**120A3051**

**Shreya Idate**

**Batch: E3**

## Experiment 9

**Aim:** Implement Naive Bayes algorithm in Python.

**Theory:**

Naive Bayes is a classification algorithm based on Bayes' theorem. It is a probabilistic model that can be used to predict the class of a new observation based on a set of features or attributes.

Naive Bayes is called "naive" because it assumes that all the features are independent of each other, even though this may not be true in reality.

The algorithm works by first learning the prior probabilities of each class based on the frequency of each class in the training data. It then estimates the likelihood of each feature given each class, which can be calculated by counting the frequency of each feature in the training data for each class. Finally, it uses Bayes' theorem to compute the posterior probability of each class given the observed features, and selects the class with the highest probability as the predicted class for the new observation.

Naive Bayes has been successfully used in many applications, such as spam filtering, sentiment analysis, and document classification.

Advantages:

1. Naive Bayes is simple and easy to implement, making it computationally efficient and fast.
2. It requires a small amount of training data to estimate the parameters necessary for classification, making it suitable for handling large datasets.
3. Naive Bayes can handle both binary and multi-class classification problems.
4. It can handle both continuous and discrete data types and is robust to irrelevant features in the data.
5. Naive Bayes is a probabilistic model, which makes it easy to interpret and explain the classification results.

Disadvantages:

1. Naive Bayes makes the strong assumption that all features are independent of each other, which may not be true in real-world scenarios.

2. The algorithm assumes that the probability distributions of the features are Gaussian, which may not be the case for all datasets.
3. Naive Bayes tends to perform poorly when there are rare events in the data, as it requires sufficient data to accurately estimate the parameters.
4. It may not work well if the features are highly correlated or if there are interactions between them.
5. Naive Bayes is a relatively simple algorithm, and there may be more complex models that can perform better on certain datasets.

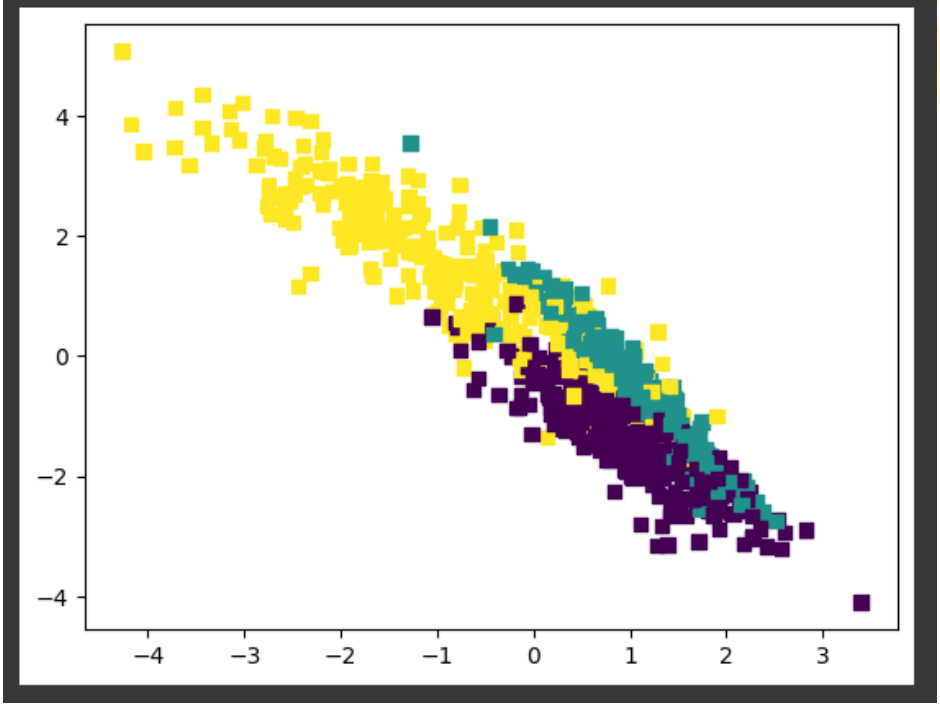
**Output:**

```
from sklearn.datasets import make_classification

X, y = make_classification(
    n_features=6,
    n_classes=3,
    n_samples=800,
    n_informative=2,
    random_state=1,
    n_clusters_per_class=1,
)
```

```
import matplotlib.pyplot as plt

plt.scatter(X[:, 0], X[:, 1], c=y, marker=",");
```



```python
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=125
)
```

```python
from sklearn.naive_bayes import GaussianNB

# Build a Gaussian Classifier
model = GaussianNB()

# Model training
model.fit(X_train, y_train)

# Predict Output
predicted = model.predict([X_test[2]])

print("Actual Value:", y_test[2])
print("Predicted Value:", predicted[0])
```

```
Actual Value: 2
Predicted Value: 2
```

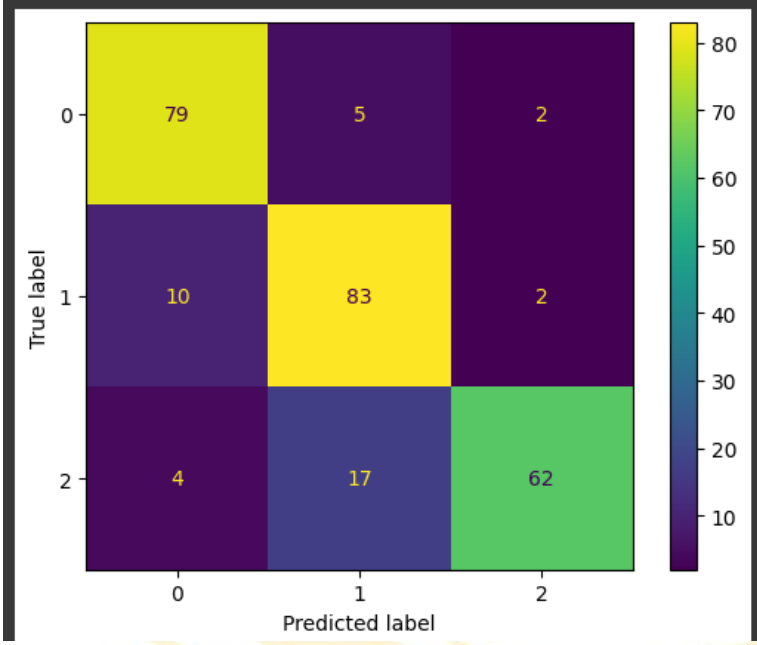
```python
from sklearn.metrics import (
    accuracy_score,
    confusion_matrix,
    ConfusionMatrixDisplay,
    f1_score,
)

y_pred = model.predict(X_test)
accuray = accuracy_score(y_pred, y_test)
f1 = f1_score(y_pred, y_test, average="weighted")

print("Accuracy:", accuray)
print("F1 Score:", f1)
```

```
Accuracy: 0.8484848484848485
F1 Score: 0.8491119695890328
```

```
labels = [0,1,2]
cm = confusion_matrix(y_test, y_pred, labels=labels)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
disp.plot();
```




```
y_pred = model.predict(X_test)
y_pred
```

```
array([0, 1, 2, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 2, 2, 1, 1, 0, 0, 0, 0, 0,
       1, 0, 0, 2, 0, 2, 0, 2, 1, 2, 0, 0, 1, 0, 2, 2, 1, 0, 2, 1, 2, 0,
       1, 0, 1, 1, 0, 1, 2, 2, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 2, 0, 2, 1,
       0, 1, 2, 1, 0, 1, 2, 1, 1, 0, 2, 1, 0, 2, 1, 2, 0, 0, 0, 0, 0, 0,
       2, 2, 1, 1, 0, 1, 1, 2, 0, 0, 1, 2, 1, 1, 0, 1, 1, 1, 2, 1, 2, 0,
       1, 2, 2, 1, 1, 2, 1, 2, 0, 2, 2, 1, 2, 2, 1, 1, 0, 1, 0, 1, 2, 1,
       1, 2, 2, 1, 1, 0, 1, 2, 0, 0, 0, 1, 1, 1, 0, 2, 2, 0, 0, 1, 1, 1,
       0, 1, 2, 1, 2, 0, 1, 1, 1, 0, 2, 0, 2, 1, 0, 0, 1, 0, 0, 0, 0, 2,
       1, 1, 1, 2, 0, 1, 0, 0, 2, 1, 1, 2, 1, 1, 0, 0, 0, 1, 0, 2, 1, 1,
       0, 1, 1, 1, 0, 0, 1, 0, 0, 2, 1, 1, 1, 1, 2, 2, 1, 2, 0, 1, 2, 1,
       1, 0, 1, 2, 1, 2, 1, 0, 2, 1, 0, 1, 0, 2, 2, 2, 1, 0, 0, 0, 0, 2,
       0, 1, 0, 1, 1, 2, 2, 0, 1, 1, 0, 1, 0, 0, 1, 2, 2, 1, 2, 0, 1, 0])
```

```
y_test
```

```
array([0, 1, 2, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 2, 2, 1, 1, 0, 0, 0, 0, 0,
       2, 0, 2, 2, 0, 2, 0, 2, 1, 0, 0, 0, 1, 0, 2, 2, 1, 1, 2, 1, 2, 0,
       1, 0, 2, 1, 0, 1, 2, 1, 2, 1, 0, 1, 1, 1, 0, 1, 0, 1, 2, 0, 2, 1,
       0, 1, 2, 1, 0, 2, 2, 1, 1, 1, 2, 1, 0, 0, 1, 2, 0, 0, 0, 0, 0, 0,
       2, 2, 1, 1, 0, 1, 1, 2, 0, 1, 1, 2, 1, 1, 0, 1, 1, 0, 2, 2, 2, 0,
       1, 2, 2, 1, 2, 2, 1, 2, 0, 2, 2, 1, 1, 2, 0, 1, 0, 1, 2, 1, 2, 1,
       2, 2, 2, 1, 2, 1, 1, 2, 0, 0, 0, 1, 1, 1, 0, 2, 2, 0, 1, 1, 1, 1,
       0, 1, 2, 1, 2, 0, 1, 2, 1, 0, 2, 0, 2, 1, 0, 0, 0, 0, 0, 0, 0, 2,
       1, 1, 0, 2, 0, 1, 0, 0, 2, 1, 1, 2, 1, 1, 0, 0, 0, 1, 0, 2, 1, 1,
       0, 1, 1, 1, 0, 0, 1, 0, 0, 2, 2, 1, 2, 2, 2, 2, 1, 2, 0, 1, 2, 1,
       1, 1, 1, 2, 1, 2, 1, 0, 2, 1, 0, 0, 0, 2, 2, 2, 2, 1, 1, 0, 0, 2,
       0, 1, 0, 1, 2, 2, 2, 0, 2, 1, 0, 1, 2, 0, 2, 2, 2, 2, 2, 0, 2, 0])
```

## Conclusion:

Successfully implemented Naive Bayes algorithm in Python.