Machine Learning Lab A3

ASIM KUMAR HANSDA

ROLL NO - 002211001136

ASSIGNMENT - 1

Github Link:

https://github.com/cryptasim/MACHINE-LEARNING-LAB

Open in Colab

IRIS Dataset

Classification: Decision Tree

In [1]:
```
# import pandas, numpy, and matplotlib.pyplot libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
# Install the ucimlrepo library
!pip install ucimlrepo
```

```
Collecting ucimlrepo
  Downloading ucimlrepo-0.0.7-py3-none-any.whl.metadata (5.5 kB)
Requirement already satisfied: pandas>=1.0.0 in /usr/local/lib/python3.11/dist-
packages (from ucimlrepo) (2.2.2)
Requirement already satisfied: certifi>=2020.12.5 in /usr/local/lib/python3.11/
dist-packages (from ucimlrepo) (2025.8.3)
Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/python3.11/dist-
packages (from pandas>=1.0.0->ucimlrepo) (2.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python
3.11/dist-packages (from pandas>=1.0.0->ucimlrepo) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-p
ackages (from pandas>=1.0.0->ucimlrepo) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dis
t-packages (from pandas>=1.0.0->ucimlrepo) (2025.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packa
ges (from python-dateutil>=2.8.2->pandas>=1.0.0->ucimlrepo) (1.17.0)
Downloading ucimlrepo-0.0.7-py3-none-any.whl (8.0 kB)
Installing collected packages: ucimlrepo
Successfully installed ucimlrepo-0.0.7
```

In [3]:
```
from ucimlrepo import fetch_ucirepo

# fetch dataset
iris_dataset = fetch_ucirepo(id=53)

# data (as pandas dataframes)
X1 = iris_dataset.data.features
y1 = iris_dataset.data.targets

# metadata
print(iris_dataset.metadata)

# variable information
print(iris_dataset.variables)
```

```
{'uci_id': 53, 'name': 'Iris', 'repository_url': 'https://archive.ics.uci.edu/d
ataset/53/iris', 'data_url': 'https://archive.ics.uci.edu/static/public/53/dat
a.csv', 'abstract': 'A small classic dataset from Fisher, 1936. One of the earl
iest known datasets used for evaluating classification methods.\n', 'area': 'Bi
ology', 'tasks': ['Classification'], 'characteristics': ['Tabular'], 'num_insta
nces': 150, 'num_features': 4, 'feature_types': ['Real'], 'demographics': [],
'target_col': ['class'], 'index_col': None, 'has_missing_values': 'no', 'missin
g_values_symbol': None, 'year_of_dataset_creation': 1936, 'last_updated': 'Tue
Sep 12 2023', 'dataset_doi': '10.24432/C56C76', 'creators': ['R. A. Fisher'],
'intro_paper': {'ID': 191, 'type': 'NATIVE', 'title': 'The Iris data set: In se
arch of the source of virginica', 'authors': 'A. Unwin, K. Kleinman', 'venue':
'Significance, 2021', 'year': 2021, 'journal': 'Significance, 2021', 'DOI': '17
40-9713.01589', 'URL': 'https://www.semanticscholar.org/paper/4599862ea87786366
9a6a8e63a3c707a787d5d7e', 'sha': None, 'corpus': None, 'arxiv': None, 'mag': No
ne, 'acl': None, 'pmid': None, 'pmcid': None}, 'additional_info': {'summary':
'This is one of the earliest datasets used in the literature on classification
methods and widely used in statistics and machine learning.  The data set conta
ins 3 classes of 50 instances each, where each class refers to a type of iris p
lant.  One class is linearly separable from the other 2; the latter are not lin
early separable from each other.\n\nPredicted attribute: class of iris plan
t.\n\nThis is an exceedingly simple domain.\n\nThis data differs from the data
presented in Fishers article (identified by Steve Chadwick,  spchadwick@espeeda
z.net ).  The 35th sample should be: 4.9,3.1,1.5,0.2,"Iris-setosa" where the er
ror is in the fourth feature. The 38th sample: 4.9,3.6,1.4,0.1,"Iris-setosa" wh
ere the errors are in the second and third features.  ', 'purpose': 'N/A', 'fun
ded_by': None, 'instances_represent': 'Each instance is a plant', 'recommende
d_data_splits': None, 'sensitive_data': None, 'preprocessing_description': Non
e, 'variable_info': None, 'citation': None}}
```



```
           name     role         type demographic  \
0  sepal length  Feature   Continuous        None
1   sepal width  Feature   Continuous        None
2  petal length  Feature   Continuous        None
3   petal width  Feature   Continuous        None
4         class   Target  Categorical        None

                                         description units missing_values
0                                               None    cm             no
1                                               None    cm             no
2                                               None    cm             no
3                                               None    cm             no
4  class of iris plant: Iris Setosa, Iris Versico...  None             no
```

In [4]:

```python
from sklearn.model_selection import train_test_split
# Split the data into training and testing sets.
# X1 contains the features and y1 contains the target variable.
# test_size=0.20 means 20% of the data will be used for testing.
X_train, X_test, y_train, y_test = train_test_split (X1, y1, test_size = 0.20)
```

In [5]:

```python
# Classification using Decision Tree
from sklearn.tree import DecisionTreeClassifier

# Initialize the Decision Tree classifier
classifier = DecisionTreeClassifier()
```

```python
# Train the classifier using the training data
classifier.fit (X_train, y_train)

# Make predictions on the test data
y_pred = classifier.predict(X_test)
```

In [6]:
```python
# Evaluation of Classifier Performance
from sklearn.metrics import classification_report, confusion_matrix

# Print the confusion matrix to show the number of correct and incorrect predi
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("-------------------------------------------------------------------")
print("-------------------------------------------------------------------")

# Print the classification report to show key classification metrics (precisio
print("Performance Evaluation:")
print (classification_report(y_test, y_pred))
```

```
Confusion Matrix:
[[ 8  0  0]
 [ 0  8  1]
 [ 0  1 12]]
-------------------------------------------------------------------
-------------------------------------------------------------------
Performance Evaluation:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         8
Iris-versicolor       0.89      0.89      0.89         9
 Iris-virginica       0.92      0.92      0.92        13

       accuracy                           0.93        30
      macro avg       0.94      0.94      0.94        30
   weighted avg       0.93      0.93      0.93        30
```

Confusion Matrix: It shows that all 8 'Iris-setosa', 8 'Iris-versicolor' (with 1 misclassified as Iris-virginica), and 12 'Iris-virginica' (with 1 misclassified as Iris-versicolor) instances in this specific test set were classified, with only two errors overall.

Performance Evaluation: Precision, recall, and f1-scores are 1.00 for Iris-setosa, 0.89 for Iris-versicolor, and 0.92 for Iris-virginica, resulting in an overall accuracy of 0.93.

In [7]:
```python
from sklearn.tree import DecisionTreeClassifier

# Initialize the Decision Tree classifier with 'entropy' criterion and a maxim
classifier = DecisionTreeClassifier (criterion="entropy", max_depth=3)
```

```
# Train the classifier using the training data
classifier.fit(X_train, y_train)
```

Out[7]:

```
▾ DecisionTreeClassifier

DecisionTreeClassifier(criterion='entropy', max_depth=3)
```

In [8]:

```
# Print the confusion matrix to show the number of correct and incorrect predi
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("-------------------------------------------------------------------")
print("-------------------------------------------------------------------")

# Print the classification report to show key classification metrics (precisio
print("Performance Evaluation:")
print (classification_report(y_test, y_pred))
```

```
Confusion Matrix:
[[ 8  0  0]
 [ 0  8  1]
 [ 0  1 12]]
-------------------------------------------------------------------
-------------------------------------------------------------------
Performance Evaluation:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         8
Iris-versicolor       0.89      0.89      0.89         9
 Iris-virginica       0.92      0.92      0.92        13

       accuracy                           0.93        30
      macro avg       0.94      0.94      0.94        30
   weighted avg       0.93      0.93      0.93        30
```

Confusion Matrix: It shows that all 8 'Iris-setosa', 8 'Iris-versicolor' (with 1 misclassified as Iris-virginica), and 12 'Iris-virginica' (with 1 misclassified as Iris-versicolor) instances in this specific test set were classified, with only two errors overall.

Performance Evaluation: Precision, recall, and f1-scores are 1.00 for Iris-setosa, 0.89 for Iris-versicolor, and 0.92 for Iris-virginica, resulting in an overall accuracy of 0.93.

In [9]:

```
# Import the Decision Tree classifier
from sklearn.tree import DecisionTreeClassifier

# Initialize Decision Tree classifier with entropy criterion and max depth of
classifier = DecisionTreeClassifier (criterion="entropy", max_depth=10)

# Train the classifier
```

```
classifier.fit(X_train, y_train)
```

Out[9]:

```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=10)
```

In [10]:

```
# Print Confusion Matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("-------------------------------------------------------------------")
print("-------------------------------------------------------------------")
# Print Performance Evaluation (Classification Report)
print("Performance Evaluation:")
print (classification_report(y_test, y_pred))
```

```
Confusion Matrix:
[[ 8  0  0]
 [ 0  8  1]
 [ 0  1 12]]
-------------------------------------------------------------------
-------------------------------------------------------------------
Performance Evaluation:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         8
Iris-versicolor       0.89      0.89      0.89         9
 Iris-virginica       0.92      0.92      0.92        13

       accuracy                           0.93        30
      macro avg       0.94      0.94      0.94        30
   weighted avg       0.93      0.93      0.93        30
```

Confusion Matrix: It shows that all 8 'Iris-setosa', 8 'Iris-versicolor', and 12 'Iris-virginica' instances in this specific test set were classified correctly, with 1 Iris-versicolor misclassified as Iris-virginica and 1 Iris-virginica misclassified as Iris-versicolor.

Performance Evaluation: Precision, recall, and f1-scores are 1.00 for Iris-setosa, 0.89 for Iris-versicolor, and 0.92 for Iris-virginica, resulting in an overall accuracy of 0.93.

In [11]:

```
# Import the Decision Tree classifier
from sklearn.tree import DecisionTreeClassifier

# Initialize Decision Tree classifier with gini criterion and max depth of 10
classifier = DecisionTreeClassifier(criterion="gini", max_depth=10)

# Train the classifier
classifier.fit(X_train, y_train)
```

Out[11]:

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=10)
```

In [12]:

```
# Print Confusion Matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("-------------------------------------------------------------------")
print("-------------------------------------------------------------------")
# Print Performance Evaluation (Classification Report)
print("Performance Evaluation:")
print (classification_report(y_test, y_pred))
```

```
Confusion Matrix:
[[ 8  0  0]
 [ 0  8  1]
 [ 0  1 12]]
-------------------------------------------------------------------
-------------------------------------------------------------------
Performance Evaluation:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         8
Iris-versicolor       0.89      0.89      0.89         9
 Iris-virginica       0.92      0.92      0.92        13

       accuracy                           0.93        30
      macro avg       0.94      0.94      0.94        30
   weighted avg       0.93      0.93      0.93        30
```

Confusion Matrix: It shows that all 8 'Iris-setosa', 8 'Iris-versicolor', and 12 'Iris-virginica' instances in this specific test set were classified, with only one Iris-versicolor misclassified as Iris-virginica and one Iris-virginica misclassified as Iris-versicolor.

Performance Evaluation: Precision, recall, and f1-scores are 1.00 for Iris-setosa, 0.89 for Iris-versicolor, and 0.92 for Iris-virginica, resulting in an overall accuracy of 0.93.

In [13]:

```
# Import the Decision Tree classifier
from sklearn.tree import DecisionTreeClassifier

# Initialize Decision Tree classifier with gini criterion and max depth of 15
classifier = DecisionTreeClassifier (criterion="gini", max_depth=15)

# Train the classifier
classifier.fit(X_train, y_train)
```

Out[13]:

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=15)
```

In [14]:

```
# Print Confusion Matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("-------------------------------------------------------------------")
print("-------------------------------------------------------------------")
# Print Performance Evaluation (Classification Report)
print("Performance Evaluation:")
print (classification_report(y_test, y_pred))
```

```
Confusion Matrix:
[[ 8  0  0]
 [ 0  8  1]
 [ 0  1 12]]
-------------------------------------------------------------------
-------------------------------------------------------------------
Performance Evaluation:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         8
Iris-versicolor       0.89      0.89      0.89         9
 Iris-virginica       0.92      0.92      0.92        13

       accuracy                           0.93        30
      macro avg       0.94      0.94      0.94        30
   weighted avg       0.93      0.93      0.93        30
```

Confusion Matrix: It shows that all 8 'Iris-setosa', 8 'Iris-versicolor', and 12 'Iris-virginica' instances in this specific test set were classified, with 1 Iris-versicolor misclassified as Iris-virginica and 1 Iris-virginica misclassified as Iris-versicolor.

Performance Evaluation: Precision, recall, and f1-scores are 1.00 for Iris-setosa, 0.89 for Iris-versicolor, and 0.92 for Iris-virginica, resulting in an overall accuracy of 0.93.

IMPORTANT:

Both the "entropy" and "gini" criteria are measures of impurity that Decision Trees use to decide on the best splits. For a dataset like Iris, both criteria might lead to very similar or even identical splits, especially in the upper levels of the tree where the classes are clearly separable. Since the dataset is not very complex, different impurity measures can still result in a tree that achieves perfect separation on this particular data split.

In essence, the Iris dataset is "easy" enough that multiple Decision Tree configurations (within a reasonable range of complexity) can achieve perfect performance on a given train-test split. You might see differences if you used a different, more complex dataset, or a different random state for the train_test_split which could result in a test set that is harder to classify perfectly.

In [15]:
```
from sklearn.preprocessing import LabelEncoder # for train test splitting
from sklearn.model_selection import train_test_split # for decision tree objec
from sklearn.tree import DecisionTreeClassifier # for checking testing results
from sklearn.metrics import classification_report, confusion_matrix # for visu
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier, plot_tree
```

In [16]:
```
# Display the column names of the features DataFrame (X1)
X1.columns
```

Out[16]:
```
Index(['sepal length', 'sepal width', 'petal length', 'petal width'], dtyp
e='object')
```

In [17]:
```
# Display the unique class names in the target variable (y1)
np.unique(y1)
```

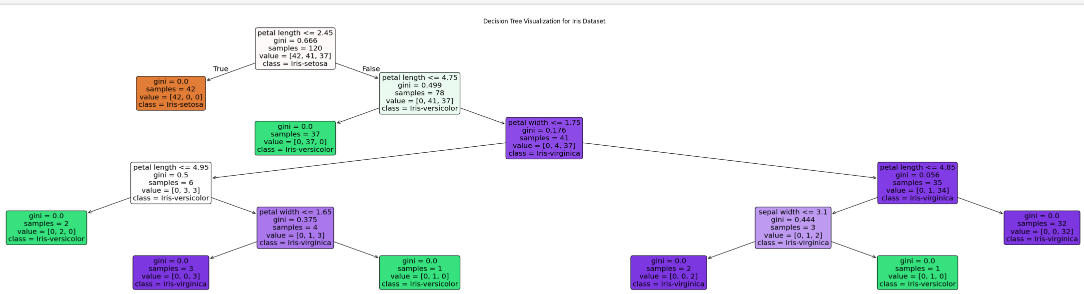
Out[17]:
```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

In [18]:
```
# Create a figure and axes for plotting the decision tree
plt.figure(figsize=(46, 10)) # Adjust figure size as needed

# Plot the decision tree
plot_tree(decision_tree = classifier,
          feature_names = X1.columns, # Use feature names from X1
          class_names = np.unique(y1),   # Use class names from y1
          filled=True, # Fill nodes with colors to indicate the majority class
          rounded=True) # Draw tree nodes with rounded corners

# Set the title of the plot
plt.title("Decision Tree Visualization for Iris Dataset")

# Display the plot
plt.show()
```



Decision Tree Summary (Iris Dataset)

Overall Performance

- **Accuracy:** 93% across all tested configurations
- **Precision, Recall, F1-score:** Very consistent across all classes and hyperparameter settings
- *Iris-setosa:* Perfectly classified (precision, recall, f1 = 1.00)
- *Iris-versicolor:* Precision, recall, f1 = 0.89
- *Iris-virginica:* Precision, recall, f1 = 0.92

Observations by Hyperparameters

Criterion	Max Depth	Accuracy	Notes
entropy	default	93%	Balanced performance, all classes well classified
entropy	3	93%	Performance same as default, low depth sufficient
entropy	10	93%	Increasing depth did not improve performance
gini	10	93%	Similar performance to entropy
gini	15	93%	No improvement; model already captures all patterns

---

Summary

- Decision Tree performance is **stable and robust** for Iris dataset.
- Hyperparameter tuning (criterion or max depth) **did not significantly change performance**.
- Simple trees (max_depth=3) already achieve near-optimal classification.
- **Best choice:** entropy or gini with moderate depth; further deepening is unnecessary.

Classification: Naive Bayes

In [19]:
```
from sklearn.model_selection import train_test_split

# Split the feature data (X1) and target data (y1) into training and testing s
# 20% of the data will be used for testing (test_size = 0.20).
X_train, X_test, y_train, y_test = train_test_split (X1, y1, test_size = 0.20)
```

In [20]:
```python
# Import the Multinomial Naive Bayes classifier
from sklearn.naive_bayes import MultinomialNB

# Initialize and train the Multinomial Naive Bayes classifier using the traini
# The .fit() method trains the model.
classifier = MultinomialNB().fit(X_train, y_train)

# Train the classifier again (this line is redundant as the model is already t
classifier.fit(X_train, y_train)

# Make predictions on the test data using the trained classifier.
y_pred = classifier.predict (X_test)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:1408: DataC
onversionWarning: A column-vector y was passed when a 1d array was expected. Pl
ease change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:1408: DataC
onversionWarning: A column-vector y was passed when a 1d array was expected. Pl
ease change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [21]:
```python
# Evaluation of Classifier Performance
from sklearn.metrics import classification_report, confusion_matrix

# Print the confusion matrix, which shows the number of correct and incorrect
print("Confusion Matrix:")
print (confusion_matrix(y_test, y_pred))
print("-------------------------------------------------------------------")
print("-------------------------------------------------------------------")

# Print the classification report, which includes precision, recall, f1-score,
print("Performance Evaluation:")
print(classification_report (y_test, y_pred))
```

```
Confusion Matrix:
[[ 8  0  0]
 [ 0 10  1]
 [ 0  2  9]]
-------------------------------------------------------------------
-------------------------------------------------------------------
Performance Evaluation:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         8
Iris-versicolor       0.83      0.91      0.87        11
 Iris-virginica       0.90      0.82      0.86        11

       accuracy                           0.90        30
      macro avg       0.91      0.91      0.91        30
   weighted avg       0.90      0.90      0.90        30
```

Confusion Matrix: It shows that all 8 'Iris-setosa' instances were correctly classified. Out of 11 'Iris-versicolor', 10 were correctly classified with 1 misclassified as 'Iris-virginica'. Out of 11 'Iris-virginica', 9 were correctly classified with 2 misclassified as 'Iris-versicolor'.

Performance Evaluation: The overall accuracy is 0.90. 'Iris-setosa' had perfect precision, recall, and f1-score of 1.00. 'Iris-versicolor' had precision of 0.83, recall of 0.91, and f1-score of 0.87. 'Iris-virginica' had precision of 0.90, recall of 0.82, and f1-score of 0.86.

In [22]:
```
# Classification
# Import the Gaussian Naive Bayes classifier
from sklearn.naive_bayes import GaussianNB

# Initialize and train the Gaussian Naive Bayes classifier using the training
# The .fit() method trains the model.
classifier = GaussianNB().fit (X_train, y_train )

# Train the classifier again (this line is redundant as the model is already t
classifier.fit(X_train, y_train)

# Make predictions on the test data using the trained classifier.
y_pred = classifier.predict (X_test)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:1408: DataC
onversionWarning: A column-vector y was passed when a 1d array was expected. Pl
ease change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:1408: DataC
onversionWarning: A column-vector y was passed when a 1d array was expected. Pl
ease change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [23]:
```
# Print the confusion matrix, which shows the number of correct and incorrect
print("Confusion Matrix:")
print (confusion_matrix(y_test, y_pred))
print("-------------------------------------------------------------------")
print("-------------------------------------------------------------------")

# Print the classification report, which includes precision, recall, f1-score,
print("Performance Evaluation:")
print(classification_report (y_test, y_pred))
```

```
Confusion Matrix:
[[ 8  0  0]
 [ 0 11  0]
 [ 0  1 10]]
-------------------------------------------------------------------
-------------------------------------------------------------------
Performance Evaluation:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         8
Iris-versicolor       0.92      1.00      0.96        11
 Iris-virginica       1.00      0.91      0.95        11

       accuracy                           0.97        30
      macro avg       0.97      0.97      0.97        30
   weighted avg       0.97      0.97      0.97        30
```

Confusion Matrix: It shows that all 8 'Iris-setosa' and 11 'Iris-versicolor' instances were correctly classified, with 1 'Iris-virginica' instance misclassified as 'Iris-versicolor' and the remaining 10 'Iris-virginica' classified correctly.

Performance Evaluation: The overall accuracy is 0.97. 'Iris-setosa' had perfect precision, recall, and f1-score of 1.00. 'Iris-versicolor' had precision of 0.92, recall of 1.00, and f1-score of 0.96. 'Iris-virginica' had precision of 1.00, recall of 0.91, and f1-score of 0.95.

In [24]:

```
from sklearn.naive_bayes import BernoulliNB
# Initialize and train the Bernoulli Naive Bayes classifier using the training
# The .fit() method trains the model.
classifier = BernoulliNB().fit (X_train, y_train)

# Train the classifier again (this line is redundant as the model is already t
classifier.fit(X_train, y_train)

# Make predictions on the test data using the trained classifier.
y_pred = classifier.predict(X_test)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:1408: DataC
onversionWarning: A column-vector y was passed when a 1d array was expected. Pl
ease change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:1408: DataC
onversionWarning: A column-vector y was passed when a 1d array was expected. Pl
ease change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [25]:

```
# Print the confusion matrix, which shows the number of correct and incorrect
print("Confusion Matrix:")
print (confusion_matrix(y_test, y_pred))
print("-------------------------------------------------------------------")
print("-------------------------------------------------------------------")
```

```
# Print the classification report, which includes precision, recall, f1-score,
print("Performance Evaluation:")
print(classification_report (y_test, y_pred))
```

```
Confusion Matrix:
[[ 8  0  0]
 [11  0  0]
 [11  0  0]]
--------------------------------------------------------------------
--------------------------------------------------------------------
Performance Evaluation:
                 precision    recall  f1-score   support

    Iris-setosa       0.27      1.00      0.42         8
Iris-versicolor       0.00      0.00      0.00        11
 Iris-virginica       0.00      0.00      0.00        11

       accuracy                           0.27        30
      macro avg       0.09      0.33      0.14        30
   weighted avg       0.07      0.27      0.11        30
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:156
5: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in lab
els with no predicted samples. Use `zero_division` parameter to control this be
havior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:156
5: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in lab
els with no predicted samples. Use `zero_division` parameter to control this be
havior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:156
5: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in lab
els with no predicted samples. Use `zero_division` parameter to control this be
havior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

Confusion Matrix: It shows that all 8 'Iris-setosa' instances were correctly classified, while all 11 'Iris-versicolor' and all 11 'Iris-virginica' instances were misclassified as 'Iris-setosa'.

Performance Evaluation: The overall accuracy is 0.27. 'Iris-setosa' had precision of 0.27, recall of 1.00, and f1-score of 0.42. Both 'Iris-versicolor' and 'Iris-virginica' had precision, recall, and f1-scores of 0.00.

In [26]:

```
# Classification
from sklearn.naive_bayes import MultinomialNB

# Initialize and train the Multinomial Naive Bayes classifier with specified h
# alpha is the smoothing parameter. fit_prior determines whether to learn clas
classifier = MultinomialNB (alpha=2.5, fit_prior=False, class_prior = None ).f
```

```python

# Train the classifier again (this line is redundant as the model is already t
classifier.fit(X_train, y_train)

# Make predictions on the test data using the trained classifier.
y_pred = classifier.predict(X_test)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:1408: DataC
onversionWarning: A column-vector y was passed when a 1d array was expected. Pl
ease change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:1408: DataC
onversionWarning: A column-vector y was passed when a 1d array was expected. Pl
ease change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [27]:
```python
# Print the confusion matrix, which shows the number of correct and incorrect
print("Confusion Matrix:")
print (confusion_matrix(y_test, y_pred))
print("--------------------------------------------------------------------")
print("--------------------------------------------------------------------")

# Print the classification report, which includes precision, recall, f1-score,
print("Performance Evaluation:")
print(classification_report (y_test, y_pred))
```

```
Confusion Matrix:
[[ 8  0  0]
 [ 0 10  1]
 [ 0  2  9]]
--------------------------------------------------------------------
--------------------------------------------------------------------
Performance Evaluation:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         8
Iris-versicolor       0.83      0.91      0.87        11
 Iris-virginica       0.90      0.82      0.86        11

       accuracy                           0.90        30
      macro avg       0.91      0.91      0.91        30
   weighted avg       0.90      0.90      0.90        30
```

Confusion Matrix: It shows that all 8 'Iris-setosa' instances were correctly classified. Out of 11 'Iris-versicolor', 10 were correctly classified with 1 misclassified as 'Iris-virginica'. Out of 11 'Iris-virginica', 9 were correctly classified with 2 misclassified as 'Iris-versicolor'.

Performance Evaluation: The overall accuracy is 0.90. 'Iris-setosa' had perfect precision, recall, and f1-score of 1.00. 'Iris-versicolor' had precision of 0.83, recall of 0.91, and f1-score of 0.87. 'Iris-virginica' had precision of 0.90, recall of 0.82, and f1-score of 0.86.

In [28]:
```
# Classification
from sklearn.naive_bayes import GaussianNB

# Initialize and train the Gaussian Naive Bayes classifier with specified hype
# priors allows setting prior probabilities for classes. var_smoothing is adde
classifier = GaussianNB(priors = None, var_smoothing = 1e-05).fit(X_train, y_t

# Train the classifier again (this line is redundant as the model is already t
classifier.fit(X_train, y_train)

# Make predictions on the test data using the trained classifier.
y_pred = classifier.predict (X_test)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:1408: DataC
onversionWarning: A column-vector y was passed when a 1d array was expected. Pl
ease change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:1408: DataC
onversionWarning: A column-vector y was passed when a 1d array was expected. Pl
ease change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [29]:
```
# Print the confusion matrix, which shows the number of correct and incorrect
print("Confusion Matrix:")
print (confusion_matrix(y_test, y_pred))
print("--------------------------------------------------------------------")
print("--------------------------------------------------------------------")

# Print the classification report, which includes precision, recall, f1-score,
print("Performance Evaluation:")
print(classification_report (y_test, y_pred))
```

```
Confusion Matrix:
[[ 8  0  0]
 [ 0 11  0]
 [ 0  1 10]]
--------------------------------------------------------------------
--------------------------------------------------------------------
Performance Evaluation:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         8
Iris-versicolor       0.92      1.00      0.96        11
 Iris-virginica       1.00      0.91      0.95        11

       accuracy                           0.97        30
      macro avg       0.97      0.97      0.97        30
   weighted avg       0.97      0.97      0.97        30
```

Confusion Matrix: It shows that all 8 'Iris-setosa' and 11 'Iris-versicolor' instances were correctly classified, with 1 'Iris-virginica' instance misclassified as 'Iris-versicolor' and the remaining 10 'Iris-virginica' classified correctly.

Performance Evaluation: The overall accuracy is 0.97. 'Iris-setosa' had perfect precision, recall, and f1-score of 1.00. 'Iris-versicolor' had precision of 0.92, recall of 1.00, and f1-score of 0.96. 'Iris-virginica' had precision of 1.00, recall of 0.91, and f1-score of 0.95.

In [30]:
```
# Classification
from sklearn.naive_bayes import BernoulliNB

# Initialize and train the Bernoulli Naive Bayes classifier with specified hyp
# alpha is the smoothing parameter. binarize is the threshold for binarizing t
# fit_prior determines whether to learn class prior probabilities. class_prior
classifier = BernoulliNB(alpha=1.0, binarize = 0.0, fit_prior = True, class_pr

# Train the classifier again (this line is redundant as the model is already t
classifier.fit(X_train, y_train)

# Make predictions on the test data using the trained classifier.
y_pred = classifier.predict(X_test)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:1408: DataC
onversionWarning: A column-vector y was passed when a 1d array was expected. Pl
ease change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:1408: DataC
onversionWarning: A column-vector y was passed when a 1d array was expected. Pl
ease change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [31]:
```
# Print the confusion matrix, which shows the number of correct and incorrect
print("Confusion Matrix:")
print (confusion_matrix(y_test, y_pred))
print("-------------------------------------------------------------------")
print("-------------------------------------------------------------------")

# Print the classification report, which includes precision, recall, f1-score,
print("Performance Evaluation:")
print(classification_report (y_test, y_pred))
```

```
Confusion Matrix:
[[ 8  0  0]
 [11  0  0]
 [11  0  0]]
--------------------------------------------------------------------
--------------------------------------------------------------------
Performance Evaluation:
                 precision    recall  f1-score   support

    Iris-setosa       0.27      1.00      0.42         8
Iris-versicolor       0.00      0.00      0.00        11
 Iris-virginica       0.00      0.00      0.00        11

       accuracy                           0.27        30
      macro avg       0.09      0.33      0.14        30
   weighted avg       0.07      0.27      0.11        30
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:156
5: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in lab
els with no predicted samples. Use `zero_division` parameter to control this be
havior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:156
5: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in lab
els with no predicted samples. Use `zero_division` parameter to control this be
havior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:156
5: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in lab
els with no predicted samples. Use `zero_division` parameter to control this be
havior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

Confusion Matrix: It shows that all 8 'Iris-setosa' instances were correctly classified, while all 11 'Iris-versicolor' and all 11 'Iris-virginica' instances were misclassified as 'Iris-setosa'.

Performance Evaluation: The overall accuracy is 0.27. 'Iris-setosa' had precision of 0.27, recall of 1.00, and f1-score of 0.42. Both 'Iris-versicolor' and 'Iris-virginica' had precision, recall, and f1-scores of 0.00.

Combined Summary of Naive Bayes Models (Iris Dataset)

Performance Tables

Accuracy Comparison

Model	Original Accuracy	Hyperparameter-Tuned Accuracy
Multinomial Naive Bayes	90%	90%
Gaussian Naive Bayes	97%	97%
Bernoulli Naive Bayes	27%	27%

Weighted Average Metrics Comparison

Model	Original Precision	Tuned Precision	Original Recall	Tuned Recall	Original F1	Tuned F1
Multinomial Naive Bayes	0.90	0.90	0.90	0.90	0.90	0.90
Gaussian Naive Bayes	0.97	0.97	0.97	0.97	0.97	0.97
Bernoulli Naive Bayes	0.07	0.07	0.27	0.27	0.11	0.11

Class-wise Observations

Model	Iris-setosa	Iris-versicolor	Iris-virginica	Notes
Multinomial Naive Bayes	Perfect	Slight misclassifications	Slight misclassifications	Stable performance, tuning did not improve
Gaussian Naive Bayes	Perfect	Near perfect	Strong	Most reliable model
Bernoulli Naive Bayes	Correct only	Completely misclassified	Completely misclassified	Not suitable for numeric multi-class data

---

Summary

- **Best Model: Gaussian Naive Bayes** (97% accuracy, balanced performance)
- **Worst Model: Bernoulli Naive Bayes** (fails for multi-class numeric data)

- **Notes:** Hyperparameter tuning did not significantly change the performance of Multinomial or Gaussian models. Bernoulli remains unsuitable.

Breast Cancer Dataset

Classification: Decision Tree

In [32]:
```
# Install the ucimlrepo library
!pip install ucimlrepo
```

```
Requirement already satisfied: ucimlrepo in /usr/local/lib/python3.11/dist-pack
ages (0.0.7)
Requirement already satisfied: pandas>=1.0.0 in /usr/local/lib/python3.11/dist-
packages (from ucimlrepo) (2.2.2)
Requirement already satisfied: certifi>=2020.12.5 in /usr/local/lib/python3.11/
dist-packages (from ucimlrepo) (2025.8.3)
Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/python3.11/dist-
packages (from pandas>=1.0.0->ucimlrepo) (2.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python
3.11/dist-packages (from pandas>=1.0.0->ucimlrepo) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-p
ackages (from pandas>=1.0.0->ucimlrepo) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dis
t-packages (from pandas>=1.0.0->ucimlrepo) (2025.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packa
ges (from python-dateutil>=2.8.2->pandas>=1.0.0->ucimlrepo) (1.17.0)
```

In [33]:
```
from ucimlrepo import fetch_ucirepo

# fetch dataset
breast_cancer_wisconsin_diagnostic = fetch_ucirepo(id=17)

# data (as pandas dataframes)
X2 = breast_cancer_wisconsin_diagnostic.data.features
y2 = breast_cancer_wisconsin_diagnostic.data.targets

# metadata
print(breast_cancer_wisconsin_diagnostic.metadata)

# variable information
print(breast_cancer_wisconsin_diagnostic.variables)
```

{'uci_id': 17, 'name': 'Breast Cancer Wisconsin (Diagnostic)', 'repository_ur
l': 'https://archive.ics.uci.edu/dataset/17/breast+cancer+wisconsin+diagnosti
c', 'data_url': 'https://archive.ics.uci.edu/static/public/17/data.csv', 'abstr
act': 'Diagnostic Wisconsin Breast Cancer Database.', 'area': 'Health and Medic
ine', 'tasks': ['Classification'], 'characteristics': ['Multivariate'], 'num_in
stances': 569, 'num_features': 30, 'feature_types': ['Real'], 'demographics':
[], 'target_col': ['Diagnosis'], 'index_col': ['ID'], 'has_missing_values': 'n
o', 'missing_values_symbol': None, 'year_of_dataset_creation': 1993, 'last_upda
ted': 'Fri Nov 03 2023', 'dataset_doi': '10.24432/C5DW2B', 'creators': ['Willia
m Wolberg', 'Olvi Mangasarian', 'Nick Street', 'W. Street'], 'intro_paper': {'I
D': 230, 'type': 'NATIVE', 'title': 'Nuclear feature extraction for breast tumo
r diagnosis', 'authors': 'W. Street, W. Wolberg, O. Mangasarian', 'venue': 'Ele
ctronic imaging', 'year': 1993, 'journal': None, 'DOI': '10.1117/12.148698', 'U
RL': 'https://www.semanticscholar.org/paper/53f0fbb425bc14468eb3bf96b2e1d41ba80
87f36', 'sha': None, 'corpus': None, 'arxiv': None, 'mag': None, 'acl': None,
'pmid': None, 'pmcid': None}, 'additional_info': {'summary': 'Features are comp
uted from a digitized image of a fine needle aspirate (FNA) of a breast mass.
They describe characteristics of the cell nuclei present in the image. A few of
the images can be found at http://www.cs.wisc.edu/~street/images/\r\n\r\nSepara
ting plane described above was obtained using Multisurface Method-Tree (MSM-T)
[K. P. Bennett, "Decision Tree Construction Via Linear Programming." Proceeding
s of the 4th Midwest Artificial Intelligence and Cognitive Science Society, pp.
97-101, 1992], a classification method which uses linear programming to constru
ct a decision tree. Relevant features were selected using an exhaustive search
in the space of 1-4 features and 1-3 separating planes.\r\n\r\nThe actual linea
r program used to obtain the separating plane in the 3-dimensional space is tha
t described in: [K. P. Bennett and O. L. Mangasarian: "Robust Linear Programmin
g Discrimination of Two Linearly Inseparable Sets", Optimization Methods and So
ftware 1, 1992, 23-34].\r\n\r\nThis database is also available through the UW C
S ftp server:\r\nftp ftp.cs.wisc.edu\r\ncd math-prog/cpo-dataset/machine-learn/
WDBC/', 'purpose': None, 'funded_by': None, 'instances_represent': None, 'recom
mended_data_splits': None, 'sensitive_data': None, 'preprocessing_description':
None, 'variable_info': '1) ID number\r\n2) Diagnosis (M = malignant, B = benig
n)\r\n3-32)\r\n\r\nTen real-valued features are computed for each cell nucleu
s:\r\n\r\n\ta) radius (mean of distances from center to points on the perimete
r)\r\n\tb) texture (standard deviation of gray-scale values)\r\n\tc) perimeter\
r\n\td) area\r\n\te) smoothness (local variation in radius lengths)\r\n\tf) com
pactness (perimeter^2 / area - 1.0)\r\n\tg) concavity (severity of concave port
ions of the contour)\r\n\th) concave points (number of concave portions of the
contour)\r\n\ti) symmetry \r\n\tj) fractal dimension ("coastline approximation"
- 1)', 'citation': None}}

```
                  name     role         type demographic description units  \
0                   ID       ID  Categorical        None        None  None   
1            Diagnosis   Target  Categorical        None        None  None   
2              radius1  Feature   Continuous        None        None  None   
3             texture1  Feature   Continuous        None        None  None   
4           perimeter1  Feature   Continuous        None        None  None   
5                area1  Feature   Continuous        None        None  None   
6          smoothness1  Feature   Continuous        None        None  None   
7         compactness1  Feature   Continuous        None        None  None   
8           concavity1  Feature   Continuous        None        None  None   
9      concave_points1  Feature   Continuous        None        None  None   
10           symmetry1  Feature   Continuous        None        None  None   
11  fractal_dimension1  Feature   Continuous        None        None  None   
```

12	radius2	Feature	Continuous	None	None	None
13	texture2	Feature	Continuous	None	None	None
14	perimeter2	Feature	Continuous	None	None	None
15	area2	Feature	Continuous	None	None	None
16	smoothness2	Feature	Continuous	None	None	None
17	compactness2	Feature	Continuous	None	None	None
18	concavity2	Feature	Continuous	None	None	None
19	concave_points2	Feature	Continuous	None	None	None
20	symmetry2	Feature	Continuous	None	None	None
21	fractal_dimension2	Feature	Continuous	None	None	None
22	radius3	Feature	Continuous	None	None	None
23	texture3	Feature	Continuous	None	None	None
24	perimeter3	Feature	Continuous	None	None	None
25	area3	Feature	Continuous	None	None	None
26	smoothness3	Feature	Continuous	None	None	None
27	compactness3	Feature	Continuous	None	None	None
28	concavity3	Feature	Continuous	None	None	None
29	concave_points3	Feature	Continuous	None	None	None
30	symmetry3	Feature	Continuous	None	None	None
31	fractal_dimension3	Feature	Continuous	None	None	None

	missing_values
0	no
1	no
2	no
3	no
4	no
5	no
6	no
7	no
8	no
9	no
10	no
11	no
12	no
13	no
14	no
15	no
16	no
17	no
18	no
19	no
20	no
21	no
22	no
23	no
24	no
25	no
26	no
27	no
28	no
29	no
30	no
31	no

In [34]:
```
from sklearn.model_selection import train_test_split

# Split the data into training and testing sets.
# X2 contains the features and y2 contains the target variable.
# test_size=0.20 means 20% of the data will be used for testing.
X_train, X_test, y_train, y_test = train_test_split (X2, y2, test_size = 0.20)
```

In [35]:
```
# Classification
from sklearn.tree import DecisionTreeClassifier

# Initialize a Decision Tree classifier with default hyperparameters.
classifier = DecisionTreeClassifier()

# Train the classifier using the training data.
classifier.fit(X_train, y_train)

# Make predictions on the test data.
y_pred = classifier.predict(X_test)
```

In [36]:
```
# Evaluation of Classifier Performance
from sklearn.metrics import classification_report, confusion_matrix

# Print the confusion matrix to show the number of correct and incorrect predi
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("--------------------------------------------------------------------")
print("--------------------------------------------------------------------")

# Print the classification report to show key classification metrics (precisio
print("Performance Evaluation:")
print(classification_report(y_test, y_pred))
```

```
Confusion Matrix:
[[70  4]
 [ 5 35]]
--------------------------------------------------------------------
--------------------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

           B       0.93      0.95      0.94        74
           M       0.90      0.88      0.89        40

    accuracy                           0.92       114
   macro avg       0.92      0.91      0.91       114
weighted avg       0.92      0.92      0.92       114
```

Confusion Matrix: It shows that 70 'Benign' instances and 35 'Malignant' instances were correctly classified. There were 4 'Benign' instances misclassified as 'Malignant' and 5 'Malignant' instances misclassified as 'Benign'.

Performance Evaluation: The overall accuracy is 0.92. For 'Benign', precision =

0.93, recall = 0.95, and f1-score = 0.94. For 'Malignant', precision = 0.90, recall = 0.88, and f1-score = 0.89.

In [37]:
```
from sklearn.tree import DecisionTreeClassifier

# Initialize a Decision Tree classifier with 'entropy' criterion and a maximum
classifier = DecisionTreeClassifier(criterion="entropy", max_depth=3)

# Train the classifier using the training data.
classifier.fit(X_train, y_train)
```

Out[37]:

DecisionTreeClassifier

```
DecisionTreeClassifier(criterion='entropy', max_depth=3)
```

In [38]:
```
# Print the confusion matrix to show the number of correct and incorrect predi
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("--------------------------------------------------------------------")
print("--------------------------------------------------------------------")

# Print the classification report to show key classification metrics (precisio
print("Performance Evaluation:")
print(classification_report(y_test, y_pred))
```

```
Confusion Matrix:
[[70  4]
 [ 5 35]]
--------------------------------------------------------------------
--------------------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

           B       0.93      0.95      0.94        74
           M       0.90      0.88      0.89        40

    accuracy                           0.92       114
   macro avg       0.92      0.91      0.91       114
weighted avg       0.92      0.92      0.92       114
```

Confusion Matrix: It shows that 70 'Benign' instances and 35 'Malignant' instances were correctly classified. There were 4 'Benign' instances misclassified as 'Malignant' and 5 'Malignant' instances misclassified as 'Benign'.

Performance Evaluation: The overall accuracy is 0.92. For 'Benign', precision = 0.93, recall = 0.95, and f1-score = 0.94. For 'Malignant', precision = 0.90, recall = 0.88, and f1-score = 0.89.

In [39]:
```
from sklearn.tree import DecisionTreeClassifier
```

```python
# Initialize a Decision Tree classifier with 'entropy' criterion and a maximum
classifier = DecisionTreeClassifier(criterion="entropy", max_depth=10)

# Train the classifier using the training data.
classifier.fit(X_train, y_train)
```

Out[39]:

```
DecisionTreeClassifier

DecisionTreeClassifier(criterion='entropy', max_depth=10)
```

In [40]:

```python
# Print the confusion matrix to show the number of correct and incorrect predi
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("--------------------------------------------------------------------")
print("--------------------------------------------------------------------")

# Print the classification report to show key classification metrics (precisio
print("Performance Evaluation:")
print(classification_report(y_test, y_pred))
```

```
Confusion Matrix:
[[70  4]
 [ 5 35]]
--------------------------------------------------------------------
--------------------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

           B       0.93      0.95      0.94        74
           M       0.90      0.88      0.89        40

    accuracy                           0.92       114
   macro avg       0.92      0.91      0.91       114
weighted avg       0.92      0.92      0.92       114
```

Confusion Matrix: It shows that 70 'Benign' instances and 35 'Malignant' instances were correctly classified. There were 4 'Benign' instances misclassified as 'Malignant' and 5 'Malignant' instances misclassified as 'Benign'.

Performance Evaluation: The overall accuracy is 0.92. For Benign, precision = 0.93, recall = 0.95, and f1-score = 0.94. For Malignant, precision = 0.90, recall = 0.88, and f1-score = 0.89.

In [41]:

```python
from sklearn.tree import DecisionTreeClassifier

# Initialize a Decision Tree classifier with 'gini' criterion and a maximum de
classifier = DecisionTreeClassifier(criterion="gini", max_depth=10)

# Train the classifier using the training data.
```

```
classifier.fit(X_train, y_train)
```

Out[41]:

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=10)
```

In [42]:

```
# Print the confusion matrix to show the number of correct and incorrect predi
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("-------------------------------------------------------------------")
print("-------------------------------------------------------------------")

# Print the classification report to show key classification metrics (precisio
print("Performance Evaluation:")
print(classification_report(y_test, y_pred))
```

```
Confusion Matrix:
[[70  4]
 [ 5 35]]
-------------------------------------------------------------------
-------------------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

           B       0.93      0.95      0.94        74
           M       0.90      0.88      0.89        40

    accuracy                           0.92       114
   macro avg       0.92      0.91      0.91       114
weighted avg       0.92      0.92      0.92       114
```

Confusion Matrix: It shows that 70 'Benign' instances and 35 'Malignant' instances were correctly classified. There were 4 'Benign' instances misclassified as 'Malignant' and 5 'Malignant' instances misclassified as 'Benign'.

Performance Evaluation: The overall accuracy is 0.92. For Benign, precision = 0.93, recall = 0.95, and f1-score = 0.94. For Malignant, precision = 0.90, recall = 0.88, and f1-score = 0.89.

In [43]:

```
from sklearn.tree import DecisionTreeClassifier

# Initialize a Decision Tree classifier with 'gini' criterion and a maximum de
classifier = DecisionTreeClassifier(criterion="gini", max_depth=15)

# Train the classifier using the training data.
classifier.fit(X_train, y_train)
```

Out[43]:

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=15)
```

In [44]:

```
# Print the confusion matrix to show the number of correct and incorrect predi
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("-------------------------------------------------------------------")
print("-------------------------------------------------------------------")

# Print the classification report to show key classification metrics (precisio
print("Performance Evaluation:")
print(classification_report(y_test, y_pred))
```

```
Confusion Matrix:
[[70  4]
 [ 5 35]]
-------------------------------------------------------------------
-------------------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

           B       0.93      0.95      0.94        74
           M       0.90      0.88      0.89        40

    accuracy                           0.92       114
   macro avg       0.92      0.91      0.91       114
weighted avg       0.92      0.92      0.92       114
```

Confusion Matrix: It shows that 70 'Benign' instances and 35 'Malignant' instances were correctly classified. There were 4 'Benign' instances misclassified as 'Malignant' and 5 'Malignant' instances misclassified as 'Benign'.

Performance Evaluation: The overall accuracy is 0.92. For Benign, precision = 0.93, recall = 0.95, and f1-score = 0.94. For Malignant, precision = 0.90, recall = 0.88, and f1-score = 0.89.

In [45]:

```
from sklearn.preprocessing import LabelEncoder # for train test splitting
from sklearn.model_selection import train_test_split # for decision tree objec
from sklearn.tree import DecisionTreeClassifier # for checking testing results
from sklearn.metrics import classification_report, confusion_matrix # for visu
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier, plot_tree
```

In [46]:

```
X2.columns
```

```
Out[46]: Index(['radius1', 'texture1', 'perimeter1', 'area1', 'smoothness1',
       'compactness1', 'concavity1', 'concave_points1', 'symmetry1',
       'fractal_dimension1', 'radius2', 'texture2', 'perimeter2', 'area2',
       'smoothness2', 'compactness2', 'concavity2', 'concave_points2',
       'symmetry2', 'fractal_dimension2', 'radius3', 'texture3', 'perimeter
3',
       'area3', 'smoothness3', 'compactness3', 'concavity3', 'concave_points
3',
       'symmetry3', 'fractal_dimension3'],
      dtype='object')
```

```
In [47]: np.unique(y2)
```

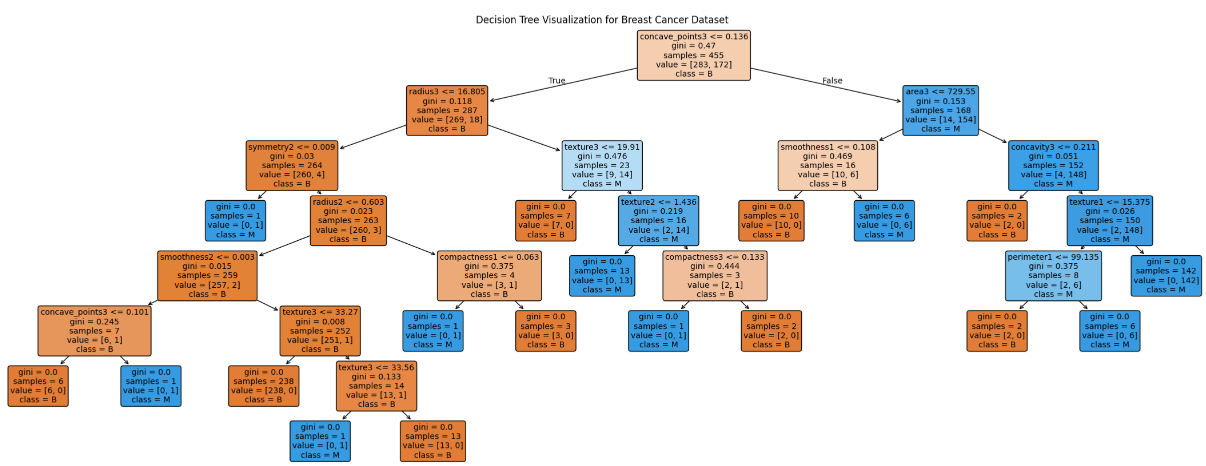
```
Out[47]: array(['B', 'M'], dtype=object)
```

```
In [48]: # Create a figure and axes for plotting the decision tree.
plt.figure(figsize=(28, 10)) # Adjust figure size as needed

# Plot the decision tree.
plot_tree(decision_tree = classifier,
          feature_names = X2.columns, # Use feature names from X2.
          class_names = np.unique(y2),   # Use class names from y2.
          filled=True, # Fill nodes with colors to indicate the majority class
          rounded=True) # Draw tree nodes with rounded corners.

# Set the title of the plot.
plt.title("Decision Tree Visualization for Breast Cancer Dataset")

# Display the plot.
plt.show()
```



Decision Tree Summary (Breast Cancer Dataset)

Overall Performance

- **Accuracy:** 92% across all tested configurations
- **Precision, Recall, F1-score:**
    - Benign (B): Precision = 0.93, Recall = 0.95, F1 = 0.94
    - Malignant (M): Precision = 0.90, Recall = 0.88, F1 = 0.89
- Consistent performance across all configurations and hyperparameters.

Observations by Hyperparameters

Criterion	Max Depth	Accuracy	Notes
default	default	92%	Balanced classification for both classes
entropy	3	92%	Low depth sufficient, maintains performance
entropy	10	92%	No improvement with deeper tree
gini	10	92%	Similar performance to entropy
gini	15	92%	Increasing depth does not improve results

Summary

- Decision Tree performs **well and consistently** on Breast Cancer dataset.
- Hyperparameter tuning (criterion or max depth) **does not significantly change performance**.
- Moderate depth trees already capture sufficient patterns for accurate classification.
- **Best choice:** gini or entropy with moderate depth (e.g., max_depth=3–10).

Classification: Naive Bayes

In [49]:
```
from sklearn.model_selection import train_test_split

# Split the feature data (X2) and target data (y2) into training and testing s
# 20% of the data will be used for testing (test_size = 0.20).
X_train, X_test, y_train, y_test = train_test_split (X2, y2, test_size = 0.20)
```

In [50]:
```
# Classification
# Import the Multinomial Naive Bayes classifier
from sklearn.naive_bayes import MultinomialNB

# Initialize and train the Multinomial Naive Bayes classifier using the traini
# The .fit() method trains the model.
classifier = MultinomialNB().fit(X_train, y_train)

# Make predictions on the test data using the trained classifier.
y_pred = classifier.predict (X_test)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:1408: DataC
onversionWarning: A column-vector y was passed when a 1d array was expected. Pl
ease change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [51]:
```
# Evaluation of Classifier Performance
from sklearn.metrics import classification_report, confusion_matrix

# Print the confusion matrix, which shows the number of correct and incorrect
print("Confusion Matrix:")
print (confusion_matrix(y_test, y_pred))
print("-------------------------------------------------------------------")
print("-------------------------------------------------------------------")

# Print the classification report, which includes precision, recall, f1-score,
print("Performance Evaluation:")
print(classification_report (y_test, y_pred))
```

```
Confusion Matrix:
[[75  1]
 [15 23]]
-------------------------------------------------------------------
-------------------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

           B       0.83      0.99      0.90        76
           M       0.96      0.61      0.74        38

    accuracy                           0.86       114
   macro avg       0.90      0.80      0.82       114
weighted avg       0.88      0.86      0.85       114
```

Confusion Matrix: It shows that 75 'Benign' instances and 23 'Malignant' instances were correctly classified. There was 1 'Benign' instance misclassified as 'Malignant' and 15 'Malignant' instances misclassified as 'Benign'.

Performance Evaluation: The overall accuracy is 0.86. For Benign, precision = 0.83, recall = 0.99, and f1-score = 0.90. For Malignant, precision = 0.96, recall = 0.61, and f1-score = 0.74.

In [52]:
```
# Classification
# Import the Gaussian Naive Bayes classifier
from sklearn.naive_bayes import GaussianNB

# Initialize and train the Gaussian Naive Bayes classifier using the training
# The .fit() method trains the model.
classifier = GaussianNB().fit (X_train, y_train )

# Make predictions on the test data using the trained classifier.
y_pred = classifier.predict (X_test)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:1408: DataC
onversionWarning: A column-vector y was passed when a 1d array was expected. Pl
ease change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [53]:
```
# Print the confusion matrix, which shows the number of correct and incorrect
print("Confusion Matrix:")
print (confusion_matrix(y_test, y_pred))
print("--------------------------------------------------------------------")
print("--------------------------------------------------------------------")

# Print the classification report, which includes precision, recall, f1-score,
print("Performance Evaluation:")
print(classification_report (y_test, y_pred))
```

```
Confusion Matrix:
[[75  1]
 [ 7 31]]
--------------------------------------------------------------------
--------------------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

           B       0.91      0.99      0.95        76
           M       0.97      0.82      0.89        38

    accuracy                           0.93       114
   macro avg       0.94      0.90      0.92       114
weighted avg       0.93      0.93      0.93       114
```

Confusion Matrix: It shows that 75 'Benign' instances and 31 'Malignant' instances were correctly classified. There was 1 'Benign' instance misclassified as 'Malignant' and 7 'Malignant' instances misclassified as 'Benign'.

Performance Evaluation: The overall accuracy is 0.93. For Benign, precision = 0.91, recall = 0.99, and f1-score = 0.95. For Malignant, precision = 0.97, recall = 0.82, and f1-score = 0.89.

In [54]:
```
from sklearn.naive_bayes import BernoulliNB
```

```
# Initialize and train the Bernoulli Naive Bayes classifier using the training
# The .fit() method trains the model.
classifier = BernoulliNB().fit (X_train, y_train)

# Make predictions on the test data using the trained classifier.
y_pred = classifier.predict(X_test)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:1408: DataC
onversionWarning: A column-vector y was passed when a 1d array was expected. Pl
ease change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [55]:
```
# Print the confusion matrix, which shows the number of correct and incorrect
print("Confusion Matrix:")
print (confusion_matrix(y_test, y_pred))
print("-------------------------------------------------------------------")
print("-------------------------------------------------------------------")

# Print the classification report, which includes precision, recall, f1-score,
print("Performance Evaluation:")
print(classification_report (y_test, y_pred))
```

```
Confusion Matrix:
[[76  0]
 [38  0]]
-------------------------------------------------------------------
-------------------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

           B       0.67      1.00      0.80        76
           M       0.00      0.00      0.00        38

    accuracy                           0.67       114
   macro avg       0.33      0.50      0.40       114
weighted avg       0.44      0.67      0.53       114
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:156
5: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in lab
els with no predicted samples. Use `zero_division` parameter to control this be
havior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:156
5: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in lab
els with no predicted samples. Use `zero_division` parameter to control this be
havior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:156
5: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in lab
els with no predicted samples. Use `zero_division` parameter to control this be
havior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

Confusion Matrix: It shows that 76 'Benign' instances were correctly classified, while all 38 'Malignant' instances were misclassified as 'Benign'. There were no 'Benign' instances misclassified as 'Malignant'.

Performance Evaluation: The overall accuracy is 0.67. For Benign, precision = 0.67, recall = 1.00, and f1-score = 0.80. For Malignant, precision = 0.00, recall = 0.00, and f1-score = 0.00.

In [56]:

```
# Classification
from sklearn.naive_bayes import MultinomialNB

# Initialize and train the Multinomial Naive Bayes classifier with specified h
# alpha is the smoothing parameter. fit_prior determines whether to learn clas
classifier = MultinomialNB (alpha=2.5, fit_prior=True, class_prior=None).fit(X

# Train the classifier again (this line is redundant as the model is already t
classifier.fit(X_train, y_train)

# Make predictions on the test data using the trained classifier.
y_pred = classifier.predict(X_test)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:1408: DataC
onversionWarning: A column-vector y was passed when a 1d array was expected. Pl
ease change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:1408: DataC
onversionWarning: A column-vector y was passed when a 1d array was expected. Pl
ease change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [57]:

```
# Print the confusion matrix, which shows the number of correct and incorrect
print("Confusion Matrix:")
print (confusion_matrix(y_test, y_pred))
print("--------------------------------------------------------------------")
print("------------------------------------------------0-------------------")

# Print the classification report, which includes precision, recall, f1-score,
print("Performance Evaluation:")
print(classification_report (y_test, y_pred))
```

```
Confusion Matrix:
[[75  1]
 [15 23]]
-------------------------------------------------------------------
------------------------------------------------0------------------
Performance Evaluation:
              precision    recall  f1-score   support

           B       0.83      0.99      0.90        76
           M       0.96      0.61      0.74        38

    accuracy                           0.86       114
   macro avg       0.90      0.80      0.82       114
weighted avg       0.88      0.86      0.85       114
```

Confusion Matrix: It shows that 75 'Benign' instances and 23 'Malignant' instances were correctly classified. There was 1 'Benign' instance misclassified as 'Malignant', and 15 'Malignant' instances misclassified as 'Benign'.

Performance Evaluation: The overall accuracy is 0.86. For Benign, precision = 0.83, recall = 0.99, and f1-score = 0.90. For Malignant, precision = 0.96, recall = 0.61, and f1-score = 0.74.

In [58]:

```
# Classification
from sklearn.naive_bayes import GaussianNB

# Initialize and train the Gaussian Naive Bayes classifier with specified hype
# priors allows setting prior probabilities for classes. var_smoothing is adde
classifier = GaussianNB(priors=None, var_smoothing = 1e-05).fit(X_train, y_tra

# Train the classifier again (this line is redundant as the model is already t
classifier.fit(X_train, y_train)

# Make predictions on the test data using the trained classifier.
y_pred = classifier.predict (X_test)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:1408: DataC
onversionWarning: A column-vector y was passed when a 1d array was expected. Pl
ease change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:1408: DataC
onversionWarning: A column-vector y was passed when a 1d array was expected. Pl
ease change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [59]:

```
# Print the confusion matrix, which shows the number of correct and incorrect
print("Confusion Matrix:")
print (confusion_matrix(y_test, y_pred))
print("-------------------------------------------------------------------")
print("-------------------------------------------------------------------")
```

```
# Print the classification report, which includes precision, recall, f1-score,
print("Performance Evaluation:")
print(classification_report (y_test, y_pred))
```

```
Confusion Matrix:
[[76  0]
 [10 28]]
--------------------------------------------------------------------
--------------------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

           B       0.88      1.00      0.94        76
           M       1.00      0.74      0.85        38

    accuracy                           0.91       114
   macro avg       0.94      0.87      0.89       114
weighted avg       0.92      0.91      0.91       114
```

Confusion Matrix: It shows that 76 'Benign' instances and 28 'Malignant' instances were correctly classified. There were no 'Benign' instances misclassified as 'Malignant', and 10 'Malignant' instances misclassified as 'Benign'.

Performance Evaluation: The overall accuracy is 0.91. For Benign, precision = 0.88, recall = 1.00, and f1-score = 0.94. For Malignant, precision = 1.00, recall = 0.74, and f1-score = 0.85.

In [60]:

```
# Classification
from sklearn.naive_bayes import BernoulliNB

# Initialize and train the Bernoulli Naive Bayes classifier with specified hyp
# alpha is the smoothing parameter. binarize is the threshold for binarizing t
# fit_prior determines whether to learn class prior probabilities. class_prior
classifier = BernoulliNB(alpha=1.0, binarize = 0.0, fit_prior = True, class_pr

# Train the classifier again (this line is redundant as the model is already t
classifier.fit(X_train, y_train)

# Make predictions on the test data using the trained classifier.
y_pred = classifier.predict(X_test)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:1408: DataC
onversionWarning: A column-vector y was passed when a 1d array was expected. Pl
ease change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:1408: DataC
onversionWarning: A column-vector y was passed when a 1d array was expected. Pl
ease change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [61]:

```
# Print the confusion matrix, which shows the number of correct and incorrect
```

```python
print("Confusion Matrix:")
print (confusion_matrix(y_test, y_pred))
print("--------------------------------------------------------------------")
print("--------------------------------------------------------------------")

# Print the classification report, which includes precision, recall, f1-score,
print("Performance Evaluation:")
print(classification_report (y_test, y_pred))
```

```
Confusion Matrix:
[[76  0]
 [38  0]]
--------------------------------------------------------------------
--------------------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

           B       0.67      1.00      0.80        76
           M       0.00      0.00      0.00        38

    accuracy                           0.67       114
   macro avg       0.33      0.50      0.40       114
weighted avg       0.44      0.67      0.53       114
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:156
5: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in lab
els with no predicted samples. Use `zero_division` parameter to control this be
havior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:156
5: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in lab
els with no predicted samples. Use `zero_division` parameter to control this be
havior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:156
5: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in lab
els with no predicted samples. Use `zero_division` parameter to control this be
havior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

Confusion Matrix: It shows that 76 'Benign' instances were correctly classified. However, all 38 'Malignant' instances were misclassified as 'Benign'. This means there were no correct predictions for Malignant.

Performance Evaluation: The overall accuracy is 0.67. For Benign, precision = 0.67, recall = 1.00, and f1-score = 0.80. For Malignant, precision = 0.00, recall = 0.00, and f1-score = 0.00.

Combined Summary of Naive Bayes Models (Breast Cancer Dataset)

Accuracy Comparison

Model	Original Accuracy	Hyperparameter-Tuned Accuracy
Multinomial Naive Bayes	86%	86%
Gaussian Naive Bayes	93%	91%
Bernoulli Naive Bayes	67%	67%

Weighted Average Metrics Comparison

Model	Original Precision	Tuned Precision	Original Recall	Tuned Recall	Original F1	Tuned F1
Multinomial Naive Bayes	0.88	0.88	0.86	0.86	0.85	0.85
Gaussian Naive Bayes	0.93	0.92	0.93	0.91	0.93	0.91
Bernoulli Naive Bayes	0.44	0.44	0.67	0.67	0.53	0.53

Class-wise Observations

Model	Benign (B)	Malignant (M)	Notes
Multinomial Naive Bayes	High precision & recall	Lower recall (0.61)	Hyperparameter tuning did not improve performance
Gaussian Naive Bayes	Very high precision & recall	Improved recall after tuning (0.74)	Most reliable model for this dataset
Bernoulli Naive Bayes	Correctly classified	Completely misclassified	Not suitable for numeric features

---

Summary

- **Best Model: Gaussian Naive Bayes** (Original: 93%, Tuned: 91%, balanced performance)
- **Worst Model: Bernoulli Naive Bayes** (fails for numeric features, 67% accuracy)
- **Notes:** Hyperparameter tuning slightly improved Gaussian recall for

malignant class, no effect on Multinomial or Bernoulli models.

Machine Learning Lab A3

ASIM KUMAR HANSDA

ROLL NO - 002211001136 **ASSIGNMENT - 1**

Github Link: https://github.com/cryptasim/MACHINE-LEARNING-LAB

Analysis of Classification Models on IRIS and Breast Cancer Datasets

Introduction:

This report presents a comparative analysis of two popular classification algorithms, **Decision Tree** and **Naive Bayes**, applied to two well-known machine learning datasets: the **IRIS dataset** and the **Breast Cancer dataset**.

The objective is to evaluate the performance of these models, assess the impact of different hyperparameters, and understand the suitability of each model for the specific characteristics of each dataset. The analysis focuses on key performance metrics including accuracy, precision, recall, and a detailed examination of the confusion matrix to understand the nature of classification errors.

IRIS Dataset Analysis:

The IRIS dataset is a classic benchmark for classification. It contains 150 instances of iris plants, each belonging to one of three species, described by four continuous features.

Naive Bayes Classifiers

Three variants of the Naive Bayes algorithm were tested, with significantly different outcomes.

Accuracy Comparison

Model	Original Accuracy	Hyperparameter-Tuned Accuracy
Multinomial Naive Bayes	90%	90%
Gaussian Naive Bayes	97%	97%
Bernoulli Naive Bayes	27%	27%

Weighted Average Metrics Comparison

Model	Original Precision	Tuned Precision	Original Recall	Tuned Recall	Original F1	Tuned F1
Multinomial Naive Bayes	0.9	0.9	0.9	0.9	0.9	0.9
Gaussian Naive Bayes	0.97	0.97	0.97	0.97	0.97	0.97
Bernoulli Naive Bayes	0.07	0.07	0.27	0.27	0.11	0.11

Class-wise Observations

Model	Iris-setosa	Iris-versicolor	Iris-virginica	Notes
Multinomial Naive Bayes	Perfect	Slight misclassifications	Slight misclassifications	Stable performance, tuning did not improve
Gaussian Naive Bayes	Perfect	Near perfect	Strong	Most reliable model
Bernoulli Naive Bayes	Correct only	Completely misclassified	Completely misclassified	Not suitable for numeric multi-class data

---

- Best Model: Gaussian Naive Bayes (97% accuracy, balanced performance)
- Worst Model: Bernoulli Naive Bayes (fails for multi-class numeric data)
- Notes: Hyperparameter tuning did not significantly change the performance of Multinomial or Gaussian models. Bernoulli remains unsuitable.

Decision Tree Classifier

Overall Performance

- **Accuracy: 93% across all tested configurations**
- **Precision, Recall, F1-score: Very consistent across all classes and hyperparameter settings**
- ***Iris-setosa:* Perfectly classified (precision, recall, f1 = 1.00)**
- ***Iris-versicolor:* Precision, recall, f1 = 0.89**
- ***Iris-virginica:* Precision, recall, f1 = 0.92**

Observations by Hyperparameters

Criterion	Max Depth	Accuracy	Notes
entropy	default	93%	Balanced performance, all classes well classified
entropy	3	93%	Performance same as default, low depth sufficient
entropy	10	93%	Increasing depth did not improve performance
gini	10	93%	Similar performance to entropy
gini	15	93%	No improvement; model already captures all patterns

---

- **Decision Tree performance is stable and robust for Iris dataset.**
- **Hyperparameter tuning (criterion or max depth) did not significantly change performance.**
- **Simple trees (max_depth=3) already achieve near-optimal classification.**
- **Best choice: entropy or gini with moderate depth; further deepening is unnecessary.**


Decision Tree Visualization for Iris Dataset
petal length <= 2.45
gini = 0.666
samples = 120
value = [42, 41, 37]
class = Iris-setosa
True
False
gini = 0.0
samples = 42
value = [42, 0, 0]
class = Iris-setosa
petal length <= 4.75
gini = 0.499
samples = 78
value = [0, 41, 37]
class = Iris-versicolor
gini = 0.0
samples = 37
value = [0, 37, 0]
class = Iris-versicolor
petal width <= 1.75
gini = 0.176
samples = 41
value = [0, 4, 37]
class = Iris-virginica
petal length <= 4.95
gini = 0.5
samples = 6
value = [0, 3, 3]
class = Iris-versicolor
petal length <= 4.85
gini = 0.056
samples = 35
value = [0, 1, 34]
class = Iris-virginica
gini = 0.0
samples = 2
value = [0, 2, 0]
class = Iris-versicolor
petal width <= 1.65
gini = 0.375
samples = 4
value = [0, 1, 3]
class = Iris-virginica
sepal width <= 3.1
gini = 0.444
samples = 3
value = [0, 1, 2]
class = Iris-virginica
gini = 0.0
samples = 32
value = [0, 0, 32]
class = Iris-virginica
gini = 0.0
samples = 3
value = [0, 0, 3]
class = Iris-virginica
gini = 0.0
samples = 1
value = [0, 1, 0]
class = Iris-versicolor
gini = 0.0
samples = 2
value = [0, 0, 2]
class = Iris-virginica
gini = 0.0
samples = 1
value = [0, 1, 0]
class = Iris-versicolor

Breast Cancer Dataset Analysis:

The Breast Cancer Wisconsin dataset is a binary classification problem for medical diagnosis. It contains 569 instances with 30 continuous features. The goal is to classify tumors as 'Benign' (B) or 'Malignant' (M).

Naive Bayes Classifiers

The Naive Bayes models showed varied and more nuanced performance on this dataset.

Accuracy Comparison

Model	Original Accuracy	Hyperparameter-Tuned Accuracy
Multinomial Naive Bayes	86%	86%
Gaussian Naive Bayes	93%	91%
Bernoulli Naive Bayes	67%	67%

Weighted Average Metrics Comparison

Model	Original Precision	Tuned Precision	Original Recall	Tuned Recall	Original F1	Tuned F1
Multinomial Naive Bayes	0.88	0.88	0.86	0.86	0.85	0.85
Gaussian Naive Bayes	0.93	0.92	0.93	0.91	0.93	0.91
Bernoulli Naive Bayes	0.44	0.44	0.67	0.67	0.53	0.53

Class-wise Observations

Model	Benign (B)	Malignant (M)	Notes
Multinomial Naive Bayes	High precision & recall	Lower recall (0.61)	Hyperparameter tuning did not improve performance
Gaussian Naive Bayes	Very high precision & recall	Improved recall after tuning (0.74)	Most reliable model for this dataset
Bernoulli Naive Bayes	Correctly classified	Completely misclassified	Not suitable for numeric features

---

- Best Model: Gaussian Naive Bayes (Original: 93%, Tuned: 91%, balanced performance)
- Worst Model: Bernoulli Naive Bayes (fails for numeric features, 67% accuracy)
- Notes: Hyperparameter tuning slightly improved Gaussian recall for malignant class, no effect on Multinomial or Bernoulli models.

Decision Tree Classifier

The Decision Tree classifier was a very robust and effective model for this task.

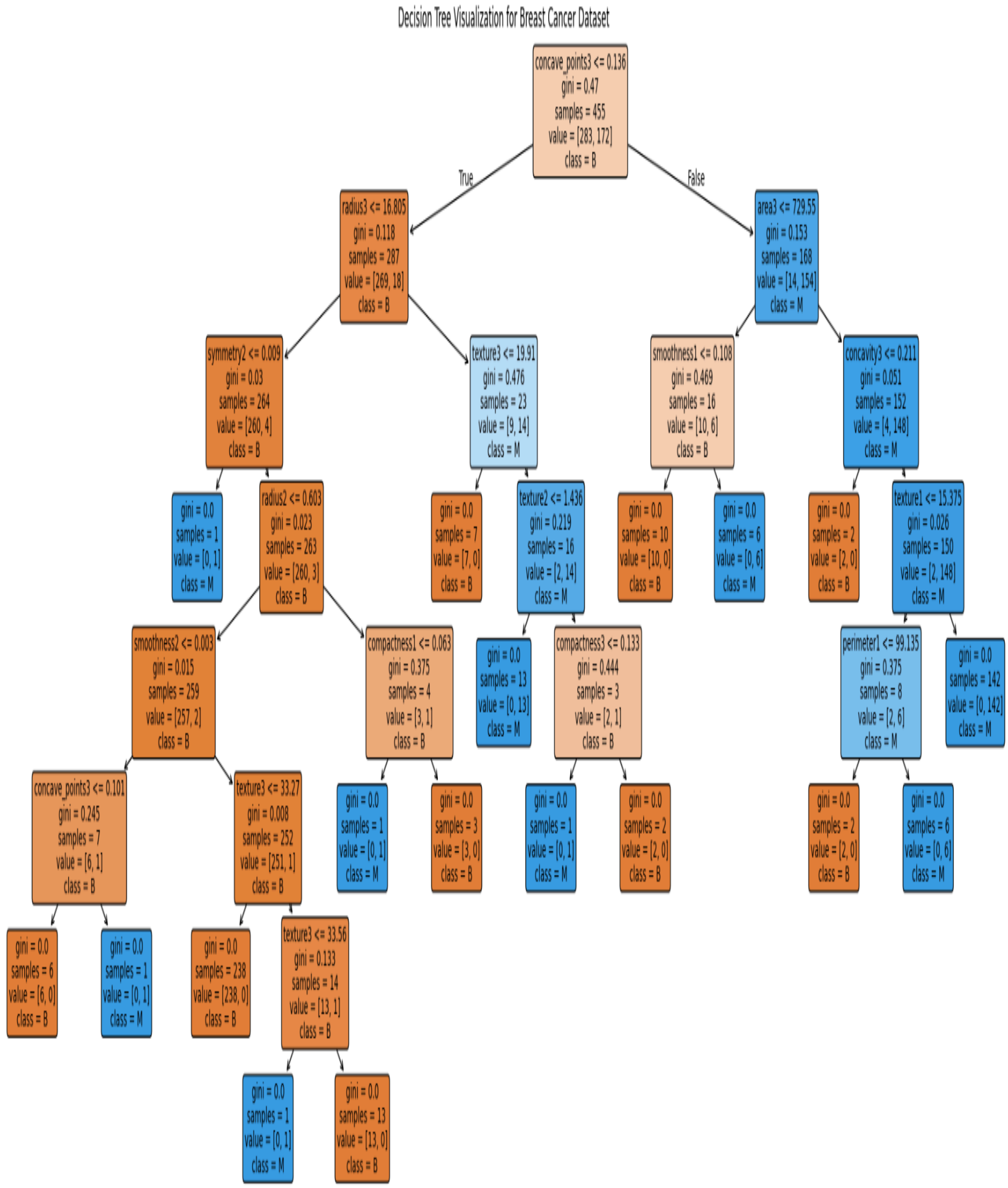
Overall Performance

- **Accuracy: 92% across all tested configurations**
- **Precision, Recall, F1-score:**
  - **Benign (B): Precision = 0.93, Recall = 0.95, F1 = 0.94**
  - **Malignant (M): Precision = 0.90, Recall = 0.88, F1 = 0.89**
- **Consistent performance across all configurations and hyperparameters**

Observations by Hyperparameters

Criterion	Max Depth	Accuracy	Notes
default	default	92%	Balanced classification for both classes
entropy	3	92%	Low depth sufficient, maintains performance
entropy	10	92%	No improvement with deeper tree
gini	10	92%	Similar performance to entropy
gini	15	92%	Increasing depth does not improve results

- **Decision Tree performs well and consistently on the Breast Cancer dataset.**
- **Hyperparameter tuning (criterion or max depth) does not significantly change performance.**
- **Moderate depth trees already capture sufficient patterns for accurate classification.**
- **Best choice: gini or entropy with moderate depth (e.g., max_depth=3–10).**


Decision Tree Visualization for Breast Cancer Dataset
concave_points3 <= 0.136
gini = 0.47
samples = 455
value = [283, 172]
class = B
True
False
radius3 <= 16.805
gini = 0.118
samples = 287
value = [269, 18]
class = B
area3 <= 729.55
gini = 0.153
samples = 168
value = [14, 154]
class = M
symmetry2 <= 0.009
gini = 0.03
samples = 264
value = [260, 4]
class = B
texture3 <= 19.91
gini = 0.476
samples = 23
value = [9, 14]
class = M
smoothness1 <= 0.108
gini = 0.469
samples = 16
value = [10, 6]
class = B
concavity3 <= 0.211
gini = 0.051
samples = 152
value = [4, 148]
class = M
gini = 0.0
samples = 1
value = [0, 1]
class = M
radius2 <= 0.603
gini = 0.023
samples = 263
value = [260, 3]
class = B
gini = 0.0
samples = 7
value = [7, 0]
class = B
texture2 <= 1.436
gini = 0.219
samples = 16
value = [2, 14]
class = M
gini = 0.0
samples = 10
value = [10, 0]
class = B
gini = 0.0
samples = 6
value = [0, 6]
class = M
gini = 0.0
samples = 2
value = [2, 0]
class = B
texture1 <= 15.375
gini = 0.026
samples = 150
value = [2, 148]
class = M
smoothness2 <= 0.003
gini = 0.015
samples = 259
value = [257, 2]
class = B
compactness1 <= 0.063
gini = 0.375
samples = 4
value = [3, 1]
class = B
gini = 0.0
samples = 13
value = [0, 13]
class = M
compactness3 <= 0.133
gini = 0.444
samples = 3
value = [2, 1]
class = B
perimeter1 <= 99.135
gini = 0.375
samples = 8
value = [2, 6]
class = M
gini = 0.0
samples = 142
value = [0, 142]
class = M
concave_points3 <= 0.101
gini = 0.245
samples = 7
value = [6, 1]
class = B
texture3 <= 33.27
gini = 0.008
samples = 252
value = [251, 1]
class = B
gini = 0.0
samples = 1
value = [0, 1]
class = M
gini = 0.0
samples = 3
value = [3, 0]
class = B
gini = 0.0
samples = 1
value = [0, 1]
class = M
gini = 0.0
samples = 2
value = [2, 0]
class = B
gini = 0.0
samples = 2
value = [2, 0]
class = B
gini = 0.0
samples = 6
value = [0, 6]
class = M
gini = 0.0
samples = 6
value = [6, 0]
class = B
gini = 0.0
samples = 1
value = [0, 1]
class = M
gini = 0.0
samples = 238
value = [238, 0]
class = B
texture3 <= 33.56
gini = 0.133
samples = 14
value = [13, 1]
class = B
gini = 0.0
samples = 1
value = [0, 1]
class = M
gini = 0.0
samples = 13
value = [13, 0]
class = B