breastcancerdetection

September 8, 2024

0.1 This is my project to learn machine learning starting off with Logistic Regression.

Basic Approach for this project is to get the Breast Cancer Dataset from kaggle.

Then train the model.

In between this i need to preprocess the dataset and many other problem need to be resolved.

Lets Start off

Based on the structure, it appears to be a breast cancer dataset with features such as: ID: Identifier for each case Diagnosis: Label indicating whether the tumor is malignant (M) or benign (B) Radius_mean, Texture_mean, Perimeter_mean, etc.: Features related to the tumor's characteristics Steps to Build the Breast Cancer Detection Model

1.Load the Dataset: Since your dataset seems to be structured as a CSV, let's start by loading it into a DataFrame using Pandas.

2.Data Preprocessing:Handle missing values (if any). Convert categorical labels (e.g., M, B) into numerical values. Split the dataset into features (X) and labels (y).

3.Data Splitting: Use train_test_split to divide the dataset into training and test sets.

4.Model Building: You can use a classification model like Logistic Regression.

5.Training: Train the model using the training data.

6.Evaluation: Evaluate the model using metrics such as accuracy, precision, recall, and F1 score on the test data. Let's start with a simple example using Logistic Regression. Below is the code to get you started:

[148]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
import seaborn as sns
```

[109]:
```
## 2.Load the dataset and print first few rows

# Load the dataset
data = pd.read_csv('breast-cancer.csv')

# Display the first few rows
print('Here are the first few data:')
#print(data.info())
print(data.head(10)) # prints first 10 rows
```

```
Here are the first few data:
         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
5    843786         M        12.45         15.70           82.57      477.1
6    844359         M        18.25         19.98          119.60     1040.0
7  84458202         M        13.71         20.83           90.20      577.9
8    844981         M        13.00         21.82           87.50      519.8
9  84501001         M        12.46         24.04           83.97      475.9

   smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0          0.11840           0.27760         0.30010              0.14710
1          0.08474           0.07864         0.08690              0.07017
2          0.10960           0.15990         0.19740              0.12790
3          0.14250           0.28390         0.24140              0.10520
4          0.10030           0.13280         0.19800              0.10430
5          0.12780           0.17000         0.15780              0.08089
6          0.09463           0.10900         0.11270              0.07400
7          0.11890           0.16450         0.09366              0.05985
8          0.12730           0.19320         0.18590              0.09353
9          0.11860           0.23960         0.22730              0.08543

   symmetry_mean  fractal_dimension_mean  radius_se  texture_se  perimeter_se  \
0         0.2419                 0.07871     1.0950      0.9053         8.589
1         0.1812                 0.05667     0.5435      0.7339         3.398
2         0.2069                 0.05999     0.7456      0.7869         4.585
3         0.2597                 0.09744     0.4956      1.1560         3.445
4         0.1809                 0.05883     0.7572      0.7813         5.438
5         0.2087                 0.07613     0.3345      0.8902         2.217
6         0.1794                 0.05742     0.4467      0.7732         3.180
7         0.2196                 0.07451     0.5835      1.3770         3.856
8         0.2350                 0.07389     0.3063      1.0020         2.406
9         0.2030                 0.08243     0.2976      1.5990         2.039
```

	area_se	smoothness_se	compactness_se	concavity_se	concave points_se	\
0	153.40	0.006399	0.04904	0.05373	0.01587	
1	74.08	0.005225	0.01308	0.01860	0.01340	
2	94.03	0.006150	0.04006	0.03832	0.02058	
3	27.23	0.009110	0.07458	0.05661	0.01867	
4	94.44	0.011490	0.02461	0.05688	0.01885	
5	27.19	0.007510	0.03345	0.03672	0.01137	
6	53.91	0.004314	0.01382	0.02254	0.01039	
7	50.96	0.008805	0.03029	0.02488	0.01448	
8	24.32	0.005731	0.03502	0.03553	0.01226	
9	23.94	0.007149	0.07217	0.07743	0.01432	

	symmetry_se	fractal_dimension_se	radius_worst	texture_worst	\
0	0.03003	0.006193	25.38	17.33	
1	0.01389	0.003532	24.99	23.41	
2	0.02250	0.004571	23.57	25.53	
3	0.05963	0.009208	14.91	26.50	
4	0.01756	0.005115	22.54	16.67	
5	0.02165	0.005082	15.47	23.75	
6	0.01369	0.002179	22.88	27.66	
7	0.01486	0.005412	17.06	28.14	
8	0.02143	0.003749	15.49	30.73	
9	0.01789	0.010080	15.09	40.68	

	perimeter_worst	area_worst	smoothness_worst	compactness_worst	\
0	184.60	2019.0	0.1622	0.6656	
1	158.80	1956.0	0.1238	0.1866	
2	152.50	1709.0	0.1444	0.4245	
3	98.87	567.7	0.2098	0.8663	
4	152.20	1575.0	0.1374	0.2050	
5	103.40	741.6	0.1791	0.5249	
6	153.20	1606.0	0.1442	0.2576	
7	110.60	897.0	0.1654	0.3682	
8	106.20	739.3	0.1703	0.5401	
9	97.65	711.4	0.1853	1.0580	

	concavity_worst	concave points_worst	symmetry_worst	\
0	0.7119	0.2654	0.4601	
1	0.2416	0.1860	0.2750	
2	0.4504	0.2430	0.3613	
3	0.6869	0.2575	0.6638	
4	0.4000	0.1625	0.2364	
5	0.5355	0.1741	0.3985	
6	0.3784	0.1932	0.3063	
7	0.2678	0.1556	0.3196	
8	0.5390	0.2060	0.4378	
9	1.1050	0.2210	0.4366	

```
   fractal_dimension_worst
0                  0.11890
1                  0.08902
2                  0.08758
3                  0.17300
4                  0.07678
5                  0.12440
6                  0.08368
7                  0.11510
8                  0.10720
9                  0.20750
```

0.2 2. Now the dataset seems to be printed.

0.3 Let's move on preprocessing where we clean unwanted columns and modify dataset to train basically we refurbuish our dataset.

1. Dropped the column id which is not required.

2.Then modified the diagnosis column with malignant and benign into numerical values ie (M=1, B=0)

3.Separate Features and Labels: Action: X contains all columns except 'diagnosis' (features), while y is the 'diagnosis' column (target variable). Basically in this step i seperate the features and labels ie diagnosis. What does this 3rd Step do?

It is required to seperate or variable or parameters which depends to predict either the diagnosis is malignant or not so i seperated the label or prediction value. Based on multiple parameter we will diagnose breast cancer either it is malignant and benign

4.Split the Data: Action: Divided the dataset into training (80%) and testing (20%) sets to train the model and evaluate its performance.

[110]:
```
# Check for missing values
print("\nMissing values:")
print(data.isnull().sum())

# Separate numeric and categorical columns
numeric_columns = data.select_dtypes(include=[np.number]).columns
categorical_columns = data.select_dtypes(exclude=[np.number]).columns

# Handle missing values
numeric_imputer = SimpleImputer(strategy='mean')
data[numeric_columns] = numeric_imputer.fit_transform(data[numeric_columns])

categorical_imputer = SimpleImputer(strategy='most_frequent')
data[categorical_columns] = categorical_imputer.
 ↪fit_transform(data[categorical_columns])

# Encode categorical variables
```

```
le = LabelEncoder()
for column in categorical_columns:
    data[column] = le.fit_transform(data[column])

# Ensure that 'id' is excluded when preparing data
X = data.drop(columns=['id', 'diagnosis'])  # Drop 'id' and 'diagnosis' (target)
y = data['diagnosis']  # Target variable

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

```
Missing values:
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
dtype: int64
```

4.All the predata processing work is done to load,clean,split and modify into numeric values as well as split for training and testing ie 80% training and 20% testing. After these all let's build a model to predict the cancerous cell . In this step you need to scale the features,print the shapes of our training and testing sets and display the summary statistics of preprocessed data.

[112]:
```
#from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Print the shapes of our training and testing sets
print("\nTraining features shape:", X_train_scaled.shape)
print("Testing features shape:", X_test_scaled.shape)
print("Training labels shape:", y_train.shape)
print("Testing labels shape:", y_test.shape)

# Display summary statistics of the preprocessed data
print("\nSummary statistics of the preprocessed features:")
print(pd.DataFrame(X_train_scaled, columns=X.columns).describe())
```

```
Training features shape: (455, 30)
Testing features shape: (114, 30)
Training labels shape: (455,)
Testing labels shape: (114,)

Summary statistics of the preprocessed features:
        radius_mean  texture_mean  perimeter_mean     area_mean  \
count  4.550000e+02  4.550000e+02    4.550000e+02  4.550000e+02
mean  -3.162306e-16 -3.513673e-17   -1.132183e-16  3.669836e-16
std    1.001101e+00  1.001101e+00    1.001101e+00  1.001101e+00
min   -1.819583e+00 -2.223500e+00   -1.809497e+00 -1.365036e+00
25%   -6.830930e-01 -7.075360e-01   -6.907613e-01 -6.602049e-01
50%   -2.314983e-01 -1.185158e-01   -2.429378e-01 -2.895973e-01
75%    4.593426e-01  5.631988e-01    4.884799e-01  3.193386e-01
max    3.961679e+00  4.715674e+00    3.976811e+00  5.208312e+00

       smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
count     4.550000e+02      4.550000e+02    4.550000e+02         4.550000e+02
mean     -2.381489e-16     -4.294489e-17   -5.953723e-17         2.225326e-16
std       1.001101e+00      1.001101e+00    1.001101e+00         1.001101e+00
min      -3.100011e+00     -1.607228e+00   -1.119899e+00        -1.269910e+00
25%      -7.132037e-01     -7.770872e-01   -7.505387e-01        -7.349048e-01
50%      -8.082013e-02     -2.413402e-01   -3.446456e-01        -3.911235e-01
75%       6.331729e-01      5.281282e-01    5.473870e-01         6.737569e-01
max       4.864642e+00      3.964311e+00    4.256736e+00         4.022271e+00
```

```
       symmetry_mean  fractal_dimension_mean     radius_se    texture_se  \
count   4.550000e+02            4.550000e+02  4.550000e+02  4.550000e+02
mean    1.444510e-16            5.055785e-16  9.564998e-17  7.027346e-17
std     1.001101e+00            1.001101e+00  1.001101e+00  1.001101e+00
min    -2.345430e+00           -1.776889e+00 -1.027104e+00 -1.556840e+00
25%    -7.010461e-01           -7.097920e-01 -5.911829e-01 -6.800068e-01
50%    -6.915087e-02           -1.772851e-01 -2.768816e-01 -1.989956e-01
75%     5.354290e-01            4.642225e-01  2.323998e-01  4.376096e-01
max     4.476124e+00            4.815921e+00  8.736037e+00  6.804586e+00

       perimeter_se       area_se  smoothness_se  compactness_se  \
count  4.550000e+02  4.550000e+02   4.550000e+02    4.550000e+02
mean   2.830459e-16  6.246530e-17  -7.808162e-18   -1.522592e-16
std    1.001101e+00  1.001101e+00   1.001101e+00    1.001101e+00
min   -1.015623e+00 -7.050913e-01  -1.727893e+00   -1.258102e+00
25%   -5.825488e-01 -4.641642e-01  -6.265241e-01   -6.943529e-01
50%   -2.761104e-01 -3.253470e-01  -1.994695e-01   -2.806072e-01
75%    1.992558e-01  7.743454e-02   3.536711e-01    3.583044e-01
max    9.242330e+00  1.064184e+01   7.906053e+00    5.905671e+00

       concavity_se  concave points_se   symmetry_se  fractal_dimension_se  \
count  4.550000e+02       4.550000e+02  4.550000e+02          4.550000e+02
mean   2.147245e-17      -2.147245e-16 -1.913000e-16          2.244847e-16
std    1.001101e+00       1.001101e+00  1.001101e+00          1.001101e+00
min   -1.022218e+00      -1.891775e+00 -1.554767e+00         -1.050856e+00
25%   -5.513403e-01      -6.684930e-01 -6.570543e-01         -5.739641e-01
50%   -2.078362e-01      -1.262792e-01 -2.270635e-01         -2.189082e-01
75%    3.033713e-01       4.375660e-01  3.242066e-01          2.453403e-01
max    1.131029e+01       6.504667e+00  5.008778e+00          9.345870e+00

       radius_worst  texture_worst  perimeter_worst    area_worst  \
count  4.550000e+02   4.550000e+02     4.550000e+02  4.550000e+02
mean  -7.027346e-17  -6.754060e-16    -2.928061e-17  1.815398e-16
std    1.001101e+00   1.001101e+00     1.001101e+00  1.001101e+00
min   -1.572438e+00  -2.230887e+00    -1.578174e+00 -1.152259e+00
25%   -6.616975e-01  -7.412292e-01    -6.853483e-01 -6.358132e-01
50%   -2.632354e-01  -5.210786e-02    -2.829543e-01 -3.357508e-01
75%    4.525400e-01   6.857059e-01     5.263332e-01  2.724862e-01
max    4.120889e+00   3.962127e+00     4.322305e+00  5.955420e+00

       smoothness_worst  compactness_worst  concavity_worst  \
count      4.550000e+02       4.550000e+02     4.550000e+02
mean      -1.561632e-17      -2.249727e-16    -7.612958e-17
std        1.001101e+00       1.001101e+00     1.001101e+00
min       -2.617938e+00      -1.455995e+00    -1.312795e+00
25%       -7.430298e-01      -6.961323e-01    -7.555873e-01
50%       -2.741590e-02      -2.753858e-01    -2.304110e-01
```

```
75%         6.296478e-01       5.738568e-01     5.383496e-01
max         3.767506e+00       4.424833e+00     4.672828e+00

       concave points_worst  symmetry_worst  fractal_dimension_worst
count          4.550000e+02    4.550000e+02             4.550000e+02
mean          -8.198570e-17    5.153387e-16             2.147245e-17
std            1.001101e+00    1.001101e+00             1.001101e+00
min           -1.749805e+00   -2.124261e+00            -1.616973e+00
25%           -7.700987e-01   -6.499846e-01            -7.189616e-01
50%           -2.386392e-01   -1.236838e-01            -2.135847e-01
75%            7.189995e-01    4.319440e-01             4.600643e-01
max            2.709674e+00    5.917679e+00             4.999482e+00
```

0.4 5. In this step we build the model and find out its accuracy

[113]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

model = LogisticRegression(random_state=42)
model.fit(X_train_scaled, y_train)

# Make predictions
y_pred = model.predict(X_test_scaled)
accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.2f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.97

Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.99      0.98        71
           1       0.98      0.95      0.96        43

    accuracy                           0.97       114
   macro avg       0.97      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114
```

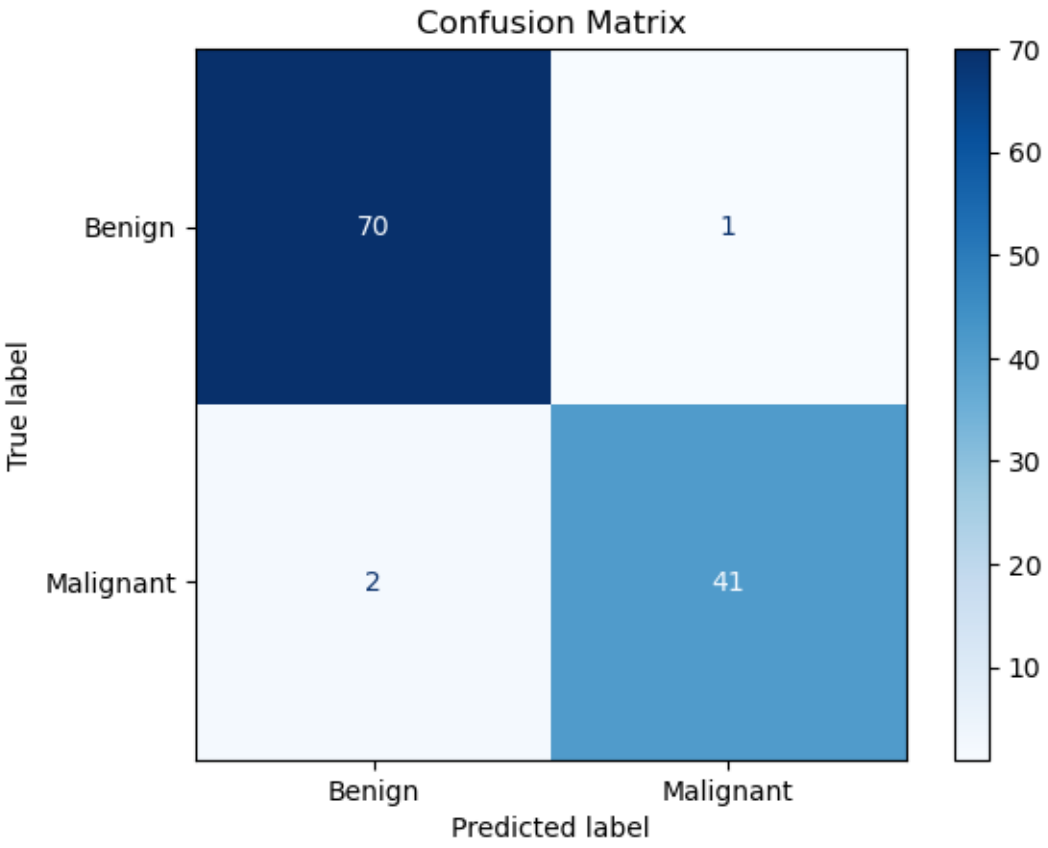
0.5 6. At this step we make all the predictions for the seperated 20% of the test data

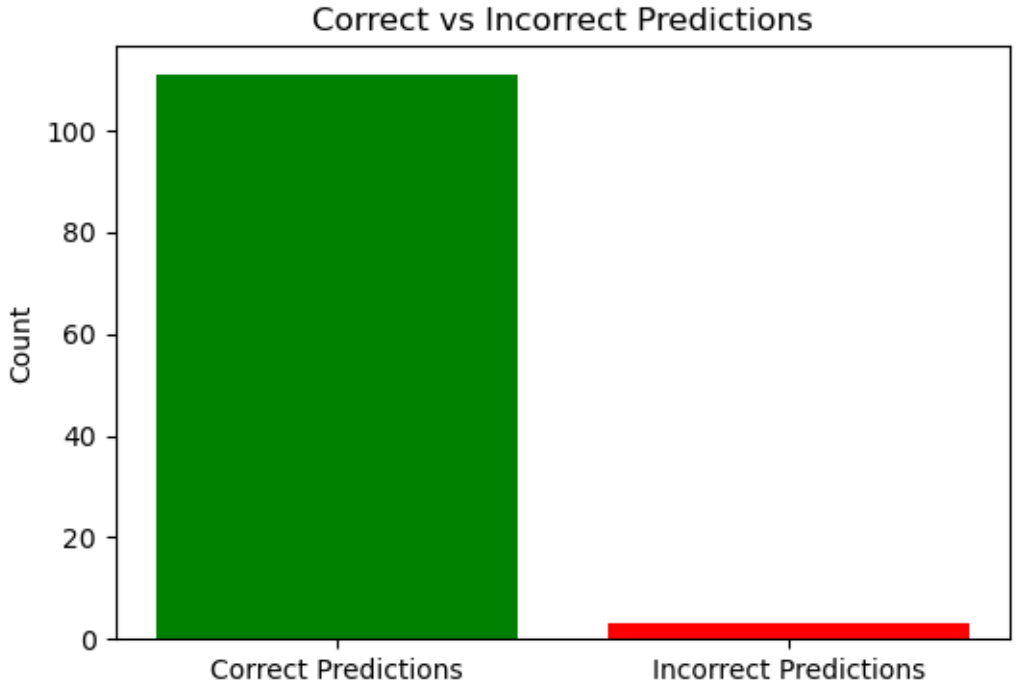
[ ]:
```
# Predicting for the test set
from sklearn.metrics import accuracy_score
predictions = model.predict(X_test_scaled)

# Mapping numeric predictions to labels

predicted_labels = ['Malignant' if pred == 1 else 'Benign' for pred in␣
↪predictions]

# Print results for each test sample
for i, (true_label, predicted_label) in enumerate(zip(y_test,␣
↪predicted_labels)):
    print(f"Sample {i+1}: Actual Diagnosis: {'Malignant' if true_label == 1␣
↪else 'Benign'}, Predicted: {predicted_label}")

def print_results(y_test, predicted_labels):
  """Prints the predictions and calculates accuracy.

  Args:
    y_test: The true labels.
    predicted_labels: The predicted labels.
  """

  # Print results for each test sample
  for i, (true_label, predicted_label) in enumerate(zip(y_test,␣
↪predicted_labels)):
      print(f"Sample {i+1}: Actual Diagnosis: {'Malignant' if true_label == 1␣
↪else 'Benign'}, Predicted: {predicted_label}")

  # Calculate and print accuracy
  accuracy = accuracy_score(y_test, predictions)
  print(f"Overall Accuracy: {accuracy:.2f}")

# Use the function
print_results(y_test, predicted_labels)
```

0.6 7. Use Confusion Matrix and show the accuracy of the model and plot the correct prediction and incorrect prediction

[147]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Step 1: Confusion Matrix Plot
def plot_confusion_matrix(y_test, predictions):
    """Plots the confusion matrix for predictions."""
```

```
    cm = confusion_matrix(y_test, predictions)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm,␣
 ↪display_labels=['Benign', 'Malignant'])

    # Plot confusion matrix heatmap
    disp.plot(cmap=plt.cm.Blues)
    plt.title('Confusion Matrix')
    plt.show()

# Step 2: Bar Plot for Correct and Incorrect Predictions
def plot_correct_incorrect(y_test, predicted_labels):
    """Plots a bar chart of correct and incorrect predictions."""
    correct = sum([1 for true, pred in zip(y_test, predictions) if true ==␣
 ↪pred])
    incorrect = len(y_test) - correct

    # Create bar plot
    plt.figure(figsize=(6, 4))
    plt.bar(['Correct Predictions', 'Incorrect Predictions'], [correct,␣
 ↪incorrect], color=['green', 'red'])
    plt.title('Correct vs Incorrect Predictions')
    plt.ylabel('Count')
    plt.show()

# Plot confusion matrix
plot_confusion_matrix(y_test, predictions)

# Plot correct and incorrect predictions
plot_correct_incorrect(y_test, predictions)
```


Confusion Matrix
Benign
Malignant
True label
70
1
2
41
Benign
Malignant
Predicted label
70
60
50
40
30
20
10


Correct vs Incorrect Predictions
Count
100
80
60
40
20
0
Correct Predictions
Incorrect Predictions

0.7 8. User Input Data to predict either the person is diagnosed with Malignant or Benign

Here the user provides the input for the same parameters as given below to predict and check manually either the prediction is correct or not.

[135]:
```
##Prediction Model for the input given by user
def predict_breast_cancer(model, scaler, input_data):
    try:
        # Convert whitespace-separated input to comma-separated list
        if isinstance(input_data, str):
            input_data = re.split(r'\s+', input_data)

        # Convert input data to DataFrame, excluding 'id' (input_data should␣
 ↪have 30 features)
        if isinstance(input_data, list):
            input_data = np.array(input_data).reshape(1, -1)

        input_data = pd.DataFrame(input_data, columns=scaler.feature_names_in_)

        # Scale the input data
        scaled_data = scaler.transform(input_data)

        # Make prediction
        prediction = model.predict(scaled_data)
        probability = model.predict_proba(scaled_data).max()

        # Interpret the result
        result = 'Malignant' if prediction[0] == 1 else 'Benign'

        return result, probability

    except Exception as e:
        print(f"An error occurred: {str(e)}")
        return None, None
```

[131]:
```
# Example input with whitespace-separated values
example_input_whitespace = "19.81        22.15        130        1260        0.
 ↪09831        0.1027        0.1479        0.09498        0.1582        0.
 ↪05395        0.7582        1.017        5.865        112.4        0.
 ↪006494        0.01893        0.03391        0.01521        0.01356        0.
 ↪001997        27.32        30.88        186.8        2398        0.
 ↪1512        0.315        0.5372        0.2388        0.2768        0.07615"

# Make the prediction
```

```
diagnosis, confidence = predict_breast_cancer(model, scaler,␣
 ↪example_input_whitespace)

if diagnosis is not None:
    print(f"Prediction: The breast mass is {diagnosis}")
    print(f"Confidence: {confidence:.2f}")
```

```
Prediction: The breast mass is Malignant
Confidence: 1.00
```

0.8 Concepts Learned while building this project

1. What ever problem occur find the reason it may be either syntax,logical or importing problems or even not handling the edge cases ie try catch and others as well.
2. Start off with printing the dataset after importing to make sure data is imported correctly.
3. Clean up the unwanted data with null values,unwanted columns for training data such as patient id and may contain other check carefully.
4. Next is to convert into numerical values if you have to for eg i had to convert ito 0's and 1's for Malignant and Benign diagnosis.
5. Split of the diagnosis column before training dataset ie what you need to find out.
6. Build a model using effective alogrithm here i used Logistic Regression which i learned about there are others as well if you are familiar check others out.
7. Note for this project(Linear Regression won't be able to predict accurately)
8. Use Feature Scaling to achieve faster convergence that means make the preprocessing of data within certain range which helps to train data without any large bais of prediction.
9. Learnt about Scikit learn which has inbuilt modules and function which make prediction and training very easy .
10. Print accuracy and test result of the 20% dataset which was seperated before training of model.
11. Learn't about Confusion matrix which has four parts ie true positive ,true negative ,false positive and false negative remember this is same as we studied assumption of hypothesis in mathematics in 3rd sem.
    - True Positive (TP): Correctly predicted positive instances.
    - True Negative (TN): Correctly predicted negative instances.
    - False Positive (FP): Incorrectly predicted positive instances (Type I error).
    - False Negative (FN): Incorrectly predicted negative instances (Type II error).
12. Lastly created a function which takes input from user and then predict the tumor cell is either malignant or benign