Machine Learning using Python : Implementation of Decision Tree and Naïve Bayes Classifier

Neelotpal Chakraborty

Department of Computer Science and Engineering

Jadavpur University

Classification: Decision Tree

Classify_DeciTree.py - H:\JU ML LAB\Python Codes\Classify_DeciTree.py (3.9.6)

File Edit Format Run Options Window Help

```python
#Decision Tree for Classification

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Dataset Preparation
dataset = pd.read_csv("H:/JU ML LAB/ML using Python/Datasets/Mall_Customers.csv")
X = dataset.drop(['CustomerID','Gender'], axis=1)
y = dataset['Gender']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)

# Classification

from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier()
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

# Evaluation of Classifier Performance

from sklearn.metrics import classification_report, confusion_matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----------------------------------------------------")
print("-----------------------------------------------------")

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

OUTPUT:

IDLE Shell 3.9.6

File Edit Shell Debug Options Window Help

```
Python 3.9.6 (tags/v3.9.6:db3ff76, Jun 28 2021, 15:26:21)
Type "help", "copyright", "credits" or "license()" for mor
>>>
============ RESTART: H:\JU ML LAB\Python Codes\Classify_D
Confusion Matrix:
[[12  9]
 [10  9]]
-----------------------------------------------------------------
-----------------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

      Female       0.55      0.57      0.56        21
        Male       0.50      0.47      0.49        19

    accuracy                           0.53        40
   macro avg       0.52      0.52      0.52        40
weighted avg       0.52      0.53      0.52        40
```

Classification: Decision Tree (Parameter tuning)

criterion : *{"gini", "entropy"}, default="gini"*

The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.

splitter : *{"best", "random"}, default="best"*

The strategy used to choose the split at each node. Supported strategies are "best" to choose the best split and "random" to choose the best random split.

max_depth : *int, default=None*

The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

min_samples_split : *int or float, default=2*

The minimum number of samples required to split an internal node:

- If int, then consider `min_samples_split` as the minimum number.
- If float, then `min_samples_split` is a fraction and `ceil(min_samples_split * n_samples)` are the minimum number of samples for each split.

Classification: Decision Tree (Parameter tuning)

min_weight_fraction_leaf : *float, default=0.0*

The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Samples have equal weight when sample_weight is not provided.

max_features : *int, float or {"auto", "sqrt", "log2"}, default=None*

The number of features to consider when looking for the best split:

- If int, then consider `max_features` features at each split.
- If float, then `max_features` is a fraction and `int(max_features * n_features)` features are considered at each split.
- If "auto", then `max_features=sqrt(n_features)`.
- If "sqrt", then `max_features=sqrt(n_features)`.
- If "log2", then `max_features=log2(n_features)`.
- If None, then `max_features=n_features`.

Note: the search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than `max_features` features.

random_state : *int, RandomState instance or None, default=None*

Controls the randomness of the estimator. The features are always randomly permuted at each split, even if `splitter` is set to `"best"`. When `max_features < n_features`, the algorithm will select `max_features` at random at each split before finding the best split among them. But the best found split may vary across different runs, even if `max_features=n_features`. That is the case, if the improvement of the criterion is identical for several splits and one split has to be selected at random. To obtain a deterministic behaviour during fitting, `random_state` has to be fixed to an integer. See Glossary for details.

Classification: Decision Tree (Parameter tuning)

max_leaf_nodes : *int, default=None*

Grow a tree with `max_leaf_nodes` in best-first fashion. Best nodes are defined as relative reduction in impurity. If None then unlimited number of leaf nodes.

min_impurity_decrease : *float, default=0.0*

A node will be split if this split induces a decrease of the impurity greater than or equal to this value.

The weighted impurity decrease equation is the following:

```
N_t / N * (impurity - N_t_R / N_t * right_impurity
                    - N_t_L / N_t * left_impurity)
```

where `N` is the total number of samples, `N_t` is the number of samples at the current node, `N_t_L` is the number of samples in the left child, and `N_t_R` is the number of samples in the right child.

`N`, `N_t`, `N_t_R` and `N_t_L` all refer to the weighted sum, if `sample_weight` is passed.

New in version 0.19.

min_impurity_split : *float, default=0*

Threshold for early stopping in tree growth. A node will split if its impurity is above the threshold, otherwise it is a leaf.

Classification: Decision Tree (Parameter tuning)

class_weight : *dict, list of dict or "balanced", default=None*

Weights associated with classes in the form `{class_label: weight}`. If None, all classes are supposed to have weight one. For multi-output problems, a list of dicts can be provided in the same order as the columns of y.

Note that for multioutput (including multilabel) weights should be defined for each class of every column in its own dict. For example, for four-class multilabel classification weights should be [{0: 1, 1: 1}, {0: 1, 1: 5}, {0: 1, 1: 1}, {0: 1, 1: 1}] instead of [{1:1}, {2:5}, {3:1}, {4:1}].

The "balanced" mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as `n_samples / (n_classes * np.bincount(y))`

For multi-output, the weights of each column of y will be multiplied.

Note that these weights will be multiplied with sample_weight (passed through the fit method) if sample_weight is specified.

ccp_alpha : *non-negative float, default=0.0*

Complexity parameter used for Minimal Cost-Complexity Pruning. The subtree with the largest cost complexity that is smaller than `ccp_alpha` will be chosen. By default, no pruning is performed. See Minimal Cost-Complexity Pruning for details.

Classification: Decision Tree (Parameter tuning)

class_weight : *dict, list of dict or "balanced", default=None*

Weights associated with classes in the form `{class_label: weight}`. If None, all classes are supposed to have weight one. For multi-output problems, a list of dicts can be provided in the same order as the columns of y.

Note that for multioutput (including multilabel) weights should be defined for each class of every column in its own dict. For example, for four-class multilabel classification weights should be [{0: 1, 1: 1}, {0: 1, 1: 5}, {0: 1, 1: 1}, {0: 1, 1: 1}] instead of [{1:1}, {2:5}, {3:1}, {4:1}].

The "balanced" mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as `n_samples / (n_classes * np.bincount(y))`

For multi-output, the weights of each column of y will be multiplied.

Note that these weights will be multiplied with sample_weight (passed through the fit method) if sample_weight is specified.

ccp_alpha : *non-negative float, default=0.0*

Complexity parameter used for Minimal Cost-Complexity Pruning. The subtree with the largest cost complexity that is smaller than `ccp_alpha` will be chosen. By default, no pruning is performed. See Minimal Cost-Complexity Pruning for details.

Classification: Decision Tree (Parameter tuning)

```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion="entropy", max_depth=3)
classifier.fit(X_train, y_train)
```

OUTPUT:

```
Confusion Matrix:
[[20  1]
 [19  0]]
------------------------------------------------------
------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

      Female       0.51      0.95      0.67        21
        Male       0.00      0.00      0.00        19

    accuracy                           0.50        40
   macro avg       0.26      0.48      0.33        40
weighted avg       0.27      0.50      0.35        40
```

Classification: Decision Tree (Parameter tuning)

```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion="entropy", max_depth=10)
classifier.fit(X_train, y_train)
```

OUTPUT:

```
Confusion Matrix:
[[19 10]
 [ 5  6]]
--------------------------------------------------------
--------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

      Female       0.79      0.66      0.72        29
        Male       0.38      0.55      0.44        11

    accuracy                           0.62        40
   macro avg       0.58      0.60      0.58        40
weighted avg       0.68      0.62      0.64        40
```

Classification: Decision Tree (Parameter tuning)

```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion="gini", max_depth=10)
classifier.fit(X_train, y_train)
```

OUTPUT:

```
Confusion Matrix:
[[19  3]
 [10  8]]
-----------------------------------------------------
-----------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

      Female       0.66      0.86      0.75        22
        Male       0.73      0.44      0.55        18

    accuracy                           0.68        40
   macro avg       0.69      0.65      0.65        40
weighted avg       0.69      0.68      0.66        40
```

Classification: Decision Tree (Parameter tuning)

```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion="gini", max_depth=15)
classifier.fit(X_train, y_train)
```

OUTPUT:

```
Confusion Matrix:
[[15  4]
 [14  7]]
-----------------------------------------------------
-----------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

      Female       0.52      0.79      0.62        19
        Male       0.64      0.33      0.44        21

    accuracy                           0.55        40
   macro avg       0.58      0.56      0.53        40
weighted avg       0.58      0.55      0.53        40
```

Classification: Decision Tree (Visualization)

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
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 plot_tree

# Dataset Preparation
dataset = pd.read_csv("E:/JU ML LAB/ML using Python/Datasets/Mall_Customers.csv")
X = dataset.drop(['CustomerID','Gender'], axis=1)
y = dataset['Gender']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)

# Classification

classifier = DecisionTreeClassifier()
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

# Evaluation of Classifier Performance

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

print("-----------------------------------------------------")
print("-----------------------------------------------------")

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

```python
# TREE REPRESENTATION

text_representation = tree.export_text(classifier)
print(text_representation)
```

Outcome:

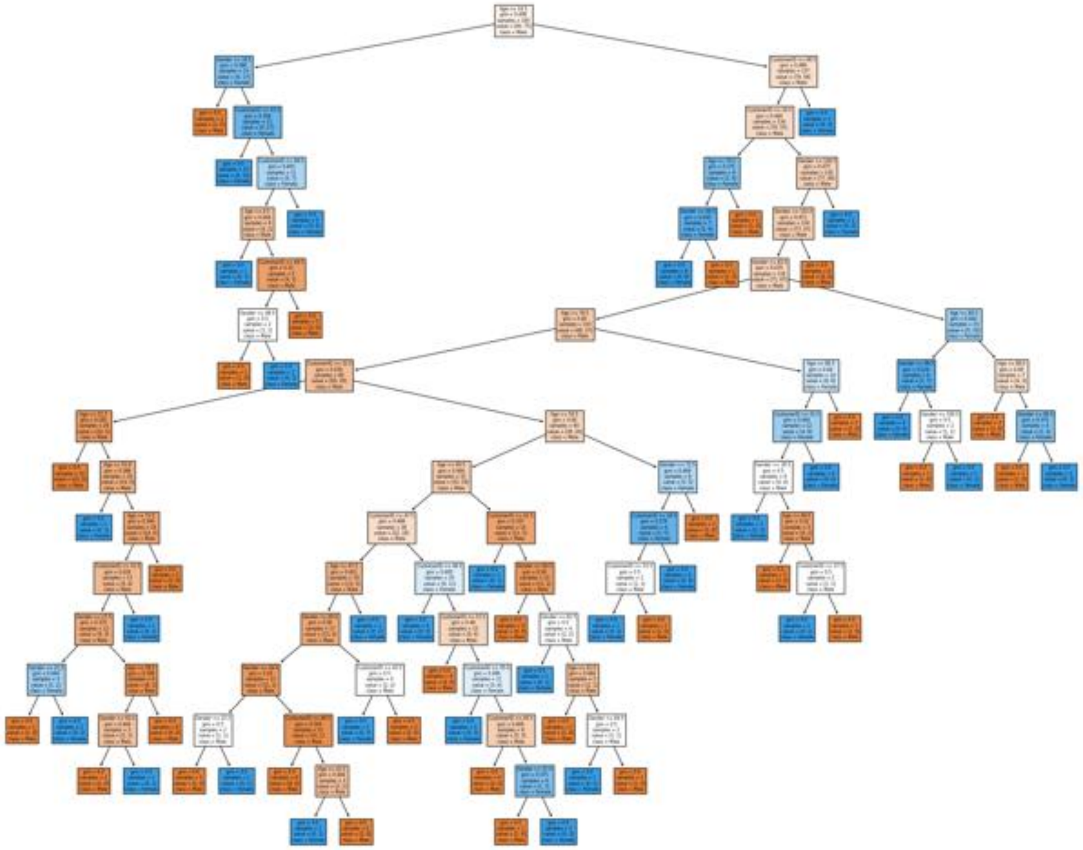
```
|--- feature_2 <= 14.50
|   |--- feature_1 <= 18.50
|   |   |--- class: Female
|   |--- feature_1 >  18.50
|   |   |--- feature_0 <= 43.00
|   |   |   |--- class: Male
|   |   |--- feature_0 >  43.00
|   |   |   |--- feature_0 <= 58.00
|   |   |   |   |--- feature_2 <= 4.50
|   |   |   |   |   |--- class: Male
|   |   |   |   |--- feature_2 >  4.50
|   |   |   |   |   |--- feature_0 <= 49.50
|   |   |   |   |   |   |--- feature_1 <= 48.00
|   |   |   |   |   |   |   |--- class: Female
|   |   |   |   |   |   |--- feature_1 >  48.00
|   |   |   |   |   |   |   |--- class: Male
|   |   |   |   |   |--- feature_0 >  49.50
|   |   |   |   |   |   |--- class: Female
|   |   |   |--- feature_0 >  58.00
|   |   |   |   |--- class: Male
|--- feature_2 >  14.50
|   |--- feature_0 <= 68.50
|   |   |--- feature_0 <= 20.50
|   |   |   |--- feature_2 <= 70.50
|   |   |   |   |--- feature_1 <= 64.50
|   |   |   |   |   |--- class: Male
```

Classification: Decision Tree (Visualization)

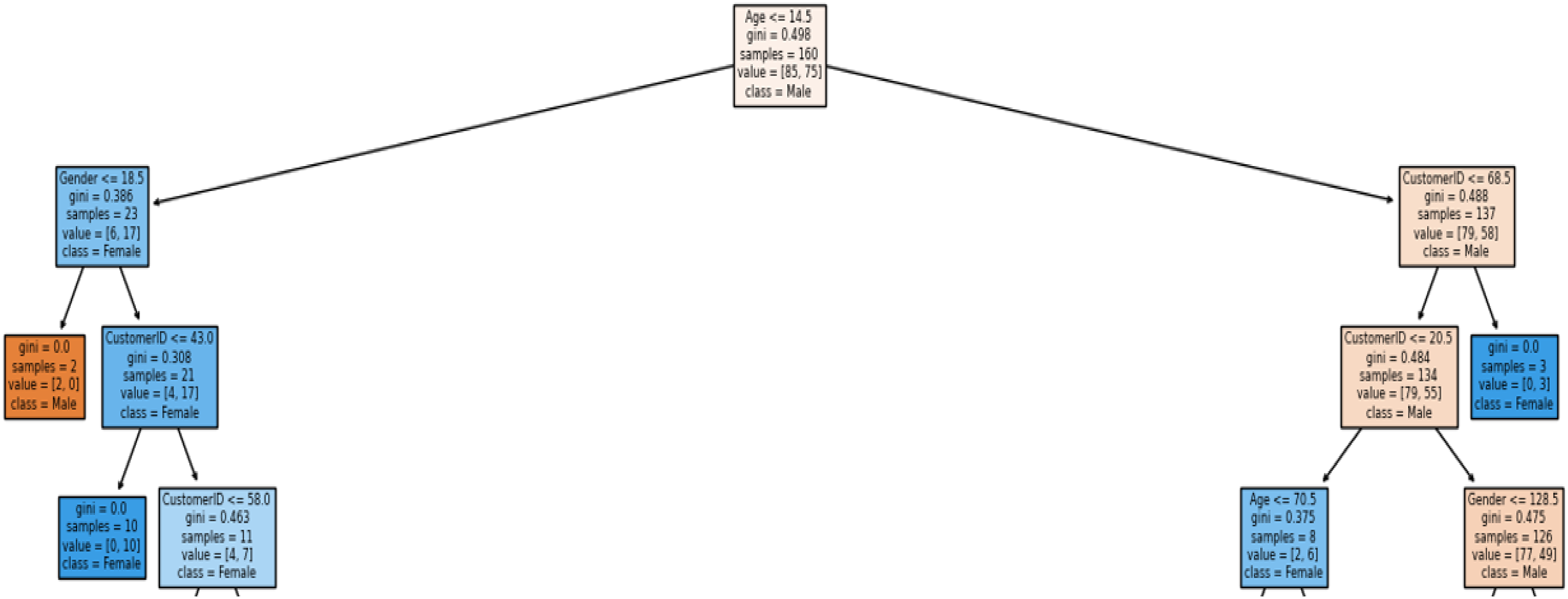
```
# Visualising the graph without the use of graphviz

fig = plt.figure(figsize=(25,20))
_ = tree.plot_tree(decision_tree=classifier, feature_names = dataset.columns,
                    class_names =["Male", "Female"] , filled = True)
fig.savefig("E:/JU ML LAB/Python Codes/decision_tree1.png")
```

Outcome:



Classification: Decision Tree (Visualization)



Classification: Naive Bayes

```python
#Naive Bayes for Classification

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Dataset Preparation
dataset = pd.read_csv("H:/JU ML LAB/ML using Python/Datasets/Mall_Customers.csv")
X = dataset.drop(['CustomerID','Gender'], axis=1)
y = dataset['Gender']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)

# Classification

from sklearn.naive_bayes import MultinomialNB
classifier = MultinomialNB().fit(X_train, y_train)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

# Evaluation of Classifier Performance

from sklearn.metrics import classification_report, confusion_matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("----------------------------------------------------")
print("----------------------------------------------------")

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

OUTPUT:

```
IDLE Shell 3.9.6
File  Edit  Shell  Debug  Options  Window  Help
Python 3.9.6 (tags/v3.9.6:db3ff76, Jun 28 2021, 15:26:21) [MSC v.1929 64
D64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=========== RESTART: H:\JU ML LAB\Python Codes\Classify_NaiveBayes.py ==
Confusion Matrix:
[[14  8]
 [11  7]]
----------------------------------------------------
----------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

      Female       0.56      0.64      0.60        22
        Male       0.47      0.39      0.42        18

    accuracy                           0.53        40
   macro avg       0.51      0.51      0.51        40
weighted avg       0.52      0.53      0.52        40

>>>
```

Classification: Naive Bayes

```
# Classification

from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB().fit(X_train, y_train)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)
```

OUTPUT:

```
=========== RESTART: H:\JU ML LAB\Python Codes\Classify_NaiveBayes.py
Confusion Matrix:
[[19  1]
 [16  4]]
--------------------------------------------------------------
--------------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

      Female       0.54      0.95      0.69        20
        Male       0.80      0.20      0.32        20

    accuracy                           0.57        40
   macro avg       0.67      0.57      0.51        40
weighted avg       0.67      0.57      0.51        40
```

Classification: Naive Bayes

```
# Classification

from sklearn.naive_bayes import BernoulliNB
classifier = BernoulliNB().fit(X_train, y_train)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)
```

OUTPUT:

```
Confusion Matrix:
[[19  0]
 [21  0]]
-----------------------------------------------------
-----------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

      Female       0.47      1.00      0.64        19
        Male       0.00      0.00      0.00        21

    accuracy                           0.48        40
   macro avg       0.24      0.50      0.32        40
weighted avg       0.23      0.47      0.31        40
```

Classification: Naive Bayes (Parameters)

Naive Bayes classifier for multinomial models:

The multinomial Naive Bayes classifier is suitable for classification with discrete features (e.g., word counts for text classification). The multinomial distribution normally requires integer feature counts. However, in practice, fractional counts such as tf-idf may also work.

Parameters:	
	alpha : *float, default=1.0* Additive (Laplace/Lidstone) smoothing parameter (0 for no smoothing). **fit_prior : *bool, default=True*** Whether to learn class prior probabilities or not. If false, a uniform prior will be used. **class_prior : *array-like of shape (n_classes,), default=None*** Prior probabilities of the classes. If specified the priors are not adjusted according to the data.

Classification: Naive Bayes (Parameters)

Naive Bayes classifier for multinomial models:

```
# Classification

from sklearn.naive_bayes import MultinomialNB
classifier = MultinomialNB(alpha=2.5, fit_prior=True, class_prior=None).fit(X_train, y_train)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)
```

OUTPUT:

```
Confusion Matrix:
[[17  5]
 [11  7]]
----------------------------------------------------
----------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

      Female       0.61      0.77      0.68        22
        Male       0.58      0.39      0.47        18

    accuracy                           0.60        40
   macro avg       0.60      0.58      0.57        40
weighted avg       0.60      0.60      0.58        40
```

Classification: Naive Bayes (Parameters)

Naive Bayes classifier for Gaussian.

Parameters:	**priors : *array-like of shape (n_classes,)***
	Prior probabilities of the classes. If specified the priors are not adjusted according to the data.
	var_smoothing : *float, default=1e-9*
	Portion of the largest variance of all features that is added to variances for calculation stability.

Classification: Naive Bayes (Parameters)

Naive Bayes classifier for Gaussian

```
# Classification

from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB(priors=None, var_smoothing=1e-05).fit(X_train, y_train)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)
```

OUTPUT:

```
Confusion Matrix:
[[16  4]
 [12  8]]
-----------------------------------------------------
-----------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

      Female       0.57      0.80      0.67        20
        Male       0.67      0.40      0.50        20

    accuracy                           0.60        40
   macro avg       0.62      0.60      0.58        40
weighted avg       0.62      0.60      0.58        40
```

Classification: Naive Bayes (Parameters)

Naive Bayes classifier for multivariate Bernoulli models.

Like MultinomialNB, this classifier is suitable for discrete data. The difference is that while MultinomialNB works with occurrence counts, BernoulliNB is designed for binary/boolean features.

Parameters:	**alpha : *float, default=1.0*** Additive (Laplace/Lidstone) smoothing parameter (0 for no smoothing). **binarize : *float or None, default=0.0*** Threshold for binarizing (mapping to booleans) of sample features. If None, input is presumed to already consist of binary vectors. **fit_prior : *bool, default=True*** Whether to learn class prior probabilities or not. If false, a uniform prior will be used. **class_prior : *array-like of shape (n_classes,), default=None*** Prior probabilities of the classes. If specified the priors are not adjusted according to the data.

Classification: Naive Bayes (Parameters)

Naive Bayes classifier for multivariate Bernoulli models.

```
# Classification

from sklearn.naive_bayes import BernoulliNB
classifier = BernoulliNB(alpha=1.0, binarize=0.0, fit_prior=True, class_prior=None).fit(X_train, y_train)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)
```

OUTPUT:

```
Confusion Matrix:
[[24  0]
 [16  0]]
----------------------------------------------------------
----------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

      Female       0.60      1.00      0.75        24
        Male       0.00      0.00      0.00        16

    accuracy                           0.60        40
   macro avg       0.30      0.50      0.37        40
weighted avg       0.36      0.60      0.45        40
```

Loading datasets from scikit-learn

File Edit Format Run Options Window Help

```
from sklearn.datasets import load_iris
from sklearn import tree
iris = load_iris()
X, y = iris.data, iris.target
```

For more details visit the link below:

https://scikit-learn.org/