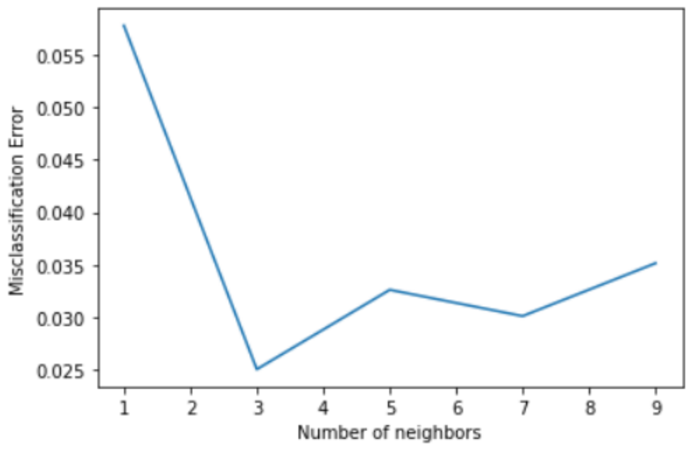


Figure 16: KNN Misclassification Error Vs Number of neighbors

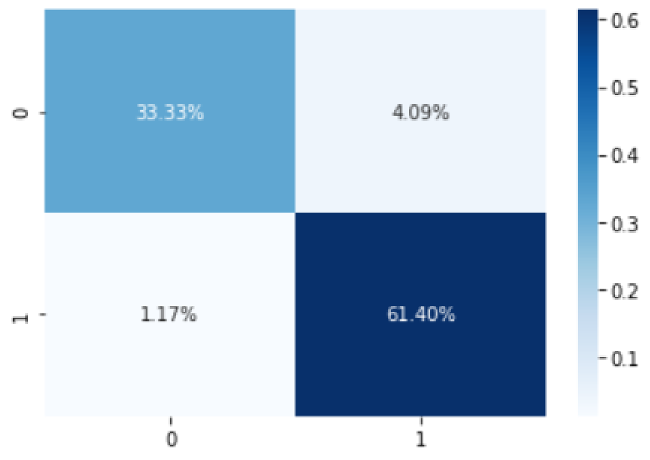
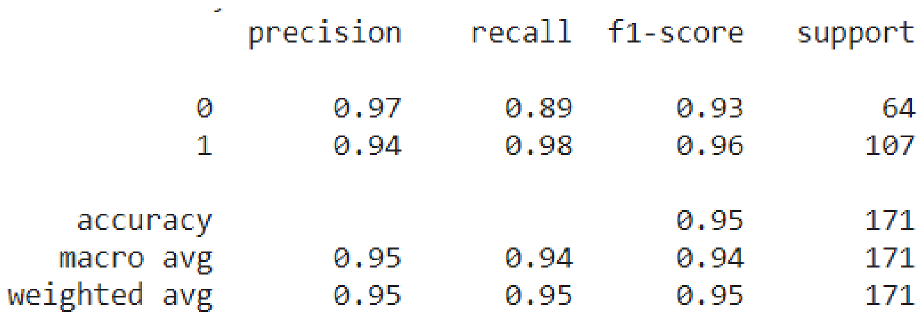


Figure 17 : Confusion Matrix



```
              precision    recall  f1-score   support

           0       0.97      0.89      0.93        64
           1       0.94      0.98      0.96       107

    accuracy                           0.95       171
   macro avg       0.95      0.94      0.94       171
weighted avg       0.95      0.95      0.95       171
```

Figure 18 : Classification Report

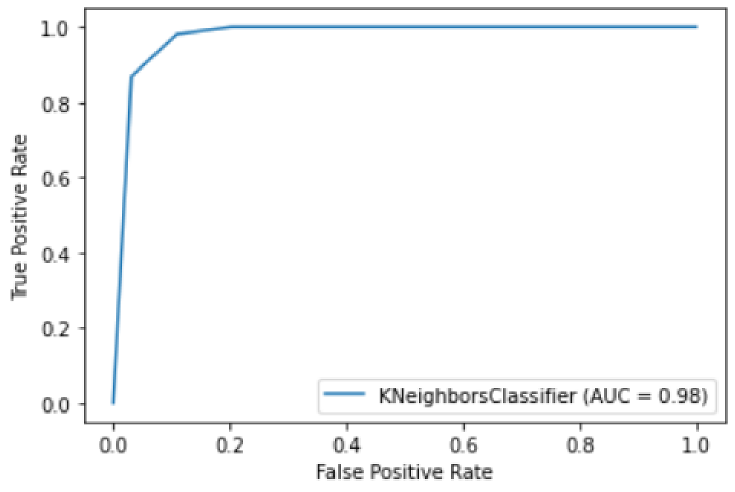


Figure 19 : ROC Curve

Support Vector Classifier

A Support Vector Machine is a classifier defined by a separating hyperplane, i.e. given a supervised learning problem, it outputs an optimal hyperplane which categorizes new samples. For instance, in a 2-dimensional space, it corresponds to a line dividing a plane in two parts - such that each class lies on either side.

Training a linear support vector classifier, like any Machine Learning algorithm is an optimization problem . We *maximize* the *margin* between - that is the *distance* separating the *closest pair of data points* belonging to *opposite classes*. These points are called support vectors because they are the data observations that support or determine the decision boundaries.

SVM comes with two different types of margin: *Hard Margin* and *Soft Margin*.

The **Hard margin** SVM is very rigid in classification and tries to always find a linearly separable hyperplane between two classes. Hard margin causes *overfitting* and a *single outliner* can determine the boundary , which makes the classifier over sensitive to noise and data. The formulation of the hard margin is :

$$y_i[< w,\ x_i > +b \geq 1]$$

Since it is not possible to have a linearly separable hyperplane, the hard margin SVM (i.e., no misclassification allowed) is not feasible.
The **Soft margin** SVM overcomes this problem by introducing slack variables ζ_i :

$$\min \frac{1}{2}||w||^2 + C\sum_i \zeta_i$$

$$\textit{subject to } y_i[< w,\ x_i > +b] \geq 1 - \zeta_i\ \zeta_i \geq 0$$

Thus, we want to tune an hyperparameter C (*penalty of the error term*) which tells how much we want to avoid misclassifying each training sample. Small values of C cause the optimizer to look for larger-margin separating hyperplanes,even though this comes at the cost of misclassifying more points. Conversely,large values of C correspond to smaller-margin hyperplanes if they do a better job at classifying all the points correctly. This strongly affects the decision boundaries of our model.
ζ (*error term*) which soften the constraint , allow for some *training points* to be *incorrectly classified* and C *controls the trade off* of how *well separated our data* is and the *size of the margin*.

Coming to the major part of SVM for which it is most famous for is the kernel trick.
The kernel is a way of computing the dot products between two vectors x and y in some very high dimensional feature space, which is why kernel functions are also sometimes called generalized dot product. For instance,we can apply support vector expansion to the linear function of SVM:

$$f(x) = \sum_i \alpha_i \, y_i \; < x_i \, x > \; + b$$

where x_i is a training example, and α_i is a coefficient for x_i .This enables us to replace x with the output of a given feature function $\phi(x)$ and the dot product with a function $k(x, x_i) = \phi(x). \, \phi(x_i)$ called a kernel.

There are various types of kernel - Linear kernel , Polynomial kernel , RBF / Gaussian Kernel. In this report, two classifiers based on the Linear and RBF kernel will be analyzed and compared.

The first one is the **Linear SVM classifier**. Its aim is to try to define linear hyperplanes to perfectly separate data so that each sector of the space contains points belonging to the same class. In the real world since data is not linearly separable we will be using the condition of soft margin , assigning some sort of penalties to points that fall into the margin and make the classification tasks feasible.

The second classifier is the **RBF-kernel based** on SVM. Such a kernel is introduced to allow a non-linear classification. This classification tries to define decision boundaries which will be non-linear. The key parameters of this classifier are C and gamma. *Gamma* defines how concentrated is the influence of a single training point: the *lower the value*,the further its influence will arrive, but also the *weaker* it will be . Too much *high value* of gamma will eventually lead to *overfitting*. Moreover,we will always remember the importance of the parameter C, that in this case acts as a *regularization factor*.

Initially we tried these two different kernels with fixed values of C and gamma , later we used grid search in order to find the right hyperparameters for our SVM model .

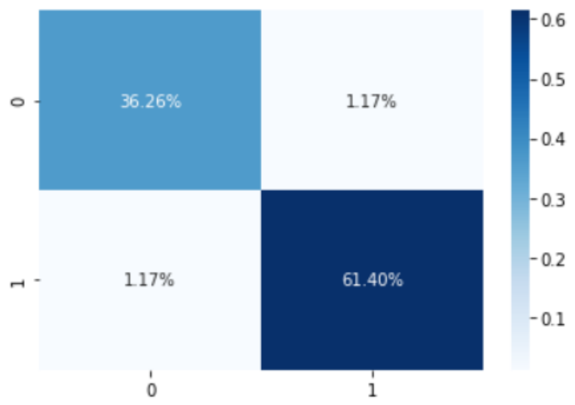


Figure 20 : Confusion Matrix

```
              precision    recall  f1-score   support

           0       0.97      0.97      0.97        64
           1       0.98      0.98      0.98       107

    accuracy                           0.98       171
   macro avg       0.98      0.98      0.98       171
weighted avg       0.98      0.98      0.98       171
```

Figure 21 : Classification Report

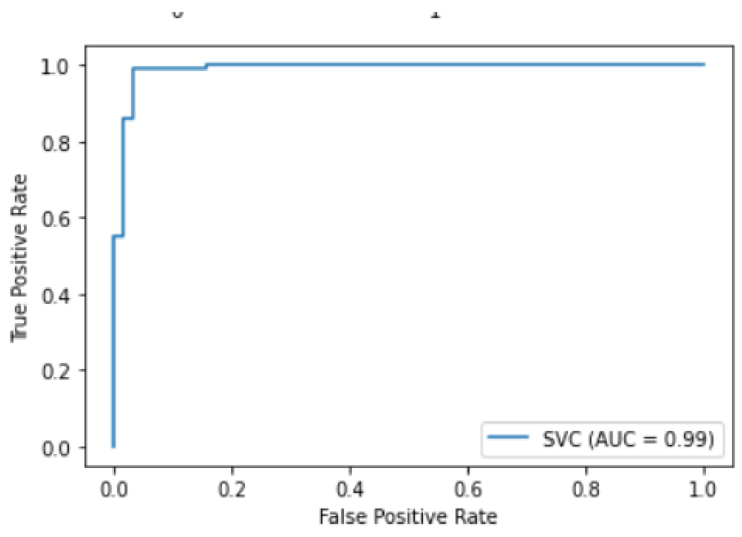


Figure 22: ROC Curve

Logistic Regression

Logistic regression is a classification algorithm used to assign observations to a discrete set of classes. Unlike linear regression which outputs continuous number values, logistic regression transforms its output using the *logistic sigmoid function* to return a probability value which can then be mapped to two or more discrete classes.

For example, given data on time spent on studying and exam scores , it could help us to predict whether a student has passed or failed. Logistic regression is discrete(only specific values or categories are allowed)

The logistic function is defined as :

$$S(z) = \frac{1}{1+e^{-z}}$$

The step from linear regression to logistic regression is kind of straightforward. In the linear regression model, we have modelled the relationship between outcome and features with a linear equation:

$$y_i^{(i)} = \beta_0 \ + \beta_1 x_1^{(i)} + \ldots\ldots + \beta_p x_p$$

For classification, we prefer probabilities between 0 and 1, so we wrap the right side of the equation into the logistic function. This forces the output to assume only values between 0 and 1.

$$P\left(y_i^{(i)} = 1\right) = \frac{1}{1+\exp\left(-\left(\beta_0 \ + \beta_1 x_1^{(i)} + \ldots\ldots + \beta_p x_p\right)\right)}$$

The interpretation of the weights in logistic regression differs from the interpretation of the weights in linear regression, since the outcome in logistic regression is a probability between 0 and 1. The weights do not influence the probability linearly any longer. The weighted sum is transformed by the logistic function to a probability. Therefore we need to reformulate the equation for the interpretation so that only the linear term is on the right side of the formula.

$$\log\left(\frac{P(y=1)}{1-P(y=1)}\right) = \log\left(\frac{P(y=1)}{P(y=0)}\right) = \beta_0 + \beta_1 x_1 + \ldots\ldots + \beta_p x_p$$

We call the log function odd (probability of event divided by probability of no even). This formula shows that the logistic regression model is a linear model for the *log odds*.
We first apply the exp() function to both sides of the equation:

$$\frac{P(y=1)}{1-P(y=1)} = odds = \exp(\beta_0 \ + \beta_1 x_1 + \ldots\ldots + \beta_p x_p)$$

Then we compare what happens when we increase one of the feature values by 1. But instead of looking at the difference, we look at the ratio of the two predictions:

$$\frac{odds x_j + 1}{odds} = \frac{\exp(\beta_0 + (\beta_1 x_1 + 1) + ... + \beta_p x_p)}{\exp(\beta_0 + \beta_1 x_1 + ... + \beta_p x_p)}$$

We apply the following rule:

$$\frac{\exp(a)}{\exp(b)} = \exp(a - b)$$

And we remove some terms :

$$\frac{odds x_j + 1}{odds} \exp(\beta_j (x_j + 1) - \beta_j x_j) = \exp(\beta_j)$$

In the end, we have something as simple as exp() of a feature weight. A change in a feature by one unit changes the odds ratio (multiplicative) by a factor of $\exp(\beta_j)$.We could also interpret it this way: A change in x_j by one unit increases the log odds ratio by the value of the corresponding weight.

We use the parameter C as our *regularization parameter*. Parameter $C = \frac{1}{\lambda}$

Lambda (λ) controls the trade-off between allowing the model to increase its complexity as much as it wants while trying to keep it simple. For example, if λ is very low or 0, the model will have enough power to increase its complexity (overfit) by assigning big values to the weights for each parameter. If, on the other hand, we increase the value of λ, the model will tend to underfit, as the model will become too simple.

Parameter C will work the other way around. For small values of C, we increase the regularization strength which will create simple models which underfit the data. For big values of C, we low the power of regularization which implies the model is allowed to increase its complexity, and therefore, overfit the data.

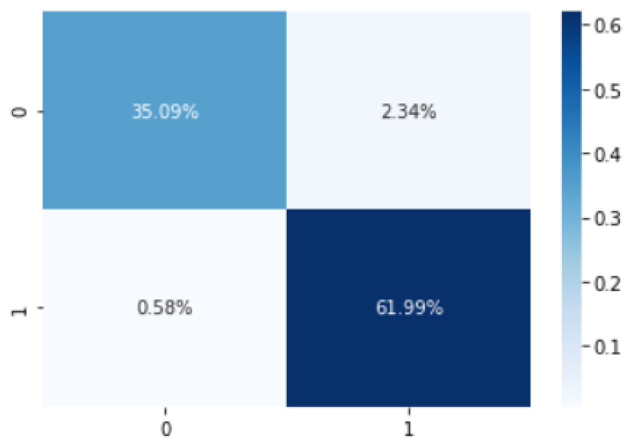
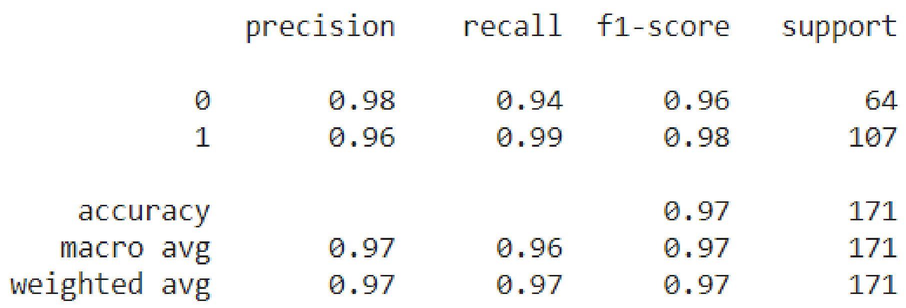


Figure 23: Confusion Matrix



```
              precision    recall  f1-score   support

           0       0.98      0.94      0.96        64
           1       0.96      0.99      0.98       107

    accuracy                           0.97       171
   macro avg       0.97      0.96      0.97       171
weighted avg       0.97      0.97      0.97       171
```

Figure 24: Classification Report

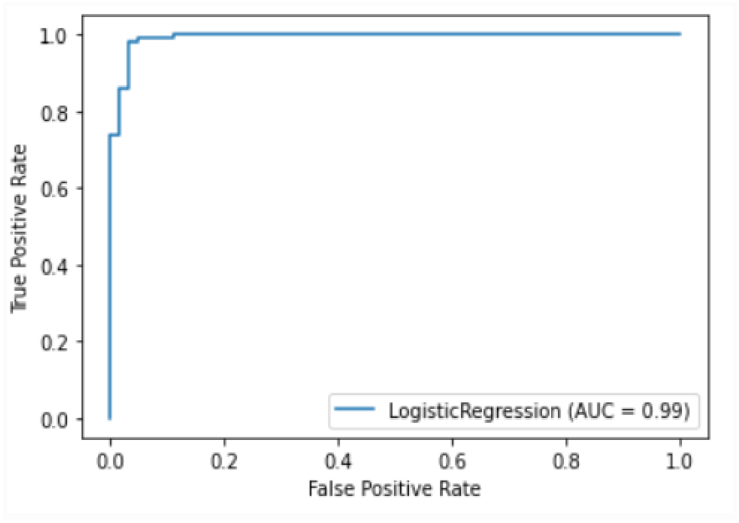


Figure 25: ROC Curve

Decision Tree

Decision tree is one of the predictive modelling approaches used in statistics, data mining and machine learning. It uses a decision tree (as a predictive model) to go from observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves). Tree models where the target variable can take a discrete set of values are called classification trees and where the target variable can take continuous value is called the regression tree. In this work, we only focus on classification trees.

For a classification tree, we predict that each observation belongs to the most commonly occurring class of training observations in the region to which it belongs.
Since in classification settings, RSS cannot be used as a criterion for making the binary splits so a natural alternative to RSS is the classification error rate. This is simply the fraction of the training observations in that region that do not belong to the most common class.

$$E = 1 - \max(pmk)$$

Here pmk represents the proportion of the training observations in the mth region that are from the kth class.
However classification error is not sufficiently sensitive for tree-growing , and in practice two other measures are preferable.

The Gini index is defined by:

$$G = \sum_{k=1}^{K} pmk(1 - pmk)$$

a measure of total variance across the K classes. The Gini index takes on a small value if all of the pmk's are close to zero or one. For this reason the Gini index is referred to as a measure of node purity - a small value indicates that a node contains predominantly observations from a single class.

An alternative to the Gini index is cross-entropy, given by

$$D = -\sum_{k=1}^{k} pmk \ \log \ pmk$$

It turns out that the Gini index and the *cross-entropy* are very similar numerically.
Decision trees are the simple and interpretable models for regression and classification. However they are often not competitive with other methods in terms of prediction accuracy.

Various methods can be implemented like *bagging, boosting and random forest* for improving the prediction accuracy of trees. They work by growing many trees on the training data and then combining the predictions of the resulting ensemble of trees.

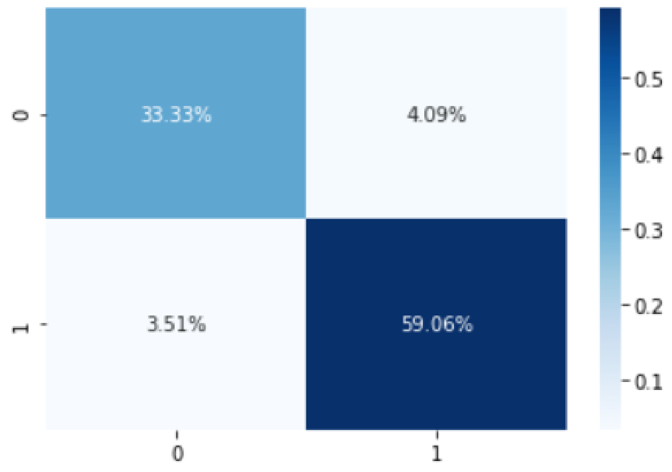
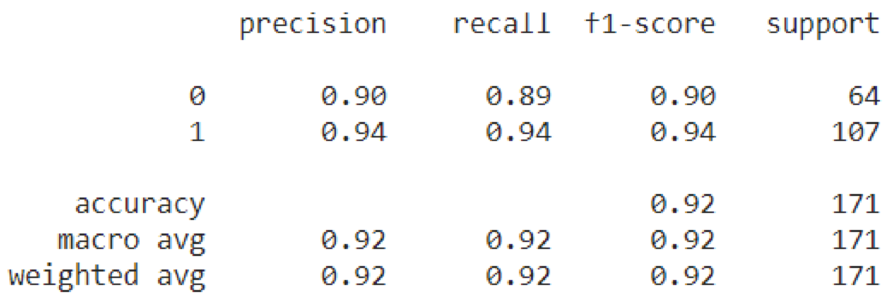


Figure 26: Confusion Matrix



```
              precision    recall  f1-score   support

           0       0.90      0.89      0.90        64
           1       0.94      0.94      0.94       107

    accuracy                           0.92       171
   macro avg       0.92      0.92      0.92       171
weighted avg       0.92      0.92      0.92       171
```

Figure 27: Classification Report

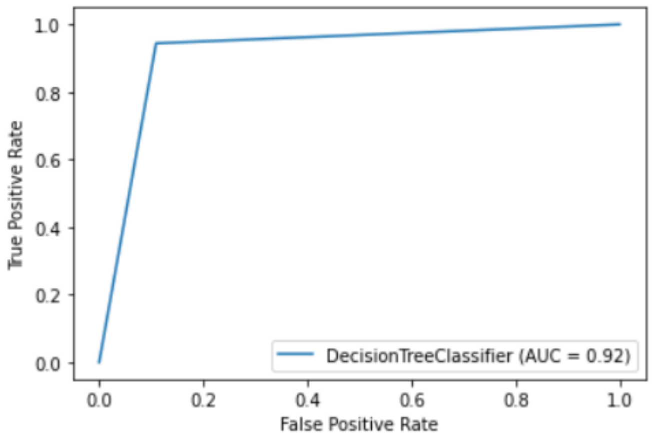


Figure 28: ROC Curve

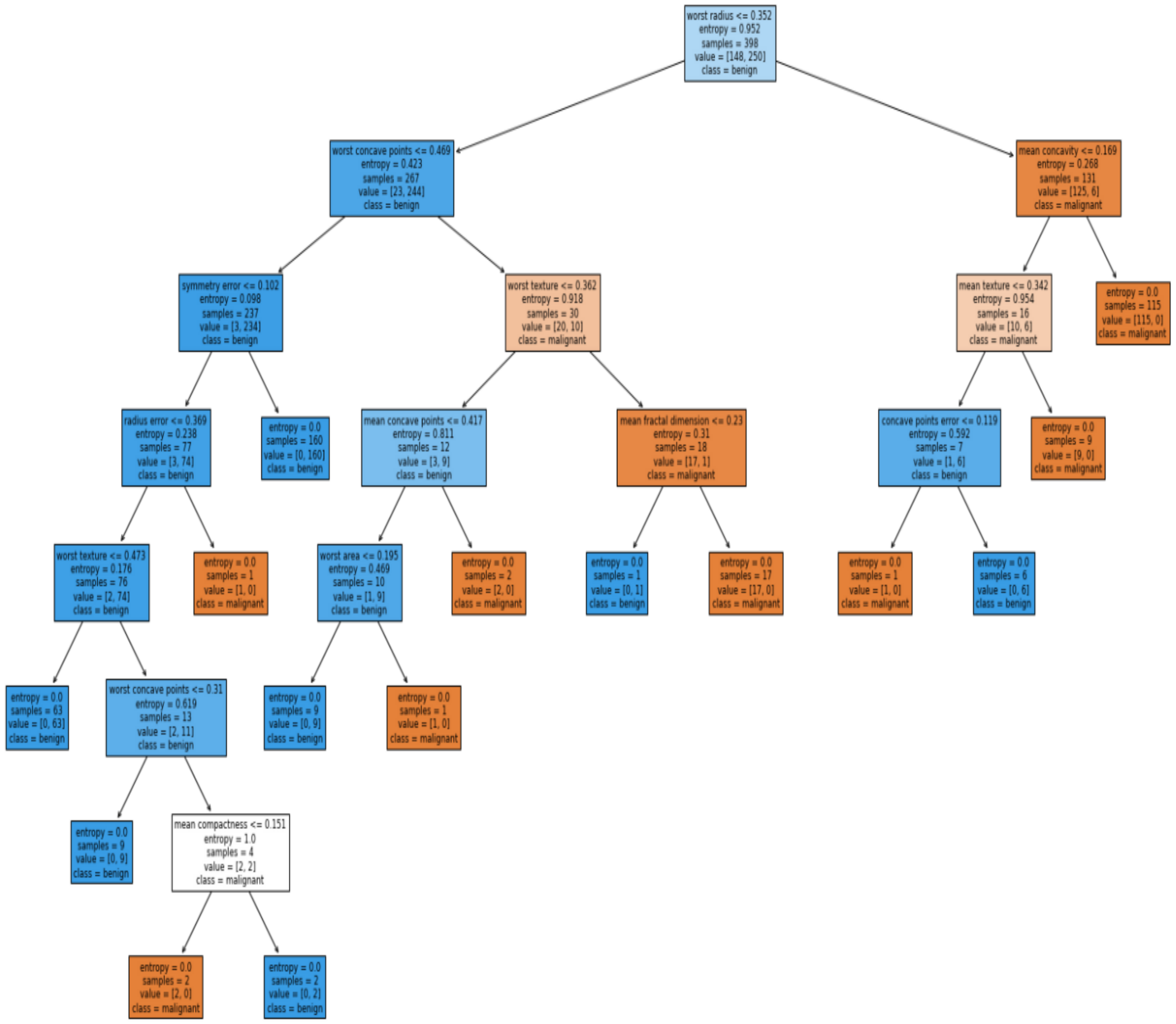


Figure 29: Decision Tree

Random Forest

Random forest is a supervised learning algorithm. The “forest” it builds is an ensemble of decision trees, usually trained with the “bagging” method. The general idea of the bagging method is that a combination of learning models increases the overall result.

Random forest builds multiple decision trees and merges them together to get more accurate and stable predictions. Random forest can be used for both classification and regression. Random forest has nearly the same hyperparameters as a decision tree or a bagging classifier.

Random forest adds additional randomness to the model, while growing the trees. Instead of searching for the most important feature while splitting a node , it searches for the best feature among a random subset of features. This results in a wide diversity that generally results in a better model.

Therefore , in random forest, only a random subset of the features is taken into consideration by the algorithms for splitting a node. Additionally it is also possible to add some random threshold for each feature rather than searching for the best possible thresholds(like a normal decision tree does.)

Random forest is a great algorithm to train early in the model development process, to see how it performs and it also avoids the problem of overfitting if there are enough trees in the forest. Random forests *($m<p$)* lead to a slight improvement over bagging *($m=p$)*.

The main limitation of random forest is that a large number of trees can make the algorithm too slow and ineffective for real-time predictions.

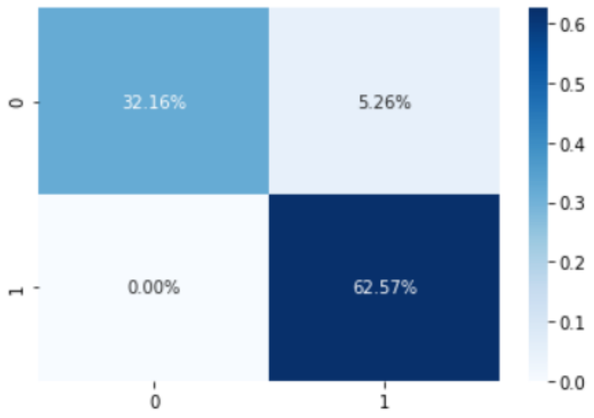
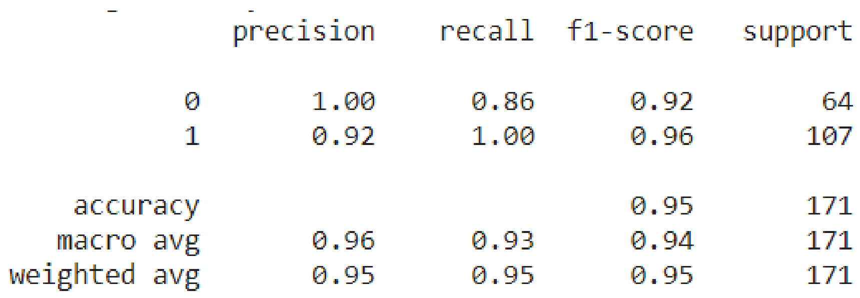


Figure 30: Confusion Matrix



```
              precision    recall  f1-score   support

           0       1.00      0.86      0.92        64
           1       0.92      1.00      0.96       107

    accuracy                           0.95       171
   macro avg       0.96      0.93      0.94       171
weighted avg       0.95      0.95      0.95       171
```

Figure 31: Random Forest Classification Report

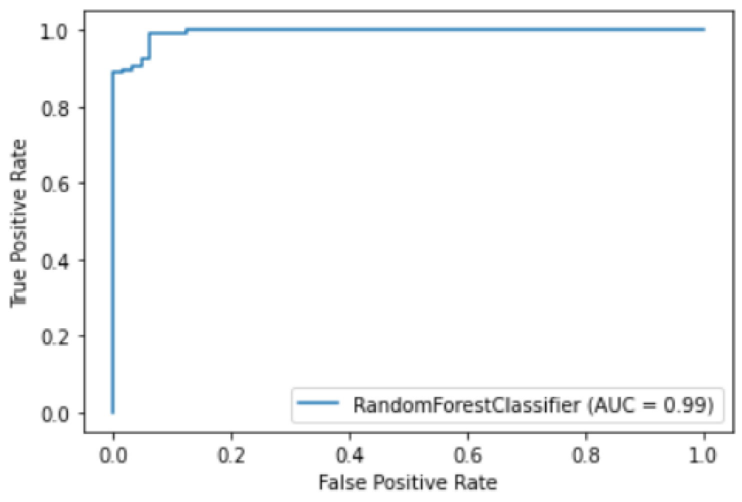


Figure 32: ROC Curve

Details Analysis of Results in Tabular Form

Classification Report	Validation Accuracy	Hyperparameters Tuning	Test Accuracy	F1 score
K-Nearest Neighbors	97.23%	K=3	94.73%	0.95
Support Vector Classifier	97.48%	{ C: 0.8, gamma: 0.1, Kernel: ‘linear’ }	97.66%	0.98
Logistic Regression	95.47%	{ C:10}	97.07%	0.97
Decision Tree	94.72%	{criterion : ‘entropy’, max_depth :100}	92.39%	0.92
Random Forest	97.48%	{bootstrap=’True’, max_depth=30, max_features=’auto’ criterion : ‘entropy’, min_samples_split:8, n_estimators:65}	94.73%	0.95

Conclusion

Various Classification techniques were applied in order to predict the Malignant and Benign. While some of the models perform particularly well (*SVM, Logistic Regression*),the main difficulties are linked to the small dimension of the dataset, the high correlation between a lot of features and the unbalance between the two classes.

There are various prospects of future improvements which can be further be performed. For example we could drop some highly correlated features or perform some sampling strategies like oversampling the minority class (positive class) or undersampling the majority class (negative class). Also increasing the fold of the cross validation.

Tools

I have used Colab to write the code. The libraries which is used are as follows:

- Matplotlib
- Seasborn
- Pandas
- Numpy
- Scikit learn