Breast Cancer Prediction

October 6, 2023

1 Breast Cancer Prediction

Breast Cancer Prediction is a classification task aimed at predicting the diagnosis of a breast mass as either malignant or benign. The dataset used for this prediction consists of features computed from a digitized image of a fine needle aspirate (FNA) of the breast mass. These features describe various characteristics of the cell nuclei present in the image.

The dataset contains the following information for each instance:

1. ID number: A unique identifier for each sample.
2. Diagnosis: The target variable indicating the diagnosis, where 'M' represents malignant and 'B' represents benign.

For each cell nucleus, ten real-valued features are computed, which are:

1. Radius: The mean distance from the center to points on the perimeter of the nucleus.
2. Texture: The standard deviation of gray-scale values in the nucleus.
3. Perimeter: The perimeter of the nucleus.
4. Area: The area of the nucleus.
5. Smoothness: A measure of local variation in radius lengths.
6. Compactness: Computed as the square of the perimeter divided by the area minus 1.0.
7. Concavity: Describes the severity of concave portions of the nucleus contour.
8. Concave points: Represents the number of concave portions of the nucleus contour.
9. Symmetry: Measures the symmetry of the nucleus.
10. Fractal dimension: This feature approximates the "coastline" of the nucleus, using the concept of fractal geometry.

These features provide quantitative measurements that can be used to assess the characteristics of cell nuclei and aid in distinguishing between malignant and benign breast masses. By training a machine learning model on this dataset, it is possible to develop a predictive model that can assist in the early detection and diagnosis of breast cancer.

[1]:
```
# Importing the library
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

[2]:
```
# Importing the dataset
df = pd.read_csv('data.csv')
df.head()
```

[2]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	\
0	842302	M	17.99	10.38	122.80	1001.0	
1	842517	M	20.57	17.77	132.90	1326.0	
2	84300903	M	19.69	21.25	130.00	1203.0	
3	84348301	M	11.42	20.38	77.58	386.1	
4	84358402	M	20.29	14.34	135.10	1297.0	

	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	\
0	0.11840	0.27760	0.3001	0.14710	
1	0.08474	0.07864	0.0869	0.07017	
2	0.10960	0.15990	0.1974	0.12790	
3	0.14250	0.28390	0.2414	0.10520	
4	0.10030	0.13280	0.1980	0.10430	

	…	texture_worst	perimeter_worst	area_worst	smoothness_worst	\
0	…	17.33	184.60	2019.0	0.1622	
1	…	23.41	158.80	1956.0	0.1238	
2	…	25.53	152.50	1709.0	0.1444	
3	…	26.50	98.87	567.7	0.2098	
4	…	16.67	152.20	1575.0	0.1374	

	compactness_worst	concavity_worst	concave points_worst	symmetry_worst	\
0	0.6656	0.7119	0.2654	0.4601	
1	0.1866	0.2416	0.1860	0.2750	
2	0.4245	0.4504	0.2430	0.3613	
3	0.8663	0.6869	0.2575	0.6638	
4	0.2050	0.4000	0.1625	0.2364	

	fractal_dimension_worst	Unnamed: 32
0	0.11890	NaN
1	0.08902	NaN
2	0.08758	NaN
3	0.17300	NaN
4	0.07678	NaN

[5 rows x 33 columns]

1.1 Data Preprocessing

[3]:
```
# droping unnecessory columns
df.drop(['Unnamed: 32','id'],axis=1,inplace=True)
```

[4]:
```
# checking for the missing values
df.isnull().sum()
```

[4]:
```
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
dtype: int64
```

[5]:
```
# checking the data types of the columns
df.dtypes
```

[5]:
```
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
dtype: object
```

1.2 Feature Engineering

[6]:
```
df['diagnosis'] = df['diagnosis'].map({'M':1, 'B':0})
```

[7]:
```
df['diagnosis'] = df['diagnosis'].astype(int)
```

1.3 Exploratory Data Analysis

[8]:
```
# checking the data description
df.describe()
```

[8]:
```
        diagnosis  radius_mean  texture_mean  perimeter_mean    area_mean  \
count  569.000000   569.000000    569.000000      569.000000   569.000000
mean     0.372583    14.127292     19.289649       91.969033   654.889104
std      0.483918     3.524049      4.301036       24.298981   351.914129
min      0.000000     6.981000      9.710000       43.790000   143.500000
25%      0.000000    11.700000     16.170000       75.170000   420.300000
50%      0.000000    13.370000     18.840000       86.240000   551.100000
75%      1.000000    15.780000     21.800000      104.100000   782.700000
max      1.000000    28.110000     39.280000      188.500000  2501.000000

       smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
```

```
count       569.000000        569.000000      569.000000           569.000000
mean          0.096360          0.104341        0.088799             0.048919
std           0.014064          0.052813        0.079720             0.038803
min           0.052630          0.019380        0.000000             0.000000
25%           0.086370          0.064920        0.029560             0.020310
50%           0.095870          0.092630        0.061540             0.033500
75%           0.105300          0.130400        0.130700             0.074000
max           0.163400          0.345400        0.426800             0.201200

       symmetry_mean  …  radius_worst  texture_worst  perimeter_worst  \
count     569.000000  …    569.000000     569.000000       569.000000
mean        0.181162  …     16.269190      25.677223       107.261213
std         0.027414  …      4.833242       6.146258        33.602542
min         0.106000  …      7.930000      12.020000        50.410000
25%         0.161900  …     13.010000      21.080000        84.110000
50%         0.179200  …     14.970000      25.410000        97.660000
75%         0.195700  …     18.790000      29.720000       125.400000
max         0.304000  …     36.040000      49.540000       251.200000

        area_worst  smoothness_worst  compactness_worst  concavity_worst  \
count   569.000000        569.000000         569.000000       569.000000
mean    880.583128          0.132369           0.254265         0.272188
std     569.356993          0.022832           0.157336         0.208624
min     185.200000          0.071170           0.027290         0.000000
25%     515.300000          0.116600           0.147200         0.114500
50%     686.500000          0.131300           0.211900         0.226700
75%    1084.000000          0.146000           0.339100         0.382900
max    4254.000000          0.222600           1.058000         1.252000

       concave points_worst  symmetry_worst  fractal_dimension_worst
count            569.000000      569.000000               569.000000
mean               0.114606        0.290076                 0.083946
std                0.065732        0.061867                 0.018061
min                0.000000        0.156500                 0.055040
25%                0.064930        0.250400                 0.071460
50%                0.099930        0.282200                 0.080040
75%                0.161400        0.317900                 0.092080
max                0.291000        0.663800                 0.207500

[8 rows x 31 columns]
```

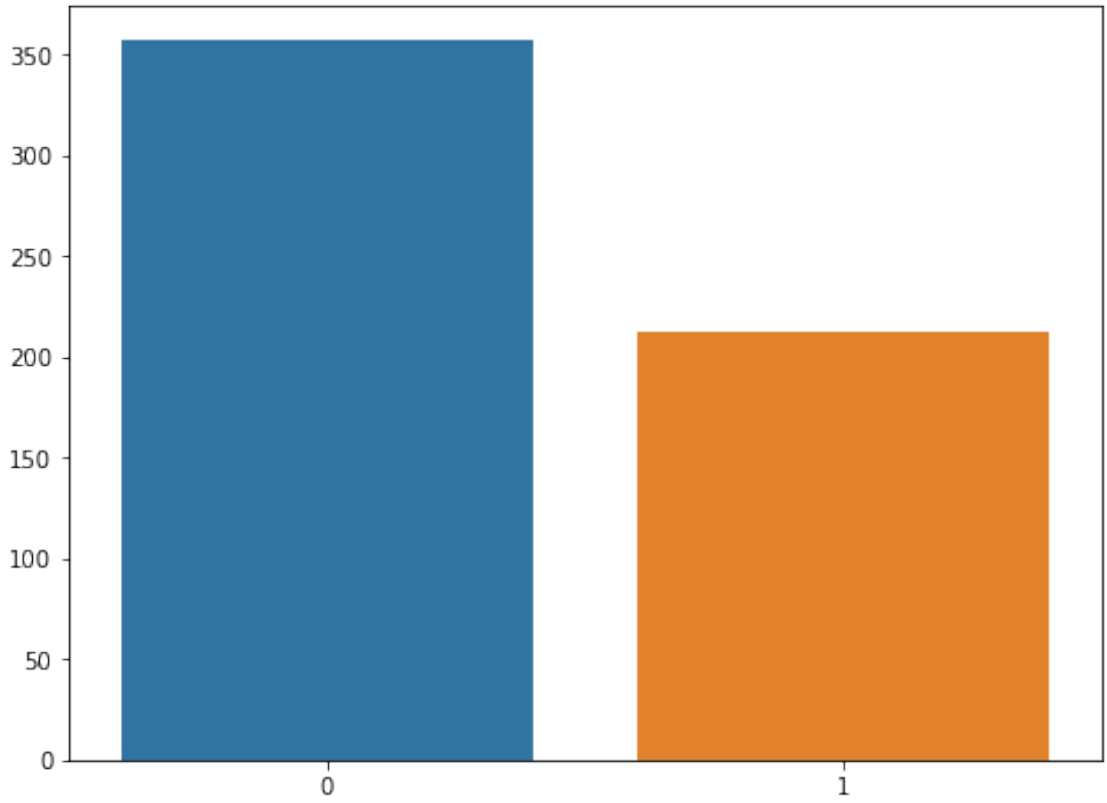
[9]:
```
# correlation between the columns diagnosis and the other columns
df.corr()['diagnosis'].sort_values()
```

[9]:
```
smoothness_se            -0.067016
fractal_dimension_mean   -0.012838
texture_se               -0.008303
```

```
symmetry_se               -0.006522
fractal_dimension_se       0.077972
concavity_se               0.253730
compactness_se             0.292999
fractal_dimension_worst    0.323872
symmetry_mean              0.330499
smoothness_mean            0.358560
concave points_se          0.408042
texture_mean               0.415185
symmetry_worst             0.416294
smoothness_worst           0.421465
texture_worst              0.456903
area_se                    0.548236
perimeter_se               0.556141
radius_se                  0.567134
compactness_worst          0.590998
compactness_mean           0.596534
concavity_worst            0.659610
concavity_mean             0.696360
area_mean                  0.708984
radius_mean                0.730029
area_worst                 0.733825
perimeter_mean             0.742636
radius_worst               0.776454
concave points_mean        0.776614
perimeter_worst            0.782914
concave points_worst       0.793566
diagnosis                  1.000000
Name: diagnosis, dtype: float64
```

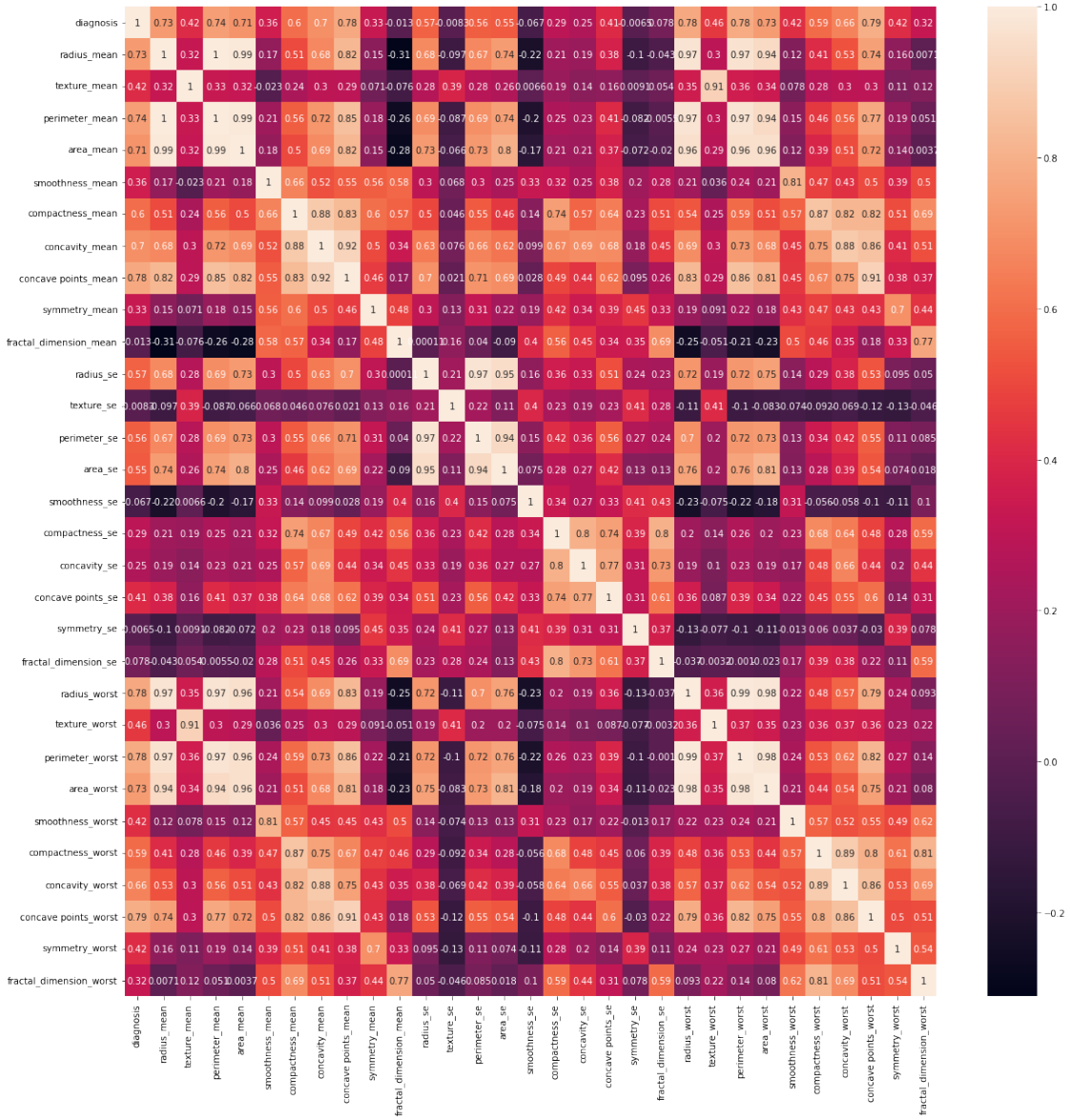
[10]:
```
# bar plot for the number of diagnosis
plt.figure(figsize=(8,6))
sns.barplot(x=df['diagnosis'].value_counts().index,y=df['diagnosis'].
 ↪value_counts().values)
```

[10]:
```
<AxesSubplot:>
```



[11]:
```
# heatmap to check the correlation
plt.figure(figsize=(20,20))
sns.heatmap(df.corr(),annot=True)
```

[11]: <AxesSubplot:>



1.4 Feature Scalling

[12]:
```
x = df.drop(['diagnosis'],axis=1)
```

[13]:
```
x.shape
```

[13]: (569, 30)

[14]:
```
y = df['diagnosis']
```

[15]:
```
y.shape
```

[15]: (569,)

1.5 Train Test Split

[16]:
```
from sklearn.model_selection import train_test_split
```

[17]:
```
X_train,X_test,y_train,y_test = train_test_split(x,y,test_size=0.
↪33,random_state=40)
```

1.6 Using Logistic Regression

[18]:
```
from sklearn.linear_model import LogisticRegression
```

[19]:
```
lr =  LogisticRegression()
lr.fit(X_train,y_train)
```

```
C:\Users\enama\anaconda3\lib\site-
packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[19]:
```
LogisticRegression()
```

[20]:
```
lr_pred = lr.predict(X_test)
```

1.7 Model Evolution

[21]:
```
# printing samples from predicted and actual values
print('Predicted values: ',lr_pred[:6])
print('Actual values: ',y_test[:6])
```

```
Predicted values:  [0 1 0 0 1 0]
Actual values:  295    0
16     1
431    0
453    0
15     1
275    0
Name: diagnosis, dtype: int32
```

[22]:
```
lr.score(X_test,y_test)
```

[22]:
```
0.9308510638297872
```

1.8 Using Decision Tree Classifier

[23]:
```
from sklearn.tree import DecisionTreeClassifier
```

[24]:
```
dtree = DecisionTreeClassifier()
dtree.fit(X_train,y_train)
```

[24]:
```
DecisionTreeClassifier()
```

[25]:
```
dtree_pred = dtree.predict(X_test)
```

1.9 Model Evolution

[26]:
```
# printing samples from predicted and actual values
print('Predicted values: ',dtree_pred[:6])
print('Actual values: ',y_test[:6])
```

```
Predicted values:  [0 1 0 0 1 0]
Actual values:  295    0
16     1
431    0
453    0
15     1
275    0
Name: diagnosis, dtype: int32
```

[27]:
```
dtree.score(X_test,y_test)
```

[27]:
```
0.9627659574468085
```

1.10 Using Random Forest Classifier

[28]:
```
from sklearn.ensemble import RandomForestClassifier
```

[29]:
```
rfr = RandomForestClassifier()
rfr.fit(X_train,y_train)
```

[29]:
```
RandomForestClassifier()
```

[30]:
```
rfr_pred = rfr.predict(X_test)
```

[31]:
```
## Model Evolution
```

[32]:
```
# printing samples from predicted and actual values
print('Predicted values: ',rfr_pred[:6])
print('Actual values: ',y_test[:6])
```

```
Predicted values:  [0 1 0 0 1 0]
Actual values:  295    0
16     1
431    0
453    0
15     1
275    0
Name: diagnosis, dtype: int32
```

[33]:
```
rfr.score(X_test,y_test)
```

[33]:
```
0.9627659574468085
```

1.11 Conclusion

From all the models we can see that the accuracy is 93.8%, 96% and 96% respectively. But we can see that the recall value for the Random Forest Classifeir and Decision tree classifier is 96% which is better than the logistic regression. So we can say that the Random forest classifier and Decision tree classifier is better then Logistic regression.