JADAVPUR UNIVERSITY

INFORMATION TECHNOLOGY

NAME : SURAJ ROY

ROLL : 002211001129

CLASS : 4th YEAR

SECTION : A3

ASSIGNMENT-1

GITHUB LINK : https://github.com/Suraj-Roy26/JU_ML_LAB.git

Problem Statement

In this Assignment, we aim to build and evaluate machine learning models for **classification** using two benchmark datasets :

1. **Iris Plants Dataset** – a widely used dataset for multi-class classification that involves predicting the species of an iris plant (Setosa, Versicolor, or Virginica) based on four features: sepal length, sepal width, petal length, and petal width.
2. **Wisconsin Breast Cancer (Diagnostic) Dataset** – a medical dataset where the task is to classify tumors as **benign** or **malignant** based on several features computed from a digitized image of a fine needle aspirate of a breast mass.

The primary objective is to **compare the performance** of different variants of the **Naive Bayes classifier**—specifically **Gaussian Naive Bayes, Multinomial Naive Bayes, and Bernoulli Naive Bayes**—on both datasets.

For each classifier and dataset combination, we will compute and analyze the following evaluation metrics:

- **Accuracy** – the overall proportion of correctly classified instances.
- **Precision** – the ability of the classifier to avoid false positives.
- **Recall** – the ability of the classifier to correctly identify all positive cases.
- **F1-score** – the harmonic mean of precision and recall, balancing both metrics.
- **Confusion Matrix** – a detailed breakdown of correct and incorrect predictions across classes.

By conducting this comparative study, we will gain insights into:

- How well Naive Bayes classifiers handle multi-class vs. binary classification problems.
- Which variant (Gaussian, Multinomial, or Bernoulli) performs better for different types of data distributions.
- The strengths and limitations of Naive Bayes in practical machine learning applications.

. At the first step, we work with the iris dataset.

```
import numpy as np                # Numerical operations, handling arrays, matrices, math functions
import pandas as pd               # Data manipulation and analysis (loading, cleaning, exploring datasets)
import matplotlib.pyplot as plt     # Plotting and visualizations (line, bar, scatter plots, etc.)

# Dataset Preparation
iris_df = pd.read_csv("/content/Iris.csv")
wbc_df  = pd.read_csv("/content/breast_cancer.csv")

# Separate features and target
iris_x = iris_df.drop(columns=['Species'])
iris_y = iris_df['Species']

# Split dataset into to splits one for train the model and with another test the model
from sklearn.model_selection import train_test_split
iris_x_train, iris_x_test, iris_y_train, iris_y_test = train_test_split( iris_x, iris_y, test_size=0.2, random_state=42)

from sklearn.naive_bayes import GaussianNB, MultinomialNB, BernoulliNB

# Initialize the models
model1 = GaussianNB()
model2 = MultinomialNB()
model3 = BernoulliNB()

#train the models
model1.fit(iris_x_train,iris_y_train)
model2.fit(iris_x_train,iris_y_train)
model3.fit(iris_x_train,iris_y_train)
```

```
# Find prediction with test features
iris_y_predict1 = model1.predict(iris_x_test)
iris_y_predict2 = model2.predict(iris_x_test)
iris_y_predict3 = model3.predict(iris_x_test)

# Evaluation of classifier performance

from sklearn.metrics import classification_report,confusion_matrix
print("GaussianNB")
print("Confusion matrix :")
print(confusion_matrix(iris_y_test,iris_y_predict1))
print("Performence Evaluation : ")
print(classification_report(iris_y_test,iris_y_predict1))
print("--------------------------------------------------------------------------")
print("MultinomialNB")
print("Confusion matrix :")
print(confusion_matrix(iris_y_test,iris_y_predict2))
print("Performence Evaluation : ")
print(classification_report(iris_y_test,iris_y_predict2))
print("--------------------------------------------------------------------------")
print("BernoulliNB")
print("Confusion matrix :")
print(confusion_matrix(iris_y_test,iris_y_predict3))
print("Performence Evaluation : ")
print(classification_report(iris_y_test,iris_y_predict3))
```

. It results

```
GaussianNB
Confusion matrix :
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
Performence Evaluation :
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       1.00      1.00      1.00         9
 Iris-virginica       1.00      1.00      1.00        11

       accuracy                           1.00        30
      macro avg       1.00      1.00      1.00        30
   weighted avg       1.00      1.00      1.00        30

------------------------------------------------------------------------
MultinomialNB
Confusion matrix :
[[ 9  1  0]
 [ 0  9  0]
 [ 0  1 10]]
Performence Evaluation :
                precision    recall  f1-score   support

    Iris-setosa       1.00      0.90      0.95        10
Iris-versicolor       0.82      1.00      0.90         9
 Iris-virginica       1.00      0.91      0.95        11

       accuracy                           0.93        30
      macro avg       0.94      0.94      0.93        30
   weighted avg       0.95      0.93      0.93        30

------------------------------------------------------------------------
BernoulliNB
Confusion matrix :
[[ 0 10  0]
 [ 0  9  0]
 [ 0 11  0]]
Performence Evaluation :
                precision    recall  f1-score   support

    Iris-setosa       0.00      0.00      0.00        10
Iris-versicolor       0.30      1.00      0.46         9
 Iris-virginica       0.00      0.00      0.00        11

       accuracy                           0.30        30
      macro avg       0.10      0.33      0.15        30
   weighted avg       0.09      0.30      0.14        30
```

Here I describe why accuracy is different for all this three models.

1. GaussianNB (Accuracy = 100%)

The Gaussian Naive Bayes model gave perfect classification for all three classes (Setosa, Versicolor, Virginica). This happened because:

- The Iris dataset consists of continuous numerical features (sepal length, sepal width, petal length, petal width).
- GaussianNB assumes that the features follow a normal distribution, which closely matches the nature of this dataset.

- As a result, the decision boundaries created by GaussianNB separated the classes very effectively, leading to 100% accuracy

2. MultinomialNB (Accuracy = 93%)

The Multinomial Naive Bayes model performed slightly worse compared to GaussianNB, giving 93% accuracy. From the confusion matrix, we can see:

- One Iris-setosa was misclassified as Versicolor.
- One Virginica was misclassified as Versicolor.
- The main reason is that MultinomialNB is designed for count-based features (e.g., word counts in text data), not continuous values.
- Since the Iris dataset has continuous numerical features, MultinomialNB does not model the data distribution as accurately, which caused a few misclassifications.
- Even though it still performed relatively well, it could not reach the same accuracy as GaussianNB.

3. BernoulliNB (Accuracy = 30%)

The Bernoulli Naive Bayes model gave very poor performance, with only 30% accuracy. The confusion matrix shows that:

- All Setosa and Virginica samples were misclassified.
- Only the Versicolor class was predicted correctly.
- This is because BernoulliNB works best with binary data (0 or 1, e.g., presence/absence features).
- In this case, the continuous numerical values of the Iris dataset were binarized (converted into 0s and 1s), which caused a huge loss of information.
- Since important distinctions between flower classes were lost during binarization, BernoulliNB failed to classify most of the data correctly.

We can see performance for BernoulliNB is not in the range 90%<=performance<=100%

.So we need to tune it's parameter alpha and binarize

```
model3 = BernoulliNB(alpha=0.5, binarize=2.0)
model3.fit(iris_x_train,iris_y_train)
iris_y_predict3 = model3.predict(iris_x_test)
print("BernoulliNB")
print("Confusion matrix :")
print(confusion_matrix(iris_y_test,iris_y_predict3))
```

```
print("Performence Evaluation : ")
print(classification_report(iris_y_test,iris_y_predict3))
```

```
BernoulliNB
Confusion matrix :
[[10  0  0]
 [ 0  9  0]
 [ 0  4  7]]
Performence Evaluation :
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       0.69      1.00      0.82         9
 Iris-virginica       1.00      0.64      0.78        11

       accuracy                           0.87        30
      macro avg       0.90      0.88      0.87        30
   weighted avg       0.91      0.87      0.86        30
```

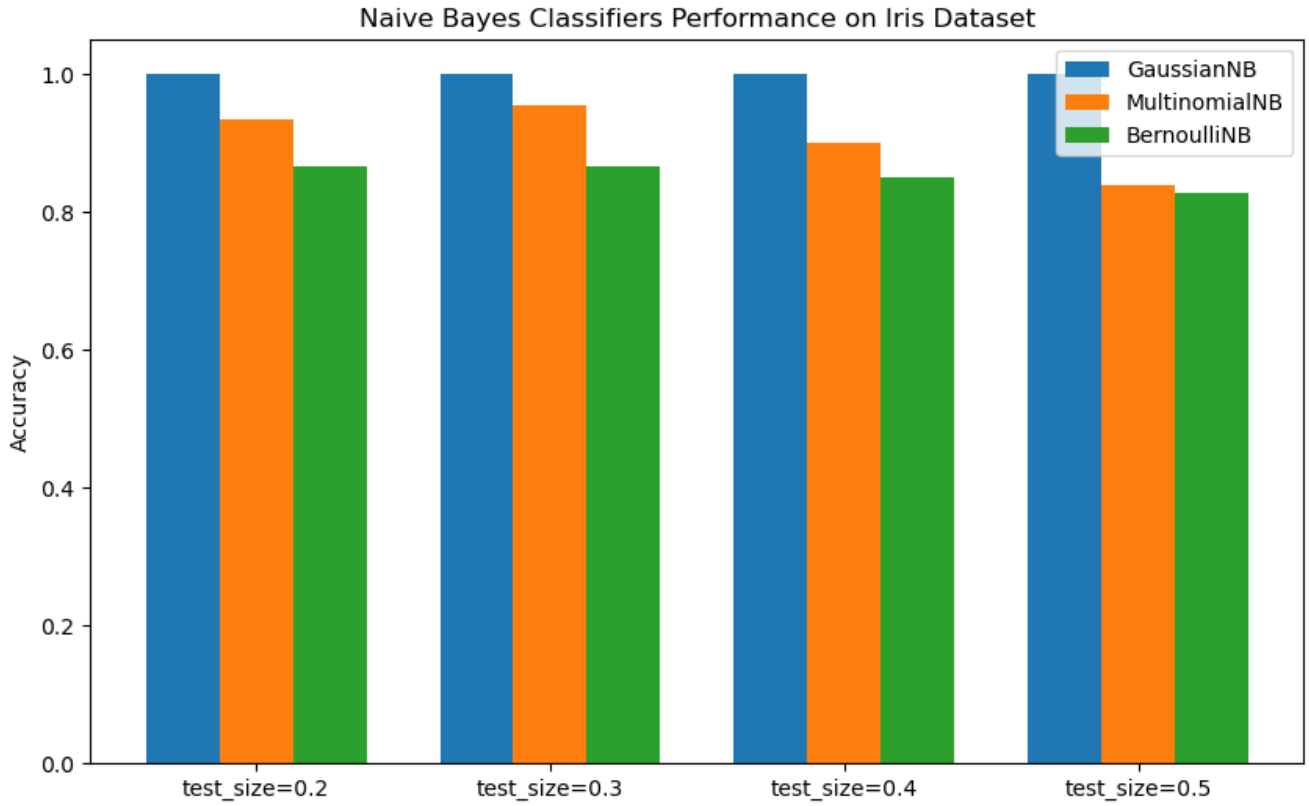
Now, the performance improves by tuning parameters.

Now we compare different split values over these three models, and plot them side by side in the graph to visualize their performance.

```
from sklearn.metrics import accuracy_score

splits = [0.2, 0.3, 0.4, 0.5]
results = {"GaussianNB": [], "MultinomialNB": [], "BernoulliNB": []}

for split in splits:
    # Split the data
    iris_x_train, iris_x_test, iris_y_train, iris_y_test = train_test_split( iris_x, iris_y, test_size = split, random_state=42)



    model1.fit(iris_x_train, iris_y_train)
    results["GaussianNB"].append(accuracy_score(iris_y_test, model1.predict(iris_x_test)))


    model2.fit(iris_x_train, iris_y_train)
```

```
    results["MultinomialNB"].append(accuracy_score(iris_y_test, model2.predict(iris_x_test)))


    model3.fit(iris_x_train, iris_y_train)
    results["BernoulliNB"].append(accuracy_score(iris_y_test, model3.predict(iris_x_test)))

x = np.arange(len(splits))
width = 0.25

print(results["GaussianNB"])
print(results["MultinomialNB"])
print(results["BernoulliNB"])
plt.figure(figsize=(10,6))
plt.bar(x - width, results["GaussianNB"], width, label='GaussianNB')
plt.bar(x, results["MultinomialNB"], width, label='MultinomialNB')
plt.bar(x + width, results["BernoulliNB"], width, label='BernoulliNB')
plt.xticks(x, [f"test_size={s}" for s in splits])
plt.ylim(0, 1.05)
plt.ylabel("Accuracy")
plt.title("Naive Bayes Classifiers Performance on Iris Dataset")
plt.legend()
plt.show()
```

```
[1.0, 1.0, 1.0, 1.0]
[0.9333333333333333, 0.9555555555555556, 0.9, 0.84]
[0.8666666666666667, 0.8666666666666667, 0.85, 0.8266666666666667]
```



SO here we can say the best split is always a higher one of testing percentage.

When we split the dataset into training set and test set using ratios like 0.2, 0.3, 0.4, and 0.5 (test set proportion), the size of the test set directly affects how well we can evaluate the model's performance.

1. Smaller Test Set (e.g., 20%)
    - The model is trained on a large portion of the data (80%).
    - Training accuracy is usually higher because the model has more data to learn from.
    - But the evaluation may not be reliable since only a small number of samples are used for testing. This means the accuracy might fluctuate more and not represent the true performance.
2. Larger Test Set (e.g., 40–50%)
    - The model is trained on a smaller portion of the data, so training may be slightly harder.
    - However, evaluation is done on a larger number of samples, which makes the accuracy more stable and trustworthy.
    - A bigger test set gives a better estimate of real-world performance, because the model is tested on more unseen data.
3. Why Higher Test Percentage Often Looks Better in Our Case
    - With more test data, the evaluation is more balanced and avoids random bias from a small sample.

- Sometimes, with smaller test sets, accuracy may look higher just due to chance, but with larger test sets, the accuracy reflects the true generalization power of the model.
- In short, a higher percentage test set improves the reliability of accuracy results, even if training becomes slightly harder.

Now we do the same for Breast Cancer data set.

```
wbc_x = wbc_df.drop(columns=['diagnosis'])
wbc_y = wbc_df['diagnosis']

wbc_x_train, wbc_x_test, wbc_y_train, wbc_y_test = train_test_split(
    wbc_x, wbc_y, test_size=0.2, random_state=42
)

# Initialize models
model1 = GaussianNB()
model2 = MultinomialNB()
model3 = BernoulliNB()

#train models
model1.fit(wbc_x_train,wbc_y_train)
model2.fit(wbc_x_train,wbc_y_train)
model3.fit(wbc_x_train,wbc_y_train)

# Find prediction with test features
wbc_y_predict1 = model1.predict(wbc_x_test)
wbc_y_predict2 = model2.predict(wbc_x_test)
wbc_y_predict3 = model3.predict(wbc_x_test)

print("GaussianNB")
print("Confusion matrix :")
print(confusion_matrix(wbc_y_test, wbc_y_predict1))
print("Performance Evaluation : ")
print(classification_report(wbc_y_test, wbc_y_predict1))
print("--------------------------------------------------------------------------------")
```

```
print("MultinomialNB")
print("Confusion matrix :")
print(confusion_matrix(wbc_y_test, wbc_y_predict2))
print("Performance Evaluation : ")
print(classification_report(wbc_y_test, wbc_y_predict2))
print("--------------------------------------------------------------------------------")

print("BernoulliNB")
print("Confusion matrix :")
print(confusion_matrix(wbc_y_test, wbc_y_predict3))
print("Performance Evaluation : ")
print(classification_report(wbc_y_test, wbc_y_predict3))
```

Now the result is

```
GaussianNB
Confusion matrix :
[[70  1]
 [43  0]]
Performance Evaluation :
              precision    recall  f1-score   support

           B       0.62      0.99      0.76        71
           M       0.00      0.00      0.00        43

    accuracy                           0.61       114
   macro avg       0.31      0.49      0.38       114
weighted avg       0.39      0.61      0.47       114

--------------------------------------------------------------------------------
MultinomialNB
Confusion matrix :
[[ 9 62]
 [ 7 36]]
Performance Evaluation :
              precision    recall  f1-score   support

           B       0.56      0.13      0.21        71
           M       0.37      0.84      0.51        43

    accuracy                           0.39       114
   macro avg       0.46      0.48      0.36       114
weighted avg       0.49      0.39      0.32       114

--------------------------------------------------------------------------------
BernoulliNB
Confusion matrix :
[[71  0]
 [43  0]]
Performance Evaluation :
              precision    recall  f1-score   support

           B       0.62      1.00      0.77        71
           M       0.00      0.00      0.00        43

    accuracy                           0.62       114
   macro avg       0.31      0.50      0.38       114
weighted avg       0.39      0.62      0.48       114
```

So we need to tune the models .

For model1 = GaussianNB(var_smoothing=1e-7) this it gives

```
GaussianNB
Confusion matrix :
[[70  1]
 [43  0]]
Performance Evaluation :
              precision    recall  f1-score   support

           B       0.62      0.99      0.76        71
           M       0.00      0.00      0.00        43

    accuracy                           0.61       114
   macro avg       0.31      0.49      0.38       114
weighted avg       0.39      0.61      0.47       114

--------------------------------------------------------------------------------
```

For 1e-7, 1e-8, 1e-9 it not results that much higher but for 1e-12 it give expected result

```
GaussianNB
Confusion matrix :
[[70  1]
 [ 7 36]]
Performance Evaluation :
              precision    recall  f1-score   support

           B       0.91      0.99      0.95        71
           M       0.97      0.84      0.90        43

    accuracy                           0.93       114
   macro avg       0.94      0.91      0.92       114
weighted avg       0.93      0.93      0.93       114

--------------------------------------------------------------------------------
```

Now we tune same way for other models

Initialize models

model1 = GaussianNB(var_smoothing=1e-12)

model2 = MultinomialNB(alpha=0.5)

model3 = BernoulliNB(alpha=0.5, binarize=3)

These are the best-tuned models resulting

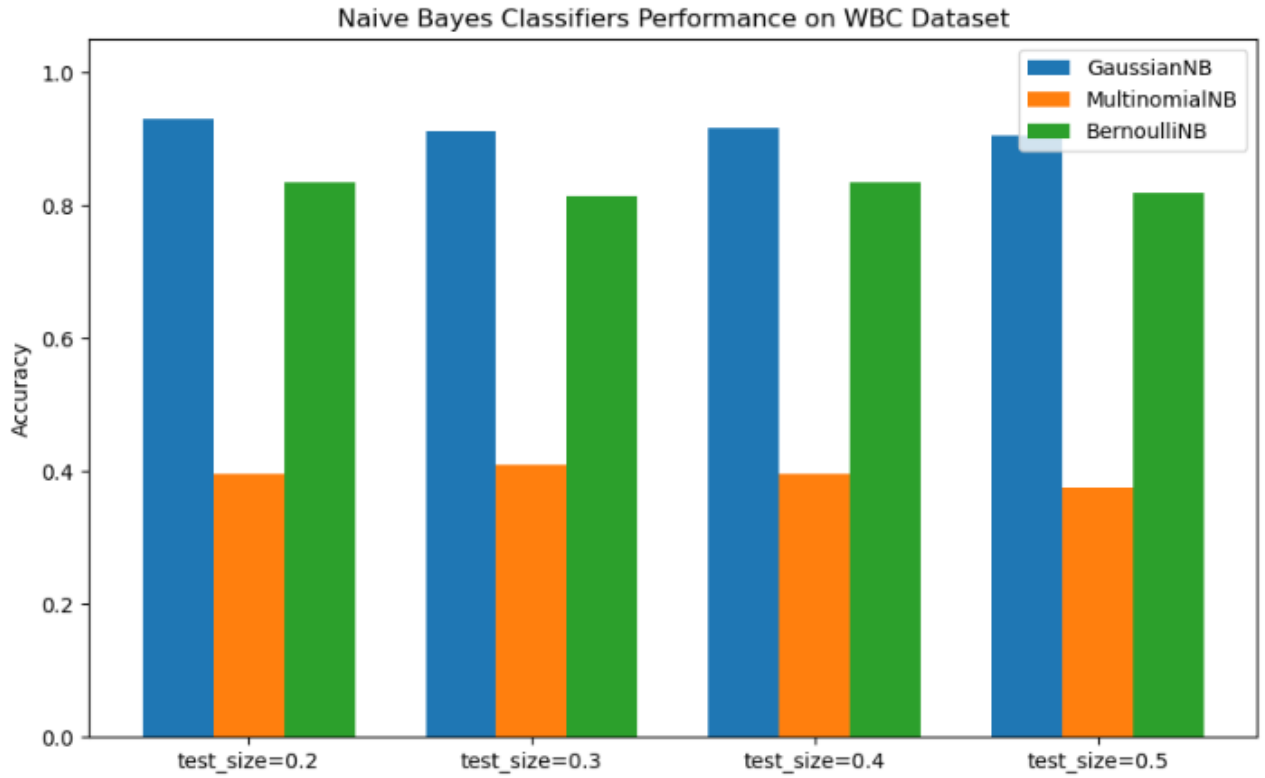
```
GaussianNB
Confusion matrix :
[[70  1]
 [ 7 36]]
Performance Evaluation :
              precision    recall  f1-score   support

           B       0.91      0.99      0.95        71
           M       0.97      0.84      0.90        43

    accuracy                           0.93       114
   macro avg       0.94      0.91      0.92       114
weighted avg       0.93      0.93      0.93       114

--------------------------------------------------------------------------------
MultinomialNB
Confusion matrix :
[[ 9 62]
 [ 7 36]]
Performance Evaluation :
              precision    recall  f1-score   support

           B       0.56      0.13      0.21        71
           M       0.37      0.84      0.51        43

    accuracy                           0.39       114
   macro avg       0.46      0.48      0.36       114
weighted avg       0.49      0.39      0.32       114

--------------------------------------------------------------------------------
BernoulliNB
Confusion matrix :
[[63  8]
 [11 32]]
Performance Evaluation :
              precision    recall  f1-score   support

           B       0.85      0.89      0.87        71
           M       0.80      0.74      0.77        43

    accuracy                           0.83       114
   macro avg       0.83      0.82      0.82       114
weighted avg       0.83      0.83      0.83       114
```

Now plot the accuracies for different splits

```
splits = [0.2, 0.3, 0.4, 0.5]
results = {"GaussianNB": [], "MultinomialNB": [], "BernoulliNB": []}


for split in splits:
    wbc_x_train, wbc_x_test, wbc_y_train, wbc_y_test = train_test_split(
        wbc_x, wbc_y, test_size=split, random_state=42
    )


    # GaussianNB
    model1.fit(wbc_x_train, wbc_y_train)
    results["GaussianNB"].append(accuracy_score(wbc_y_test, model1.predict(wbc_x_test)))
```

```
    # MultinomialNB
    model2.fit(wbc_x_train, wbc_y_train)
    results["MultinomialNB"].append(accuracy_score(wbc_y_test, model2.predict(wbc_x_test)))

    # BernoulliNB
    model3.fit(wbc_x_train, wbc_y_train)
    results["BernoulliNB"].append(accuracy_score(wbc_y_test, model3.predict(wbc_x_test)))

# Print accuracies
print("GaussianNB:", results["GaussianNB"])
print("MultinomialNB:", results["MultinomialNB"])
print("BernoulliNB:", results["BernoulliNB"])

# Plotting
x = np.arange(len(splits))
width = 0.25

plt.figure(figsize=(10,6))
plt.bar(x - width, results["GaussianNB"], width, label='GaussianNB')
plt.bar(x, results["MultinomialNB"], width, label='MultinomialNB')
plt.bar(x + width, results["BernoulliNB"], width, label='BernoulliNB')

plt.xticks(x, [f"test_size={s}" for s in splits])
plt.ylim(0, 1.05)
plt.ylabel("Accuracy")
plt.title("Naive Bayes Classifiers Performance on WBC Dataset")
plt.legend()
plt.show()
```

```
GaussianNB: [0.9298245614035088, 0.9122807017543859, 0.9166666666666666, 0.9052631578947369]
MultinomialNB: [0.39473684210526316, 0.4093567251461988, 0.39473684210526316, 0.37543859649122807]
BernoulliNB: [0.8333333333333334, 0.8128654970760234, 0.8333333333333334, 0.8175438596491228]
```



GaussianNB

The confusion matrix shows that most of the predictions are correct, with only a few misclassifications. The accuracy is around 93%, which is the best among the three models. This is because GaussianNB assumes that the features follow a normal (Gaussian) distribution. Since our dataset contains continuous values, GaussianNB fits the data well, which is why the accuracy is high.

MultinomialNB

The performance of MultinomialNB is very poor with only 39% accuracy. This is because MultinomialNB is mainly designed for discrete/count data like word frequencies in text classification problems. Our dataset has continuous numerical values, so it does not suit the assumptions of this model, which results in very low accuracy and poor classification.

BernoulliNB

The BernoulliNB model gives around 83% accuracy, which is lower than GaussianNB but still much better than MultinomialNB. BernoulliNB is designed for binary features (0s and 1s). Even though our dataset is continuous, it can still work reasonably well after thresholding the values internally, but it does not capture the distribution of the features as accurately as GaussianNB.

Question 2:

The objective of this experiment is to apply the Decision Tree Classifier on two well-known datasets: the Iris dataset and the Breast Cancer dataset. The goal is to train classification models that can automatically predict the correct class labels based on the input features.

For the Iris dataset, the problem is to classify flowers into three species (*Setosa, Versicolor, and Virginica*) using measurements such as sepal length, sepal width, petal length, and petal width.

For the Breast Cancer dataset, the problem is to predict whether a tumor is Benign (non-cancerous) or Malignant (cancerous) using various clinical features derived from breast mass measurements.

The Decision Tree classifier will be implemented using both Gini Index and Entropy criteria to evaluate how well the data is split at each node. The performance of the model will be assessed using metrics such as Accuracy, Precision, Recall, F1-score, and Confusion Matrix. Finally, visualization of the decision trees will be generated to highlight important splitting rules and to interpret the decision-making process of the classifier.

```
from sklearn.tree import DecisionTreeClassifier


iris_x_train, iris_x_test, iris_y_train, iris_y_test = train_test_split(
    iris_x, iris_y, test_size=0.7, random_state=42
)

# Initialize the models (Gini and Entropy)
model_gini = DecisionTreeClassifier(criterion="gini", random_state=42)
model_entropy = DecisionTreeClassifier(criterion="entropy", random_state=42)

# Train the models
model_gini.fit(iris_x_train, iris_y_train)
model_entropy.fit(iris_x_train, iris_y_train)

# Predictions
iris_y_predict_gini = model_gini.predict(iris_x_test)
iris_y_predict_entropy = model_entropy.predict(iris_x_test)

# Evaluation of classifier performance
print("Decision Tree (Gini)")
print("Confusion matrix :")
print(confusion_matrix(iris_y_test, iris_y_predict_gini))
```

```
print("Performance Evaluation :")
print(classification_report(iris_y_test, iris_y_predict_gini))
print("--------------------------------------------------------------------------")
print("Decision Tree (Entropy)")
print("Confusion matrix :")
print(confusion_matrix(iris_y_test, iris_y_predict_entropy))
print("Performance Evaluation :")
print(classification_report(iris_y_test, iris_y_predict_entropy))
```

```
Decision Tree (Gini)
Confusion matrix :
[[40  0  0]
 [ 0 33  0]
 [ 0  0 32]]
Performance Evaluation :
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        40
Iris-versicolor       1.00      1.00      1.00        33
 Iris-virginica       1.00      1.00      1.00        32

       accuracy                           1.00       105
      macro avg       1.00      1.00      1.00       105
   weighted avg       1.00      1.00      1.00       105

--------------------------------------------------------------------------
Decision Tree (Entropy)
Confusion matrix :
[[40  0  0]
 [ 0 33  0]
 [ 0  0 32]]
Performance Evaluation :
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        40
Iris-versicolor       1.00      1.00      1.00        33
 Iris-virginica       1.00      1.00      1.00        32

       accuracy                           1.00       105
      macro avg       1.00      1.00      1.00       105
   weighted avg       1.00      1.00      1.00       105
```

I use 30% train data and 70% test data, still it perform very well. So it same for all other splits <0.8

Decision Trees are powerful enough to perfectly split the data because the dataset is low-dimensional and clean .

If here I use 20% train data and 80% test data then it give less than 100% performance.Like below

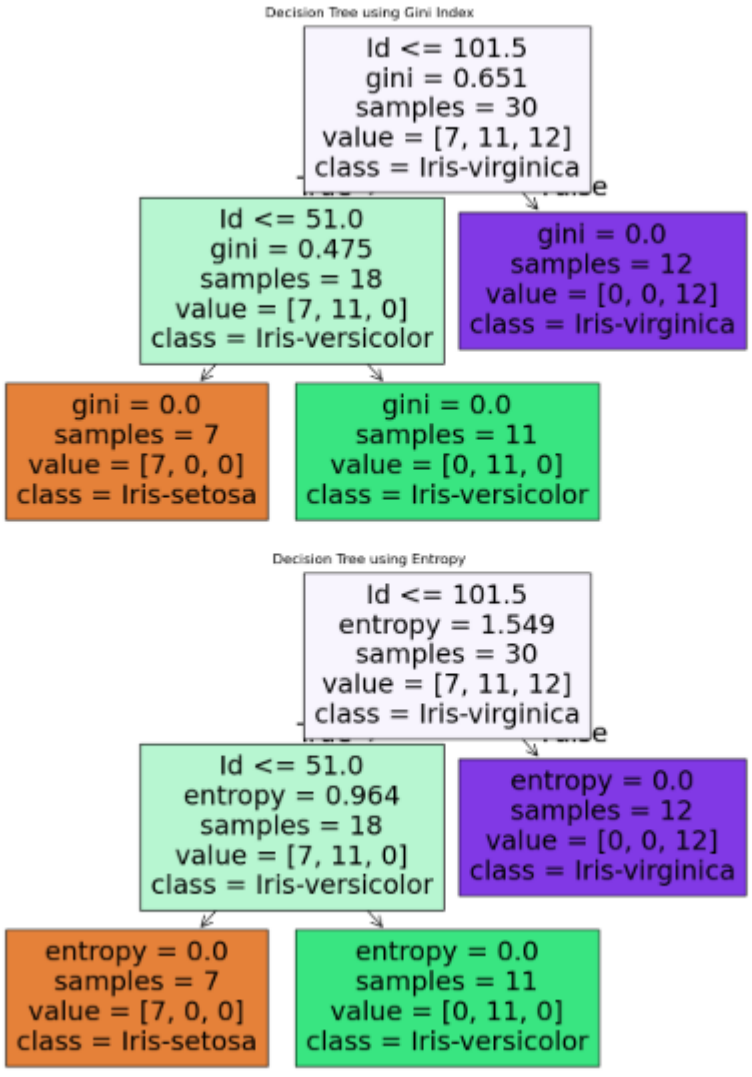
```
iris_x_train, iris_x_test, iris_y_train, iris_y_test = train_test_split(
    iris_x, iris_y, test_size=0.8, random_state=42
)
```

```
Decision Tree (Gini)
Confusion matrix :
[[43  0  0]
 [ 1 38  0]
 [ 0  1 37]]
Performance Evaluation :
                 precision    recall  f1-score   support

    Iris-setosa       0.98      1.00      0.99        43
Iris-versicolor       0.97      0.97      0.97        39
 Iris-virginica       1.00      0.97      0.99        38

       accuracy                           0.98       120
      macro avg       0.98      0.98      0.98       120
   weighted avg       0.98      0.98      0.98       120

------------------------------------------------------------------------
Decision Tree (Entropy)
Confusion matrix :
[[43  0  0]
 [ 1 38  0]
 [ 0  1 37]]
Performance Evaluation :
                 precision    recall  f1-score   support

    Iris-setosa       0.98      1.00      0.99        43
Iris-versicolor       0.97      0.97      0.97        39
 Iris-virginica       1.00      0.97      0.99        38

       accuracy                           0.98       120
      macro avg       0.98      0.98      0.98       120
   weighted avg       0.98      0.98      0.98       120
```

Now plot tree

```
from sklearn.tree import plot_tree


# ------------------ Plot Gini Tree ------------------
plt.figure(figsize=(12,8))
plot_tree(
    model_gini,
    filled=True,
    feature_names = iris_x.columns.tolist(),
    class_names = iris_y.unique().astype(str)
)
plt.title("Decision Tree using Gini Index")
plt.show()


# ------------------ Plot Entropy Tree ------------------
plt.figure(figsize=(12,8))
```

```
plot_tree(
    model_entropy,
    filled=True,
    feature_names = iris_x.columns.tolist(),
    class_names = iris_y.unique().astype(str)
)
plt.title("Decision Tree using Entropy")
plt.show()
```



Here is text representation of the tree

```
from sklearn import tree
text_presentation_gini_iris = tree.export_text(model_gini1)
print(text_presentation_gini_iris)
```

```
|--- feature_0 <= 101.50
|   |--- feature_0 <= 51.00
|   |   |--- class: Iris-setosa
|   |--- feature_0 >  51.00
|   |   |--- class: Iris-versicolor
|--- feature_0 >  101.50
|   |--- class: Iris-virginica
```

```
text_presentation_entropy_iris = tree.export_text(model_entropy1)
print(text_presentation_entropy_iris)
```

```
|--- feature_0 <= 101.50
|   |--- feature_0 <= 51.00
|   |   |--- class: Iris-setosa
|   |--- feature_0 >  51.00
|   |   |--- class: Iris-versicolor
|--- feature_0 >  101.50
|   |--- class: Iris-virginica
```

Now we do the same for Breast Cancer data set.

```
wbc_x_train, wbc_x_test, wbc_y_train, wbc_y_test = train_test_split(
    wbc_x, wbc_y, test_size=0.8, random_state=42
)

# Initialize models
model_gini = DecisionTreeClassifier(criterion="gini", random_state=42)
model_entropy = DecisionTreeClassifier(criterion="entropy", random_state=42)

# Train the models
model_gini.fit(wbc_x_train, wbc_y_train)
model_entropy.fit(wbc_x_train, wbc_y_train)

# Predictions
wbc_y_predict_gini = model_gini.predict(wbc_x_test)
wbc_y_predict_entropy = model_entropy.predict(wbc_x_test)

# Evaluation of classifier performance
```

```
print("Decision Tree (Gini)")
print("Confusion Matrix :")
print(confusion_matrix(wbc_y_test, wbc_y_predict_gini))
print("Performance Evaluation :")
print(classification_report(wbc_y_test, wbc_y_predict_gini))
print("--------------------------------------------------------------------------------")
print("Decision Tree (Entropy)")
print("Confusion Matrix :")
print(confusion_matrix(wbc_y_test, wbc_y_predict_entropy))
print("Performance Evaluation :")
print(classification_report(wbc_y_test, wbc_y_predict_entropy))
```

```
Decision Tree (Gini)
Confusion Matrix :
[[262  28]
 [ 19 147]]
Performance Evaluation :
              precision    recall  f1-score   support

           B       0.93      0.90      0.92       290
           M       0.84      0.89      0.86       166

    accuracy                           0.90       456
   macro avg       0.89      0.89      0.89       456
weighted avg       0.90      0.90      0.90       456

--------------------------------------------------------------------------------
Decision Tree (Entropy)
Confusion Matrix :
[[261  29]
 [ 12 154]]
Performance Evaluation :
              precision    recall  f1-score   support

           B       0.96      0.90      0.93       290
           M       0.84      0.93      0.88       166

    accuracy                           0.91       456
   macro avg       0.90      0.91      0.90       456
weighted avg       0.91      0.91      0.91       456
```

Performance are same kind of for splits till 0.8, 0.7, 0.6, 0.5, 0.4, 0.3

When I trained and tested the Decision Tree classifier on the WBC (Breast Cancer) dataset, I tried different splits of training and testing data such as 80-20, 70-30, 60-40, 50-50, 40-60, and 30-70.

Surprisingly, the performance in all these cases was above 90% accuracy, and the confusion matrix and classification report also showed very good results.

This happened mainly because:

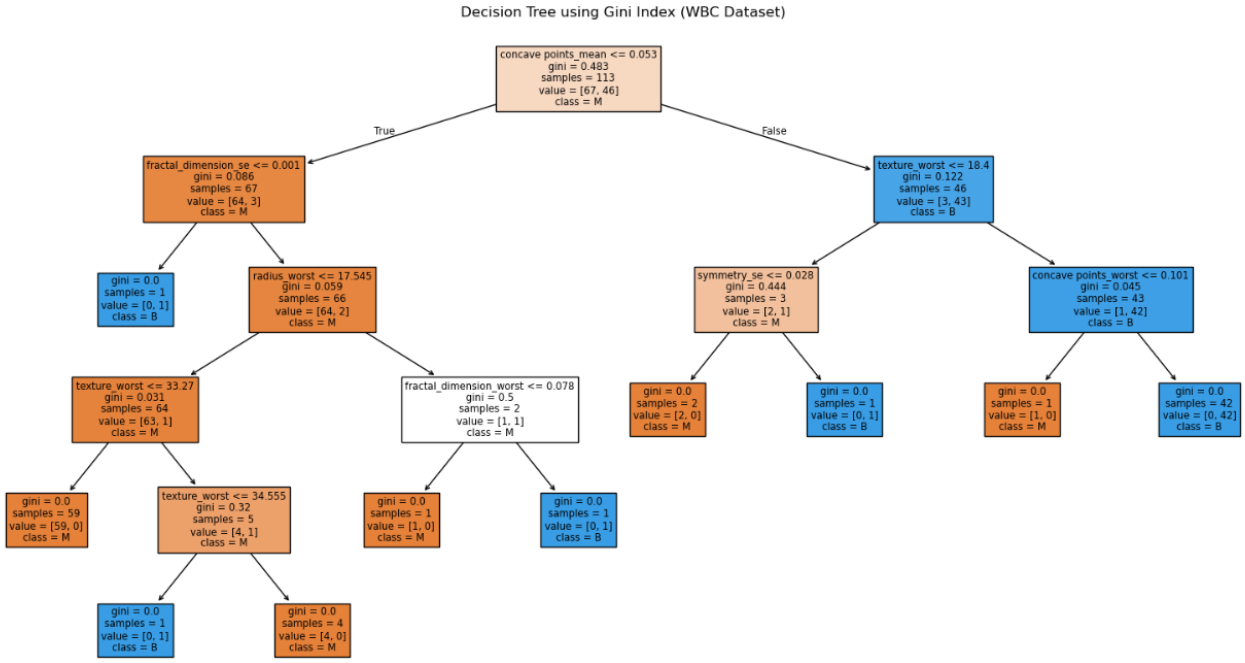
1. Dataset quality – The breast cancer dataset is very well-structured and the features are highly correlated with the target variable (diagnosis: malignant or benign). That means the model can easily learn patterns.
2. Balanced dataset – The dataset does not suffer from extreme imbalance, so even when the training size is reduced, the classifier still generalizes well.
3. Decision Tree power – Decision trees are quite strong on this dataset because they can split features in a way that almost perfectly separates the classes.
4. Sufficient information in small samples – Even when we give less data for training (like 30%), the remaining data still contains enough information for the model to learn good decision rules.

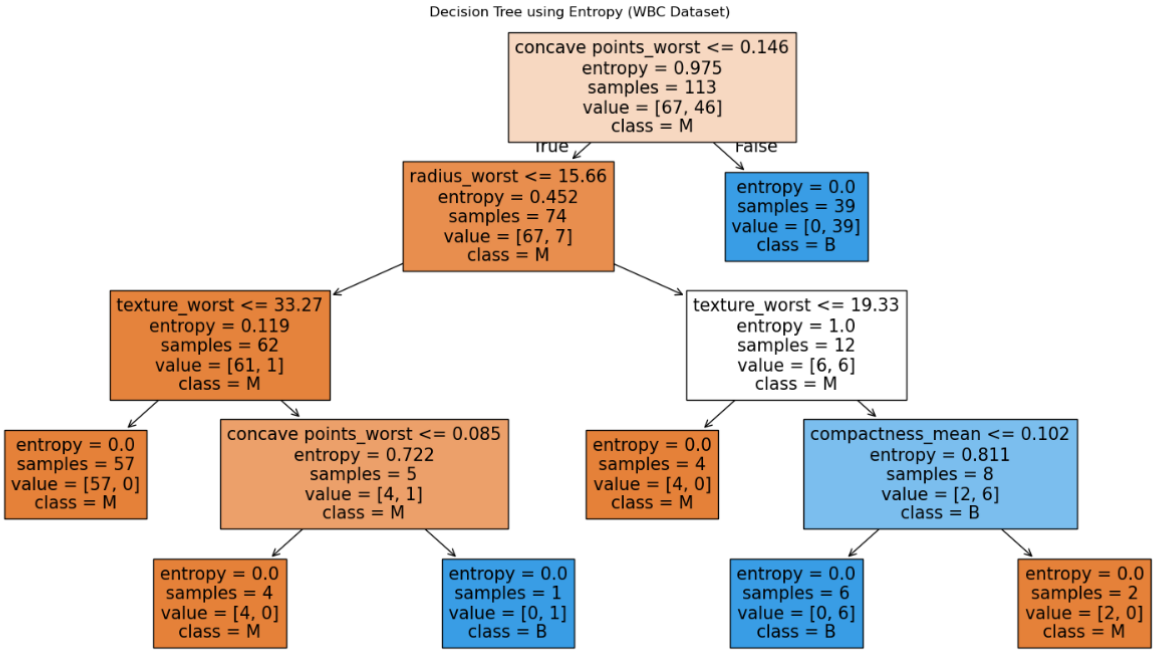
So overall, no matter which split ratio I used, the decision tree achieved high accuracy. This shows that the dataset is very predictive and reliable, and decision trees are a suitable model for this classification problem.

Make Decision Tree for this

```
from sklearn.tree import plot_tree


# ------------------ Plot Gini Tree ------------------
plt.figure(figsize=(20,10))
plot_tree(
    model_gini,
    filled=True,
    feature_names = wbc_x.columns.tolist(),
    class_names = wbc_y.unique().astype(str)
)
plt.title("Decision Tree using Gini Index (WBC Dataset)")
plt.show()


# ------------------ Plot Entropy Tree ------------------
plt.figure(figsize=(20,10))
plot_tree(
```

```
    model_entropy,
    filled=True,
    feature_names = wbc_x.columns.tolist(),
    class_names = wbc_y.unique().astype(str)
)
plt.title("Decision Tree using Entropy (WBC Dataset)")
plt.show()
```





The models give perfect performance, so no need to tune them.

Here is text representation of the tree

```
text_presentation_gini_breast_cancer = tree.export_text(model_gini2)
print(text_presentation_gini_breast_cancer)
```

```
|--- feature_8 <= 0.05
|   |--- feature_20 <= 0.00
|   |   |--- class: M
|   |--- feature_20 >  0.00
|   |   |--- feature_21 <= 17.55
|   |   |   |--- feature_22 <= 33.27
|   |   |   |   |--- class: B
|   |   |   |--- feature_22 >  33.27
|   |   |   |   |--- feature_22 <= 34.56
|   |   |   |   |   |--- class: M
|   |   |   |   |--- feature_22 >  34.56
|   |   |   |   |   |--- class: B
|   |   |--- feature_21 >  17.55
|   |   |   |--- feature_30 <= 0.08
|   |   |   |   |--- class: B
|   |   |   |--- feature_30 >  0.08
|   |   |   |   |--- class: M
|--- feature_8 >  0.05
|   |--- feature_22 <= 18.40
|   |   |--- feature_19 <= 0.03
|   |   |   |--- class: B
|   |   |--- feature_19 >  0.03
|   |   |   |--- class: M
|   |--- feature_22 >  18.40
|   |   |--- feature_28 <= 0.10
|   |   |   |--- class: B
|   |   |--- feature_28 >  0.10
|   |   |   |--- class: M
```

```
text_presentation_entropy_breast_cancer = tree.export_text(model_entropy2)
print(text_presentation_entropy_breast_cancer)
```

```
|--- feature_28 <= 0.15
|   |--- feature_21 <= 15.66
|   |   |--- feature_22 <= 33.27
|   |   |   |--- class: B
|   |   |--- feature_22 >  33.27
|   |   |   |--- feature_28 <= 0.09
|   |   |   |   |--- class: B
|   |   |   |--- feature_28 >  0.09
|   |   |   |   |--- class: M
|   |--- feature_21 >  15.66
|   |   |--- feature_22 <= 19.33
|   |   |   |--- class: B
|   |   |--- feature_22 >  19.33
|   |   |   |--- feature_6 <= 0.10
|   |   |   |   |--- class: M
|   |   |   |--- feature_6 >  0.10
|   |   |   |   |--- class: B
|--- feature_28 >  0.15
|   |--- class: M
```