Breast Cancer Diagnosis Using Machine Learning: A KNN Classifier Approach

Aryan Singh
209310499
CSE: F

Introduction

- Breast cancer is a leading cause of death among women worldwide, with early diagnosis being critical for improved survival rates.
- Traditional methods of diagnosis can be subjective and time-consuming, leading to delays in treatment and increased patient anxiety.
- In recent years, machine learning algorithms have shown great promise in improving the accuracy and efficiency of breast cancer diagnosis.
- The KNN classifier algorithm is a popular and effective method for developing machine learning models for medical diagnosis.
- In this project, we explore the use of the KNN classifier algorithm to develop a model for diagnosing breast cancer, using the Breast Cancer Wisconsin (Diagnostic) dataset.

Project Description

Breast Cancer Classification using KNN

This project demonstrates an analysis of a breast cancer dataset using the K-Nearest Neighbors (KNN) classification algorithm. The dataset is loaded using the Scikit-learn library, and then preprocessed by splitting it into training and testing sets, scaling the features using the StandardScaler, and fitting a KNN classifier with a default train-test ratio of 75-25. The performance of the classifier is then evaluated using a confusion matrix, ROC curve, and a scatter plot and histogram for visualizing the data. The script also contains a loop that analyzes the classifier for different train-test ratios and calculates various performance metrics such as specificity, sensitivity, accuracy, precision, and F1-score. Overall, this project aims to develop a KNN-based classification model for breast cancer detection and to evaluate its performance using various performance metrics.

K-Nearest Neighbors (KNN) Classifier

K-Nearest Neighbors (KNN) Classifier

The KNN classifier algorithm is a popular machine learning algorithm that can be used for classification problems, such as the diagnosis of breast cancer. The algorithm works by comparing a new instance to all existing instances in the training dataset, and identifying the k nearest neighbors based on a similarity metric, such as Euclidean distance. The algorithm then assigns the new instance to the class that is most common among its k neighbors.

The KNN algorithm is a type of lazy learning algorithm, which means that it does not build a model during the training phase but instead, it stores the entire training dataset in memory. During the prediction phase, the algorithm calculates the distance between the input data point and all the training examples and selects the K closest neighbors. The class label of the input data point is then determined by the most frequent class label among the K neighbors.

KNN is a non-parametric algorithm, meaning it does not assume any underlying probability distribution of the input data. KNN is widely used in various applications such as image recognition, text classification, and recommendation systems.

Advantages

1.Simple and easy to understand: KNN is a straightforward algorithm that is easy to understand and implement, making it a good choice for beginners.

2.No assumption about data distribution: KNN does not make any assumptions about the distribution of data, making it suitable for datasets that are not normally distributed.

3.Non-parametric: KNN is a non-parametric algorithm, meaning that it does not make any assumptions about the underlying data distribution, making it more flexible than other parametric algorithms.

4.Can handle multi-class classification: KNN can handle multi-class classification problems with ease, as it simply assigns a class to a test instance based on the majority class of its nearest neighbors.

5.Robust to noisy data: KNN is relatively robust to noisy data, as it relies on the proximity of instances in feature space, rather than fitting a model to the data.

Limitations

1.Computationally expensive: KNN is computationally expensive, as it requires the calculation of distances between all instances in the training set and the test instance for each classification task.

2.Sensitive to the choice of distance metric: KNN is sensitive to the choice of distance metric used, and different distance metrics can produce different results.

3.Curse of dimensionality: KNN suffers from the curse of dimensionality, meaning that as the number of dimensions or features increases, the distance between instances becomes increasingly large, leading to a decrease in accuracy.

4.Requires a large amount of memory: KNN requires a large amount of memory to store the entire training dataset, which can be an issue for large datasets.

5.Imbalanced data can lead to poor performance: KNN can perform poorly on imbalanced datasets, where the majority class dominates the training set and the minority class is underrepresented.

Dataset Description

The Breast Cancer Wisconsin (Diagnostic) dataset is a public dataset that contains measurements from digitized images of breast tissue. The dataset includes information on 569 patients, with 30 attributes per patient, including patient ID, diagnosis, and various measures of the cells present in the tissue images. The dataset is widely used in machine learning studies for breast cancer diagnosis.

The dataset contains 569 instances, with each instance representing a patient with a diagnosis of either malignant or benign breast cancer. The diagnosis is provided in the "diagnosis" column, which is the target variable in our analysis.

SOURCE CODE

In [26]:
```python
import pandas as pd
import numpy as np
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, roc_curve, auc
import matplotlib.pyplot as plt
```

In [27]:
```python
# Load the dataset
data = load_breast_cancer()
X = data.data
y = data.target
```

In [28]:
```python
# Create a KNN classifier with random state 0 and default train test ratio
knn = KNeighborsClassifier(n_neighbors=5)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
knn.fit(X_train, y_train)
```

Out[28]:
```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

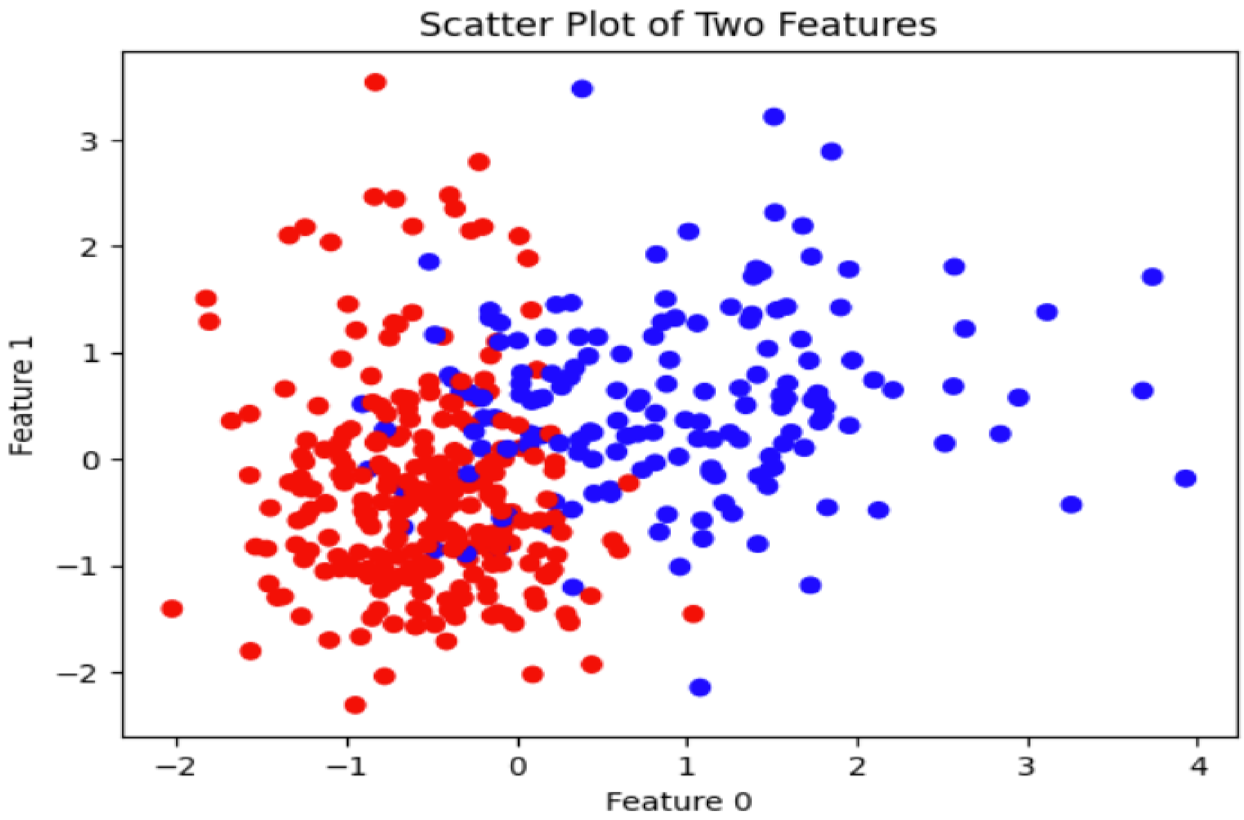
In [29]:
```python
# Evaluate the classifier using confusion matrix
y_pred = knn.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
print("Confusion matrix:\n", cm)
```

```
Confusion matrix:
 [[47  6]
 [ 1 89]]
```

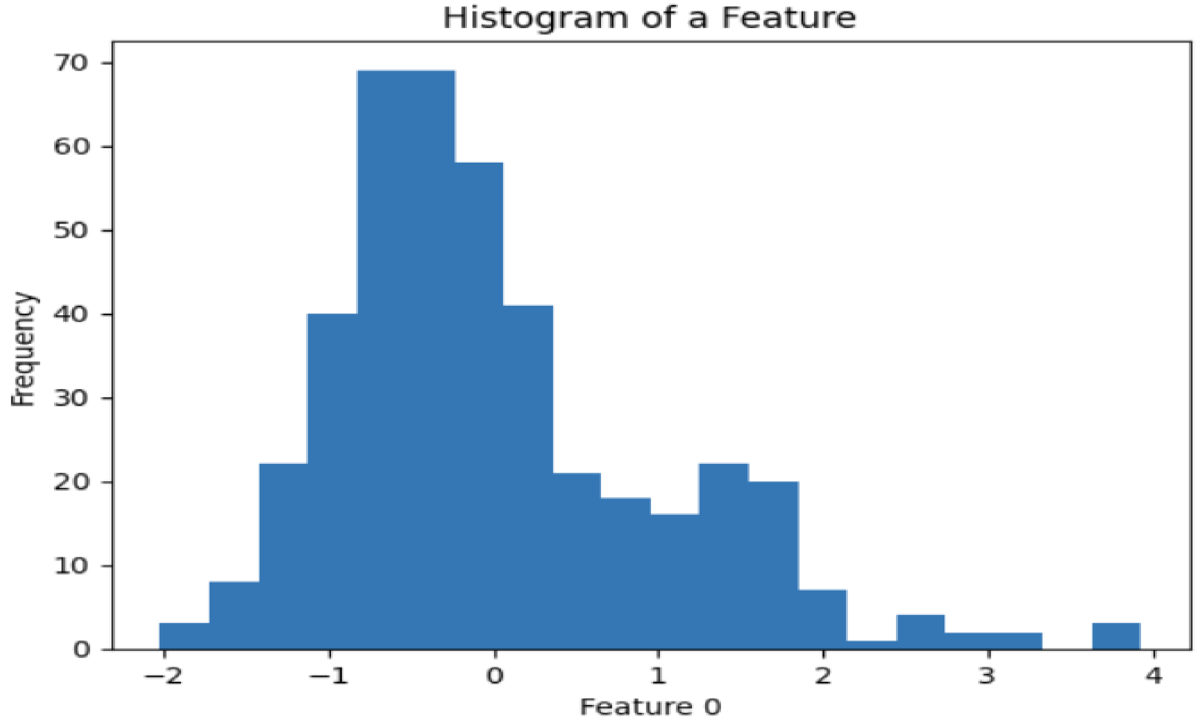
In [30]:
```python
# Evaluate the classifier using ROC curve
y_score = knn.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_score)
roc_auc = auc(fpr, tpr)
print("ROC AUC:", roc_auc)
```

```
ROC AUC: 0.9821802935010483
```

In [31]:
```
# Visualize the classifier using scatter plot and histogram
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap='bwr')
plt.xlabel('Feature 0')
plt.ylabel('Feature 1')
plt.title('Scatter Plot of Two Features')
plt.show()
```



In [32]:
```
plt.hist(X_train[:, 0], bins=20)
plt.xlabel('Feature 0')
plt.ylabel('Frequency')
plt.title('Histogram of a Feature')
plt.show()
```

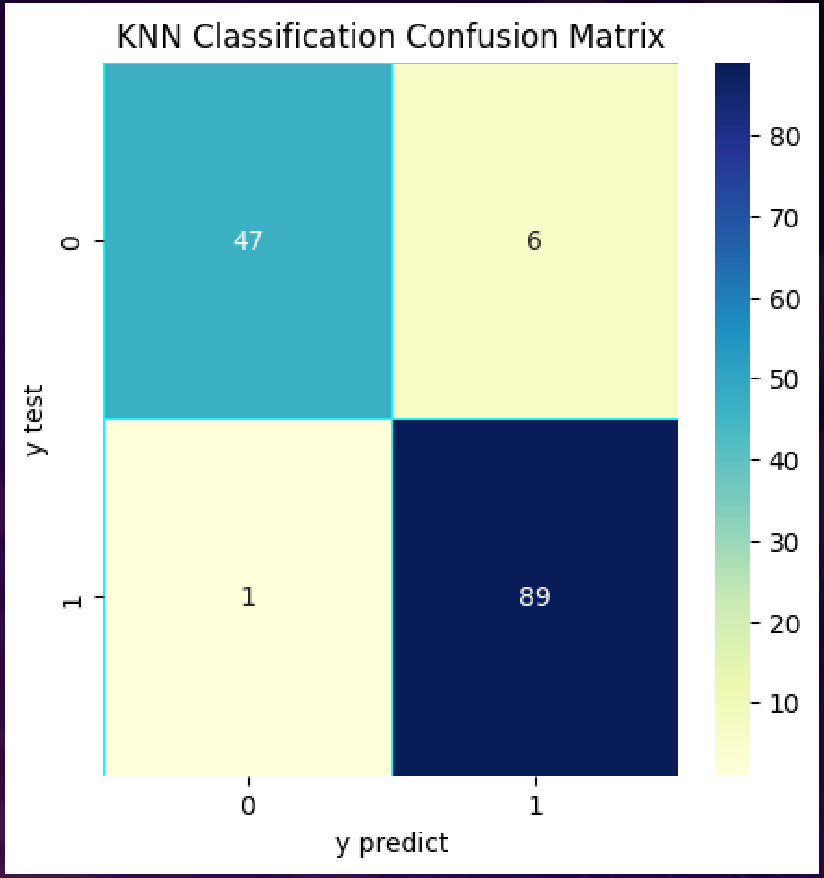


In [33]:
```
# Analyze the classifier for each train test ratio
train_test_ratios = [0.7, 0.8, 0.6]
for ratio in train_test_ratios:
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=ratio, random_state=0)
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
    specificity = tn / (tn + fp)
    sensitivity = tp / (tp + fn)
    accuracy = (tp + tn) / (tp + tn + fp + fn)
    precision = tp / (tp + fp)
    fpr = fp / (fp + tn)
    fnr = fn / (fn + tp)
    npv = tn / (tn + fn)
    fdr = fp / (fp + tp)
    f1_score = 2 * (precision * sensitivity) / (precision + sensitivity)
```

```
    fnr = fn / (fn + tp)
    npv = tn / (tn + fn)
    fdr = fp / (fp + tp)
    f1_score = 2 * (precision * sensitivity) / (precision + sensitivity)
    mcc = (tp*tn - fp*fn) / np.sqrt((tp+fp)*(tp+fn)*(tn+fp)*(tn+fn))
    print("Train-Test Ratio:", ratio)
    print("Specificity:", specificity)
    print("Sensitivity:", sensitivity)
    print("Accuracy:", accuracy)
    print("Precision:", precision)
    print("False Positive Rate:", fpr)
    print("False Negative Rate:", fnr)
    print("Negative Predictive Value:", npv)
    print("False Discovery Rate:", fdr)
    print("F1-Score:", f1_score)
    print("Matthews Correlation Coefficient:", mcc)
    print("\t")
```

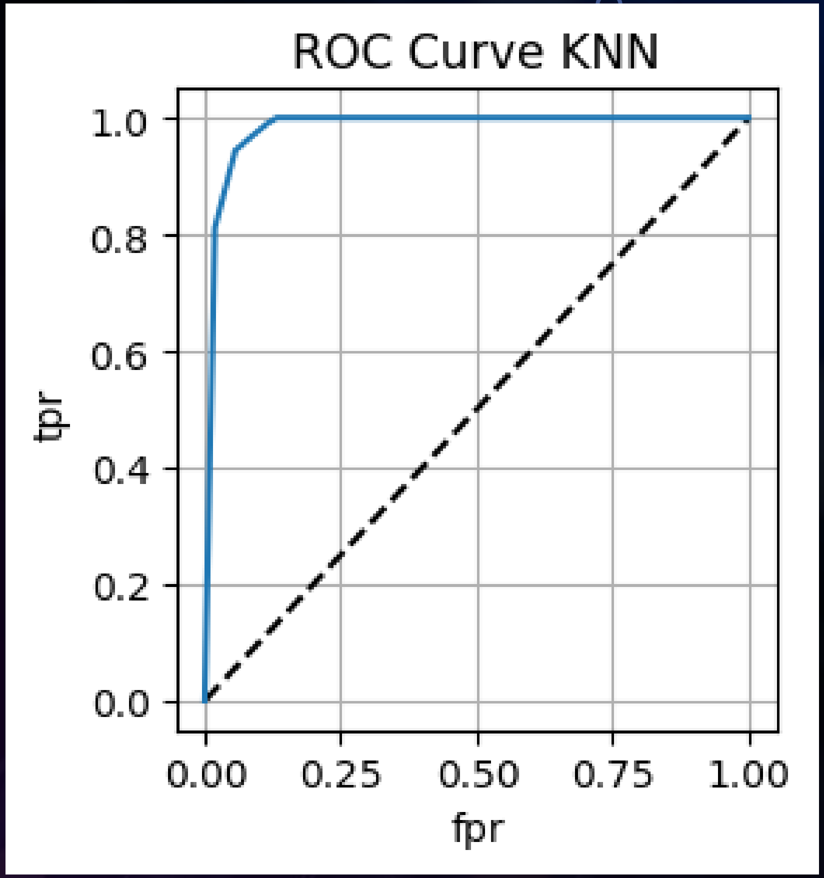
```
Train-Test Ratio: 0.7
Specificity: 0.8904109589041096
Sensitivity: 0.9841897233201581
Accuracy: 0.949874686716792
Precision: 0.9396226415094340
False Positive Rate: 0.1095890410958904
False Negative Rate: 0.015810276679841896
Negative Predictive Value: 0.9701492537313433
False Discovery Rate: 0.06037735849056604
F1-Score: 0.9613899613899612
Matthews Correlation Coefficient: 0.892012959685031

Train-Test Ratio: 0.8
Specificity: 0.8402366863905325
Sensitivity: 0.9860627177700348
Accuracy: 0.9320175438596491
Precision: 0.9129032258064517
False Positive Rate: 0.15976331360946747
False Negative Rate: 0.013937282229965157
Negative Predictive Value: 0.9726027397260274
False Discovery Rate: 0.08709677419354839
F1-Score: 0.9480737018425461
Matthews Correlation Coefficient: 0.8553905842947509

Train-Test Ratio: 0.6
Specificity: 0.8943089430894309
Sensitivity: 0.9908675799086758
Accuracy: 0.956140350877193
Precision: 0.9434782608695652
False Positive Rate: 0.10569105691056911
False Negative Rate: 0.0091324200913242
Negative Predictive Value: 0.9821428571428571
False Discovery Rate: 0.05652173913043478
F1-Score: 0.9665924276169265
Matthews Correlation Coefficient: 0.90517295742629
```

Result Analysis

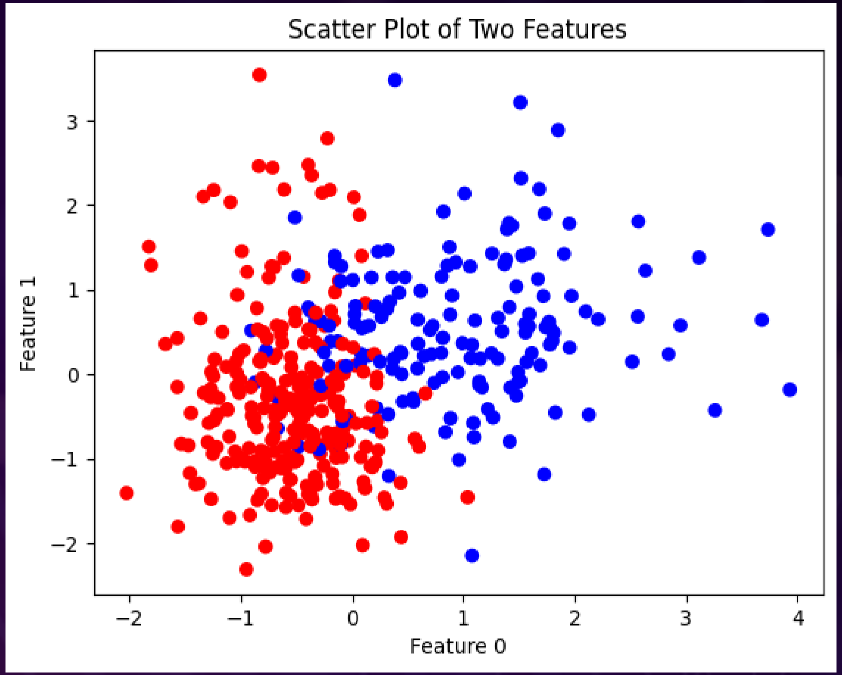


Confusion Matrix

ROC Curve

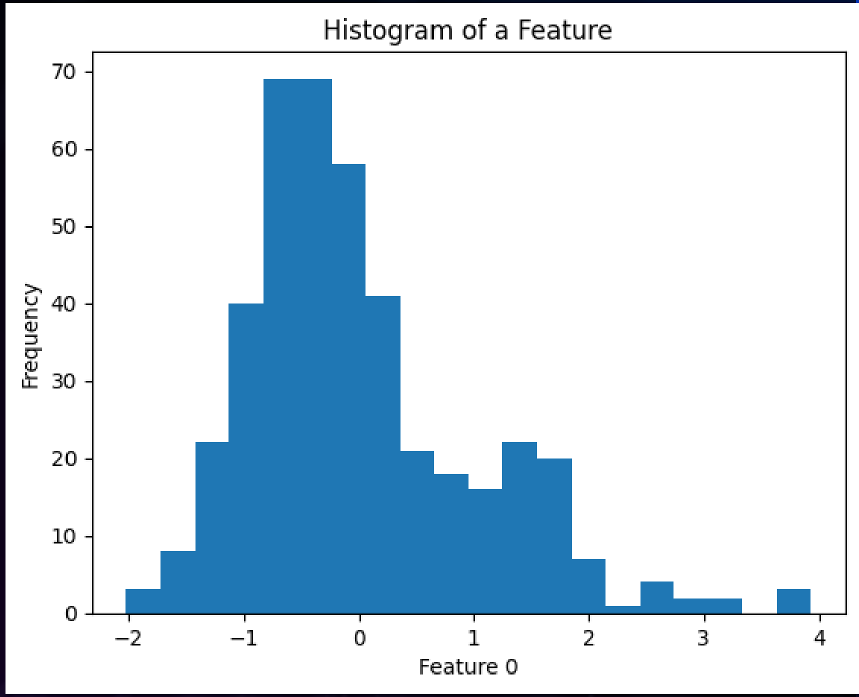


Data Visualization



Scatter Plot

Histogram



Evaluation

MEASURES	70:30 Train Test Ratio	80:20 Train Test Ratio	60:40 Train Test Ratio
Specifity	0.8904109589041096	0.8402366863905325	0.8943089430894309
Sensitivity	0.9841897233201581	0.9860627177700348	0.9908675799086758
Accuracy	0.949874686716792	0.9320175438596491	0.956140350877193
Precision	0.939622641509434	0.9129032258064517	0.9434782608695652
False Positive Rate	0.1095890410958904	0.15976331360946747	0.10569105691056911
False Negative Rate	0.015810276679841896	0.013937282229965157	0.0091324200913242
Negative Predicitve Value	0.9701492537313433	0.9726027397260274	0.9821428571428571
False Discovery Rate	0.06037735849056604	0.08709677419354839	0.05652173913043478
F1-Score	0.9613899613899612	0.9480737018425461	0.9665924276169265
Matthews Correlation Coefficient	0.892012959685031	0.8553905842947509	0.90517295742629

Thank you

Aryan Singh

209301499

CSE F

aryan.209301499@muj.manipal.edu