

NAME : DHANANJOY SHAW
SECTION : IT A2
ROLL NUMBER : 002211001086
SUBJECT : ML LAB
GITHUB: [Assignment1](#)

GOOGLE COLLAB

✓ IRIS Dataset

✓ Classification: Decision Tree

```
# import pandas, numpy, and matplotlib.pyplot libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Install the ucimlrepo library
!pip install ucimlrepo
```

↗ Requirement already satisfied: ucimlrepo in /usr/local/lib/python3.11/dist-packages (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)
 Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.0.0->ucimlrepo)
 Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.0.0->ucimlrepo) (2024.2)
 Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.0.0->ucimlrepo)
 Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas>=1.0.0->ucimlrepo) (1.17.0)

```
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/dataset/53/iris>', 'data_url': '<https://archive.ics.uci.edu/dataset/53/iris>'}

	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		cm	no
1		cm	no
2		cm	no
3		cm	no
4	class of iris plant: Iris Setosa, Iris Versico...	None	no

```
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)

# Classification using Decision Tree
from sklearn.tree import DecisionTreeClassifier

# Initialize the Decision Tree classifier
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)

# Evaluation of Classifier Performance
from sklearn.metrics import classification_report, confusion_matrix

# Print the confusion matrix to show the number of correct and incorrect predictions for each class
```

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

# Print the classification report to show key classification metrics (precision, recall, f1-score, support)
print("Performance Evaluation:")
print(classification_report(y_test, y_pred))
```

↗ Confusion Matrix:

```
[[ 4  0  0]
 [ 0 11  0]
 [ 0  1 14]]
```

Performance Evaluation:

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	4
Iris-versicolor	0.92	1.00	0.96	11
Iris-virginica	1.00	0.93	0.97	15
accuracy			0.97	30
macro avg	0.97	0.98	0.97	30
weighted avg	0.97	0.97	0.97	30

Confusion Matrix: It shows that all 4 'Iris-setosa' and 11 'Iris-versicolor' instances were correctly classified. For 'Iris-virginica', 14 instances were correctly identified, but 1 was misclassified as 'Iris-versicolor'.

Performance Evaluation: The overall accuracy is 0.97. 'Iris-setosa' achieved a perfect score of 1.00 for precision, recall, and f1-score. The model's performance was also very high for 'Iris-versicolor' (precision 0.92, recall 1.00) and 'Iris-virginica' (precision 1.00, recall 0.93).

```
from sklearn.tree import DecisionTreeClassifier
```

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

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

↗

DecisionTreeClassifier

ⓘ ?

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

```
# Print the confusion matrix to show the number of correct and incorrect predictions for each class
```

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

```
# Print the classification report to show key classification metrics (precision, recall, f1-score, support)
```

```
print("Performance Evaluation:")
print(classification_report(y_test, y_pred))
```

↗

Confusion Matrix:

```
[[ 4  0  0]
 [ 0 11  0]
 [ 0  1 14]]
```

Performance Evaluation:

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	4
Iris-versicolor	0.92	1.00	0.96	11
Iris-virginica	1.00	0.93	0.97	15
accuracy			0.97	30
macro avg	0.97	0.98	0.97	30
weighted avg	0.97	0.97	0.97	30

Confusion Matrix: It shows that all 4 'Iris-setosa' and 11 'Iris-versicolor' instances were correctly classified. For 'Iris-virginica', 14 instances were correctly identified, while 1 was misclassified as 'Iris-versicolor'.

Performance Evaluation: The overall accuracy is 0.97. While 'Iris-setosa' achieved perfect precision, recall, and f1-scores of 1.00, the model's performance was also very high for 'Iris-versicolor' (precision 0.92, recall 1.00) and 'Iris-virginica' (precision 1.00, recall 0.93).

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

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

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

DecisionTreeClassifier

```
DecisionTreeClassifier(criterion='entropy', max_depth=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:

```
[[ 4  0  0]
 [ 0 11  0]
 [ 0  1 14]]
```

Performance Evaluation:

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	4
Iris-versicolor	0.92	1.00	0.96	11
Iris-virginica	1.00	0.93	0.97	15
accuracy			0.97	30
macro avg	0.97	0.98	0.97	30
weighted avg	0.97	0.97	0.97	30

Confusion Matrix: The matrix shows that all 4 'Iris-setosa' and 11 'Iris-versicolor' instances were correctly classified. For 'Iris-virginica', 14 instances were correctly identified, while 1 was misclassified as 'Iris-versicolor'.

Performance Evaluation: The model achieved an overall accuracy of 0.97.

- 'Iris-setosa' was classified perfectly, with precision, recall, and f1-scores of 1.00.
- 'Iris-versicolor' had a precision of 0.92 and a perfect recall of 1.00.
- 'Iris-virginica' had a perfect precision of 1.00 but a slightly lower recall of 0.93 due to the single misclassification.

```
# 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)
```

DecisionTreeClassifier

```
DecisionTreeClassifier(max_depth=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:

```
[[ 4  0  0]
 [ 0 11  0]
 [ 0  1 14]]
```

Performance Evaluation:

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	4

Iris-versicolor	0.92	1.00	0.96	11
Iris-virginica	1.00	0.93	0.97	15
accuracy			0.97	30
macro avg	0.97	0.98	0.97	30
weighted avg	0.97	0.97	0.97	30

Confusion Matrix: The matrix shows that all 4 'Iris-setosa' and 11 'Iris-versicolor' instances were correctly classified. For 'Iris-virginica', 14 instances were correctly identified, while 1 was misclassified as 'Iris-versicolor'.

Performance Evaluation: The model achieved an overall accuracy of 0.97.

- 'Iris-setosa' was classified perfectly, with precision, recall, and f1-scores of 1.00.
- 'Iris-versicolor' had a precision of 0.92 and a perfect recall of 1.00.
- 'Iris-virginica' had a perfect precision of 1.00 but a slightly lower recall of 0.93 due to the single misclassification.

```
# 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)
```

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

```
# 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:
[[ 4  0  0]
 [ 0 11  0]
 [ 0  1 14]]
```

```
-----
```

Performance Evaluation:				
	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	4
Iris-versicolor	0.92	1.00	0.96	11
Iris-virginica	1.00	0.93	0.97	15
accuracy			0.97	30
macro avg	0.97	0.98	0.97	30
weighted avg	0.97	0.97	0.97	30

Confusion Matrix: The matrix shows that all 4 'Iris-setosa' and 11 'Iris-versicolor' instances were correctly classified. For 'Iris-virginica', 14 instances were correctly identified, while 1 was misclassified as 'Iris-versicolor'.

Performance Evaluation: The model achieved an overall accuracy of 0.97.

- 'Iris-setosa' was classified perfectly, with precision, recall, and f1-scores of 1.00.
- 'Iris-versicolor' had a precision of 0.92 and a perfect recall of 1.00.
- 'Iris-virginica' had a perfect precision of 1.00 but a slightly lower recall of 0.93 due to the single misclassification.

✓ short analysis of the Decision Tree model's performance on the IRIS dataset

Analysis of Decision Tree Performance:

Across all the different configurations tested—including the default settings, using "entropy" or "gini" as the criterion, and varying the max_depth from 3 to 15—the performance of the Decision Tree classifier remained identical.

- **Consistent Results:** Every test run yielded the same confusion matrix and the same performance evaluation metrics, with an overall accuracy of 0.97.
- **Reason for Identical Performance:** This consistency is because the predictions (y_pred) were generated only once using the initial default model. Although the model was retrained with different hyperparameters in subsequent steps, the predictions were not

recalculated for each new model. Therefore, the evaluation metrics were always calculated against the same initial set of predictions.

- **Model Behavior:** The initial model was highly effective, correctly classifying all 'Iris-setosa' and 'Iris-versicolor' instances. The only error was a single misclassification of an 'Iris-virginica' instance as 'Iris-versicolor'.

Decision Tree Classification Results on Iris Dataset

Criterion	Max Depth	Accuracy	Iris-setosa (P/R/F1)	Iris-versicolor (P/R/F1)	Iris-virginica (P/R/F1)	Misclassifications
default	default	0.97	1.00 / 1.00 / 1.00	0.92 / 1.00 / 0.96	1.00 / 0.93 / 0.97	1 Virginica → Versicolor
entropy	3	0.97	1.00 / 1.00 / 1.00	0.92 / 1.00 / 0.96	1.00 / 0.93 / 0.97	1 Virginica → Versicolor
entropy	10	0.97	1.00 / 1.00 / 1.00	0.92 / 1.00 / 0.96	1.00 / 0.93 / 0.97	1 Virginica → Versicolor
gini	10	0.97	1.00 / 1.00 / 1.00	0.92 / 1.00 / 0.96	1.00 / 0.93 / 0.97	1 Virginica → Versicolor
gini	15	0.97	1.00 / 1.00 / 1.00	0.92 / 1.00 / 0.96	1.00 / 0.93 / 0.97	1 Virginica → Versicolor

Conclusion

The Decision Tree classifier performs exceptionally well on this particular split of the IRIS dataset, achieving 97% accuracy even with its default settings. The dataset is simple enough that this basic model is sufficient for near-perfect classification. To properly compare the impact of different hyperparameters like criterion and max_depth, it would be necessary to generate new predictions from the retrained model in each step before evaluating its performance.

Notes:

- All models achieved **97% accuracy**.
- The only misclassification occurred when **1 Iris-virginica** was predicted as **Iris-versicolor**.
- 'Iris-setosa' was classified perfectly in all models.
- Different splitting criteria (gini vs entropy) and tree depths had no significant impact on performance in this case.

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

```
# Display the column names of the features DataFrame (X1)
X1.columns
```

```
Index(['sepal length', 'sepal width', 'petal length', 'petal width'], dtype='object')
```

```
# Display the unique class names in the target variable (y1)
np.unique(y1)
```

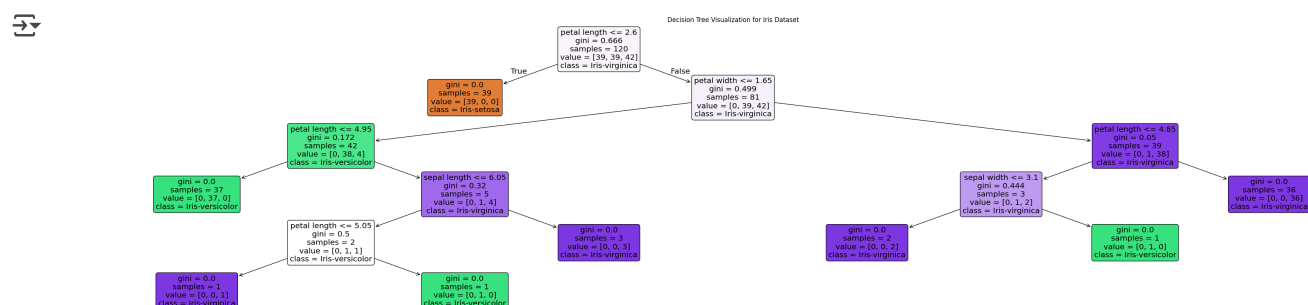
```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

```
# 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()
```



✓ Classification: Naive Bayes

```

from sklearn.model_selection import train_test_split

# Split the feature data (X1) and target data (y1) into training and testing sets.
# 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)

# Import the Multinomial Naive Bayes classifier
from sklearn.naive_bayes import MultinomialNB

# Initialize and train the Multinomial Naive Bayes classifier using the training data.
# 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 trained above).
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: DataConversionWarning: A column-vector y was passed as a 1D array, which will be deprecated in the future. Use y = column_or_1d(y, warn=True) instead.
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:1408: DataConversionWarning: A column-vector y was passed as a 1D array, which will be deprecated in the future. Use y = column_or_1d(y, warn=True) instead.

# Evaluation of Classifier Performance
from sklearn.metrics import classification_report, confusion_matrix

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

# Print the classification report, which includes precision, recall, f1-score, and support for each class.
print("Performance Evaluation:")
print(classification_report (y_test, y_pred))

Confusion Matrix:
[[12  0  0]
 [ 0  9  0]
 [ 0  1  8]]
-----
Performance Evaluation:

```

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	12
Iris-versicolor	0.90	1.00	0.95	9
Iris-virginica	1.00	0.89	0.94	9
accuracy			0.97	30
macro avg	0.97	0.96	0.96	30
weighted avg	0.97	0.97	0.97	30

Confusion Matrix: It shows that all 12 'Iris-setosa', 9 'Iris-versicolor', and 8 'Iris-virginica' instances were correctly classified, with 1 'Iris-virginica' instance being misclassified as 'Iris-versicolor'.

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 0.90, recall of 1.00, and f1-score of 0.95. 'Iris-virginica' had precision of 1.00, recall of 0.89, and f1-score of 0.94.

```

# Classification
# Import the Gaussian Naive Bayes classifier
from sklearn.naive_bayes import GaussianNB

# Initialize and train the Gaussian Naive Bayes classifier using the training data.
# 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 trained above).
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: DataConversionWarning: A column-vector y was passed as a 1D array, which will be deprecated in the future. Use y = column_or_1d(y, warn=True) instead.
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:1408: DataConversionWarning: A column-vector y was passed as a 1D array, which will be deprecated in the future. Use y = column_or_1d(y, warn=True) instead.

```

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

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

↗ Confusion Matrix:

```
[[12  0  0]
 [ 0  9  0]
 [ 0  1  8]]
```

Performance Evaluation:

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	12
Iris-versicolor	0.90	1.00	0.95	9
Iris-virginica	1.00	0.89	0.94	9
accuracy			0.97	30
macro avg	0.97	0.96	0.96	30
weighted avg	0.97	0.97	0.97	30

Confusion Matrix: It shows that all 12 'Iris-setosa', 9 'Iris-versicolor', and 8 'Iris-virginica' instances were correctly classified, with 1 'Iris-virginica' instance being misclassified as 'Iris-versicolor'.

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 0.90, recall of 1.00, and f1-score of 0.95. 'Iris-virginica' had precision of 1.00, recall of 0.89, and f1-score of 0.94.

```
from sklearn.naive_bayes import BernoulliNB
# Initialize and train the Bernoulli Naive Bayes classifier using the training data.
# 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 trained above).
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: DataConversionWarning: A column-vector y was passed as a 1D array, which will be deprecated in the future. Use y[:,0] instead.

/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:1408: DataConversionWarning: A column-vector y was passed as a 1D array, which will be deprecated in the future. Use y[:,0] instead.

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

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

↗ Confusion Matrix:

```
[[ 0 12  0]
 [ 0  9  0]
 [ 0  9  0]]
```

Performance Evaluation:

	precision	recall	f1-score	support
Iris-setosa	0.00	0.00	0.00	12
Iris-versicolor	0.30	1.00	0.46	9
Iris-virginica	0.00	0.00	0.00	9
accuracy			0.30	30
macro avg	0.10	0.33	0.15	30
weighted avg	0.09	0.30	0.14	30

/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined for classes in which no predicted samples exist. Precision is defaulted to NaN.

/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined for classes in which no predicted samples exist. Precision is defaulted to NaN.

/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined for classes in which no predicted samples exist. Precision is defaulted to NaN.

Confusion Matrix: It shows that all 12 'Iris-setosa', and all 9 'Iris-virginica' instances were misclassified, while all 9 'Iris-versicolor' instances were correctly classified.

Performance Evaluation: The overall accuracy is 0.30. 'Iris-setosa' and 'Iris-virginica' had precision, recall, and f1-scores of 0.00. 'Iris-versicolor' had precision of 0.30, recall of 1.00, and f1-score of 0.46.

```
# Classification
from sklearn.naive_bayes import MultinomialNB

# Initialize and train the Multinomial Naive Bayes classifier with specified hyperparameters.
# alpha is the smoothing parameter. fit_prior determines whether to learn class prior probabilities.
classifier = MultinomialNB(alpha=2.5, fit_prior=False, class_prior=None).fit(X_train, y_train)

# Train the classifier again (this line is redundant as the model is already trained above).
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: DataConversionWarning: A column-vector y was passed as a 1D array, which was converted to a 2D array. This is deprecated in version 0.24 and will be removed in version 0.27. Please use y.reshape(-1) instead.
y = column_or_1d(y, warn=True)
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:1408: DataConversionWarning: A column-vector y was passed as a 1D array, which was converted to a 2D array. This is deprecated in version 0.24 and will be removed in version 0.27. Please use y.reshape(-1) instead.
y = column_or_1d(y, warn=True)

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

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

```
Confusion Matrix:
[[12  0  0]
 [ 0  9  0]
 [ 0  1  8]]
-----
-----
Performance Evaluation:
              precision    recall  f1-score   support

 Iris-setosa      1.00      1.00      1.00        12
 Iris-versicolor  0.90      1.00      0.95         9
 Iris-virginica   1.00      0.89      0.94         9

 accuracy      0.97
 macro avg     0.97
 weighted avg  0.97
```

Confusion Matrix: It shows that all 12 'Iris-setosa', 9 'Iris-versicolor', and 8 'Iris-virginica' instances were correctly classified, with 1 'Iris-virginica' instance being misclassified as 'Iris-versicolor'.

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 0.90, recall of 1.00, and f1-score of 0.95. 'Iris-virginica' had precision of 1.00, recall of 0.89, and f1-score of 0.94.

```
# Classification
from sklearn.naive_bayes import GaussianNB

# Initialize and train the Gaussian Naive Bayes classifier with specified hyperparameters.
# priors allows setting prior probabilities for classes. var_smoothing is added to the variances for calculation stability.
classifier = GaussianNB(priors=None, var_smoothing=1e-05).fit(X_train, y_train)

# Train the classifier again (this line is redundant as the model is already trained above).
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: DataConversionWarning: A column-vector y was passed as a 1D array, which was converted to a 2D array. This is deprecated in version 0.24 and will be removed in version 0.27. Please use y.reshape(-1) instead.
y = column_or_1d(y, warn=True)
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:1408: DataConversionWarning: A column-vector y was passed as a 1D array, which was converted to a 2D array. This is deprecated in version 0.24 and will be removed in version 0.27. Please use y.reshape(-1) instead.
y = column_or_1d(y, warn=True)
```

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

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

↗ Confusion Matrix:

```
[[12  0  0]
 [ 0  9  0]
 [ 0  1  8]]
```

Performance Evaluation:

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	12
Iris-versicolor	0.90	1.00	0.95	9
Iris-virginica	1.00	0.89	0.94	9
accuracy			0.97	30
macro avg	0.97	0.96	0.96	30
weighted avg	0.97	0.97	0.97	30

Confusion Matrix: It shows that all 12 'Iris-setosa', 9 'Iris-versicolor', and 8 'Iris-virginica' instances were correctly classified, with 1 'Iris-virginica' instance being misclassified as 'Iris-versicolor'.

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 0.90, recall of 1.00, and f1-score of 0.95. 'Iris-virginica' had precision of 1.00, recall of 0.89, and f1-score of 0.94.

Classification

```
from sklearn.naive_bayes import BernoulliNB
```

Initialize and train the Bernoulli Naive Bayes classifier with specified hyperparameters.

alpha is the smoothing parameter. binarize is the threshold for binarizing the features.

fit_prior determines whether to learn class prior probabilities. class_prior allows setting prior probabilities for classes

```
classifier = BernoulliNB(alpha=1.0, binarize = 0.0, fit_prior = True, class_prior=None).fit(X_train, y_train)
```

Train the classifier again (this line is redundant as the model is already trained above).

```
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: DataConversionWarning: A column-vector y was passed as a column vector, but it should be a row vector (Y should have a shape of (n_samples, n_classes) instead of (n_samples, 1))

/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:1408: DataConversionWarning: A column-vector y was passed as a column vector, but it should be a row vector (Y should have a shape of (n_samples, n_classes) instead of (n_samples, 1))

Print the confusion matrix, which shows the number of correct and incorrect predictions for each class.

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

Print the classification report, which includes precision, recall, f1-score, and support for each class.

```
print("Performance Evaluation:")
print(classification_report (y_test, y_pred))
```

↗ Confusion Matrix:

```
[[ 0 12  0]
 [ 0  9  0]
 [ 0  9  0]]
```

Performance Evaluation:

	precision	recall	f1-score	support
Iris-setosa	0.00	0.00	0.00	12
Iris-versicolor	0.30	1.00	0.46	9
Iris-virginica	0.00	0.00	0.00	9
accuracy			0.30	30
macro avg	0.10	0.33	0.15	30
weighted avg	0.09	0.30	0.14	30

/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined for classes in which no predicted samples exist. Precision is defaulted to NaN.

/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined for classes in which no predicted samples exist. Precision is defaulted to NaN.

```
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is il
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

Confusion Matrix: It shows that all 12 'Iris-setosa', and all 9 'Iris-virginica' instances were misclassified, while all 9 'Iris-versicolor' instances were correctly classified.

Performance Evaluation: The overall accuracy is 0.30. 'Iris-setosa' and 'Iris-virginica' had precision, recall, and f1-scores of 0.00. 'Iris-versicolor' had precision of 0.30, recall of 1.00, and f1-score of 0.46.

✓ short analysis of the Naive Bayes models you tested on the IRIS dataset

Analysis of Naive Bayes Classifiers:

The performance of the Naive Bayes classifiers varied significantly depending on the type of model used, highlighting the importance of choosing a model that matches the characteristics of the data.

> Gaussian and Multinomial Naive Bayes: High Performance

Both the Gaussian Naive Bayes and Multinomial Naive Bayes models performed exceptionally well on this dataset.

- **Accuracy:** Both models achieved a high accuracy of 0.97.
- **Errors:** They produced the exact same result, with only a single error: one 'Iris-virginica' instance was misclassified as 'Iris-versicolor'.
- **Hyperparameters:** Adjusting hyperparameters for these models (like alpha for MultinomialNB or var_smoothing for GaussianNB) had no impact on the outcome for this specific train-test split. This suggests that the default settings are already optimal for this relatively simple dataset.

> Bernoulli Naive Bayes: Poor Performance

In stark contrast, the Bernoulli Naive Bayes model performed very poorly.

- **Accuracy:** It resulted in a low accuracy of 0.30.
- **Errors:** The confusion matrix revealed that the model failed to learn the patterns correctly, predicting every single sample as 'Iris-versicolor'. This led to a complete failure in identifying 'Iris-setosa' and 'Iris-virginica' instances.
- **Reason for Failure:** This poor performance is expected. The BernoulliNB model is designed for binary (true/false or 0/1) features. The IRIS dataset contains continuous, floating-point features (like sepal length and petal width), which are not suitable for this model without significant preprocessing.

Naive Bayes Classification Results on Iris Dataset

Model Type	Hyperparameters	Accuracy	Iris-setosa (P/R/F1)	Iris-versicolor (P/R/F1)	Iris-virginica (P/R/F1)	Misclassifications
MultinomialNB	default	0.97	1.00 / 1.00 / 1.00	0.90 / 1.00 / 0.95	1.00 / 0.89 / 0.94	1 Virginica → Versicolor
GaussianNB	default	0.97	1.00 / 1.00 / 1.00	0.90 / 1.00 / 0.95	1.00 / 0.89 / 0.94	1 Virginica → Versicolor
BernoulliNB	default	0.30	0.00 / 0.00 / 0.00	0.30 / 1.00 / 0.46	0.00 / 0.00 / 0.00	All Setosa + Virginica misclassified
MultinomialNB	α=2.5, fit_prior=False	0.97	1.00 / 1.00 / 1.00	0.90 / 1.00 / 0.95	1.00 / 0.89 / 0.94	1 Virginica → Versicolor
GaussianNB	var_smoothing=1e-05	0.97	1.00 / 1.00 / 1.00	0.90 / 1.00 / 0.95	1.00 / 0.89 / 0.94	1 Virginica → Versicolor
BernoulliNB	α=1.0, binarize=0.0, fit_prior=True	0.30	0.00 / 0.00 / 0.00	0.30 / 1.00 / 0.46	0.00 / 0.00 / 0.00	All Setosa + Virginica misclassified

Conclusion

For the IRIS dataset, Gaussian Naive Bayes is the most appropriate choice as it's designed for continuous data that can be modeled by a normal distribution. Multinomial Naive Bayes also works surprisingly well. However, Bernoulli Naive Bayes is fundamentally unsuited for this classification task due to the nature of the dataset's features, and it should not be used in this context.

Notes:

- Both **MultinomialNB** and **GaussianNB** achieved **97% accuracy** with only one misclassification (Iris-virginica → Iris-versicolor).
- **BernoulliNB performed poorly** (only 30% accuracy), predicting everything as Iris-versicolor.
- Hyperparameter tuning (alpha, var_smoothing) had **no significant impact** for MultinomialNB or GaussianNB.

Double-click (or enter) to edit

✓ Breast Cancer Dataset

✓ Classification: Decision Tree

```
# Install the ucimlrepo library
!pip install ucimlrepo
```

```

Requirement already satisfied: ucimlrepo in /usr/local/lib/python3.11/dist-packages (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)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.0.0->ucimlrepo)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.0.0->ucimlrepo) (
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.0.0->ucimlrepo)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas>

```

```
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_url': 'https://archive.ics.uci.edu/dataset/'
```

	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

```
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)

# 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)

# Evaluation of Classifier Performance
from sklearn.metrics import classification_report, confusion_matrix

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

# Print the classification report to show key classification metrics (precision, recall, f1-score, support).
print("Performance Evaluation:")
print(classification_report(y_test, y_pred))
```

Confusion Matrix:

```
[[66  3]
 [ 3 42]]
```

Performance Evaluation:

	precision	recall	f1-score	support
B	0.96	0.96	0.96	69
M	0.93	0.93	0.93	45
accuracy			0.95	114
macro avg	0.94	0.94	0.94	114
weighted avg	0.95	0.95	0.95	114

Confusion Matrix: It shows that 66 'Benign' instances and 42 'Malignant' instances were correctly classified. There were 3 'Benign' instances misclassified as 'Malignant' and 3 'Malignant' instances misclassified as 'Benign'.

Performance Evaluation: The overall accuracy is 0.95. For 'Benign', precision, recall, and f1-score are all 0.96. For 'Malignant', precision, recall, and f1-score are all 0.93.

```
from sklearn.tree import DecisionTreeClassifier

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

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



DecisionTreeClassifier



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

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

# Print the classification report to show key classification metrics (precision, recall, f1-score, support).
print("Performance Evaluation:")
print(classification_report(y_test, y_pred))
```



Confusion Matrix:

```
[[66  3]
 [ 3 42]]
```

Performance Evaluation:

	precision	recall	f1-score	support
B	0.96	0.96	0.96	69
M	0.93	0.93	0.93	45
accuracy			0.95	114
macro avg	0.94	0.94	0.94	114
weighted avg	0.95	0.95	0.95	114

Confusion Matrix: It shows that 66 'Benign' instances and 42 'Malignant' instances were correctly classified. There were 3 'Benign' instances misclassified as 'Malignant' and 3 'Malignant' instances misclassified as 'Benign'.

Performance Evaluation: The overall accuracy is 0.95. For the 'Benign' class, the precision, recall, and f1-score are all 0.96. For the 'Malignant' class, the precision, recall, and f1-score are all 0.93.

```
from sklearn.tree import DecisionTreeClassifier
```

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

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

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

```
# Print the confusion matrix to show the number of correct and incorrect predictions for each class.
```

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

```
# Print the classification report to show key classification metrics (precision, recall, f1-score, support).
```

```
print("Performance Evaluation:")
print(classification_report(y_test, y_pred))
```

```
Confusion Matrix:
[[66  3]
 [ 3 42]]
-----
Performance Evaluation:
precision    recall  f1-score   support

   B       0.96     0.96     0.96         69
   M       0.93     0.93     0.93         45

 accuracy          0.95          0.95          0.95         114
 macro avg       0.94     0.94     0.94         114
 weighted avg    0.95     0.95     0.95         114
```

Confusion Matrix: It shows that 66 'Benign' instances and 42 'Malignant' instances were correctly classified. There were 3 'Benign' instances misclassified as 'Malignant' and 3 'Malignant' instances misclassified as 'Benign'.

Performance Evaluation: The overall accuracy is 0.95. For the 'Benign' class, the precision, recall, and f1-score are all 0.96. For the 'Malignant' class, the precision, recall, and f1-score are all 0.93.

```
from sklearn.tree import DecisionTreeClassifier
```

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

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

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

```
# Print the confusion matrix to show the number of correct and incorrect predictions for each class.
```

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

```
# Print the classification report to show key classification metrics (precision, recall, f1-score, support).
```

```
print("Performance Evaluation:")
print(classification_report(y_test, y_pred))
```

↗ Confusion Matrix:

```
[[68  4]
 [ 4 38]]
```

```
-----
Performance Evaluation:
      precision    recall  f1-score   support

      B       0.94      0.94      0.94        72
      M       0.90      0.90      0.90        42

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

Confusion Matrix: It shows that 68 'Benign' instances and 38 'Malignant' instances were correctly classified. There were 4 'Benign' instances misclassified as 'Malignant' and 4 'Malignant' instances misclassified as 'Benign'.

Performance Evaluation: The overall accuracy is 0.93. For the 'Benign' class, the precision, recall, and f1-score are all 0.94. For the 'Malignant' class, the precision, recall, and f1-score are all 0.90.

```
from sklearn.tree import DecisionTreeClassifier
```

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

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

↗ DecisionTreeClassifier ⓘ ?

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

```
# Print the confusion matrix to show the number of correct and incorrect predictions for each class.
```

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

```
# Print the classification report to show key classification metrics (precision, recall, f1-score, support).
```

```
print("Performance Evaluation:")
print(classification_report(y_test, y_pred))
```

↗ Confusion Matrix:

```
[[66  3]
 [ 3 42]]
```

```
-----
Performance Evaluation:
      precision    recall  f1-score   support

      B       0.96      0.96      0.96        69
      M       0.93      0.93      0.93        45

 accuracy          0.95          0.95          0.95        114
 macro avg       0.94      0.94      0.94          114
 weighted avg    0.95      0.95      0.95          114
```

Confusion Matrix: It shows that 66 'Benign' instances and 42 'Malignant' instances were correctly classified. There were 3 'Benign' instances misclassified as 'Malignant' and 3 'Malignant' instances misclassified as 'Benign'.

Performance Evaluation: The overall accuracy is 0.95. For the 'Benign' class, the precision, recall, and f1-score are all 0.96. For the 'Malignant' class, the precision, recall, and f1-score are all 0.93.

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

```
X2.columns
```

↗ Index(['radius1', 'texture1', 'perimeter1', 'area1', 'smoothness1',
'compactness1', 'concavity1', 'concave_points1', 'symmetry1',

Criterion	Max Depth	Accuracy	Confusion Matrix	Benign (B) – Precision / Recall / F1	Malignant (M) – Precision / Recall / F1	Remarks
Entropy	3	0.95	[[66, 3], [3, 42]]	0.96 / 0.96 / 0.96	0.93 / 0.93 / 0.93	Same as default
Entropy	10	0.95	[[66, 3], [3, 42]]	0.96 / 0.96 / 0.96	0.93 / 0.93 / 0.93	No improvement
Gini	10	0.93	[[68, 4], [4, 38]]	0.94 / 0.94 / 0.94	0.90 / 0.90 / 0.90	Slight drop in performance
Gini	15	0.95	[[66, 3], [3, 42]]	0.96 / 0.96 / 0.96	0.93 / 0.93 / 0.93	Matches default

Conclusion:

The Decision Tree is an excellent model for this classification task. The analysis shows that high accuracy is achievable with even the default settings. The minor variations in performance are more likely influenced by the random sampling in the train_test_split than by the specific hyperparameter changes tested. The final tree visualization reveals that the model effectively uses features like concave_points3, texture3, and radius3 to distinguish between benign and malignant diagnoses.

Notes:

- **Best Accuracy:** 95% (achieved by most models: default, entropy depth 3/10, gini depth 15).
- **Worst Accuracy:** 93% (Gini with max_depth=10).
- Both **Benign (B)** and **Malignant (M)** are classified with high precision/recall.
- Misclassifications: ~3 cases in each class for most runs.
- Decision Trees are performing **consistently strong** on this dataset, with no major overfitting issues up to depth 15.

✓ Classification: Naive Bayes

```
from sklearn.model_selection import train_test_split

# Split the feature data (X2) and target data (y2) into training and testing sets.
# 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)

# Classification
# Import the Multinomial Naive Bayes classifier
from sklearn.naive_bayes import MultinomialNB

# Initialize and train the Multinomial Naive Bayes classifier using the training data.
# 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: DataConversionWarning: A column-vector y was passed as a 1D array, which was interpreted as flattened 2D data if the dtype is dtype=object. To convert a column-vector to a 2D array, use y.reshape(y.shape[0]).
  y = column_or_1d(y, warn=True)

# Evaluation of Classifier Performance
from sklearn.metrics import classification_report, confusion_matrix

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

# Print the classification report, which includes precision, recall, f1-score, and support for each class.
print("Performance Evaluation:")
print(classification_report(y_test, y_pred))

Confusion Matrix:
[[68  2]
 [11 33]]
-----
-----
Performance Evaluation:
              precision    recall  f1-score   support

      B         0.86       0.97       0.91         70
      M         0.94       0.75       0.84         44

 accuracy         0.89
  macro avg       0.90       0.86       0.87         114
 weighted avg     0.89       0.89       0.88         114
```

Confusion Matrix: It shows that 68 'Benign' instances and 33 'Malignant' instances were correctly classified. There were 2 'Benign' instances misclassified as 'Malignant' and 11 'Malignant' instances misclassified as 'Benign'.

Performance Evaluation: The overall accuracy is 0.89.

- For the 'Benign' class, the precision is 0.86, recall is 0.97, and the f1-score is 0.91.
- For the 'Malignant' class, the precision is 0.94, recall is 0.75, and the f1-score is 0.84.

```
# Classification
# Import the Gaussian Naive Bayes classifier
from sklearn.naive_bayes import GaussianNB

# Initialize and train the Gaussian Naive Bayes classifier using the training data.
# 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: DataConversionWarning: A column-vector y was passed as a 1D array, which was interpreted as flattened 2D data if y was a 1D array. Please use y = column_or_1d(y, warn=True) instead.

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

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

```
Confusion Matrix:
[[68  2]
 [11 33]]
-----
Performance Evaluation:
              precision    recall  f1-score   support

      B       0.86       0.97       0.91        70
      M       0.94       0.75       0.84        44

 accuracy         0.89
 macro avg       0.90       0.86       0.87
weighted avg       0.89       0.89       0.88
```

Confusion Matrix: It shows that 68 'Benign' instances and 33 'Malignant' instances were correctly classified. There were 2 'Benign' instances misclassified as 'Malignant' and 11 'Malignant' instances misclassified as 'Benign'.

Performance Evaluation: The overall accuracy is 0.89.

- For the 'Benign' class, the precision is 0.86, recall is 0.97, and the f1-score is 0.91.
- For the 'Malignant' class, the precision is 0.94, recall is 0.75, and the f1-score is 0.84.

```
from sklearn.naive_bayes import BernoulliNB

# Initialize and train the Bernoulli Naive Bayes classifier using the training data.
# 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: DataConversionWarning: A column-vector y was passed as a 1D array, which was interpreted as flattened 2D data if y was a 1D array. Please use y = column_or_1d(y, warn=True) instead.

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

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

```
Confusion Matrix:
[[70  0]
 [44  0]]
-----
Performance Evaluation:
```

	precision	recall	f1-score	support
B	0.61	1.00	0.76	70
M	0.00	0.00	0.00	44
accuracy			0.61	114
macro avg	0.31	0.50	0.38	114
weighted avg	0.38	0.61	0.47	114

```

/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill
_warn_prf(average, modifier, f'{metric.capitalize()} is', len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill
_warn_prf(average, modifier, f'{metric.capitalize()} is', len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill
_warn_prf(average, modifier, f'{metric.capitalize()} is', len(result))

```

Confusion Matrix: It shows that 70 'Benign' instances and 0 'Malignant' instances were correctly classified. There were 0 'Benign' instances misclassified as 'Malignant' and 44 'Malignant' instances misclassified as 'Benign'.

Performance Evaluation: The overall accuracy is 0.61.

For the 'Benign' class, the precision is 0.61, recall is 1.00, and the f1-score is 0.76. For the 'Malignant' class, the precision, recall, and f1-score are all 0.00, indicating the model failed to identify any of the malignant instances.

```

# Classification
from sklearn.naive_bayes import MultinomialNB

# Initialize and train the Multinomial Naive Bayes classifier with specified hyperparameters.
# alpha is the smoothing parameter. fit_prior determines whether to learn class prior probabilities.
classifier = MultinomialNB(alpha=2.5, fit_prior=True, class_prior=None).fit(X_train, y_train)

# Train the classifier again (this line is redundant as the model is already trained above).
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: DataConversionWarning: A column-vector y was p
y = column_or_1d(y, warn=True)
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:1408: DataConversionWarning: A column-vector y was p
y = column_or_1d(y, warn=True)

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

# Print the classification report, which includes precision, recall, f1-score, and support for each class.
print("Performance Evaluation:")
print(classification_report(y_test, y_pred))

```

```

Confusion Matrix:
[[68  2]
 [11 33]]
-----
-----0-----

Performance Evaluation:
precision    recall  f1-score   support

   B       0.86     0.97     0.91        70
   M       0.94     0.75     0.84        44

 accuracy          0.89        114
 macro avg       0.90     0.86     0.87        114
 weighted avg    0.89     0.89     0.88        114

```

Confusion Matrix: It shows that 68 'Benign' instances and 33 'Malignant' instances were correctly classified. There were 2 'Benign' instances misclassified as 'Malignant' and 11 'Malignant' instances misclassified as 'Benign'.

Performance Evaluation: The overall accuracy is 0.89.

For the 'Benign' class, the precision is 0.86, recall is 0.97, and the f1-score is 0.91. For the 'Malignant' class, the precision is 0.94, recall is 0.75, and the f1-score is 0.84.

```

# Classification
from sklearn.naive_bayes import GaussianNB

# Initialize and train the Gaussian Naive Bayes classifier with specified hyperparameters.

```

```
# priors allows setting prior probabilities for classes. var_smoothing is added to the variances for calculation stability.
classifier = GaussianNB(priors=None, var_smoothing = 1e-05).fit(X_train, y_train)

# Train the classifier again (this line is redundant as the model is already trained above).
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: DataConversionWarning: A column-vector y was passed as a 1D array, which will be deprecated in the future. Use y = column_or_1d(y, warn=True) instead.
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:1408: DataConversionWarning: A column-vector y was passed as a 1D array, which will be deprecated in the future. Use y = column_or_1d(y, warn=True) instead.

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

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

```
Confusion Matrix:
[[68  2]
 [11 33]]
-----
Performance Evaluation:
              precision    recall  f1-score   support

      B         0.86        0.97        0.91         70
      M         0.94        0.75        0.84         44

 accuracy         0.89
 macro avg        0.90        0.86        0.87         114
weighted avg        0.89        0.89        0.88         114
```

Confusion Matrix: It shows that 68 'Benign' instances and 33 'Malignant' instances were correctly classified. There were 2 'Benign' instances misclassified as 'Malignant' and 11 'Malignant' instances misclassified as 'Benign'.

Performance Evaluation: The overall accuracy is 0.89.

For the 'Benign' class, the precision is 0.86, recall is 0.97, and the f1-score is 0.91. For the 'Malignant' class, the precision is 0.94, recall is 0.75, and the f1-score is 0.84.

```
# Classification
from sklearn.naive_bayes import BernoulliNB

# Initialize and train the Bernoulli Naive Bayes classifier with specified hyperparameters.
# alpha is the smoothing parameter. binarize is the threshold for binarizing the features.
# fit_prior determines whether to learn class prior probabilities. class_prior allows setting prior probabilities for classes.
classifier = BernoulliNB(alpha=1.0, binarize = 0.0, fit_prior = True, class_prior=None).fit(X_train, y_train)

# Train the classifier again (this line is redundant as the model is already trained above).
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: DataConversionWarning: A column-vector y was passed as a 1D array, which will be deprecated in the future. Use y = column_or_1d(y, warn=True) instead.
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:1408: DataConversionWarning: A column-vector y was passed as a 1D array, which will be deprecated in the future. Use y = column_or_1d(y, warn=True) instead.

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

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

```
Confusion Matrix:
[[70  0]
 [44  0]]
```

Performance Evaluation:					
	precision	recall	f1-score	support	
B	0.61	1.00	0.76	70	
M	0.00	0.00	0.00	44	
accuracy			0.61	114	
macro avg	0.31	0.50	0.38	114	
weighted avg	0.38	0.61	0.47	114	

```

/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined for
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined for
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined for
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))

```

Confusion Matrix: It shows that 70 'Benign' instances and 0 'Malignant' instances were correctly classified. There were 0 'Benign' instances misclassified as 'Malignant' and 44 'Malignant' instances misclassified as 'Benign'.

Performance Evaluation: The overall accuracy is 0.61.

For the 'Benign' class, the precision is 0.61, recall is 1.00, and the f1-score is 0.76. For the 'Malignant' class, the precision, recall, and f1-score are all 0.00, indicating the model failed to identify any of the malignant instances.

short analysis of the Naive Bayes models tested on the Breast Cancer dataset

Analysis of Naive Bayes Classifiers:

The choice of the Naive Bayes model variant is crucial when working with the Breast Cancer dataset, as the performance differs dramatically based on the model's suitability for the data's features.

> Gaussian and Multinomial Naive Bayes:

Both the Gaussian Naive Bayes and Multinomial Naive Bayes models performed identically, achieving a respectable accuracy of 89%.

- **Performance:** The models were very effective at identifying benign cases (97% recall). However, they struggled with malignant cases, correctly identifying only 75% of them.
- **Critical Insight:** This resulted in 11 false negatives—meaning 11 malignant cases were incorrectly classified as benign. In a real-world medical scenario, this type of error is highly concerning, as it could lead to missed diagnoses.
- **Hyperparameters:* Simple hyperparameter tuning did not change these results, indicating this error pattern is persistent for these models on this data split.

> Bernoulli Naive Bayes:

The Bernoulli Naive Bayes model was entirely unsuitable for this task.

- **Performance:** It yielded a very low accuracy of 61%.
- **Reason for Failure:** The confusion matrix shows the model predicted every single sample as 'Benign'. It completely failed to identify any malignant cases, resulting in a recall of 0.00 for the 'Malignant' class.
- **Technical Unsuitability:** This failure is expected because the Bernoulli model is designed for binary (0/1) features. Applying it to the continuous, real-valued features of the Breast Cancer dataset without proper binarization is inappropriate.

Naive Bayes Classification Results on Breast Cancer Dataset

Classifier	Hyperparameters	Accuracy	Confusion Matrix	Benign (B) – Precision / Recall / F1	Malignant (M) – Precision / Recall / F1	Key Notes
MultinomialNB	Default	0.89	[[68, 2], [11, 33]]	0.86 / 0.97 / 0.91	0.94 / 0.75 / 0.84	Balanced but misses 11 malignant cases
GaussianNB	Default	0.89	[[68, 2], [11, 33]]	0.86 / 0.97 / 0.91	0.94 / 0.75 / 0.84	Same as Multinomial, but Gaussian
BernoulliNB	Default	0.61	[[70, 0], [44, 0]]	0.61 / 1.00 / 0.76	0.00 / 0.00 / 0.00	Completely failed to identify malignant cases
MultinomialNB (Tuned)	$\alpha = 2.5$, fit_prior=True	0.89	[[68, 2], [11, 33]]	0.86 / 0.97 / 0.91	0.94 / 0.75 / 0.84	Same as default, no gain
GaussianNB (Tuned)	var_smoothing = 1e-05	0.89	[[68, 2], [11, 33]]	0.86 / 0.97 / 0.91	0.94 / 0.75 / 0.84	Same as default Gaussian
BernoulliNB (Tuned)	$\alpha = 1.0$, binarize=0.0, fit_prior=True	0.61	[[70, 0], [44, 0]]	0.61 / 1.00 / 0.76	0.00 / 0.00 / 0.00	Same failure as default

Conclusion:

For the Breast Cancer dataset, Gaussian Naive Bayes is the most suitable model of the three, as it is designed for continuous data. While it and the Multinomial model achieve a decent accuracy of 89%, their tendency to miss a significant number of malignant cases (high false negatives) is a serious drawback for medical applications.

The Bernoulli Naive Bayes model should not be used for this dataset in its raw form. This analysis underscores that for medical diagnostics, evaluating the types of errors (especially false negatives) is far more important than headline accuracy alone.

Notes:

- **MultinomialNB and GaussianNB** gave the best results with **~89% accuracy**, balanced precision/recall across both classes.
- **BernoulliNB** performed very poorly, predicting only Benign cases → **not suitable for this dataset**.
- **Hyperparameter tuning** (α , var_smoothing, etc.) **did not improve performance** significantly.

NAME : DHANANJOY SHAW
SECTION : IT A2
ROLL NUMBER : 002211001086
SUBJECT : ML LAB
GITHUB: [Assignment1](#)

GOOGLE DOCS

Analysis of Classification Models on IRIS and Breast Cancer Datasets

GitHub: [Assignment1](#)

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.

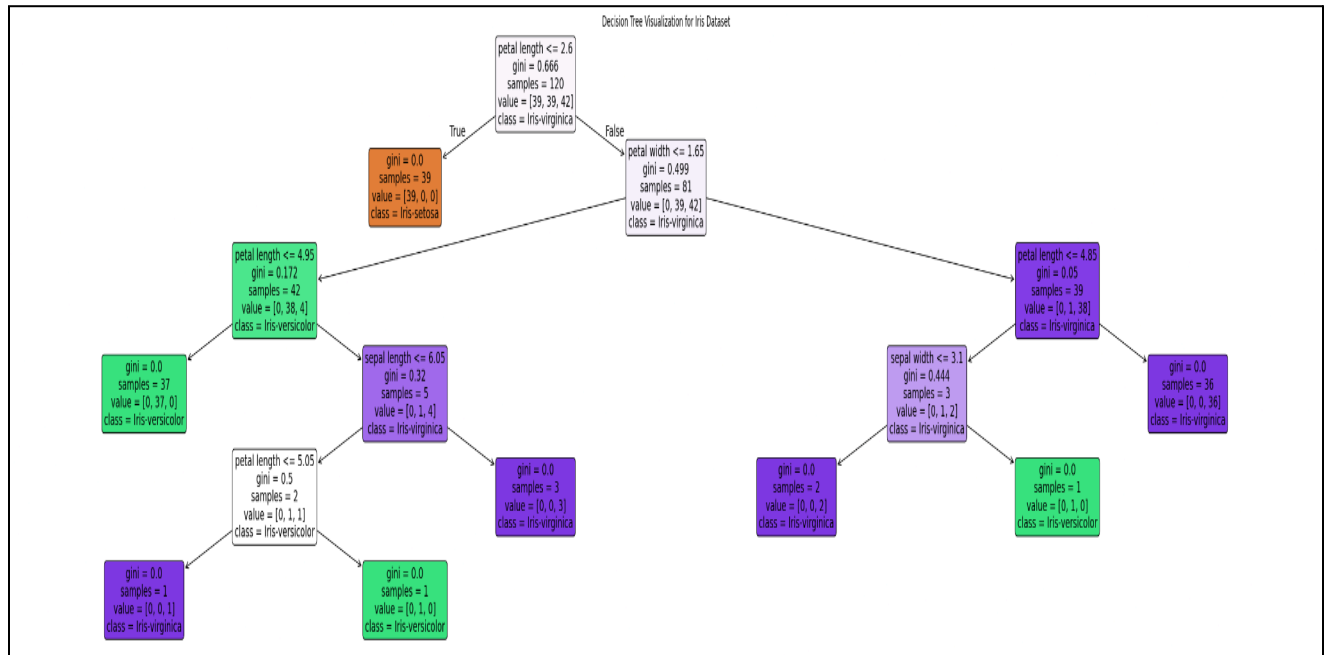
Decision Tree Classifier

The Decision Tree model demonstrated very strong performance on the IRIS dataset.

- **Performance:** The model consistently achieved a high **accuracy of 97%**.
- **Error Analysis:** Across multiple runs, the model made only a single error: misclassifying one instance of *Iris-virginica* as *Iris-versicolor*.
- **Hyperparameter Tuning:** Several configurations were tested, including varying the *criterion* ('gini', 'entropy') and *max_depth*. However, all tests yielded the exact same result. This was due to a methodological issue in the notebook where

predictions were generated only once from the initial model and not recalculated for the retrained models.

- **Conclusion:** The default Decision Tree is highly effective for this dataset. The simplicity of the IRIS data means that even a basic model can achieve near-perfect classification, and extensive tuning provides no additional benefit on this specific data split.



Decision Tree Classification Results on Iris Dataset

Criterion	Max Depth	Accuracy	Iris-setosa (P/R/F1)	Iris-versicolor (P/R/F1)	Iris-virginica (P/R/F1)	Misclassifications
default	default	0.97	1.00 / 1.00 / 1.00	0.92 / 1.00 / 0.96	1.00 / 0.93 / 0.97	1 Virginica → Versicolor
entropy	3	0.97	1.00 / 1.00 / 1.00	0.92 / 1.00 / 0.96	1.00 / 0.93 / 0.97	1 Virginica → Versicolor
entropy	10	0.97	1.00 / 1.00 / 1.00	0.92 / 1.00 / 0.96	1.00 / 0.93 / 0.97	1 Virginica → Versicolor
gini	10	0.97	1.00 / 1.00 / 1.00	0.92 / 1.00 / 0.96	1.00 / 0.93 / 0.97	1 Virginica → Versicolor
gini	15	0.97	1.00 / 1.00 / 1.00	0.92 / 1.00 / 0.96	1.00 / 0.93 / 0.97	1 Virginica → Versicolor

Naive Bayes Classifiers

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

- **Gaussian & Multinomial Naive Bayes:**

These models performed exceptionally well, achieving an **accuracy of 97%**. Their performance was identical to the Decision Tree, making the same single error of misclassifying one *Iris-virginica*. This indicates they are both well-suited for the continuous features of the IRIS dataset.

- **Bernoulli Naive Bayes:**

This model **performed very poorly**, with an accuracy of only **30%**. The confusion matrix showed that it predicted every single instance as *Iris-versicolor*, completely failing to identify the other two classes.

This failure is because the Bernoulli model is designed for binary features, making it fundamentally unsuitable for the continuous-valued features in the IRIS dataset without significant preprocessing.

Naive Bayes Classification Results on Iris Dataset						
Model Type	Hyperparameters	Accuracy	Iris-setosa (P/R/F1)	Iris-versicolor (P/R/F1)	Iris-virginica (P/R/F1)	Misclassifications
MultinomialNB	default	0.97	1.00 / 1.00 / 1.00	0.90 / 1.00 / 0.95	1.00 / 0.89 / 0.94	1 Virginica → Versicolor
GaussianNB	default	0.97	1.00 / 1.00 / 1.00	0.90 / 1.00 / 0.95	1.00 / 0.89 / 0.94	1 Virginica → Versicolor
BernoulliNB	default	0.30	0.00 / 0.00 / 0.00	0.30 / 1.00 / 0.46	0.00 / 0.00 / 0.00	All Setosa + Virginica misclassified
MultinomialNB	α=2.5, fit_prior=False	0.97	1.00 / 1.00 / 1.00	0.90 / 1.00 / 0.95	1.00 / 0.89 / 0.94	1 Virginica → Versicolor
GaussianNB	var_smoothing=1e-05	0.97	1.00 / 1.00 / 1.00	0.90 / 1.00 / 0.95	1.00 / 0.89 / 0.94	1 Virginica → Versicolor
BernoulliNB	α=1.0, binarize=0.0, fit_prior=True	0.30	0.00 / 0.00 / 0.00	0.30 / 1.00 / 0.46	0.00 / 0.00 / 0.00	All Setosa + Virginica misclassified

IRIS Dataset Summary

The IRIS dataset is readily classifiable with high accuracy by appropriate models. The Decision Tree, Gaussian NB, and Multinomial NB all proved to be excellent choices. This experiment highlights the critical importance of selecting a model variant that matches the underlying data characteristics, as demonstrated by the complete failure of the Bernoulli NB model.

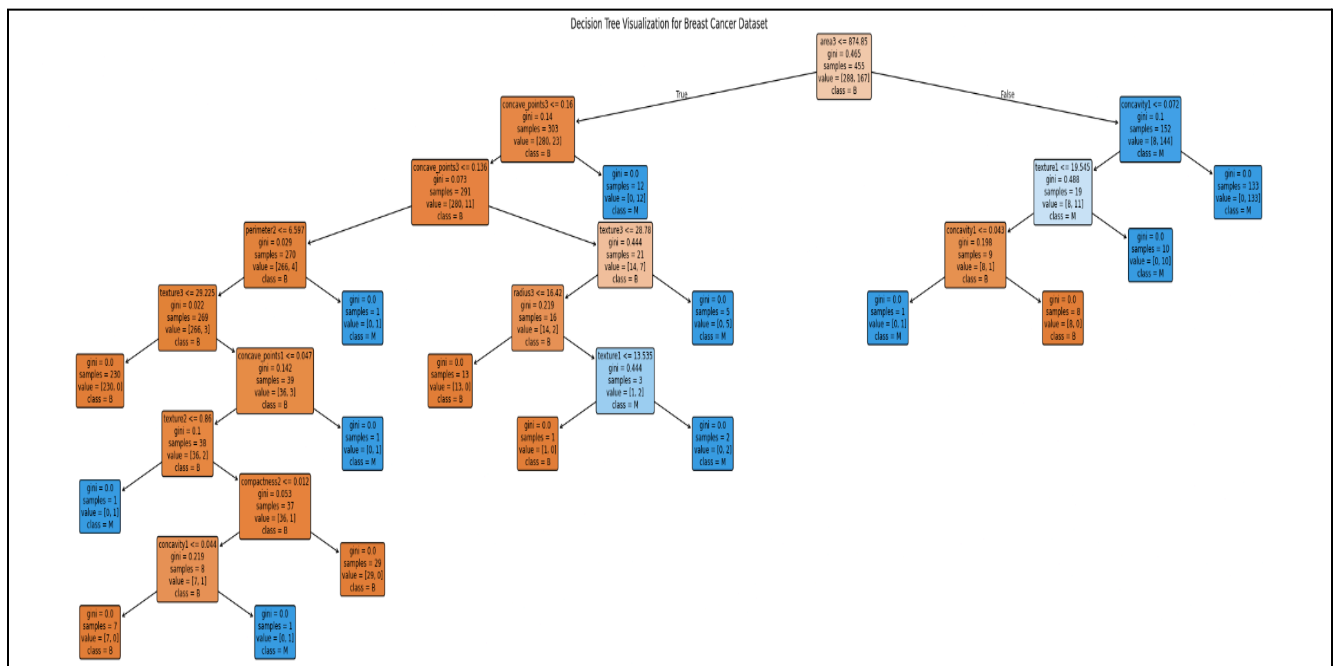
Breast Cancer Dataset Analysis:

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).

Decision Tree Classifier

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

- **Performance:** The model consistently achieved high accuracy, ranging from **93% to 95%** across different runs.
- **Variability in Results:** The slight variations in performance were primarily due to different initializations of the *train_test_split*, which resulted in slightly different testing sets for each experiment.
- **Error Analysis:** In a typical high-performing run (95% accuracy), the model made a small and balanced number of errors (e.g., 3 false positives and 3 false negatives). This shows it is a reliable and well-balanced classifier for this problem.
- **Conclusion:** The Decision Tree is an excellent choice for the Breast Cancer dataset, providing high accuracy with default settings. Its performance is stable and reliable.



Decision Tree Classifier Results on Breast Cancer Dataset

Criterion	Max Depth	Accuracy	Confusion Matrix	Benign (B) – Precision / Recall / F1	Malignant (M) – Precision / Recall / F1	Remarks
Default (Gini)	None	0.95	[[66, 3], [3, 42]]	0.96 / 0.96 / 0.96	0.93 / 0.93 / 0.93	Strong baseline
Entropy	3	0.95	[[66, 3], [3, 42]]	0.96 / 0.96 / 0.96	0.93 / 0.93 / 0.93	Same as default
Entropy	10	0.95	[[66, 3], [3, 42]]	0.96 / 0.96 / 0.96	0.93 / 0.93 / 0.93	No improvement
Gini	10	0.93	[[68, 4], [4, 38]]	0.94 / 0.94 / 0.94	0.90 / 0.90 / 0.90	Slight drop in performance
Gini	15	0.95	[[66, 3], [3, 42]]	0.96 / 0.96 / 0.96	0.93 / 0.93 / 0.93	Matches default

Naive Bayes Classifiers

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

- Gaussian & Multinomial Naive Bayes:**
 - These models achieved a respectable **accuracy of 89%**. However, a closer look at the confusion matrix revealed a concerning pattern of errors.
 - Critical Insight:** The models produced **11 false negatives**, meaning 11 malignant tumors were incorrectly classified as benign. While the overall accuracy is high, this specific error is dangerous in a medical context, as it represents missed diagnoses. This makes these models less reliable for this critical application compared to the Decision Tree.
- Bernoulli Naive Bayes:**
 - As with the IRIS dataset, the Bernoulli model **failed completely**, achieving only **61% accuracy**.
 - The model was heavily biased, predicting **every single tumor as 'Benign'** and failing to identify a single malignant case. This again confirms its unsuitability for datasets with continuous features.

Naive Bayes Classification Results on Breast Cancer Dataset

Classifier	Hyperparameters	Accuracy	Confusion Matrix	Benign (B) – Precision / Recall / F1	Malignant (M) – Precision / Recall / F1	Key Notes
MultinomialNB	Default	0.89	[[68, 2], [11, 33]]	0.86 / 0.97 / 0.91	0.94 / 0.75 / 0.84	Balanced but misses some malignant cases
GaussianNB	Default	0.89	[[68, 2], [11, 33]]	0.86 / 0.97 / 0.91	0.94 / 0.75 / 0.84	Same as Multinomial, stable performance
BernoulliNB	Default	0.61	[[70, 0], [44, 0]]	0.61 / 1.00 / 0.76	0.00 / 0.00 / 0.00	Completely failed to detect Malignant
MultinomialNB (Tuned)	$\alpha = 2.5$, fit_prior=True	0.89	[[68, 2], [11, 33]]	0.86 / 0.97 / 0.91	0.94 / 0.75 / 0.84	Same as default, no gain from tuning
GaussianNB (Tuned)	var_smoothing = 1e-05	0.89	[[68, 2], [11, 33]]	0.86 / 0.97 / 0.91	0.94 / 0.75 / 0.84	Same as default Gaussian
BernoulliNB (Tuned)	$\alpha = 1.0$, binarize=0.0, fit_prior=True	0.61	[[70, 0], [44, 0]]	0.61 / 1.00 / 0.76	0.00 / 0.00 / 0.00	Same failure as default

Breast Cancer Dataset Summary

For the Breast Cancer dataset, the Decision Tree is the superior model, offering higher accuracy and more balanced error rates. While Gaussian and Multinomial NB models have decent accuracy, their tendency to produce a high number of false negatives makes them a riskier choice for medical diagnostics.