Improving Breast Cancer Classification Using Machine Learning Algorithms

Roshan Dhanasekaran
12135637

Improving Breast Cancer Classification using Machine Learning Algorithms

Roshan Dhanasekaran
Coventry University
Coventry, England
dhanasekaran@coventry.ac.uk

Suleiman Adamu Yakubu
Coventry University
Coventry, England
yakubus6@coventry.ac.uk

Abstract-- ***This paper focuses on classifying breast cancer diagnosis based on several variables like radius mean, perimeter mean and area means, and 17 other variables. This was achieved by using classification algorithms like Naive Bayas, SVC, Random Forest, and MLP Classifier. Multiple pre-processing methods were deployed covering standard scaling, Min-Max scaling, feature selection, and PCA to improve the performance of the models and bring the best from the models and finally compare the models and choose the best suitable model to achieve the final goal. All the codes used in this coursework were taken from Sklearn, python libraries(pandas and NumPy) were used and matplotlib was utilised to visualize the data.***

Keywords-- Radius mean, perimeter mean and area means, Naive Bayas, SVC, Random Forest, MLP Classifier, from standard scaling, Min-Max scaling, feature selection, and PCA, from Sklearn, python, pandas and NumPy, matplotlib.

1. INTRODUCTION

According to world health origination (WHO) Breast cancer (BC) was the major contributor to women's death and cancer cases are at an alarming high (El Agouri et al., 2022). BC is a universal disease that affects women between the age of 25 and 50 (Vaka et al., 2020) It was found that 1 in 8 women have the chance of developing BC and a 17% chance of developing BC in a women's lifetime. The primary cause of BC can be from mutation of cells and other factors like increasing age, obesity, and postmenopausal hormonal imbalances (Vaka et al., 2020).

There is no prevention currently available in medicine at the moment, but early detection by self-checking can reduce the chance of cancer proceeding to further stages. Although it can be very difficult for a person to detect a cancer cell by touching it. Hears where machine learning comes into play.

The ground-breaking Machine learning (ML) and Artificial Intelligence (AI) algorithms, we are able to classify the diagnosis with more precision given there is a huge number of the dataset. This paper mainly focuses on deploying support vector machine (SVM), *Naive Bayas Random Forest, and MLP Classifier* algorithms to find the accuracy and tune the modules with feature selection, min-max scaling, standard scaling, PCA to improve the accuracy and plot the results in a graph comparing all the models and checking the accuracy using confusion matrix and give statical analysis on the output.

The paper is split into 6 parts where 1St part is the introduction to the topic 2nd part describes the dataset and gives us more insights into the dataset and all the variables in the dataset. 3rd part we prepare the data

by using some pre-processing techniques and deploying it into the machine learning algorithms. 4Th part is where we introduce machine learning and use the prepared data in our models to get primary results and improve results by tuning the model. 5Th will be future studies on the subject and finally the conclusion.

2. THE DATASET

The dataset which we are using in this paper was obtained from Kaggle ("Breast Cancer Wisconsin (Diagnostic) Data Set", 2022) the dataset consist of 569 instances and 33 attributes. Each instance refers to the shape and size of cancer for each induvial. The second attribute demotes the diagnosis required by the patient therefore diagnosis is the independent variable and the rest of the columns are the dependent variables.

NO	Description	Type	UNIT
1	Diagnosis	String	
2	Radius mean	Int	Um
3	Texture mean	Int	Um
4	Perimeter mean	Int	Um
5	Area mean	Int	Um
6	Smoothness mean	Int	Um
7	Compactness mean	Int	Um
8	Concavity mean	Int	Um
9	concave points mean	Int	Um
10	Symmetry mean	Int	Um
11	Fractal dimension mean	Int	Um
12	Radius se	Int	Um
13	Text size se	Int	Um
14	Perimeter se	Int	Um
15	Area size	Int	Um
16	Smoothness se	Int	Um
17	Compactness se	Int	Um
18	Concavity se	Int	Um
19	concave points se	Int	Um
20	Symmetry se	Int	Um
21	Fractal dimension se	Int	Um
22	radius worst	Int	Um
23	texture worst	Int	Um
24	perimeter worst	Int	Um
25	area worst	Int	Um
26	smoothness worst	Int	Um
27	compactness worst	Int	Um
29	concavity worst	Int	Um
30	concave points worst	Int	Um
31	Symmetry worst	Int	Um
32	Fractal dimension worst	Int	Um

Table 1. Dataset

3. DATA PREPARATION

This section covers the analysis of the data finding correlation in the data, scaling down data using min-max scalar and standard scalar and selecting reverent features using recursive feature elimination with cross-validation and reducing the dimensions of the variables by using PCA.

1. Correlation heat map

A correlation. Heatmap is a graphical representation of a correlation matrix that can represent a correlation between different variables. It can be used to understand the linear relationship between several variables in a dataset. We can use this technique to drop out highly correlated variables in the dataset

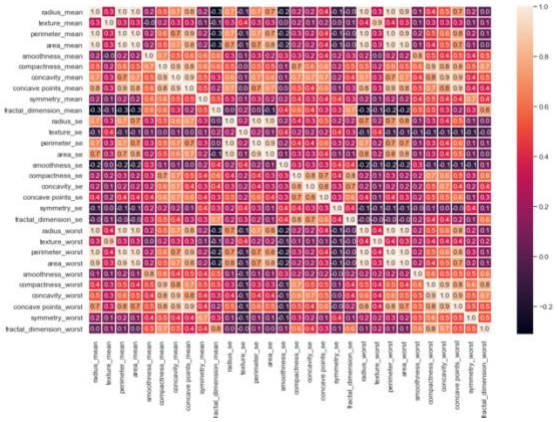


Figure 1. Correlation heatmap

2. CLASSIFICATION OF DATA

We intend to classify the data as benign and malignant based on radius worst and texture worst by plotting we can get see how the data is classified and develop ways to improve classification with data pre-processing techniques.

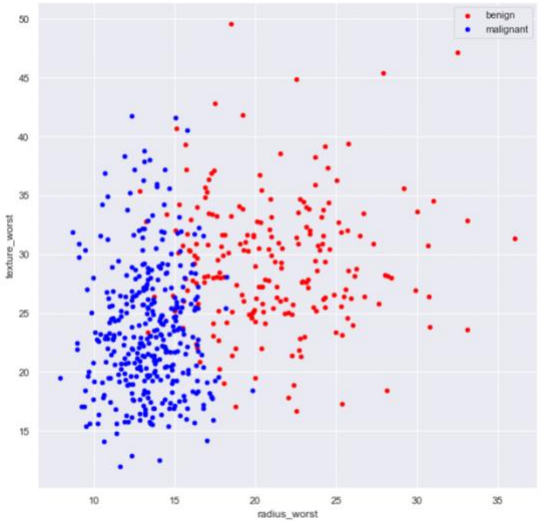


Figure 2. classification using scatter plot

A. Class Imbalance

(Johnson & Khoshgoftaar, 2019) have suggested that class imbalance is very common in binary classification datasets and there are several techniques to address the class imbalance. The concept of class imbalance can also be extended to the multiclass problem since it is possible to convert a multi-class problem into a binary class problem by using class decomposition. I also found that addressing the class imbalance in the pre-processing stage can improve the accuracy of the model. In this paper, we use SMOT from sklearn to tackle class imbalance.

1) SMOTE Over-sampling

SMOTE is an over-sampling technique where artificial samples are created from the minority class. This algorithm helps to eliminate the overfitting problem posed by random oversampling. It utilises the KNN algorithm to create synthetic values.

```
B    357
M    212
Name: diagnosis, dtype: int64
```

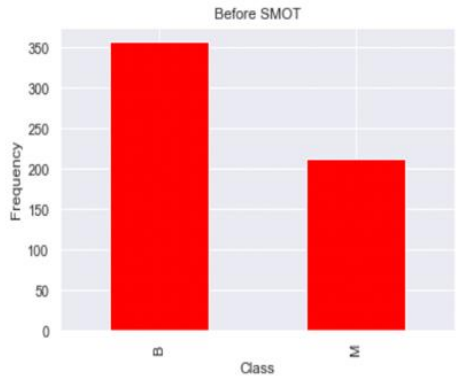


Figure 3. data before performing SMOTE

As we can clearly see the imbalance in the dataset where Benign has 357 instances where malignant with only 212. After performing SMOTE

Figure 4. data after performing SMOTE

After performing SMOT we address other problems in the dataset.

B. Categorical Label encoding

It is always advisable to have insights over the dataset before applying a machine learning(ML). Dropping irrelevant features and removing Nan values is crustal before performing ML Although some variables needed to perform ML can be categorical in these instances we need to convert the categorical variables into a machine-readable format therefore we use a label encoder to address the issue

Diagnosis
M
B
B
M

Diagnosis
1
0
0
1

Table 2. Converting categorical variables into numeric

After converting the categorical variables into numeric we can drop the categorical columns and replace them with the converted columns.

C. Standard scaling

Standard scaling is a type of scaling technique wherein the variables are centred around the mean value with standard deviation(SD) resulting in the mean of the attributes becoming zero and the resultant distribution having a standard deviation (Abramo et al., 2012).

$$Z = (X-U) / S$$

In this paper, we use standard scaling from the sklearn library.

D. Min-Max normalization

Min-max normalization is used to rescale the data in simple terms the range of features is scaled from [0,1] or [-1,1] selecting the target variable depending on the data. By applying this method we are able to bring more accuracy to machine models. We import a min-max scalar from the sklearn library to perform the function.

E. Recursive feature Elimination

Recursive feature elimination(RFE) is a popular feature selection technique, it uses more efficient in selecting the feature in the dataset which are more relevant and contribute to delivering better accuracy. Another reason for using RFE is that it is also very easy to configure. In this paper, we used RFE from the sklearn library and performed in a pipeline

F. Principle Component Analysis

Principle component analysis(PCA) is based on linear algebra PCA can be performed for several reasons ns. the primary reason is when the learning algorithm is slow we can reduce the input value dimensions to decrease model latency. ("PCA using Python (scikit-learn)", 2022)

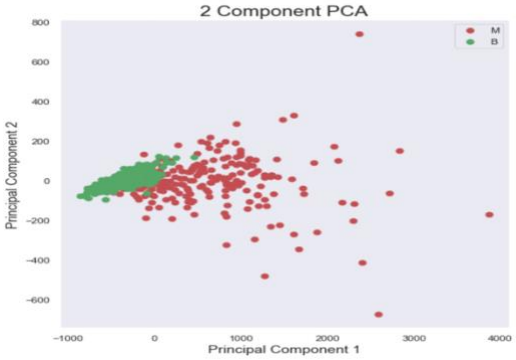


Figure 5 . visualization of PCA data

We have used PCA model pre-processing from the sklearn library to perform on the dataset.

G. Pipelines

In machine learning, the pipeline is defined as an end to end construct which has orchestrated the flow of the data. A pipeline can hold multiple transformers, estimators and models and we can fit out training data to find the output in a much faster way using a pipeline can be helpful to construct the code which is a much more readable format for a 3rd person. In this project, we have used pipelines to incorporate our previously discussed data pre-processing models and perform an evaluation

Note: Not all the above pre-processing techniques were used on all the models. Selective techniques were used to bring the best accuracy possible in a particular model.

4. MACHINE LEARNING CLASSIFICATION TECHNIQUES

A. Support Vector Machine

A support vector machine(SVM) is a supervised learning technique used for classification and regression problems. It belongs to a family of linear classification. A stand put the property of SVM is that it can reduce empirical classification error and increase geometrical margin it is also called a maximum margin classifier and SVM can plot input variables in a higher dimensional space where a maximum separation hyperplane is constructed. SVM consists of various kernels like linear, poly, RBF, sigmoid and precomputed. SVM kernels provide shortcuts to avoid complex calculations higher dimensions can be given to have a much smoother calculation in the model. C parameter helps the SVM model to be optimized to reduce misclassification for each training example ("Breast Cancer Classification using Support Vector Machine and Neural Network", 2016) Figure 4 depicts how an SVM can classify a binary classification problem.

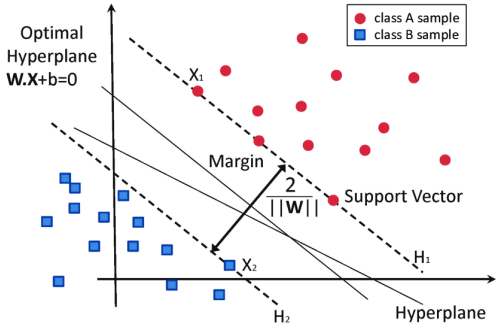


Figure 6. SVM plotting example

B. Random Forest classifier

Random forest classification is a type of classification algorithm which is made up of several decision trees it uses randomness to build up each tree to perform uncorrelated forests, which then uses forest prediction to come up with a more accurate conclusion. The random forest has almost similar hyperparameters as the decision tree and it is easy to configure ("A Complete Guide to the Random Forest Algorithm", 2022)

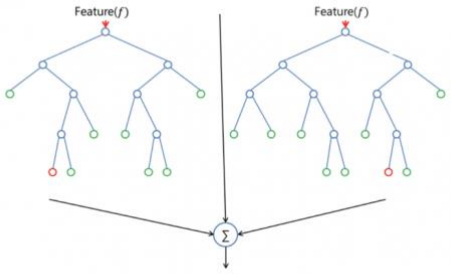


Figure 7. Random Forest Classifier

C. Stochastic Gradient Descent

Stochastic gradient descent is a widely used machine learning classifier, it

is a simple yet very effective classifier. It can be used in classification problems like binary classification, and text classification. In this paper, we use one of the powerful classifiers from sklearn. library to compare with our previously discussed methods.

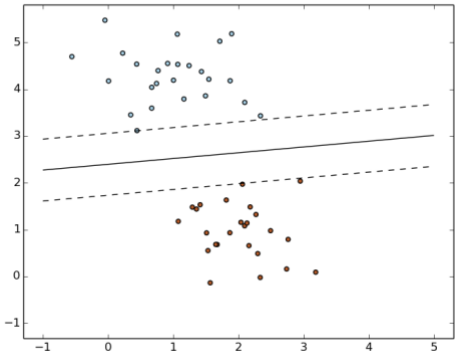


Figure 8. Stochastic Gradient Descent

D. Multilayer perception

The name multilayer perception (MLP) itself connects to the neural network, unlike other classifiers MLP uses neural networks to classify the data. We know MLP can perform better than other classifiers discussed in the paper. In this paper, we intend to match the level of MLP accuracy from our previous model by tuning our models to perform well. The codes used for MLP are taken from sklearn library.

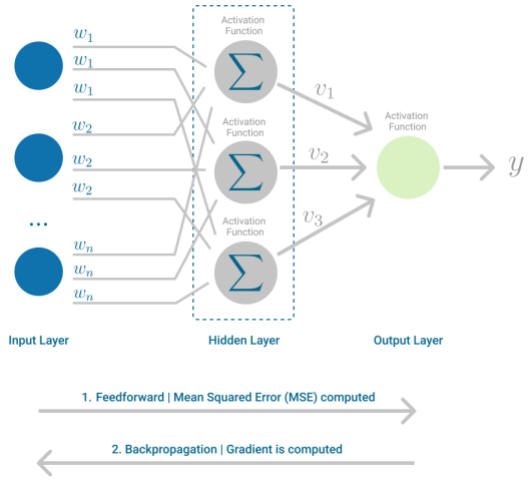


Figure 9. inside MLP classifier

5. Application of Techniques and Results

Algorithms were implemented and results were obtained from an M1 MacBook pro using jupyter notebook.
The models have trained primarily without any data pre-processing techniques and the results were noted the same models were used after data pre-processing the results were measured with various metrics and accuracy was tested with a confusion matrix.

Model evaluation metrics:

1) Model Accuracy

2) Model Precision

3) Macro average

4) Sensitivity / Recall

5) F1 Measure

6) Weighted average

8) Metrics calculations

A. Support Vector Machine

Using the SVM classifier from Sklearn and the results obtained in *fig. 10* and *Table. 3*

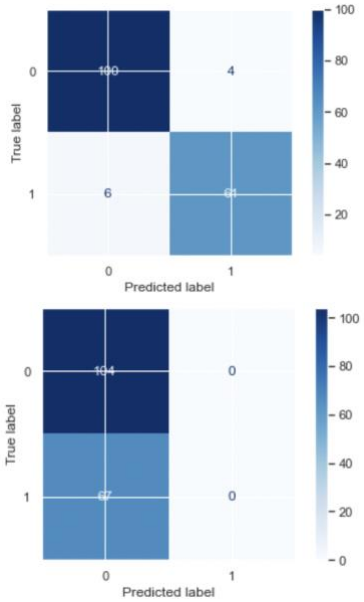


Fig 10 Confusion matrix pre and post-pre-processing

Fig 8 ROC curve for SVM

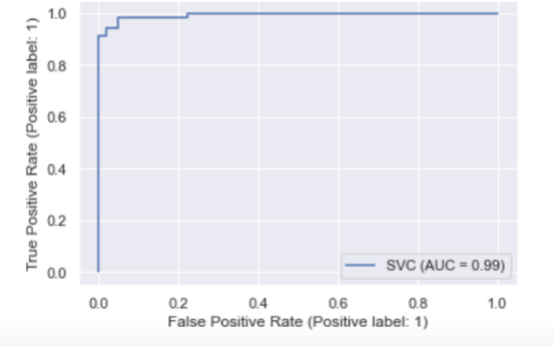


Table 3 Support vector-matrix result summary

Support Vector Machine		
Algorithm parameters		
C	2	2
Kernel	linear	sigmoid
Gamma	auto	auto
Results		
	Before	After
No of samples	357	252
Accuracy	0.9532	0.9766
precision	0.95	0.97
Macro avg	0.95	0.97
Weighted avg	0.95	0.96
F1 score	0.95	0.97
Support	171	171
Confusion matrix		
No of samples	357	252
True Positive	61	104
True Negative	100	0
False Positive	6	0
False Negative	4	67

The results show that SVM ran well with the sigmoid kernel and the pre-processing methods used in the SVM model are SMOTE, Feature selection and min-max scalar. It was evident that a 2% increase was shown in the model when after pre-processing the data Check *fig.8* and *Table 4*

B. Random Forest Classification

We have taken the codes from sklearn to show the model and the confusion matrix.

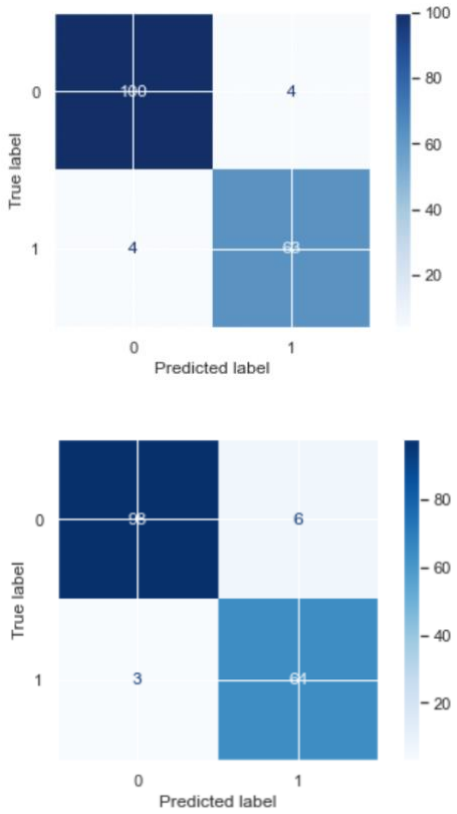


Figure 9 Confusion matrix for RF model

Random Forest		
Algorithm parameters		
Max Depth	2	4
Random state	0	2
Results		
	Before	After
No of samples	357	252
Accuracy	0.941	0.9473
Precision	0.95	0.96
Macro avg	0.94	0.97
Weighted avg	0.94	0.97
F1 score	0.94	0.95
Support	171	171
Confusion matrix		
No of samples	357	252
True Positive	63	61
True Negative	100	98
False Positive	4	3
False Negative	4	6

Table 4 Random Forest result summary

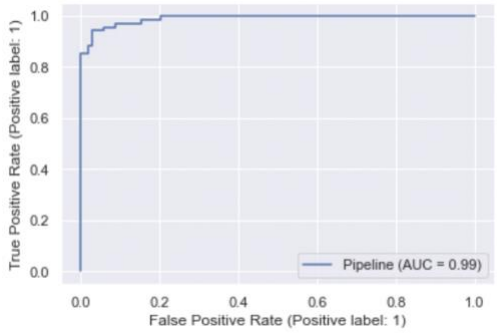
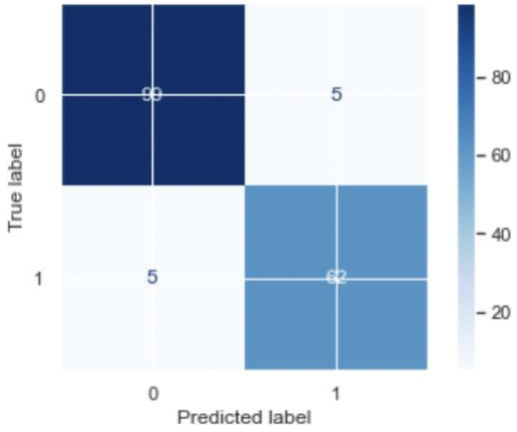


Figure 10. ROC curve for RF model

The results of the random forest classifier show minimal changes after applying data pre-processing. We have used feature scaling and min-max scalar to improve the results. The results clearly show random forest is a strong classification algorithm and we can achieve higher accuracy by tailoring the model.

C. Stochastic Gradient Descent

The algorithm used is taken from sklearn library the model results are shown in *fig 11* and *Table 5*



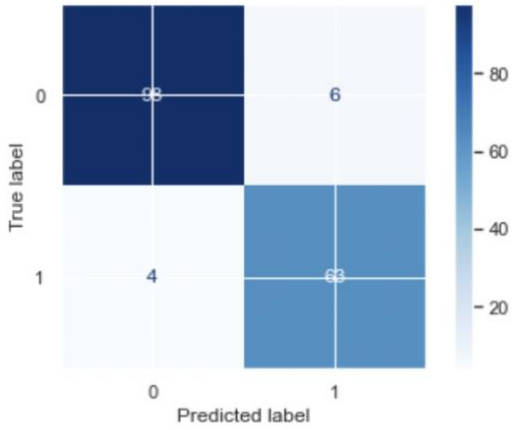


Figure 11 Confusion matrix for SGD

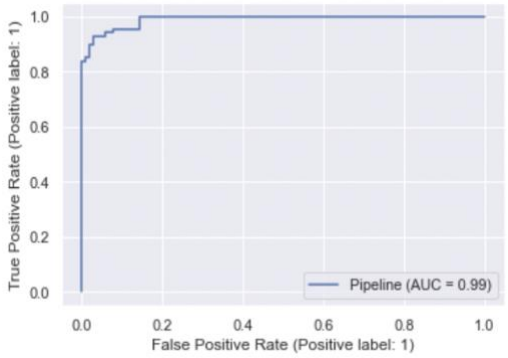


Figure 12 ROC curve for SGD

Stochastic Gradient Descent		
Algorithm parameters		
Loss	hinge	hinge
Penalty	12	12
Max_iter	5	5
Results		
	Before	After
No of samples	357	252
Accuracy	0.877	0.941
Precision	0.94	0.96
Macro avg	0.87	0.94
Weighted avg	0.88	0.92
F1 score	0.88	0.92
Support	171	171
Confusion matrix		
No of samples	357	252
True Positive	62	63
True Negative	99	98
False Positive	5	4
False Negative	5	6

Table 5 SGD result and summary

The results clearly indicate a significant increase in inaccuracy in the model after pre-processing the data. Implementation of feature selection, min-max scalar and PCA help the model to classify and give better results than before.

D. Multilayer perception

MLP is used in this paper is taken from sklearn library although this model belongs to neural networks we have implemented to compare our ML models. the results of the model are given in *fig 13* and *Table 6*

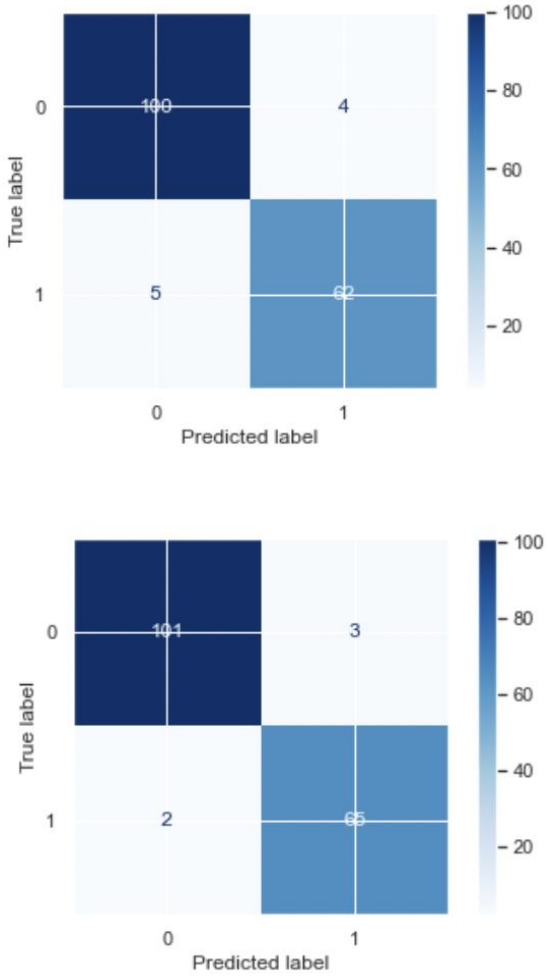


Figure 13 Confusion matrix for MLP

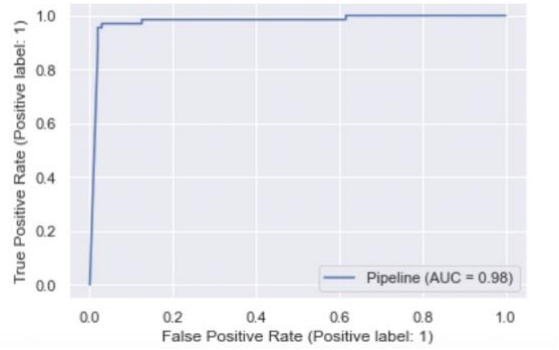


Figure 14 ROC curve for MLP model

Multilayer perception		
Algorithm parameters		
Solver	lbfgs	lbfgs
alpha	1e-5	1e-5
Hidden layer	(5,2)	(5,2)
Random state	1	1
Results		
	Before	After
No of samples	357	252
Accuracy	0.947	0.970
Precision	0.95	0.96
Macro avg	0.95	0.96
Weighted avg	095	0.96
F1 score	0.94	0.97
Support	171	171
Confusion matrix		
No of samples	357	252
True Positive	62	65
True Negative	100	101
False Positive	5	2
False Negative	4	3

Table 6 MLP results and summary

It is quite clear that MLP can give better results than other models but we have tuned our SVM to perform as good as the MLP model. The data pre-processing techniques used in this model are standard scalar and feature selection to obtain the above results.

5. DISCUSSION

In this paper, we have applied different machine learning models to classify breast cancer diagnosis although there are several

studies related to this subject the attempt is to bring out the best model possible to classify breast cancer. It was found Support vector machine and Multilayer perception has the best accuracy when the data provided to them are well pre-processed. It was also found that feature selection and SMOTE have parlayed a major role in improving the model accuracy. Compression of the model used above is given as a graph in *fig 15*

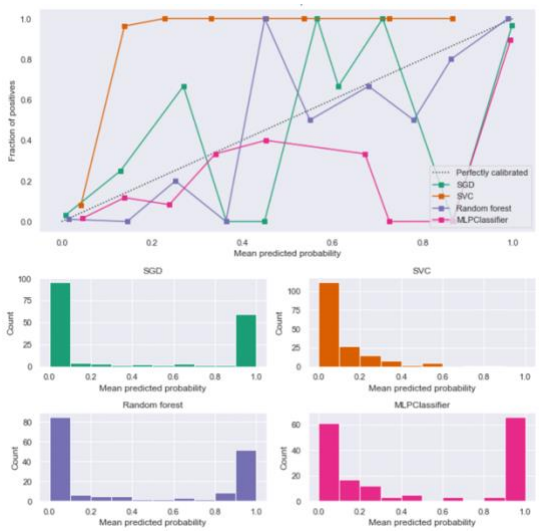


Figure 15. Comparison of models

6. FUTURE RESEARCH

All our models perform well but the research can be improved by using deep neural networks (DNN) and creating a model which is capable of handling missing variables and giving much higher accuracy. DNN in breast cancer can also be used to study images and recognise features where the model will not be subjected to human errors, and this will be beneficial to doctors in cancer diagnosis.

7. REFERENCES

El Agouri, H., Azizi, M., El Attar, H., El Khannoussi, M., Ibrahimi, A., Kabbaj, R., Kadiri, H., BekarSabein, S., EchCharif, S., Mounjid, C., & El Khannoussi, B. (2022). Assessment of deep learning algorithms to predict histopathological diagnosis of breast cancer: First moroccan prospective study on a private dataset. *BMC Research Notes*, *15*(1), 66. https://doi.org/10.1186/s13104-022-05936-1

Park, K. W., Kim, S. W., Han, H., Park, M., Han, B., Ko, E. Y., Choi, J. S., Cho, E. Y., Cho, S. Y., & Ko, E. S. (2022). Ductal carcinoma in situ: A risk prediction model for the underestimation of invasive breast cancer.*NPJ Breast Cancer*, *8*(1), 8. https://doi.org/10.1038/s41523-021-00364-z

Singh, H., Sharma, V., & Singh, D. (2022). Comparative analysis of proficiencies of various textures and geometric features in breast mass classification using k-nearest neighbor. *Visual Computing for Industry, Biomedicine and Art*, *5*(1), 3. https://doi.org/10.1186/s42492-021-00100-1

Srwa Hasan Abdulla, Ali Makki Sagheer, & Hadi Veisi. (2021). Breast cancer classification using machine learning techniques: A review. *Turkish Journal of Computer and Mathematics Education*, *12*(14), 1970-1979. https://search.proquest.com/docview/2623927246

Vaka, A. R., Soni, B., & K, S. R. (2020). Breast cancer detection by leveraging machine learning. *ICT Express*, *6(4)*, *6*(4), 320-324. https://doi.org/10.1016/j.icte.2020.04.009

learning. *ICT Express*, *6(4)*, *6*(4), 320-324. https://doi.org/10.1016/j.icte.2020.04.009

Breast Cancer Wisconsin (Diagnostic) Data Set. Kaggle.com. (2022). Retrieved 13 April 2022, from https://www.kaggle.com/datasets/uciml/breast-cancer-wisconsin-data.

Johnson, J., & Khoshgoftaar, T. (2019). Survey on deep learning with class imbalance. *Journal Of Big Data*, *6*(1). https://doi.org/10.1186/s40537-019-0192-5

Abramo, G., Cicero, T., & D'Angelo, C. (2012). How important is choice of the scaling factor in standardizing citations?. *Journal Of Informetrics*, *6*(4), 645-654. https://doi.org/10.1016/j.joi.2012.07.002

Breast Cancer Classification using Support Vector Machine and Neural Network. (2016), *5*(3), 1-6. https://doi.org/10.21275/v5i3.nov161719

A Complete Guide to the Random Forest Algorithm. Built In. (2022). Retrieved 15 April 2022, from https://builtin.com/data-science/random-forest-algorithm.

APPENDIX :

Dataset Link : https://www.kaggle.com/datasets/uciml/breast-cancer-wisconsin-data

Codes for machine learning model and data visualization

Data Importation and data cleaning:

```
import pandas as pd
import numpy as np

df=pd.read_csv("data.csv")

df.head()

df.isnull().sum()

df.dtypes

df['radius_mean'] = df.radius_mean.astype(int)

df.dtypes
df.shape
dummy =pd.get_dummies(df['diagnosis'])
dummy.head()
df2 = pd.concat((df, dummy),axis=1)
df2.drop(['diagnosis'],axis=1)
df2=df2.drop(['Unnamed: 32'],axis=1)
df2=df2.drop(['B'],axis=1)
df2['diagnosis'].value_counts()
```

Basic classification plot:

```
malignant_df = df2[df2['M']==0][0:500]
benign_df = df2[df2['M']== 1][0:500]
fig_dims=(10,10)
axes = benign_df.plot(kind='scatter', x='radius_worst',y = 'texture_worst',color =
"red",label ='benign',figsize=fig_dims )
malignant_df.plot(kind='scatter', x='radius_worst',y = 'texture_worst',color =
"blue",label ='malignant',ax=axes )
```

Train Test Split

```
feature_df = df2[['radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst']]
#cell_data has 100 rows
# -2 colums id and diagonsis

# independent varibel
x= np.asarray(feature_df)

# dependent varible
```

```
y = np.asarray(df2['M'])

# train and test

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3, random_state
= 2)
```

ML Without Pre Processing:

```
#RF
clf_rf = RandomForestClassifier(max_depth=2, random_state=0)
clf_rf.fit(x_train,y_train)
y_pred=clf_rf.predict(x_test)
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print(classification_report(y_test,y_pred))


from collections import Counter
from sklearn.datasets import make_classification
from imblearn.over_sampling import RandomOverSampler

from sklearn import svm

classifier_svm=svm.SVC(kernel='linear',gamma = 'auto', C=2)
classifier_svm.fit(x_train,y_train)
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print(classification_report(y_test,y_pred))

from sklearn import metrics

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

from sklearn.metrics import classification_report


print(classification_report(y_test,y_pred))

pred_y_df=pd.DataFrame({'Actual Value':y_test,'Predicted
value':y_pred,'Difference': y_test-y_pred })

pred_y_df[0:10]

import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.metrics import confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import seaborn as sns
sns.set()


knn = KNeighborsClassifier(n_neighbors=5, metric='euclidean')
knn.fit(x_train, y_train)

y_pred = knn.predict(x_test)
confusion_matrix(y_test, y_pred)

from sklearn.metrics import classification_report
```

```
print(classification_report(y_test,y_pred))

from sklearn import metrics

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

import pandas as pd
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
from sklearn.model_selection import train_test_split # Import train_test_split
function
from sklearn import metrics #Import scikit-learn metrics module for accuracy calc

# Create Decision Tree classifer object
clf = DecisionTreeClassifier()

# Train Decision Tree Classifer
clf = clf.fit(x_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(x_test)

print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

print(classification_report(y_test,y_pred))

#naive_bayes
from sklearn.naive_bayes import GaussianNB

#Create a Gaussian Classifier
gnb = GaussianNB()

#Train the model using the training sets
gnb.fit(x_train, y_train)

#Predict the response for test dataset
y_pred = gnb.predict(x_test)

from sklearn import metrics

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))

#Stochastic Gradient Descent
from sklearn.linear_model import SGDClassifier
clf = SGDClassifier(loss="hinge", penalty="l2", max_iter=5)
clf.fit(x_train, y_train)

y_pred=clf.predict(x_test)

from sklearn import metrics

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
#Neural network models
from sklearn.neural_network import MLPClassifier
y_pred= clf_mlp.predict(x_test)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
clf_rf = RandomForestClassifier(max_depth=2, random_state=0)
clf_rf.fit(x, y)
y_pred= clf_rf.predict(x_test)

print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

ML Prediction Post Pre Processing:

```
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import VarianceThreshold
from sklearn.svm import LinearSVC
from sklearn.feature_selection import SelectKBest
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
from sklearn.preprocessing import PowerTransformer
from sklearn.feature_selection import RFECV
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import BaggingRegressor
from imblearn.over_sampling import SMOTE
from sklearn.metrics import f1_score

sm = SMOTE(random_state=42)

x_res, y_res = sm.fit_resample(x_train,y_train)
print('Resampled dataset shape %s' % Counter(y_res))

X1 = pd.get_dummies(df).values
y1 = df['diagnosis']

pd.value_counts(y1).plot.bar(color = "red")
plt.title('Before SMOT')
plt.xlabel('Class')
plt.ylabel('Frequency')
y1.value_counts()

X1 = x_res #pd.get_dummies(df).values
y1 = y_res #df['diagnosis']

pd.value_counts(y1).plot.bar(color = 'g')
plt.title('SMOT')
plt.xlabel('Class')
plt.ylabel('Frequency')

scaling = MinMaxScaler()
scaling.fit_transform(df[['radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst']])

df2.var()
import seaborn as sn
import matplotlib.pyplot as plt
fig_dims=(15,10)
fig, ax = plt.subplots(figsize=fig_dims)

sn.heatmap(df2.corr(),ax=ax)
```

```
plt.show()
abs(df2.corr()['radius_mean'])
```

Model Pipeline :

```
from sklearn.svm import SVC
svc = SVC(kernel="linear")

pipeline_svm = Pipeline([#('scalar1',StandardScaler()),

('Feature_selection',RFECV(estimator=svc,step=1,scoring="accuracy")),
                         ('scalar8',MinMaxScaler()),
                         #('pca1', PCA(n_components=2)),
                         ('SVM',svm.SVC(kernel='sigmoid',gamma = 'auto', C=2))])

pipeline_MLP =Pipeline([('scalar2',StandardScaler()),
                         ('Feature_selection',RFECV(estimator=
svc,step=1,scoring="accuracy")),
                         #('scalar2',MinMaxScaler()),
                         #('pca2', PCA(n_components=2)),
                         ("MLP", MLPClassifier(solver='lbfgs', alpha=1e-5,
hidden_layer_sizes=(5, 2), random_state=1))])

pipeline_SGD =Pipeline([#('scalar3',StandardScaler()),

('Feature_selection',RFECV(estimator=svc,step=1,scoring="accuracy")),
                         #('scalar3',PowerTransformer(standardize=True)),
                         ('scalar2',MinMaxScaler()),
                         ('pca3', PCA(n_components=2)),
                         ("SGB",SGDClassifier(loss="hinge", penalty="l2",
max_iter=5))])

pipline_RF = Pipeline([

('Feature_selection',RFECV(estimator=svc,step=1,scoring="accuracy")),
                         ('scalar2',MinMaxScaler()),
                         #('pca2', PCA(n_components=2)),
                         ("RF",RandomForestClassifier(max_depth=4,
random_state=2))])

pipelines = [pipeline_svm, pipeline_MLP, pipeline_SGD, pipline_RF ]

pipe_dict ={0: 'SVM', 1: 'MLP', 2: 'SGB', 3:'RF'}

for pipe in pipelines:
    pipe.fit(x_res, y_res)

for i,model in enumerate(pipelines):
    print("{} Test Accuracy: {}". format(pipe_dict[i],model.score(x_test,y_test)))
```

Confusion Matrix And ROC Plot:

```
#SVM
plot_confusion_matrix(clf, x_test, y_test,  cmap= plt.cm.Blues)
plt.show()


plot_confusion_matrix(pipeline_svm, x_test,y_test, cmap= plt.cm.Blues)

plt.show()


plot_confusion_matrix(clf_mlp, x_test, y_test,  cmap= plt.cm.Blues)
plt.show()
```

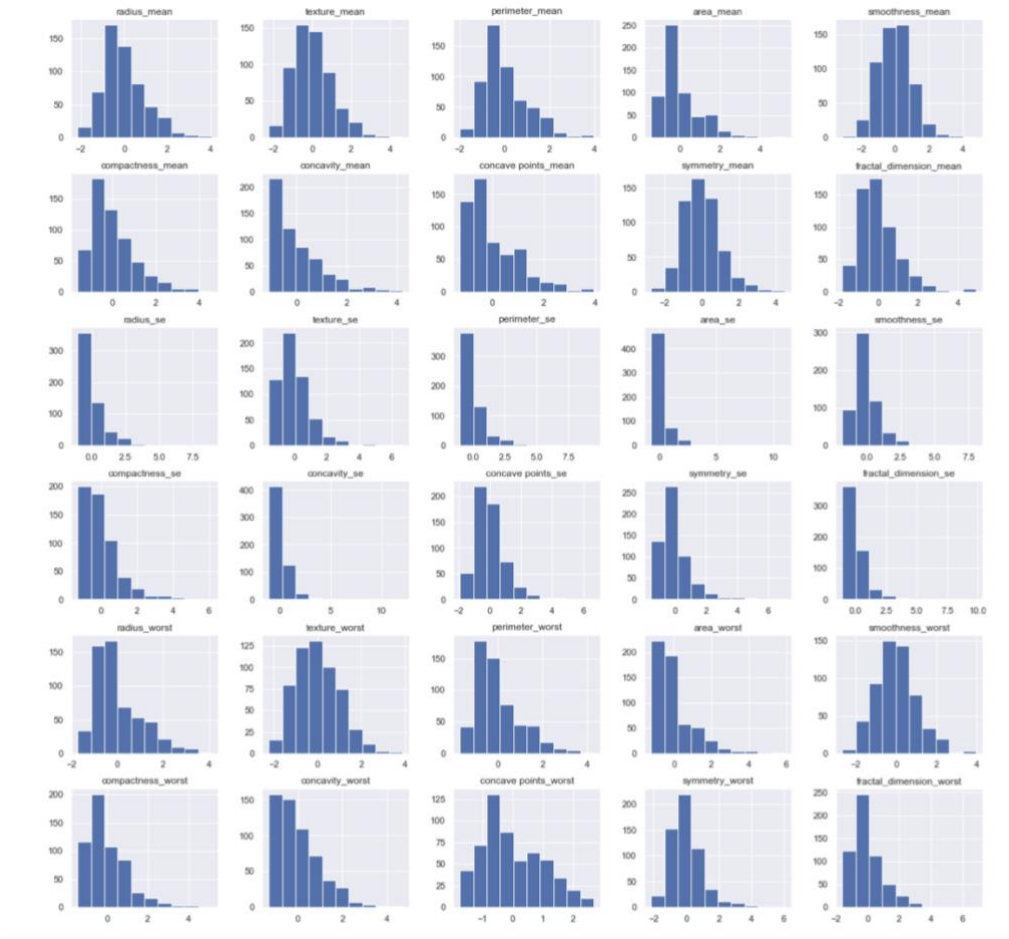
```
plot_confusion_matrix(pipeline_MLP, x_test, y_test,  cmap= plt.cm.Blues)
plt.show()

plot_confusion_matrix(gnb, x_test, y_test,  cmap= plt.cm.Blues)
plt.show()

plot_confusion_matrix(pipeline_SGD, x_test, y_test,  cmap= plt.cm.Blues)
plt.show()


plot_confusion_matrix(clf_rf, x_test, y_test,  cmap= plt.cm.Blues)
plt.show()

plot_confusion_matrix(pipline_RF, x_test, y_test,  cmap= plt.cm.Blues)
plt.show()

from sklearn.metrics import RocCurveDisplay
from sklearn.ensemble import RandomForestClassifier

ax = plt.gca()
rfc_disp = RocCurveDisplay.from_estimator(classifier_svm, x_test, y_test, ax=ax,
alpha=0.8)
svc_disp = RocCurveDisplay.from_estimator(classifier_svm, x_test, y_test)

svc_disp.plot(ax=ax, alpha=0.8)

import numpy as np

from sklearn.svm import LinearSVC


class NaivelyCalibratedLinearSVC(LinearSVC):
    """LinearSVC with `predict_proba` method that naively scales
    `decision_function` output."""

    def fit(self, X, y):
        super().fit(X, y)
        df = self.decision_function(X)
        self.df_min_ = df.min()
        self.df_max_ = df.max()

    def predict_proba(self, X):
        """Min-max scale output of `decision_function` to [0,1]."""
        df = self.decision_function(X)
        calibrated_df = (df - self.df_min_) / (self.df_max_ - self.df_min_)
        proba_pos_class = np.clip(calibrated_df, 0, 1)
        proba_neg_class = 1 - proba_pos_class
        proba = np.c_[proba_neg_class, proba_pos_class]
        return proba

from sklearn.calibration import CalibrationDisplay
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier

# Create classifiers
#sgd = SGDClassifier(loss="hinge", penalty="l2", max_iter=5,"Naive Bayes")
lr = LogisticRegression()
gnb = GaussianNB()
svc = NaivelyCalibratedLinearSVC(C=1.0)
rfc = RandomForestClassifier()
MPL = MLPClassifier()
clf_list = [
    (lr, "SGD"),
    #(SGDClassifier(loss="hinge", penalty="l2", max_iter=5), "Naive Bayes"),
    #(sgd, "SGD"),
```

```
    (svc, "SVC"),
    (rfc, "Random forest"),
    (MPL, "MLPClassifier")
]

import matplotlib.pyplot as plt
from matplotlib.gridspec import GridSpec

fig = plt.figure(figsize=(10, 10))
gs = GridSpec(4, 2)
colors = plt.cm.get_cmap("Dark2")

ax_calibration_curve = fig.add_subplot(gs[:2, :2])
calibration_displays = {}
for i, (clf, name) in enumerate(clf_list):
    clf.fit(x_train, y_train)
    display = CalibrationDisplay.from_estimator(
        clf,
        x_test,
        y_test,
        n_bins=10,
        name=name,
        ax=ax_calibration_curve,
        color=colors(i),
    )
    calibration_displays[name] = display

ax_calibration_curve.grid()
ax_calibration_curve.set_title("Calibration plots")

# Add histogram
grid_positions = [(2, 0), (2, 1), (3, 0), (3, 1)]
for i, (_, name) in enumerate(clf_list):
    row, col = grid_positions[i]
    ax = fig.add_subplot(gs[row, col])

    ax.hist(
        calibration_displays[name].y_prob,
        range=(0, 1),
        bins=10,
        label=name,
        color=colors(i),
    )
    ax.set(title=name, xlabel="Mean predicted probability", ylabel="Count")

plt.tight_layout()
plt.show()

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

pca = PCA(n_components=2)

principalComponents = pca.fit_transform(x)

principalDf = pd.DataFrame(data = principalComponents
             , columns = ['principal component 1', 'principal component 2'])

principalDf.head(5)

finalDf = pd.concat([principalDf, df[['diagnosis']]], axis = 1)
finalDf.head(5)

fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 Component PCA', fontsize = 20)
```

```
targets = ['M', 'B']
colors = ['r', 'g', 'b']
for target, color in zip(targets,colors):
    indicesToKeep = finalDf['diagnosis'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
               , finalDf.loc[indicesToKeep, 'principal component 2']
               , c = color
               , s = 50)
ax.legend(targets)
ax.grid()

pca.explained_variance_ratio_
```

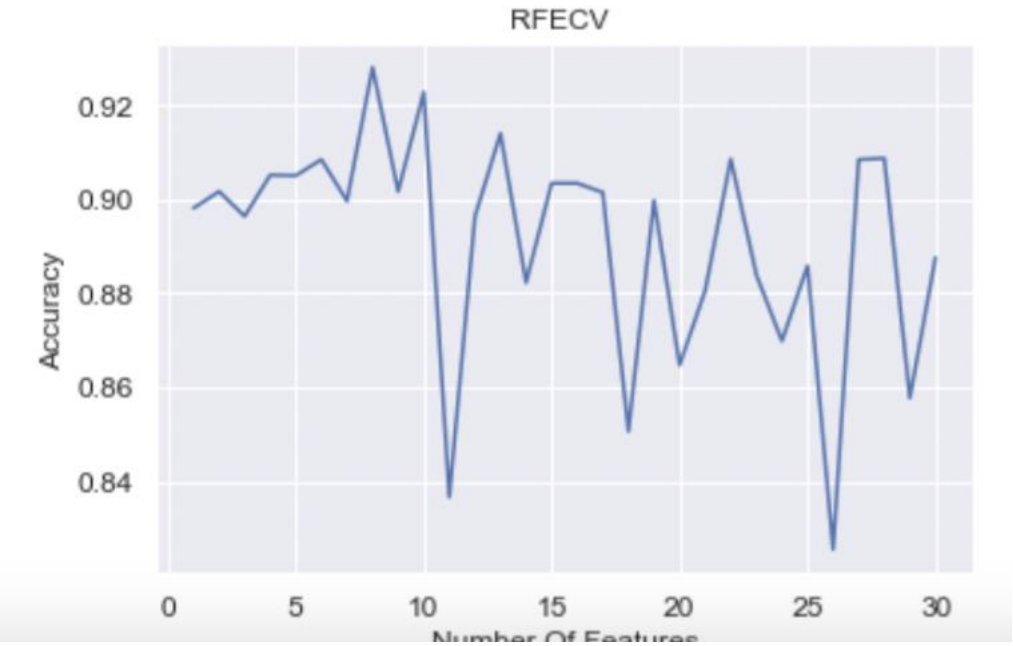
Normalizing Data And Plot:

```
X=df.copy()
y=X.pop('diagnosis')

X.drop(['id','Unnamed: 32'],axis=1,inplace=True)

X_norm=(X - X.mean()) / (X.std())

X_norm=X_norm.join(y)

X_norm.hist(figsize=(20,20))
plt.show()
```



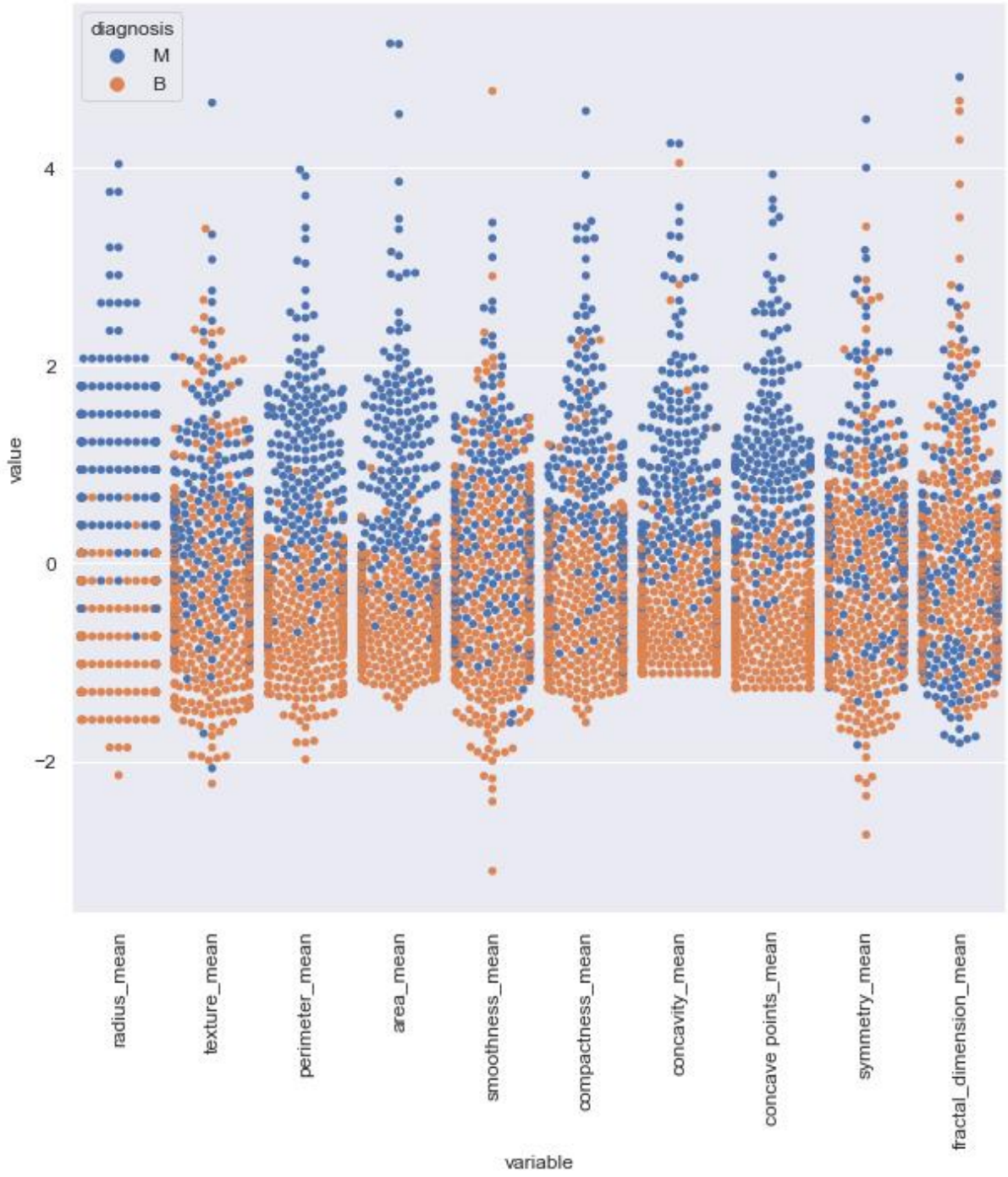
Feature selection(RFEC) :

```
from sklearn.feature_selection import RFECV
rfecv=RFECV(estimator=svc,step=1,cv=10,scoring='accuracy')
#since it uses cross validation, we should fit it to the whole dataset
rfecv.fit(X,y)

list(X.columns[rfecv.support_])

plt.title("RFECV ")
plt.xlabel("Number Of Features")
plt.ylabel("Accuracy")
plt.plot(range(1, len(rfecv.cv_results_['mean_test_score']) + 1),
rfecv.cv_results_['mean_test_score'])
plt.show()
```



Swarm Plotting Diagnosi:

```
plt.rc('xtick', labelsize=12)    # fontsize of the tick labels
plt.rc('ytick', labelsize=12)
plt.rc('axes', labelsize=12) # fontsize of the x and y labels
plt.rc('legend', fontsize=12)    # legend fontsize
swarm(X_mlt1)
```


diagnosis
M
B
value
4
2
0
−2
radius_mean
texture_mean
perimeter_mean
area_mean
smoothness_mean
compactness_mean
concavity_mean
concave points_mean
symmetry_mean
fractal_dimension_mean
variable