IT4060 - MACHINE LEARNING

Assignment 2 Report

Breast Cancer Detection using Random Forest Classifier Algorithm

Submitted by:

IT18153750 – Herath H.M.R.K.R.

IT18378658 – Perera A.P.A.D.

IT18085686 – Samaranath T.I.

Bachelor of Science Special (honors)

In Information Technology

Department of Information Technology

Sri Lanka Institute of Information Technology

Sri Lanka

May 2021

TABLE OF CONTENT

TABLE OF FIGURES

1 INTRODUCTION

1.1 Breast Cancer

Breast cancer is the one of the most well-known cancer among women worldwide. **Breast cancer** begins when healthy cells in the **breast** change and become out of control. There are lot of facts cause to breast cancer [3].

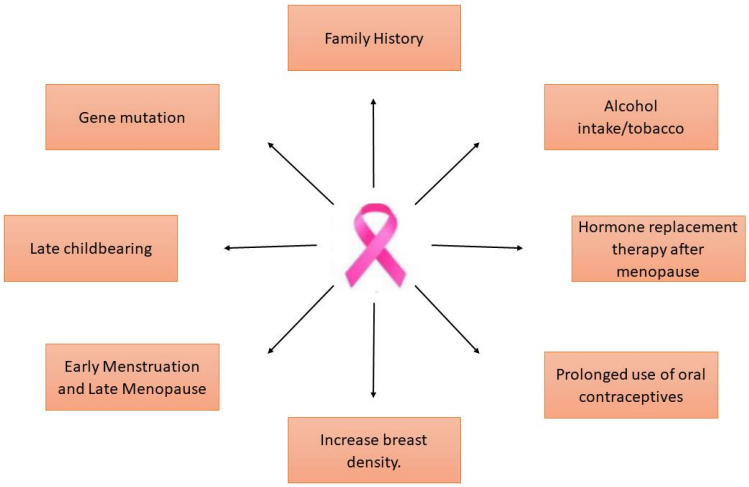

Figure 1.1.1: The facts that cause breast cancer.

The improvement of breast cancer is a multi-step process, including different cell types. Preventing it remains challenge in the world. Diagnosis can be classify as following.

1.**benign(B)** – This is not cancerous because their cells are close to a healthy appearance. They grow slowly, and they do not spread all throughout the body.

2.**Malignant(M)** – This is a cancerous because their cells can grow rapidly, attack and obliterate close by ordinary tissues, and spread throughout the body.
Early detection is a way deal with control the breast cancer. Early detection and reduction of mortality rates are handled in several ways. We'll try to develop a framework for early detection of patients by assessing several factors affecting the breast cancer using Machine learning techniques.

2 DATASET

In order to develop the early detection system for breast cancer, the data set was taken from the Kaggle website [1].

D**ATASET** : https://www.kaggle.com/uciml/breast-cancer-wisconsin-data/data

Features are calculated from a digitized image of a FNA of a breast mass.
[FNA – Fine Needle Aspire].They describe attributes of the cell nuclei present in the image.
The characteristic of the data set as follow.

Attribute Characteristics	Real
Number of Instances	569
Number of Attributes	30
Missing Values	No
Associated Tasks	Classification
Class distribution	Benign (357) , Malignant (212)

10 real-valued features are calculated for each cell nucleus:

Real-valued features	Description
Radius	mean of distances from center to points on the perimeter
Texture	standard deviation of gray-scale values
Perimeter	Boundary around a shape
Area	measurement of a surface
Smoothness	local variation in radius lengths
Compactness	perimeter^2 / area - 1.0
Concavity	severity of concave portions of the contour
Concave points	number of concave portions of the contour
Symmetry	Correct correspondence between different things
Fractal dimension	"coastline approximation" - 1

Attribute Information:

1) ID number
2) Diagnosis

M = malignant, B = benign
3-32)

1.Mean
2.standard error (SE)
3.worst /The mean of the three largest values / largest

calculated for each image, it has 30 features as result.

E.g.: field 3 is Mean Radius, field13 is Radius SE(Standard error), field 23 is Worst Radius.

3 METHODOLOGY

3.1 Random Forest Classification Algorithm

Random forest algorithm is a supervised learning algorithm [4] . It is used for the following.

1.Classification

2. Regression

This algorithm makes a forest with several trees at random. It is a collection of Decision Trees, trained with the bagging method. Random forest combines with many decision trees to make more accurate predictions. This algorithm is more flexible and simple to use.

We can understand the functionality of Random Forest algorithm with the help of following steps.

Step 1	Random sample selection from a given data set.
Step 2	This algorithm will create a decision tree for every sample. Then each decision tree will have a predictive result.
Step 3	Vote for each of the predicted results.
Step 4	Finally, choose the most voted prediction result

The diagram below illustrates its functionality.[4]

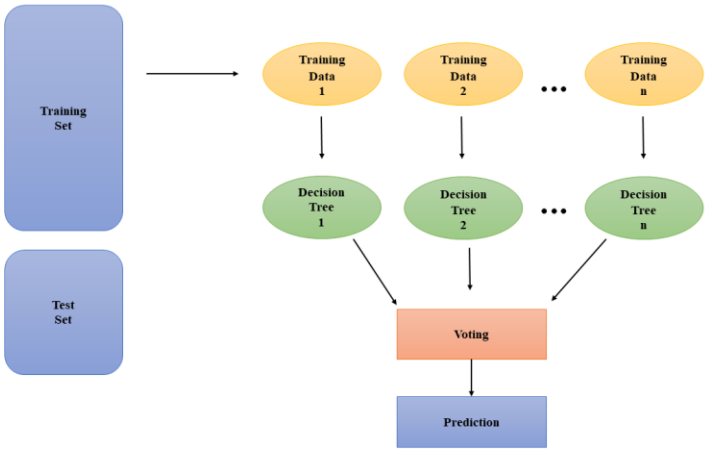

Figure 3.1.1: The workflow of Random Forest Algorithm

4 IMPLEMENTATIONS

4.1 Importing Data.

First step is to import the related common libraries required. As shown in Figure 4.1.1, imported all common python libraries such as numpy, pandas, matplotlib. Pyplot and seaborn for the execution of code segments. **numpy** is used for data statistical analysis, **pandas** is used for data manipulation using dataframes, **matplotlib.pyplot** is used for data visualization. Seaborn is used for statistical data visualization. **sklearn.metrics** includes scores, performance metrics, pair metrics and distance clculations.

```
In [1]: #Import libraries
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sb
        import sklearn.metrics as sm
        %matplotlib inline
```

Figure 4.1.1: Importing common libraries

As shown in figure 4.1.2, Load data in the data.csv to the python program. Following command to open the file**. df_cancer** is a **DataFrame.object**. **Read_csv** function in Pandas library is utilized to load the data.

```
In [2]: # Read in data
        df_cancer = pd.read_csv("data.csv")
```

Figure 4.1.2: Load data from csv file

The following command is used to display the first five lines of the data frame.

```
In [3]: #Read the data and display 5 raws
        df_cancer.head()
```

Out[3]:

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

5 rows × 33 columns

Figure 4.1.3: output of the head (5) function

4.2 Exploring Data.

In Python program, the number of rows/columns is given by shape () function in pandas library, explore the data types of the columns which could be done using dtypes () function and describe () function calculates a statistical summary for the columns of the DataFrame. The outputs of **shape** function , **dtypes ()** function and **describe ()** function will be shown in figure 4.2.3, figure 4.2.4 and figure 4.2.5 accordingly.

```
In [6]: # Get the new count of the number of rows and cols
        df_cancer.shape

Out[6]: (569, 33)
```

Figure 4.2.1: output of the shape function

```
In [7]: # Display the data types
        df_cancer.dtypes

Out[7]: id                         int64
        diagnosis                 object
        radius_mean              float64
        texture_mean             float64
        perimeter_mean           float64
        area_mean                float64
        smoothness_mean          float64
        compactness_mean         float64
        concavity_mean           float64
        concave points_mean      float64
        symmetry_mean            float64
        fractal_dimension_mean   float64
        radius_se                float64
        texture_se               float64
        perimeter_se             float64
        area_se                  float64
        smoothness_se            float64
        compactness_se           float64
        concavity_se             float64
        concave points_se        float64
        symmetry_se              float64
        fractal_dimension_se     float64
        radius_worst             float64
        texture_worst            float64
        perimeter_worst          float64
        area_worst               float64
        smoothness_worst         float64
        compactness_worst        float64
        concavity_worst          float64
        concave points_worst     float64
        symmetry_worst           float64
        fractal_dimension_worst  float64
        Unnamed: 32              float64
        dtype: object
```

Figure 4.2.2: output of the dtypes() function

```
In [8]: # Descriptive statistics for each column
        df_cancer.describe()
```

Out[8]:

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

8 rows × 32 columns

Figure 4.2.3: output of the describe() function

4.3. Data preprocessing

In python programming, check null values using below as shown in Figure 4.3.1.

```
In [9]: # Count the empty values in each column
        df_cancer.isnull().sum()

Out[9]: id                          0
        diagnosis                   0
        radius_mean                 0
        texture_mean                0
        perimeter_mean              0
        area_mean                   0
        smoothness_mean             0
        compactness_mean            0
        concavity_mean              0
        concave points_mean         0
        symmetry_mean               0
        fractal_dimension_mean      0
        radius_se                   0
        texture_se                  0
        perimeter_se                0
        area_se                     0
        smoothness_se               0
        compactness_se              0
        concavity_se                0
        concave points_se           0
        symmetry_se                 0
        fractal_dimension_se        0
        radius_worst                0
        texture_worst               0
        perimeter_worst             0
        area_worst                  0
        smoothness_worst            0
        compactness_worst           0
        concavity_worst             0
        concave points_worst        0
        symmetry_worst              0
        fractal_dimension_worst     0
        Unnamed: 32               569
        dtype: int64
```

Figure 4.3.1: check for null values

The column ‘Unnamed: 32’ from the original data set can be deleted because it does not add any value. The ID column will not help us make predictions about the cancer. It can be dropped.

```
In [11]: # Since "id" column has no direct impact on target value (just patient id or ind
         df_cancer.drop('id',axis=1,inplace=True)
         df_cancer.drop('Unnamed: 32',axis=1,inplace=True)
```

Figure 4.3.2: clean the data set

It checks the duplicate value count/values.

```
In [13]: df_cancer.duplicated().sum()

Out[13]: 0

In [14]: # Checking if any duplicate values in the df
         df_cancer.duplicated()

Out[14]: 0      False
         1      False
         2      False
         3      False
         4      False
                ...
         564    False
         565    False
         566    False
         567    False
         568    False
         Length: 569, dtype: bool
```

Figure 4.3.3: checking if there any duplicates values in the dataframe

It then calculates number of rows/columns from the cleand data set. The output is shown in Figure 4.3.5 below.

```
In [17]: df_cancer.shape
Out[17]: (569, 31)
```

Figure 4.3.4: get the new count of rows/columns from the cleaned dataset

As shown in figure 4.2.2 above, it is necessary to transformed/encoded the data type in the **"diagnosis"** column, which is the categorical data represented as a python object.

As shown in Figure 4.3.5, all categorical data are encrypted by changing the values in the 'diagnostic' column from M and B to 1 and 0.

```
In [18]: # Mapping string value feature to numerical value feature using map function
         cancer_mapping = {'B':0, 'M':1}
         df_cancer.diagnosis = df_cancer.diagnosis.map(cancer_mapping)
```

Figure 4.3.5: Mapping string value feature to numerical value feature using map function.

The first 5 rows of the dataset after the encoding process are shown in figure 4.3.6.

```
In [19]: df_cancer.head(5)
```

Out[19]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean
0	1	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419
1	1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812
2	1	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069
3	1	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597
4	1	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809

5 rows × 31 columns

Figure 4.3.6: Display first 5 rows

4.4 Split the data

First, the data set for the model should be divided into two parts (e.g.: feature and target).
Feature - independent (**X**) dataset
Target - dependent (**Y**) dataset.
The "diagnosis" column is considered target data, and the remaining columns are considered feature data.

Then, Split the x and y data again as training and testing data.

```
In [25]: # drop the target label coloumns
         X = df_cancer.drop(['diagnosis'],axis=1)
```

```
In [27]: # Define output values - this is the target
         Y = df_cancer['diagnosis']
         Y
```

```
In [30]: # splitting the data into test and train / using 20% of the dataset
         from sklearn.model_selection import train_test_split
         X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20, random_state=5)
```

Figure 4.4.1: Split data into training and testing data

As shown in Figure 4.4.1, this is used to divide the data arrays into two sub-sets called training data and testing data. In **Sklearn** model, **train_test_split fuction** was used .

There are several parameters of that function. Gets the data selected to use the **X, y** parameters**. test_size** will specify the size of the test dataset. In this case, the test size is 20 percent and train size is changed to 80 percent accordingly. If the random state parameter of the **train_test_split** function is not defined, it will divide the arrays into random subsets.

4.5 Feature Scaling

Scale the data to get all features to the same magnitude point.This means tha the feature / independent data will be a different range, e.g. 0–100 or 0–1.

```
In [36]: # Feature Scaling
         from sklearn.preprocessing import StandardScaler
         scaler = StandardScaler()
         X_train = scaler.fit_transform(X_train)
         X_test = scaler.transform(X_test)
```

Figure 4.5.1: Feature Scaling

4.6 Build the Random Forest classifier

A random forest is a meta estimator that suits multiple decision tree classification on different dataset sub-samples and uses an average to boost predictive accuracy and control overfitting.

```
In [37]: # Using Random Forest Classification algorithm
         from sklearn.ensemble import RandomForestClassifier
         rf_classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state =
         0)
         rf_classifier.fit(X_train, Y_train)
Out[37]: RandomForestClassifier(criterion='entropy', n_estimators=10, random_state=0)
```

```
In [38]: # Print model accuracy on the training data
         print('Random Forest Classifier Training Accuracy:', rf_classifier.score(X_train, Y_train))
         Random Forest Classifier Training Accuracy: 0.9956043956043956
```

Figure 4.6.1: Build the Random Forest classifier

As shown in Figure 4.6.1, the RandomForestClassifier() function which is used to create a classification, has several parameters. **n_estimators** are used to process the number of trees in the forest, criteria is used to measure the consistency of divisions, and the support criteria for obtaining details are "entropy". **random_state** can be used to control the randomness.

There are several approaches to use to train the model of classifier. As Figure 4.6.1, there is a 0.995 accuracy score on the training data for the random forest classifier model to classify whether or not a patient has cancer.

4.7 Testing

Uses 20 percent of the data for the predictions. Figure 4.7.1 shows that the classifier obtained an 0.956 accuracy score.

```
In [40]: # Get the accuracy of the test data
         Y_prediction = rf_classifier.predict(X_test)
         test_accuracy=sm.accuracy_score(Y_test, Y_prediction)
         print('Testing Accuracy:',str(test_accuracy))

         Testing Accuracy: 0.956140350877193
```

Figure 4.7.1: Test accuracy

4.8 Classification report

The classification report lists the most appropriate classification metrics for each category. This provides an in-depth understanding of classifier behavior towards global accuracy which can obscure the functional limitations in a multi-level problem class.

As shown in Figure 4.8.1, The report for the cancer dataset shows the key classification metrics precision, recall, f1-score and support per class.

```
In [42]: #Generate classification report based on the predicted values
         from sklearn import metrics
         print("Classification Report : \n\n", metrics.classification_report(Y_prediction
         target_names = ["Malignant","Benign"]))

         Classification Report :

                       precision    recall  f1-score   support

            Malignant       1.00      0.93      0.96        71
               Benign       0.90      1.00      0.95        43

             accuracy                           0.96       114
            macro avg       0.95      0.96      0.95       114
         weighted avg       0.96      0.96      0.96       114
```

Figure 4.8.1: Classifier report

- **Precision** - The ability of a classifier to indicate what is actually negative in a positive case.
- **Recall** – It is possible to identify all positive instances by a classification. It is defined as the ratio of true positives to the sum of true positives & false negatives for each class.
- **f1 score** - When classifying samples belonging to this class, this gives the class accuracy related to other classes
- **Support** - Enter the actual number of class occurrences in the specified dataset.

4.9 Visualization

Get a count of patients with cancerous Malignant (M) and non-cancerous Benign (B) cells and Visualize a count plot by using that count as shown in figure 4.9.1 below.

```
In [22]: # 357 benign (1), 212 malignant (0)
         sb.countplot(df_cancer['diagnosis'], label = "Count")

         C:\Users\dilak\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
           warnings.warn(

Out[22]: <AxesSubplot:xlabel='diagnosis', ylabel='count'>
```

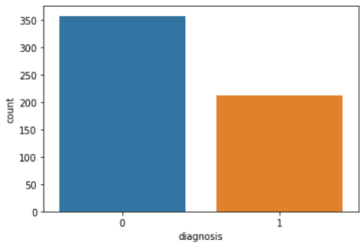

Figure 4.9.1: Visualize a count plot

Get the correlation of the columns in dataset and Visualize the correlation by creating a heat map as shown in figure 4.9.2.

```
In [24]: # Visualize the correlation
         plt.figure(figsize=(24,13))
         sb.heatmap(df_cancer.corr(), annot=True)

Out[24]: <AxesSubplot:>
```

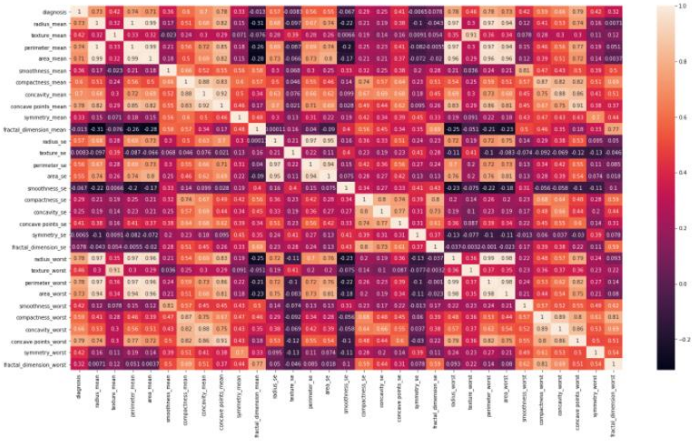

Figure 4.9.2: Visualize the correlation

Figure 4.9.3 despites the confusion matrix for the test results. The confusion matrix shows how many patients have been misdiagnosed and the number of correct diagnoses, the true positive and the true negative.

```
In [43]: # Based on the test values this code phase generate the confusion matrix
         from sklearn.metrics import confusion_matrix
         sb.set()
         get_ipython().run_line_magic('matplotlib', 'inline')
         matrx = confusion_matrix(Y_test, Y_prediction)
         sb.heatmap(matrx.T, square=True, annot=True, fmt='d', cbar=False)
         plt.xlabel('true class')
         plt.ylabel('predicted class')

Out[43]: Text(89.18, 0.5, 'predicted class')
```

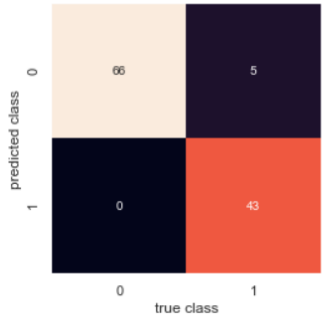

Figure 4.9.3: Confusion Matrix

5 EVALUATION

5.1 Critical analysis

Random Forest Classifier showed the best performance of the test data with a precision score of approximately 95.6 percent. Therefore, this would be the good way for patients to detect cancer cells.

It Allows to predict/classify the test data. figure 5.1.1 shows both the classification/prediction model for the Random Forest Classifier and actual values of the patient that prove they have cancer either or not.

```
In [44]: # Compare the Predictions of Random Forest Classifier model and the actual class
         print("The Predictions of Random Forest Classifier model: \n\n", Y_prediction)

The Predictions of Random Forest Classifier model:

 [1 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 1 1 1 1 0 0 1
 0 0 1 0 0 0 1 0 0 1 1 0 1 0 0 0 0 0 1 1 1 0 1 1 1 0 0 0 0 0 0 0 1 0 1 0 0
 0 0 0 0 1 1 1 0 1 1 1 0 1 0 1 1 1 1 0 0 1 1 0 0 0 0 0 1 0 0 1 1 0 1 0 0 0
 1 1 0]
```

```
In [45]: print("\nThe actual classification of the patients: \n\n", Y_test)
```

```
The actual classification of the patients:

 28     1
163    0
123    0
361    0
549    0
      ..
414    1
515    0
186    1
3      1
261    1
Name: diagnosis, Length: 114, dtype: int64
```

Figure 5.1.1: Compare the Predictions of Random Forest Classifier model and the actual classification of the patients

5.2 k-Fold Cross Validation

Uses k-fold cross validation to find and justify the best model for the predictions. Whenever model predictions are discuss, it is important to understand the errors (bias and variance) of the prediction.

Bias = Average prediction of the model - the correct value that is attempting to predict.

Variance = Measure of variability.

The High-biased model pays no attention to training data and oversimplifies the process. This can leads to significant errors in training and testing results.

Bias: 0.956-0.940 = 0.016
Variance: 0.04
In supervised learning, the models that have low bias and low variance have good balance.

```
In [46]: # Applying k-Fold Cross Validation
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator=rf_classifier, X=X_train, y=Y_train, cv=10)
print("Mean: ", accuracies.mean())
print("Standard Deviation: ", accuracies.std())
```

```
Mean:  0.9405797101449276
Standard Deviation:  0.0454234743880831
```

Figure 5.2.1: k-fold cross validation

6 CONCLUSION

The random forest (RF) classification algorithm for this dataset provides an accuracy of 95% for the breast cancer detection. This model will be useful for control the breast cancer and reduction of mortality rates. In the future, our study will look additional attributes and risk factors for the breast cancer detection which are to be included be included in the dataset.

7 REFERENCES

[1] Kaggle.com. [Online]. Available: https://www.kaggle.com/uciml/breast-cancer-wisconsin-data/data. [Accessed: 10-May-2021].

[2] Z. Lateef, "Complete tutorial on Random Forest In R with examples," Edureka.co, 19-Sep-2014. [Online]. Available: https://www.edureka.co/blog/random-forest-classifier/. [Accessed: 10-May-2021].

[3] S. Salaria, "Breast cancer detection using machine learning - DataDrivenInvestor," DataDrivenInvestor, 08-Feb-2019. [Online]. Available: https://medium.datadriveninvestor.com/breast-cancer-detection-using-machine-learning-475d3b63e18e. [Accessed: 10-May-2021].

[4] "Classification Algorithms - Random Forest," Tutorialspoint.com. [Online]. Available: https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_with_python_classification_algorithms_random_forest.htm. [Accessed: 10-May-2021].

8 APPENDICES

8.1 Appendix – A: Source Code

```
# 1.IMPORTING DATA
#Import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
import sklearn.metrics as sm
%matplotlib inline

# Read in data
df_cancer = pd.read_csv("data.csv")

#Read the data and display 5 raws
df_cancer.head()

#2.EXPLORING THE DATA
# Key values in the dictionary
df_cancer.keys()

#count the raws ans columns in the data set
df_cancer.diagnosis.value_counts()

# Get the new count of the number of rows and cols
df_cancer.shape
```

```
# Display the data types
df_cancer.dtypes

# Descriptive statistics for each column
df_cancer.describe()

# 3.DATA PREPROCESSING
# Count the empty values in each column
df_cancer.isnull().sum()

# Checking if any duplicates in "id" column by finding unique values and if their frequency is greater than 1
df_cancer.id.value_counts().unique()

# Since "id" column has no direct impact on target value (just patient id or index)
df_cancer.drop('id',axis=1,inplace=True)
df_cancer.drop('Unnamed: 32',axis=1,inplace=True)

df_cancer.columns
df_cancer.duplicated().sum()

# Checking if any duplicate values in the df
df_cancer.duplicated()
# To check presence of missing (NaN) values
df_cancer.isnull().any()
# Get the new count of first 5 raws
df_cancer.head()
# Get the new count of the number of rows and columns
df_cancer.shape
```

```
# 4.VISUALIZING THE DATA

# Visualising selected features by target:

sb.pairplot(df_cancer, hue = 'diagnosis', vars = ['radius_mean', 'texture_mean', 'area_mean',
'perimeter_mean', 'smoothness_mean'] )


# Getting insight of data distribution based on frequency of unique values in the features

df_cancer.hist(figsize=(20,20))

plt.show()


# 357 benign (1), 212 malignant (0)

sb.countplot(df_cancer['diagnosis'], label = "Count")


sb.scatterplot(x = 'area_mean', y = 'smoothness_mean', hue = 'diagnosis', data = df_cancer)


# Visualize the correlation

plt.figure(figsize=(24,13))

sb.heatmap(df_cancer.corr(), annot=True)


# 5.MODEL TRAINING

# drop the target label coloumns

X = df_cancer.drop(['diagnosis'],axis=1)

X


# Define output values - this is the target

Y = df_cancer['diagnosis']

Y


df_cancer_list=list(X.columns)

df_cancer_list
```

```
# splitting the data into test and train / using 20% of the dataset
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20, random_state=5)
X_train
X_test
Y_train
Y_test

print(X_train.shape,' The shape of Training Features')
print(Y_train.shape,' The shape of Training Lables')
print(X_test.shape,'  The shape of Testing Features')
print(Y_test.shape,'  The shape of Testing Lables')

# 6.EVALUATING THE MODEL
# Feature Scaling
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Using Random Forest Classification algorithm
from sklearn.ensemble import RandomForestClassifier
rf_classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state =
0)
rf_classifier.fit(X_train, Y_train)

# Print model accuracy on the training data
```

```
print('Random Forest Classifier Training Accuracy:', rf_classifier.score(X_train, Y_train))


#sklearn.metrics includes score functions, performance metrics and pairwise metrics and distance computations.
import sklearn.metrics as sm


# Get the accuracy of the test data
Y_prediction = rf_classifier.predict(X_test)
test_accuracy=sm.accuracy_score(Y_test, Y_prediction)
print('Testing Accuracy:',str(test_accuracy))


rf_classifier.score(X_test,Y_test)


#Generate classification report based on the predicted values
from sklearn import metrics
print("Classification Report : \n\n", metrics.classification_report(Y_prediction, Y_test,
target_names = ["Malignant","Benign"]))


# Based on the test values this code phase generate the confusion matrix
from sklearn.metrics import confusion_matrix
sb.set()
get_ipython().run_line_magic('matplotlib', 'inline')
matrx = confusion_matrix(Y_test, Y_prediction)
sb.heatmap(matrx.T, square=True, annot=True, fmt='d', cbar=False)
plt.xlabel('true class')
plt.ylabel('predicted class')


# Compare the Predictions of Random Forest Classifier model and the actual classification of the patients
```

```
print("The Predictions of Random Forest Classifier model: \n\n", Y_prediction)

print("\nThe actual classification of the patients: \n\n", Y_test)

# Applying k-Fold Cross Validation
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator=rf_classifier, X=X_train, y=Y_train, cv=10)
print("Mean: ", accuracies.mean())
print("Standard Deviation: ", accuracies.std())
```