Breast Cancer Diagnosis Using k-Nearest Neighbors

Rajat Shubhra Biswas

University of Information Technology & Sciences (UITS)

Dhaka, Bangladesh

E-mail: rajatshubhra5010@gmail.com

Alif Ibne Alam

University of Information Technology & Sciences (UITS)

Dhaka, Bangladesh

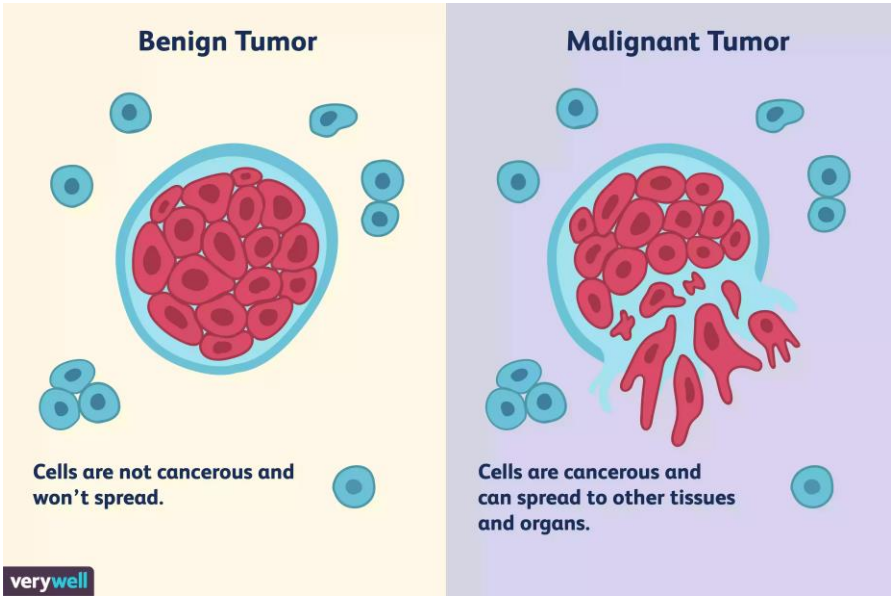
E-mail: alifibnealam2016@gmail.com

***Abstract*: Breast cancer is the most common invasive cancer in women, and the second main cause of cancer death in women, after lung cancer. Early can increase the chance of finding breast cancer before it spreads. We can predict whether the tumor is malignant or benign by using 'Machine Learning'. To do so, we can use k-Nearest Neighbors model. This paper contains the necessary steps to implement k-Nearest Neighbors model that is actually predicting whether the tumor is malignant or benign.**

Introduction

This is an implementation of k-Nearest Neighbors model in python to diagnosis breast Cancer classification. The implementation allows users to get to know whether it is in malignant or benign stage.

If you have been diagnosed with a tumor, the first step your doctor will take is to find out whether it is malignant or benign, as this will affect your treatment plan. In short, the meaning of malignant is cancerous and the meaning of benign is non-cancerous.



Necessary Steps to implement

Importing library:

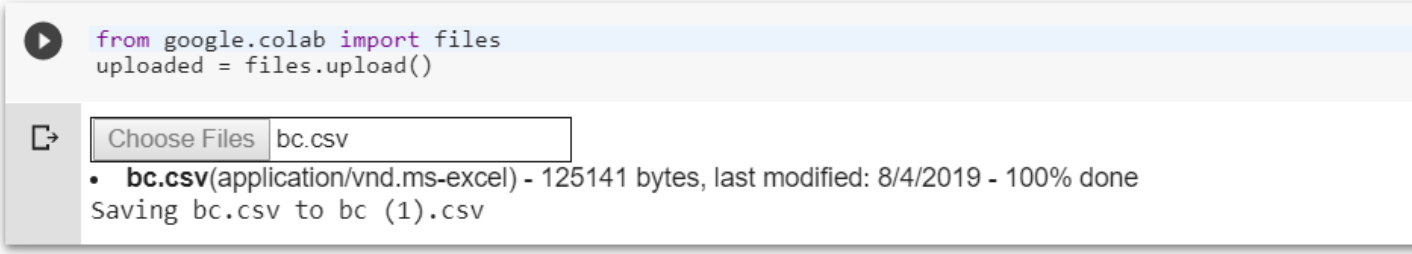
The first step is to import the necessary libraries so that we can use specific function of those libraries.

```
[ ]  # Packages for analysis
     import pandas as pd
     import numpy as np
     from sklearn import svm

     # Packages for visuals
     import matplotlib.pyplot as plt
     import seaborn as sns; sns.set(font_scale=1.2)

     # Allows charts to appear in the notebook
     %matplotlib inline

     # Pickle package
     import pickle
```

Importing the data dataset

To predict breast cancer some relevant dataset needs to be imported. To import the dataset from the storage of computer following code is to be executed.

```
from google.colab import files
uploaded = files.upload()
```




```
[ ]  import io
     dataset = pd.read_csv(io.BytesIO(uploaded['bc.csv']))
```

Preprocessing:

The next step is to split the dataset into its attributes and labels. To split the dataset, have to execute the following code:

```
[ ]  X = dataset.iloc[:, 2:31].values
     y = dataset.iloc[:, 1].values
```

Train Test Split

Training and testing on the same data is not an optimal approach, so we do split the data into two pieces, training set and testing set. We use 'train_test_split' function to split the data. Optional parameter 'test-size' determines the split percentage.

```
[ ]  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)
```

The above script splits the dataset into 80% train data and 20% test data. This means that out of total 150 records, the training set will contain 120 records and the test set contains 30 of those records.

feature scaling:

Before making any actual predictions, it is always a good practice to scale the features so that all of them can be uniformly evaluated. For feature scaling, have to execute the following code:

```
[ ]  from sklearn.preprocessing import StandardScaler
     scaler = StandardScaler()
     scaler.fit(X_train)

     X_train = scaler.transform(X_train)
     X_test = scaler.transform(X_test)
```

Training and prediction: It is extremely straight forward to train the KNN algorithm and make predictions with it, especially when using Scikit-Learn.

'KNeighborsClassifier' is to be imported first class from the 'sklearn.neighbors' library. In the second step this class is initialized with one parameter that is nothing but the value for the K. Here, we have taken k=5. The final step is to make predictions on our test data. Codes are:

```
[ ]  from sklearn.neighbors import KNeighborsClassifier
     classifier = KNeighborsClassifier(n_neighbors=5)
     classifier.fit(X_train, y_train)

     y_pred = classifier.predict(X_test)
```

Evaluating the Algorithm:

For evaluating an algorithm, confusion matrix, precision, recall and f1 score are the most commonly used metrics. The confusion_matrix and classification_report methods of the sklearn.metrics can be used to calculate these metrics. To evaluate the algorithm the following code is to be written:

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
result = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(result)
result1 = classification_report(y_test, y_pred)
print("Classification Report:",)
print (result1)
result2 = accuracy_score(y_test,y_pred)
print("Accuracy:",result2)
```

The output of the above script looks like this:

```
Confusion Matrix:
[[63  1]
 [ 4 46]]
Classification Report:
              precision    recall  f1-score   support

           B       0.94      0.98      0.96        64
           M       0.98      0.92      0.95        50

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

Accuracy: 0.956140350877193
```

The accuracy of the output is approximately 96%, that is satisfactory.

CONCLUSIONS:

Breast cancer can't be prevented. *But early detection* can increase the chance of finding breast cancer before it spreads and life can be saved. This model can predict whether tumor is malignant or benign with great accuracy that can be helpful for the doctor to take quick decision.

References:

https://www.verywellhealth.com/what-does-malignant-and-benign-mean-514240

https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)

https://www.medicalnewstoday.com/articles/37136.php