Report on

Machine Learning Model for Prognosis of Breast Cancers

Developed by **Yasiru Makavita** for the partial fulfillment of the capstone project of the Machine learning foundation by Data Science Academy

2021.11.25

Contents

1.0 Dataset 3

1.1 Attributes 3

1.2 Possible Predictions 4

2.0 Methodology 5

2.1 Platform 5

2.2 Libraries 5

2.3 Creating the dataframe 5

2.4 Data exploration 5

2.5 Balancing the dataset 9

2.6 Feature selection 9

2.7 Model development 12

2.7.1 Train Test Split 12

2.7.2 Model building 12

2.7.3 Best model selection 12

2.7.4 Tuning the best model 13

2.7.5 Scoring the model 14

2.7.6 Saving model 14

3.0 Discussion and conclusions 15

3.1 summery 16

Abbreviations

M	- Mean
SE	- Standard Error
ML	- Machine Learning
TS	- Tumor Size
LNS	- Lymph Node Status

1.0 Dataset

Dataset name : Wisconsin Prognostic Breast Cancer Dataset

Repository: UCI machine learning repository

Data Set Characteristics:	Multivariate	**Number of Instances:**	198
Attribute Characteristics:	Real	**Number of Attributes:**	35
Associated Tasks:	Classification, Regression	**Missing Values?**	Yes

This dataset was made using the data collected by Dr. Wolberg on consecutive patients seen by him since 1984. This only includes records of patients who were cases exhibiting invasive breast cancer and no evidence of distant metastases at the time of diagnosis.

1.1 Attributes

1. ID Number - ID number of the patient
2. Outcome - (R = recur, N = non-recur) This means if the cancer had recurred or not.
3. Time - recurrence time if Outcome = R, disease-free time if Outcome = N
4. – 33. - these 30 features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image.

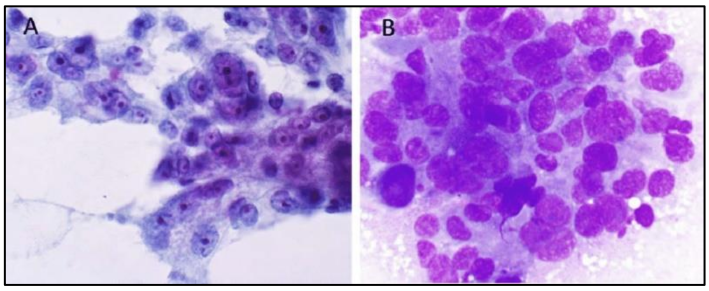


Figure 1: Samples of cancerous cells

Following 10 features of the cell nuclei are given in the dataset.

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)

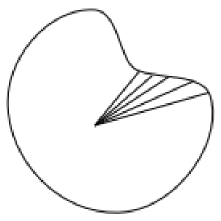


Figure 5: Line segments used to calculate the radius

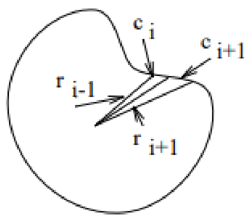


Figure 5:Line segments used to calculate the Smoothness

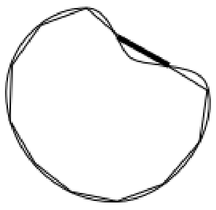


Figure 5: Line segments used to compute Concavity and concave points

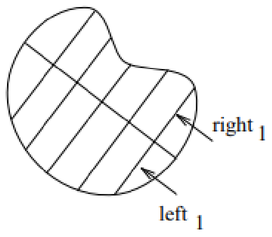


Figure 5: Line segments used to compute symmetry

The mean, standard error, and "worst" or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features. For example, field 4 is Mean Radius, field 14 is Radius SE, field 24 is Worst Radius. All 30 features are recorded in 4 digits.

34. Tumor size	- diameter of the excised tumor in centimeters
35. Lymph node status	- number of positive axillary lymph nodes observed at time of surgery

1.2 Possible Predictions

Two possible predictions can be done using this dataset

a) Predicting **Outcome** (field 2) - **binary classification** model can be trained to predict if the outcome is Recurrent (R) or non-recurrent (N).

b) Predicting **Time** to Recur (field 3) – if the outcome is predicted as 'Recurrent', then a **regression model** can be trained to predict the time for recurrence.

As for this project, only the 'Outcome' prediction was done.

2.0 Methodology

2.1 Platform

The ‘Googles Colab’ python development environment was used for coding this ML project.

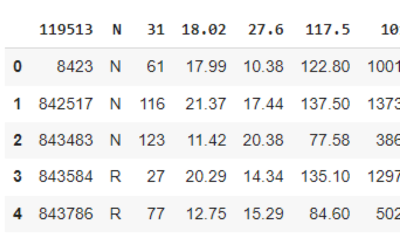
2.2 Libraries

- Pandas – This library was used for loading, analyzing and manipulation of data.
- NumPy – Used to perform mathematical operations (on arrays and matrices).
- Matplotlib – Extension of NumPy used for plotting
- Seaborn – Used for data visualization
- Scikit-learn - Library used for machine learning. Includes ML models and other necessary functions and tools.

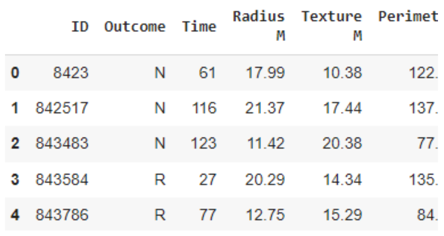
2.3 Creating the dataframe

Dataset was imported into the Colab environment directly from the UCI ML repo using the Pands read_csv feature. But the dataframe had no column names. The ‘column_header’ list was created by including the column names extracted from readme file in the repo.

“ Data.columns = column_headers “ was used to assign the column names to the dataframe.



	119513	N	31	18.02	27.6	117.5	10
0	8423	N	61	17.99	10.38	122.80	1001
1	842517	N	116	21.37	17.44	137.50	1373
2	843483	N	123	11.42	20.38	77.58	386
3	843584	R	27	20.29	14.34	135.10	1297
4	843786	R	77	12.75	15.29	84.60	502

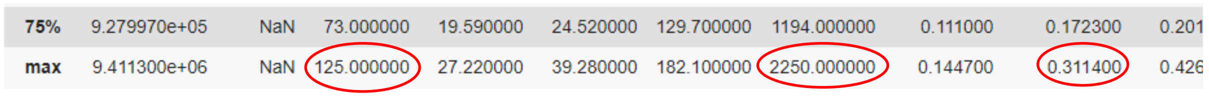


	ID	Outcome	Time	Radius M	Texture M	Perimet
0	8423	N	61	17.99	10.38	122.
1	842517	N	116	21.37	17.44	137.
2	843483	N	123	11.42	20.38	77.
3	843584	R	27	20.29	14.34	135.
4	843786	R	77	12.75	15.29	84.

Figure 6: Dataframe before & after labeling

2.4 Data exploration

- Exploration started by checking the dimensions of the dataframe. The observation was that it contained 35 columns and 197 rows. This matched with the dataset description confirmation that all data has been imported.
- Next the data types were inspected. ID & Time were integers (int64), Outcome & LNS were objects and all other attributes were floating points (float64).
- Summery of the dataframe was taken using describe (). It was observed that data are spread over different scales.



75%	9.279970e+05	NaN	73.000000	19.590000	24.520000	129.700000	1194.000000	0.111000	0.172300	0.201
max	9.411300e+06	NaN	125.000000	27.220000	39.280000	182.100000	2250.000000	0.144700	0.311400	0.426

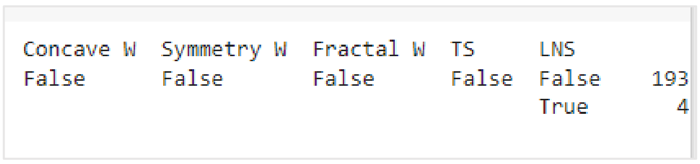
Figure 7: Part from dataframe description

- Next the Outcome column was inspected. Data[‘outcome’].value_counts() was used for this. It was observed that there were 150 Non-recurrence cases and only 47 Recurrence cases.

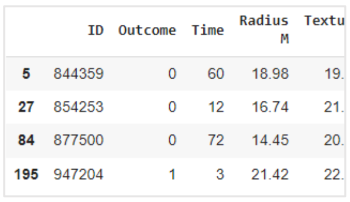
Therefore, it can be said that data set in imbalanced. Recurrence cases are only close to 25% of the total dataset.

- Before plotting the distribution of values under each attribute, the dataset was checked for any missing values. In this dataset, missing values were indicated in ‘?’.

data.isin(['?']).value_counts() was used to check where and how many missing values are there. 4 missing values were found in LNS. data[data.values == '?'] was used to identify the rows with missing data.



```
Concave W  Symmetry W  Fractal W  TS     LNS
False      False       False      False  False    193
                                         True       4
```



	ID	Outcome	Time	Radius M	Textu
5	844359	0	60	18.98	19.
27	854253	0	12	16.74	21.
84	877500	0	72	14.45	20.
195	947204	1	3	21.42	22.

Figure 8: Missing values

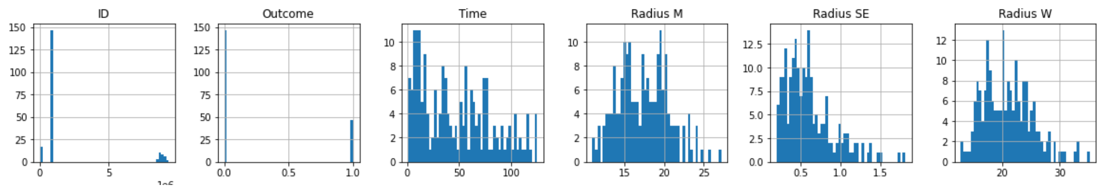
Only 1 of the minority class was having a missing value. It was decided to remove all the rows with missing values since there was no significant damage to the minority class. Other option was to remove the majority class rows with missing values and replace the remaining on with the mean or the mode [0]. This was left out to be tried during future developments.

Since it was only 4 columns, index numbers were used to select to rows to drop. Dropping was done using data.drop(labels=[5, 27, 84, 195], axis=0, inplace=True).

- LNS column was also classified as an object because it contained ‘?’ symbol. Even after removing the missing values, LNS was classified was an object type. Therefore, it was also converted into a numerical value using, data['LNS'] = pd.to_numeric(data['LNS'])

- The outcome column was the type ‘Object’ [categorical]. It was converted into a numerical variable using, data['Outcome'] = data['Outcome'].replace({"N": 0, "R": 1}).

- Next, histograms were used to observed how the values have spread within each attribute. data.hist(bins=50, figsize=(20, 20)) was used for plotting the graphs.



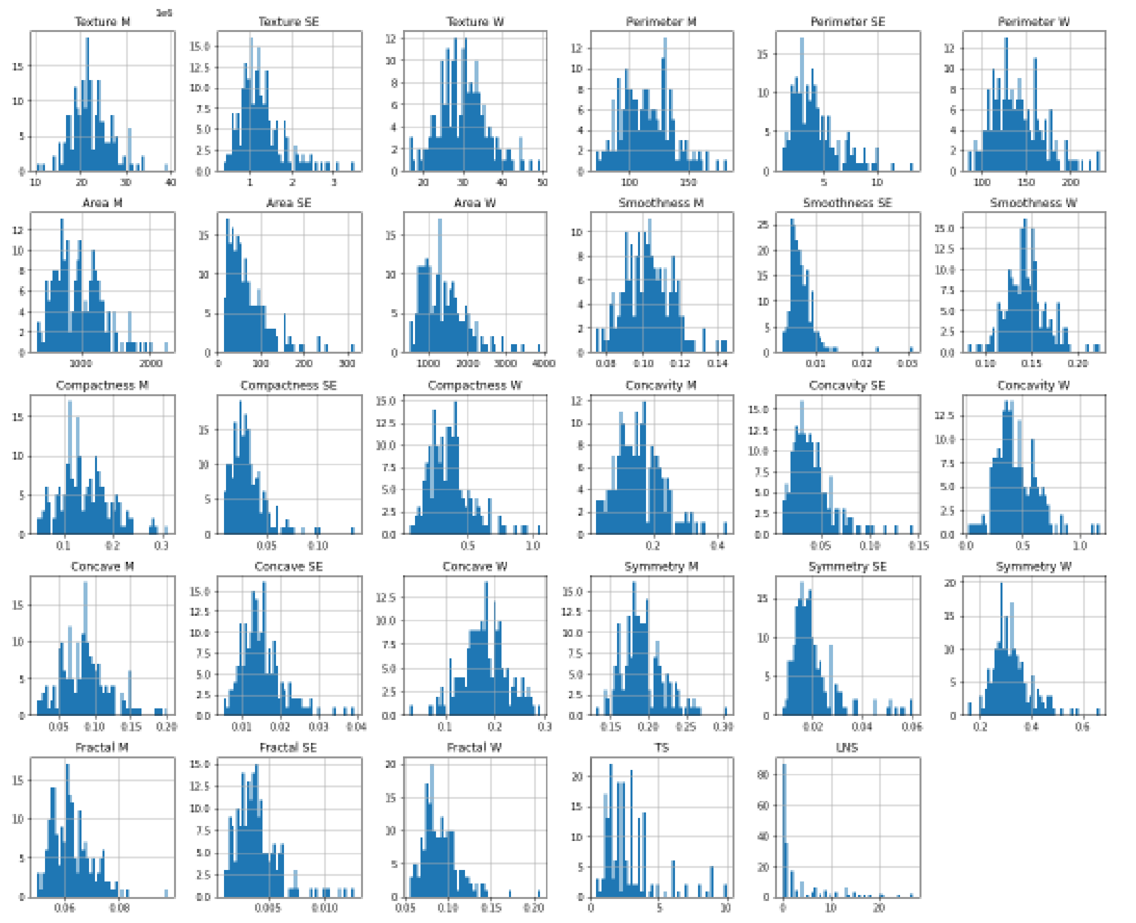


Figure 9: Histograms

It was observed that almost all the distributions are skewed. Only few like mean texture and worst concavity are close to being symmetrical. To check for outliers, box plots were used. Since attributes are not in the same scale, they had to be plotted in several groups.

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f02bd1a4f10>
```

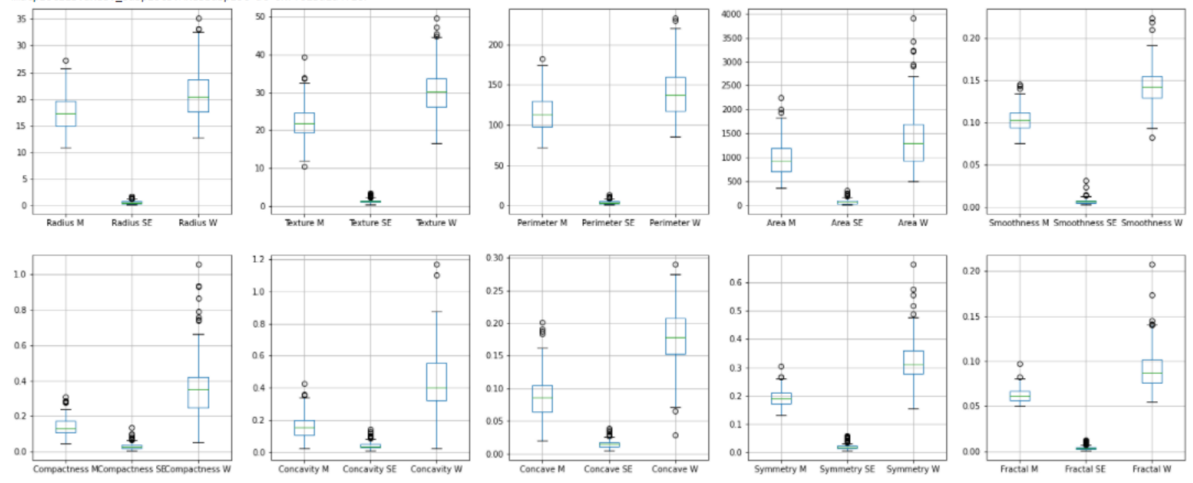
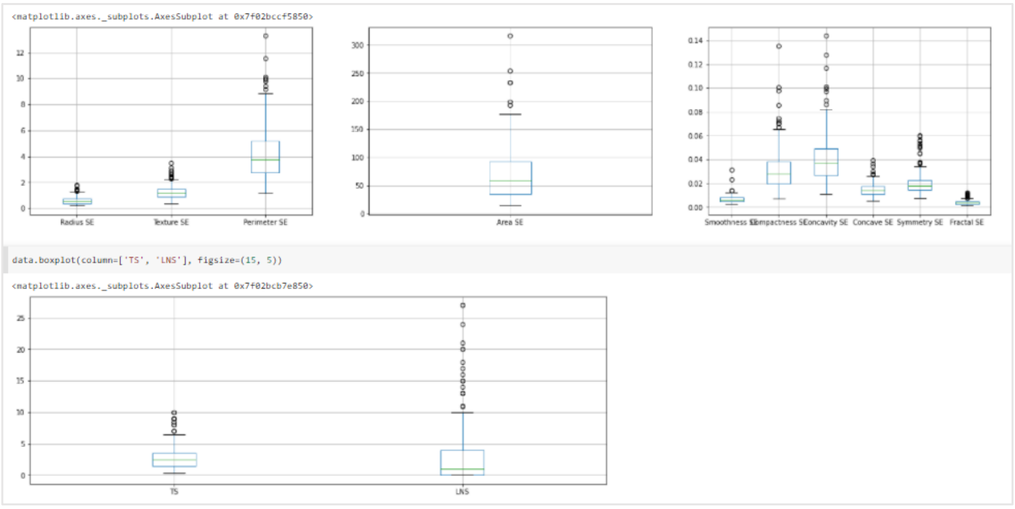


Figure 10: Box plots

Standard error (SE), TS and LNS were again plotted separately.



```
<matplotlib.axes._subplots.AxesSubplot at 0x7f02bccf5850>
```

```
data.boxplot(column=['TS', 'LNS'], figsize=(15, 5))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f02bcb7e850>
```

Figure 11: Boxplot – extended

These plots show that all the attributes have outliers. The 'mean value' columns have the least number of outliers and 'standard error' columns have the most number of outliers. LNS has the largest set of outliers.

It can be seen in most cases that the outliers are clustered closer to the upper whisker of the box plot. But Worst - area, compactness, symmetry, fractal & SE – parameter, area, compactness, concavity & LNS has outliers spread far from the upper whisker.

Due to the nature of the dataset, it is hard to decide how to treat these outliers.

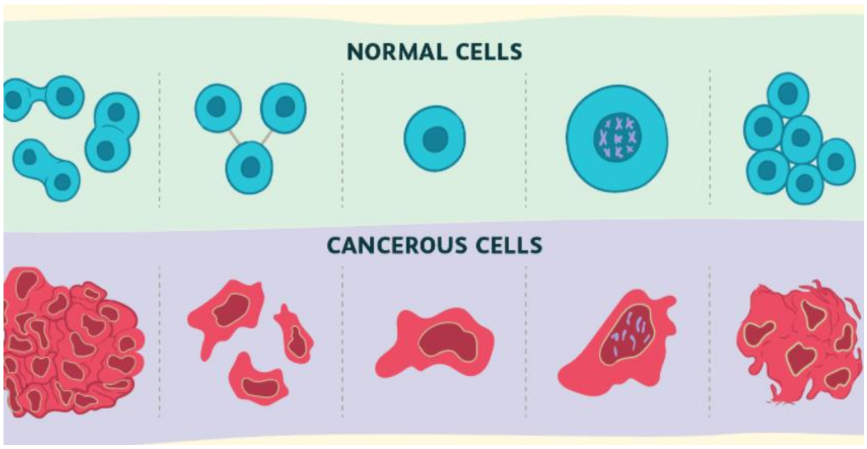


Figure 12: Normal vs cancerous cells

As shown in figure 12, there is specific shape or features cancerous cells. This explains the considerable number of outliers. Removing them might affect the accuracy of the model.

The most distanced outliers can be removed using the z score method. But this should be done one column at time while observing how the model accuracy chances with it.

This will be left for future developments considering the required extra amount of time and more expertise on each attribute. But the availability of high outliers will be considered during feature selection.

2.5 Balancing the dataset

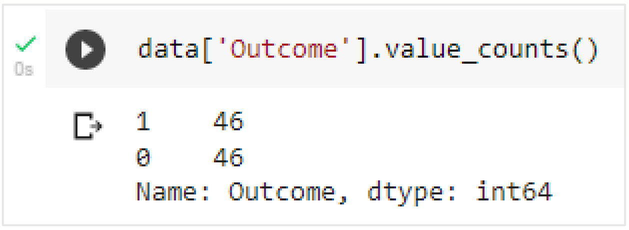
Due to the 3:1 nature of the N & R datapoints of the Outcome column, the model might generate predictions biased to the majority (non-recurrent) class. Therefore, dataset be changed to have at least a ratio of 3:2. There are 2 approaches are available for this.

- Under sampling the majority class – random datapoints can be removed until required ratio of data is achieved
- Oversampling the minority class – data can be synthesized until the dataset is balanced.

For the easiness, under sampling the majority class was done. Datapoints were removed until a 1:1 ratio was archived.

To reduce the existing 147 N data points to 46, 101 random rows had to be deleted. They were selected by, data['NB_Outcome']= data.query("Outcome==0").sample(101).

Above line of code created a new column in the data frame with the value '0.0' for 101 samples and 'NaN' for the rest. Rows with 0.0 were removed using, data = data[data.NB_Outcome != 0.0]. Actually what it did was rewriting the data frame with the rows which weren't '0.0'. A balanced dataset was created with this.



```
data['Outcome'].value_counts()

1    46
0    46
Name: Outcome, dtype: int64
```

Figure 13: Balanced Output column

2.6 Feature selection

- 'Outcome' was selected as the Y variable since model will be predicting it.

- Out of the remaining 34 attributes, time and ID columns can be removed since patient ID has no relation to the recurrence of cancer and so does the time to recurrence.

Therefor 32 attributes remain for selection. Correlation matrix was generated using correlation_matrix = data.corr(). And the heatmap of correlation matrix was generated using sns.heatmap(correlation_matrix). This data was used to select features.

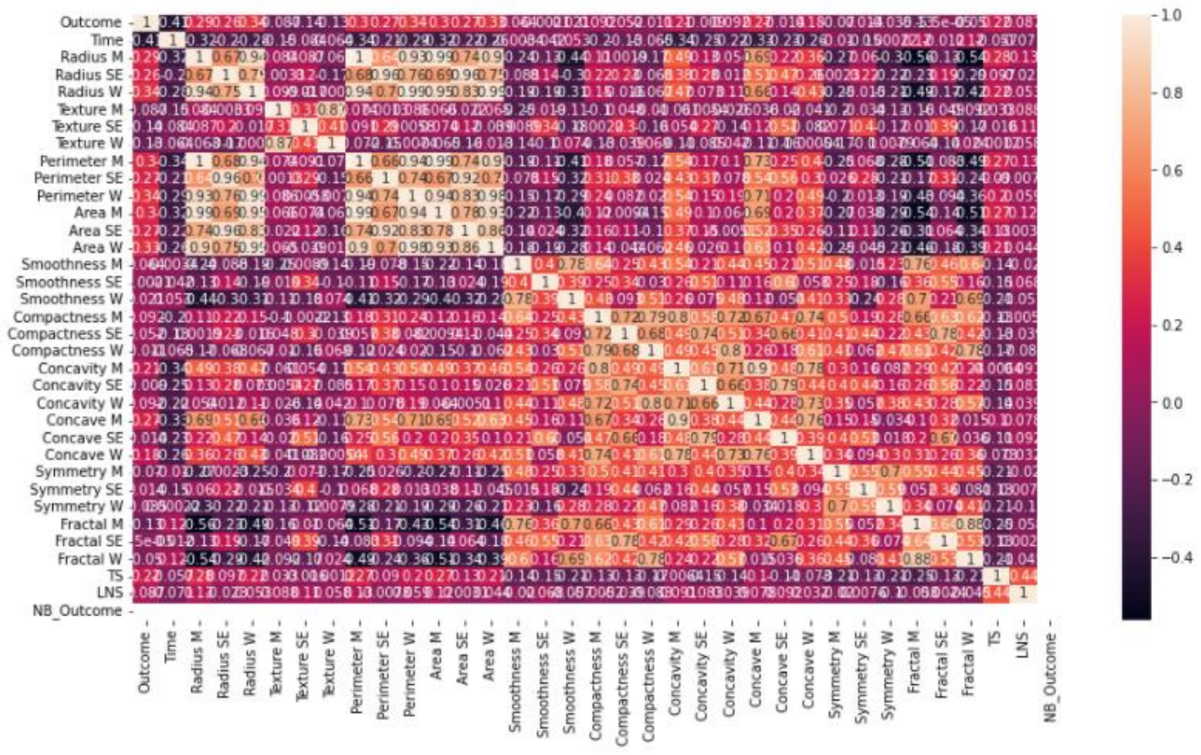


Figure 14: Correlation matrix

The columns which have a correlation above 0.5 with the y variable can be considered as good features. But according to this correlation matrix map, none of the columns have a correlation greater than 0.5 with the 'Outcome' column. Therefor the columns which are closest to 0.5 correlation were selected.

Column	Correlation value	Column	Correlation value
Radius Mean	0.285	Area Mean	0.298
Radius Worst	0.336	Area Standard Error	0.274
Concave Mean	0.271	Area Worst	0.327
		Perimeter Mean	0.295
		Perimeter Standard Error	0.271
		Perimeter Worst	0.343

Relationship between these columns were inspected again using the above correlation map and the below pair plot.

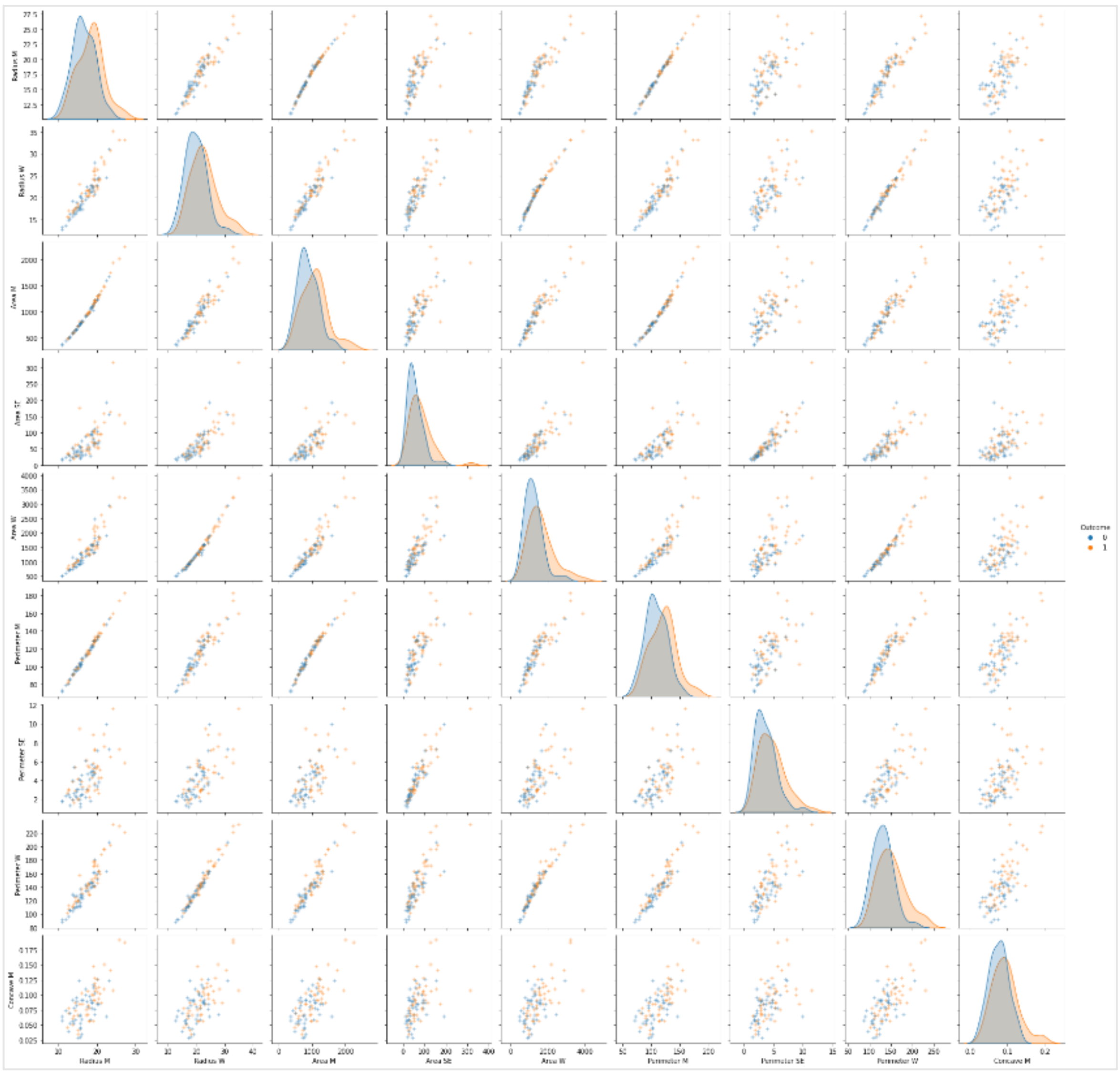


Figure 15:Pair plot

Based on these 2 plots, it can be observed that Radius Mean, Area Mean & Perimeter Mean have high correlation. So does the Radius Worst, Area Worst & Perimeter Worst. Perimeter SE and Area SE also have a high correlation but not much as the above two cases.

Based on the correlation with the 'outcome' column, following were selected as features (X variables).

- **Area Mean** – out of Radius Mean, Area Mean & Perimeter Mean
- **Perimeter Worst** – out of Radius Worst, Area Worst & Perimeter Worst
- **Area Standard Error** – out of Perimeter SE and Area SE
- **Concave Mean**

Selection was entered using `X_variables = ['Area M','Concave M', 'Area SE', 'Perimeter W']`.

'Outcome' column was selected as the y variable. Only the values in these columns were extracted into arrays for training the and testing the ML model, using the `.value` function of Pandas library.

2.7 Model development

2.7.1 Train Test Split

The dataset was next slip into 2. One for training the model and the other for testing the model. Training dataset includes 70% of the original set. Split was done using the scikit learn function train_test_split. the output are X_train, X_test, y_train, and y_test.

2.7.2 Model building

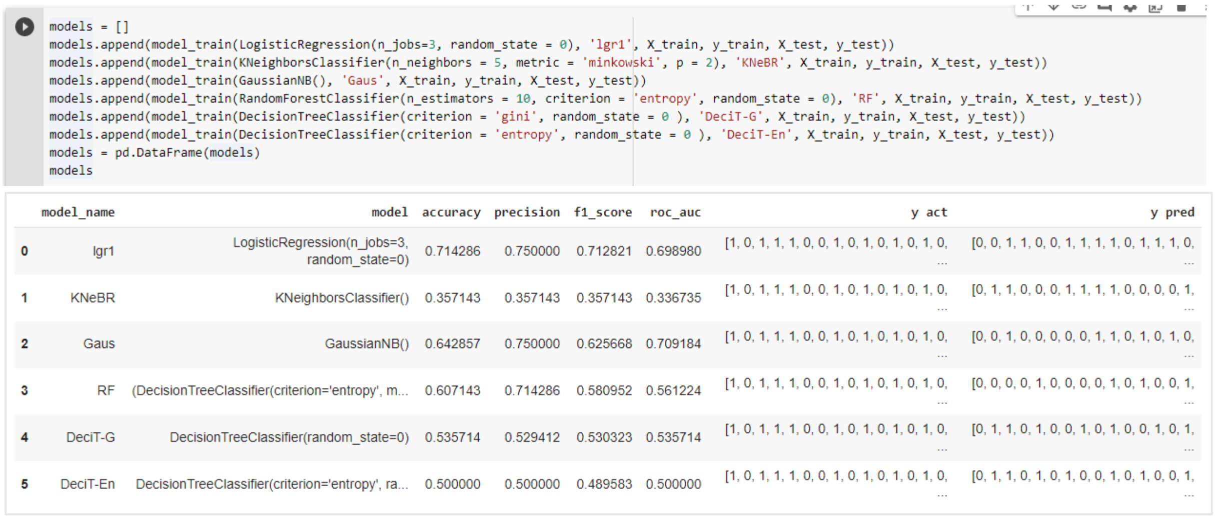
A function was written for training the model. Inputs for the training function are 'the model (imported from Scikit Learn), the name of the mode, X_train, y_train, X_test and y_test data.

Model.fit() was used to push the X_train and y_train data into the model and train it. Model.predict() was used to predict the 'Outcome' for the X_test data.

Scikitlearn metrics scoring functions were using to score the model output. The output of the model training function are 'model name, model details from scikit learn, accuracy, precission, f1 score, area under the roc curve, actual test dataset & predicted data set'.

2.7.3 Best model selection

Since this is a classification problem, several ML classification model were trained and scored to select the model with best predictions. This was done by creating a list called 'models'. Each element of the list the code calling the model training function for different ML models. Below is the list of models and the output from the model train function.

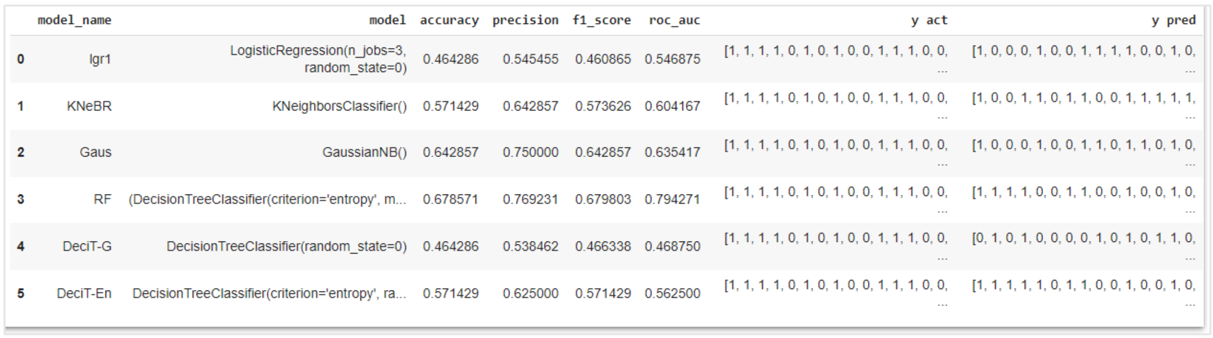


```
models = []
models.append(model_train(LogisticRegression(n_jobs=3, random_state = 0), 'lgr1', X_train, y_train, X_test, y_test))
models.append(model_train(KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2), 'KNeBR', X_train, y_train, X_test, y_test))
models.append(model_train(GaussianNB(), 'Gaus', X_train, y_train, X_test, y_test))
models.append(model_train(RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 0), 'RF', X_train, y_train, X_test, y_test))
models.append(model_train(DecisionTreeClassifier(criterion = 'gini', random_state = 0 ), 'DeciT-G', X_train, y_train, X_test, y_test))
models.append(model_train(DecisionTreeClassifier(criterion = 'entropy', random_state = 0 ), 'DeciT-En', X_train, y_train, X_test, y_test))
models = pd.DataFrame(models)
models
```

	model_name	model	accuracy	precision	f1_score	roc_auc	y act	y pred
0	lgr1	LogisticRegression(n_jobs=3, random_state=0)	0.714286	0.750000	0.712821	0.698980	[1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, ...	[0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, ...
1	KNeBR	KNeighborsClassifier()	0.357143	0.357143	0.357143	0.336735	[1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, ...	[0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, ...
2	Gaus	GaussianNB()	0.642857	0.750000	0.625668	0.709184	[1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, ...	[0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, ...
3	RF	(DecisionTreeClassifier(criterion='entropy', m...	0.607143	0.714286	0.580952	0.561224	[1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, ...	[0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, ...
4	DeciT-G	DecisionTreeClassifier(random_state=0)	0.535714	0.529412	0.530323	0.535714	[1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, ...	[0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, ...
5	DeciT-En	DecisionTreeClassifier(criterion='entropy', ra...	0.500000	0.500000	0.489583	0.500000	[1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, ...	[0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, ...

Figure 16:Model scores

Here we can see that the logistic regression has the most impressive scores. But when the program was run again, a different score (shown below) was given. The reasons behind these changes are analyzed later.



	model_name	model	accuracy	precision	f1_score	roc_auc	y act	y pred
0	lgr1	LogisticRegression(n_jobs=3, random_state=0)	0.464286	0.545455	0.460865	0.546875	[1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, ...	[1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, ...
1	KNeBR	KNeighborsClassifier()	0.571429	0.642857	0.573626	0.604167	[1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, ...	[1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, ...
2	Gaus	GaussianNB()	0.642857	0.750000	0.642857	0.635417	[1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, ...	[1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, ...
3	RF	(DecisionTreeClassifier(criterion='entropy', m...	0.678571	0.769231	0.679803	0.794271	[1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, ...	[1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, ...
4	DeciT-G	DecisionTreeClassifier(random_state=0)	0.464286	0.538462	0.466338	0.468750	[1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, ...	[0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, ...
5	DeciT-En	DecisionTreeClassifier(criterion='entropy', ra...	0.571429	0.625000	0.571429	0.562500	[1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, ...	[1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, ...

Figure 17: Model score 2

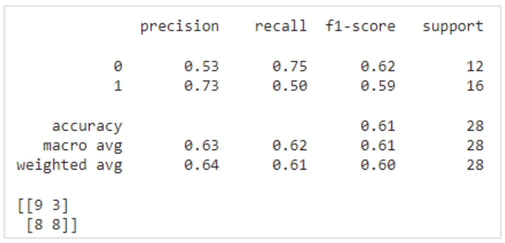
Since rerunning the program generated different scores, 3 models which generated good results multiple times were selected for tuning.

2.7.4 Tuning the best model

Hyperparameters of the selected models were tuned using the grid search method. After selecting the best parameters, a prediction was done for each model.

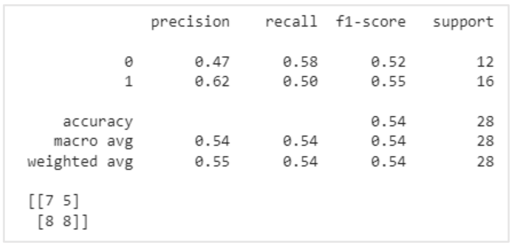
Best parameters and pedictions:

Decision tree: `{'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf': 17, 'min_samples_split': 2}`



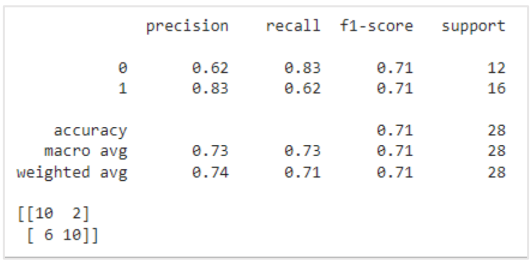
```
              precision    recall  f1-score   support

           0       0.53      0.75      0.62        12
           1       0.73      0.50      0.59        16

    accuracy                           0.61        28
   macro avg       0.63      0.62      0.61        28
weighted avg       0.64      0.61      0.60        28

[[9 3]
 [8 8]]
```

Logistic regression: `{'C': 100, 'penalty': 'none', 'solver': 'sag'}`



```
              precision    recall  f1-score   support

           0       0.47      0.58      0.52        12
           1       0.62      0.50      0.55        16

    accuracy                           0.54        28
   macro avg       0.54      0.54      0.54        28
weighted avg       0.55      0.54      0.54        28

[[7 5]
 [8 8]]
```

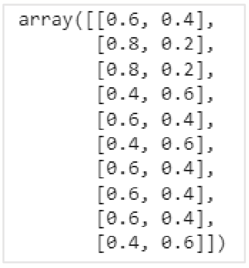
KNeighbor Classifier: `{'leaf_size': 1, 'n_neighbors': 21, 'p': 1}`



```
              precision    recall  f1-score   support

           0       0.62      0.83      0.71        12
           1       0.83      0.62      0.71        16

    accuracy                           0.71        28
   macro avg       0.73      0.73      0.71        28
weighted avg       0.74      0.71      0.71        28

[[10  2]
 [ 6 10]]
```

2.7.5 Scoring the model

A sample dataset was taken from the main dataset and was fed into the model for predictions. The probability of prediction outcome as 1 or 0 was extracted using model.predict_proba.

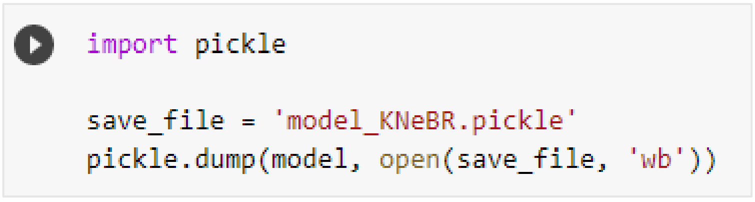


```
array([[0.6, 0.4],
       [0.8, 0.2],
       [0.8, 0.2],
       [0.4, 0.6],
       [0.6, 0.4],
       [0.4, 0.6],
       [0.6, 0.4],
       [0.6, 0.4],
       [0.6, 0.4],
       [0.4, 0.6]])
```

Figure 18: Prediction probability

2.7.6 Saving model

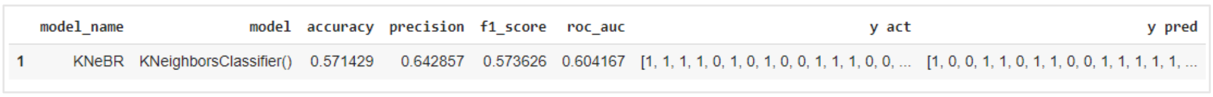
After tuning the models, KNeighbourClassifier was selected as the best model as it has the best recall and f1 scores. In order to store the trained model, pickle was used.



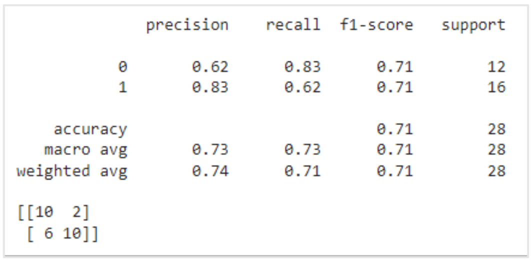
```
import pickle

save_file = 'model_KNeBR.pickle'
pickle.dump(model, open(save_file, 'wb'))
```

Figure 19: Saving the tuned model

3.0 Discussion and conclusions



	model_name	model	accuracy	precision	f1_score	roc_auc	y act	y pred
1	KNeBR	KNeighborsClassifier()	0.571429	0.642857	0.573626	0.604167	[1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, ...	[1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, ...



```
              precision    recall  f1-score   support

           0       0.62      0.83      0.71        12
           1       0.83      0.62      0.71        16

    accuracy                           0.71        28
   macro avg       0.73      0.73      0.71        28
weighted avg       0.74      0.71      0.71        28

[[10  2]
 [ 6 10]]
```

Figure 20: Model scores

Model has a true positive rate of 62.5% and true negative rate of 83.4%. The precision and F1 scores have improved after tuning the model parameters. But this cannot be considered as a reliable model because the precision and recall are just a little over 0.5. But this is the best model out of the tested models. But as stated earlier, when the whole program is run again, these scores change. At one of those instances, an accuracy of **0.834** was given for the **Gaussian** model.

1st possible reason for these changes is deducted as this: every time the program is run from the beginning, 101 random values are removed from the dataset to balance the classes. Therefore, the final dataset is not the same.

The highest accuracy of 0.834 was must have generated when the worst datapoints were removed during balancing. This concludes that proper cleaning of this dataset can improve the model.

The reason it wasn’t done earlier is that treating the outliers was tried before feature selection and considerable amount of the minority class was also removed when tried. A possible reason for this is that all the features were considered at that time resulting in large of outliers removed.

What should be done next is to only remove the outliers in the selected features and test the model again.

It would also be better to try and remove the outliers of the selected features only for the datapoints where ‘Outcome’ is ‘0’ (Majority class). Then scale up the minority class by synthesis of data.

2nd reason for low accuracy and recall might be the low amount of data points after cleaning

3rd reason can be the unpredictable behavior of the cancer cells parameters. This reason makes this dataset very hard to predict.

At the current stage, this model shows that despite the other 32 parameters, mean area and concaveness, standard error of the area and the worst perimeter values plays an important role in predicting if the cancer can reoccur or not.

This was verified by once training the testing the model with all the available parameters. It resulted in very lower accuracy and precision than the current values. In future, the model should be scored while adding the remaining parameters one by one. This will give a good insight on how the other parameter relate cancer reoccurrence on an individual level.

3.1 summery

- As a summery, the model predicts negative cases more precisely than predicting positive cases. 62.5% true positive model is not reliable enough for the medical field where the model will be used.
- Minority class should be oversamples and the outlier removed majority class should be under sampled a little to balance the dataset without just under sampling the majority class to reach the size of the minority class.
- Clear signs are there indicating that further data cleansing can improve the model accuracy. Only the model with highest precision should be saved. (I accidently saved a lower precision model over the 0.834 precise model due executing the run all function)

Even though the model isn't accurate, it proved to be working. This achieves the objective of building the model which is to test if the course content was property understood or not.