Naive Bayes Classifier[1]

Data Science and Analytics Lab (CSL DC205)

Fall 2024

Tavneet Singh
23dcs002@smvdu.ac.in

1 Naive Bayes Classifier Explanation

1.1 Assumption of Independence

- The "naive" part of Naive Bayes refers to the assumption that features (input variables) are conditionally independent given the class label.
- In other words, the presence or absence of one feature does not affect the likelihood of other features occurring.
- This simplification allows us to calculate probabilities more efficiently.

1.2 Bayes' Theorem

- Bayes' Theorem is the foundation of Naive Bayes. It relates the posterior probability of an event (class) given evidence (features) to the prior probability of the event and the likelihood of the evidence.
- Mathematically, it looks like this:

$$P(Z|X) = \frac{P(X|Z) \cdot P(Z)}{P(X)}$$

  where ,

  - $(P(Z|X))$ is the posterior probability of class (Z) given features (X).
  - $(P(X|Z))$ is the likelihood of observing features (X) given class (Z).
  - $(P(Z))$ is the prior probability of class (Z).
  - $(P(X))$ is the evidence (probability of observing features (X)).

1.3 Classification Process

- Given a set of features (X), Naive Bayes calculates the posterior probability for each possible class.
- The class with the highest posterior probability is assigned to the observation.
- For example, in spam detection, we compare the probabilities of "spam" and "not spam" to classify an email.

[1]Last Updated: September 30, 2024

1.4 Types of Naive Bayes:

There are different variants of Naive Bayes, such as:

1.4.1 Gaussian Naive Bayes(`GaussianNB`):

- **Description:** This classifier is used when the features are continuous and assumes that they follow a Gaussian (normal) distribution. It calculates the mean and standard deviation for each feature in each class.
- **Use Cases:** Commonly used in scenarios where the data is continuous, such as predicting numerical outcomes or classifying data points based on continuous attributes.
- **Example:** Predicting whether an email is spam based on features like the length of the email and the frequency of certain words, assuming these features are normally distributed.

1.4.2 Multinomial Naive Bayes(`MultinomialNB`):

- **Description:** This variant is specifically designed for discrete data, typically used for document classification tasks. It assumes that features (like word counts) follow a multinomial distribution.
- **Use Cases:** Ideal for text classification problems such as spam detection, sentiment analysis, and topic categorization.
- **Example:** Classifying emails as "spam" or "not spam" by analyzing the frequency of words present in the emails.

1.4.3 Bernoulli Naive Bayes (`BernoulliNB`):

- **Description:** Similar to Multinomial Naive Bayes, but it uses binary features (0 or 1) to indicate whether a feature is present or absent. It assumes that each feature follows a Bernoulli distribution.
- **Use Cases:** Particularly useful for binary/boolean features in text classification tasks where only the presence or absence of words matters.
- **Example:** Classifying documents based on whether specific keywords appear (yes/no) rather than how many times they appear.

1.4.4 Complement Naive Bayes (`ComplementNB`):

- **Description:** An adaptation of Multinomial Naive Bayes that is particularly suited for imbalanced datasets. Calculate the probability of a class by considering all other classes as complements.

- **Use Cases:** Effective in scenarios where one class significantly outnumbers others, such as detecting rare events in datasets.
- **Example:** Classifying rare diseases in medical data sets where healthy cases vastly outnumber sick cases.

1.4.5 Categorical Naive Bayes (CategoricalNB):

- **Description:** Categorical Naive Bayes is a variant of the Naive Bayes classifier specifically designed for categorical data. It is particularly useful in situations where features are discrete and can take on a limited number of values, such as in text classification and other applications involving categorical variables.
- **Use Cases:** Commonly used in applications such as text classification (e.g., spam detection), sentiment analysis, and any scenario in which features are categorical (e.g., predicting customer preferences based on categorical attributes).
- **Example:** *Text classification:* In a news categorization task, Categorical Naive Bayes can classify articles into categories such as politics, sports, or technology based on the presence of specific keywords. The features of each article would be the words it contains, treated as categorical variables, allowing the model to predict the most likely category based on training data.

1.4.6 Out-Of-Core Naive Bayes

- **Description:** Designed to handle large-scale classification problems where the complete training dataset cannot fit into memory.
- **Use Cases:** Useful for big data applications where traditional methods would fail due to memory constraints.
- **Example:** Classifying user behavior on large-scale web applications.

Table 1: Summary of Types of Naive Bayes Classifiers

Type	**Data Type**	**Assumption**	**Common Use Cases**	**Example**	**Limitations**
Gaussian Naive Bayes	Continuous	Features follow Gaussian distribution	Predicting numerical outcomes	Classifying flower species based on petal length and width.	Assumes independence; sensitive to outliers; not suitable for categorical data.

Table 1: Summary of Types of Naive Bayes Classifiers

Type	**Data Type**	**Assumption**	**Common Use Cases**	**Example**	**Limitations**
Multinomial Naive Bayes	Discrete (counts)	Features follow multinomial distribution	Text classification (spam detection)	Classifying emails based on word frequency.	Requires large training data; assumes independence; can be sensitive to irrelevant features.
Bernoulli Naive Bayes	Binary	Features follow Bernoulli distribution	Document classification (presence/absence of words)	Classifying documents based on keyword presence.	Cannot handle continuous data; limited expressive power; requires feature selection.
Complement Naive Bayes	Discrete	Adjusted for imbalanced datasets	Rare event detection	Detecting rare diseases in medical datasets.	Not effective for balanced datasets; requires careful tuning.
Categorical Naive Bayes	Categorical	Features are conditionally independent given the class label	Text classification, sentiment analysis, customer preference prediction	Classifying emails as spam or not based on keyword categories.	Assumes independence; may struggle with highly correlated features; not suitable for continuous data.
Out-of-Core Naive Bayes	Large-scale	Handles large datasets	Big data applications	Classifying user behavior in large web applications.	May require significant preprocessing; can be complex to implement.

1.5 Applications:

- Naive Bayes is commonly used in text classification tasks, where data has high dimensionality (each word represents a feature).

  Examples include:

  - **Sapm Filtering:** Classifying emails as spam or not.
  - **Sentiment Analysis:** Determining sentiment (positive/negative) from text.
  - **Rating Classification:** Categorizing product reviews (e.g., 5 stars, 4 stars, etc.).

1.6 Advantages and Limitations:

1.6.1 Advantages:

- Simple and computationally efficient.
- Works well with high-dimensional data.
- Performs surprisingly well in practice, especially for text classification.

1.6.2 Limitations:

- Assumes independence (which may not hold in all cases).
- Sensitive to irrelevant features.
- Requires labeled training data.

Q1. To predict if a person will purchase a product on a specific combination of Day, Discount and Free delivery using Naïve Bayesian Classifier.

Consider Sample Data:

Table 2: Sample Purchase Data

Day	**Discount**	**Free Delivery**	**Purchase**
Monday	0	1	No
Tuesday	1	0	Yes
Wednesday	1	1	Yes
Thursday	0	0	No
Friday	1	1	Yes
Saturday	0	0	No
Sunday	1	1	Yes
Monday	0	0	No
Tuesday	1	1	Yes
Wednesday	1	0	Yes

Let's calculate the probabilities step by step:

1. **Calculate Prioir Probabilities:** These are just the probabilities of the outcome events, which in this case are making a purchase (Yes) and not making a purchase (No). Count the number of 'Yes' and 'No' in the Purchase column and divide by the total number of days.
   - $P(Yes)$ = Number of 'Yes' / Total Days = 6/10
   - $P(No)$ = Number of 'No' / Total Days = 4/10

2. **Calculate Likelihoods:**
   These are the probabilities of the features given the outcome. In this case, we need to calculate the probability of it being Monday, there being a discount, and there being free delivery given that a purchase is made and not made. In general terms we have to do this for all days of the week but here as we are only taking Monday as example so we are doing only for that.

   Firstly we calaculate likehood of Day:

   - $P(Monday|Yes)$ = Number of 'Yes' on Monday / Total 'Yes' = 0/6 = 0
   - $P(Monday|No)$ = Number of 'No' on Monday / Total 'No' = 2/4 = 0.5

   - $P(Tuesday|Yes)$ = Number of 'Yes' on Tuesday / Total 'Yes' = 1/6 = 0.1667
   - $P(Tuesday|No)$ = Number of 'No' on Tuesday / Total 'No' = 0/4 = 0

   - $P(Wednsday|Yes)$ = Number of 'Yes' on Wednesday / Total 'Yes' = 2/6 = 0.3333
   - $P(Wednsday|No)$ = Number of 'No' on Wednesday / Total 'No' = 0/4 = 0

   - $P(Thursday|Yes)$ = Number of 'Yes' on Thursday / Total 'Yes' = 0/6 = 0
   - $P(Thursday|No)$ = Number of 'No' on Thursday / Total 'No' = 1/4 = 0.25

   - $P(Friday|Yes)$ = Number of 'Yes' on Friday / Total 'Yes' = 1/6 = 0.1667
   - $P(Friday|No)$ = Number of 'No' on Friday / Total 'No' = 0/4 = 0

   - $P(Saturday|Yes)$ = Number of 'Yes' on Saturday / Total 'Yes' = 0/6 = 0
   - $P(Saturday|No)$ = Number of 'No' on Saturday / Total 'No' = 1/4 = 0.25

   - $P(Sunday|Yes)$ = Number of 'Yes' on Sunday / Total 'Yes' = 1/6 = 0.1667
   - $P(Sunday|No)$ = Number of 'No' on Sunday / Total 'No' = 0/4 = 0

   Now we do the same for Discount:

   - $P(NoDiscount|Yes)$ = Number of 'Yes' without Discount / Total 'Yes' = 0/6 = 0
   - $P(NoDiscount|No)$ = Number of 'No' without Discount / Total 'No' = 4/4 = 1

   - $P(WithDiscount|Yes)$ = Number of 'Yes' with Discount / Total 'Yes' = 6/6 = 1
   - $P(WithDiscount|No)$ = Number of 'No' with Discount / Total 'No' = 0/4 = 0

   Now we do the same for Delivery:

   - $P(FreeDelivery|Yes)$ = Number of 'Yes' with Free Delivery / Total 'Yes' = 4/6 = 0.6667
   - $P(FreeDelivery|No)$ = Number of 'No' with Free Delivery / Total 'No' = 1/4 = 0.25

- $P(NoFreeDelivery|Yes)$ = Number of 'Yes' with No Free Delivery / Total 'Yes' = 2/6 = 0.3333
- $P(NoFreeDelivery|No)$ = Number of 'No' with No Free Delivery / Total 'No' = 3/4 = 0.75

3. **Apply Bayes' Theorem:**

$$P(Yes|Monday, NoDiscount, FreeDelivery)$$
$$= P(Monday|Yes)xP(NoDiscount|Yes)xP(FreeDelivery|Yes)xP(Yes)$$

$$P(No|Monday, NoDiscount, FreeDelivery)$$
$$= P(Monday|No)xP(NoDiscount|No)xP(FreeDelivery|No)xP(No)$$

4. **Make Prediction:**
If $P(Yes|Monday, NoDiscount, FreeDelivery) > P(No|Monday, NoDiscount, FreeDelivery)$, we predict 'Yes'; otherwise, we predict 'No'.

Let's assume the following probabilities based on the data:

- $P(Yes) = 6/10 = 0.6$
- $P(No) = 4/10 = 0.4$
- $P(Monday|Yes) = 0$
- $P(NoDiscount|Yes) = 0$
- $P(FreeDelivery|Yes) = 0.6667$
- $P(Monday|No) = 0.5$
- $P(NoDiscount|No) = 1$
- $P(FreeDelivery|No) = 0.25$

Now, calculate the posterior probabilities:

$$P(Yes|Monday, NoDiscount, FreeDelivery)$$
$$= P(Monday|Yes)xP(NoDiscount|Yes)xP(FreeDelivery|Yes)xP(Yes)$$
$$= 0x0x0.6667x0.6 = 0$$

$$P(No|Monday, NoDiscount, FreeDelivery)$$
$$= P(Monday|No)xP(NoDiscount|No)xP(FreeDelivery|No)xP(No)$$
$$= 0.5x1x0.25x0.4 = 0.05$$

Since $P(Yes|Monday, NoDiscount, FreeDelivery) < P(No|Monday, NoDiscount, FreeDelivery)$, we predict **'No'**, i.e., a purchase will not be made.

Q1. Why we neglect denominator for doing final calculations?
In the Naive Bayes formula:

$$P(c|x_1, x_2, ..., x_n) = \frac{P(x_1|c) * P(x_2|c) * ... * P(x_n|c) * P(c)}{P(x_1) * P(x_2) * ... * P(x_n)} \tag{1}$$

The denominator,

$$P(x_1) * P(x_2) * ... * P(x_n) \tag{2}$$

, is the total probability of the features.

When we're using Naive Bayes for classification, we're interested in comparing the *relative probabilities* of the different classes given the features. We want to find the class c that maximizes.

$$P(c|x_1, x_2, ..., x_n) \tag{3}$$

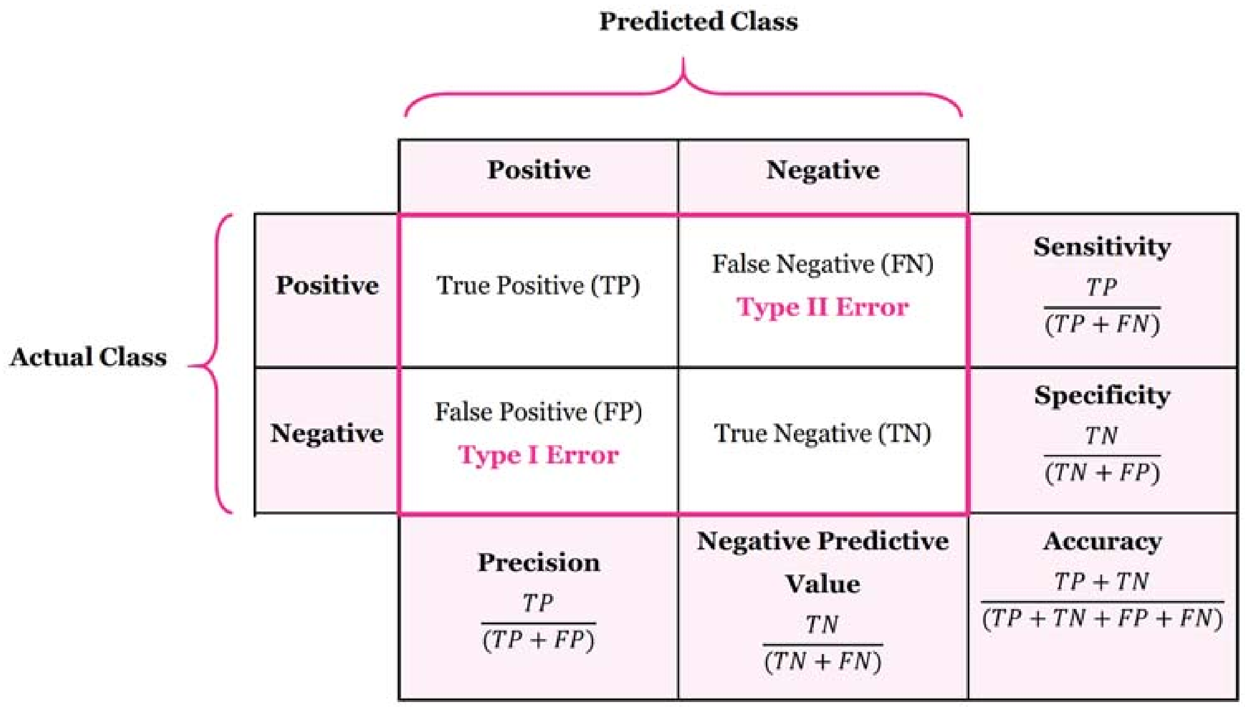
The denominator doesn't depend on c, it's the same for all classes. So, when we're comparing these probabilities to find the class that maximizes.

$$P(c|x_1, x_2, ..., x_n) \tag{4}$$

, we can ignore the denominator because it's a constant factor that doesn't affect which class has the maximum probability.

In other words, we're not interested in the exact probability, but in which class has the highest probability. That's why we can neglect the denominator when using Naive Bayes for classification.

1.7 Confusion Matrix:



		Predicted Class		
		Positive	**Negative**	
Actual Class	**Positive**	True Positive (TP)	False Negative (FN) **Type II Error**	**Sensitivity** $\frac{TP}{(TP+FN)}$
	Negative	False Positive (FP) **Type I Error**	True Negative (TN)	**Specificity** $\frac{TN}{(TN+FP)}$
		Precision $\frac{TP}{(TP+FP)}$	**Negative Predictive Value** $\frac{TN}{(TN+FN)}$	**Accuracy** $\frac{TP+TN}{(TP+TN+FP+FN)}$

Figure 1: Confusion Matrix (Source)

1.7.1 Key Terms:

- **True Positive (TP):** The model correctly predicts the positive class.
- **True Negative (TN):** The model correctly predicts the negative class.
- **False Positive (FP):** The model incorrectly predicts the positive class (Type I error).
- **False Negative (FN):** The model incorrectly predicts the negative class (Type II error).

To calculate True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN) from a confusion matrix, follow these definitions:

- **True Positive (TP):** The number of instances correctly predicted as the positive class. For example, if a model correctly predicts that a patient has a disease when they actually do, this counts as a TP.
- **True Negative (TN):** The number of instances correctly predicted as the negative class. For example, if a model correctly predicts that a patient does not have a disease when they actually do not, this counts as a TN.
- **False Positive (FP):** The number of instances incorrectly predicted as the positive class. This occurs when the model predicts a positive outcome for an instance that is actually negative. For example, predicting that a healthy patient has a disease.
- **False Negative (FN):** The number of instances incorrectly predicted as the negative class. This occurs when the model predicts a negative outcome for an instance that is actually positive. For example, predicting that a patient with a disease is healthy.

1.7.2 Example of Confusion Matrix:

Suppose you have the following confusion matrix:

Table 3: Confusion Matrix

	Predicted Positive	**Predicted Negative**
Actual Positive	40	10
Actual Negative	5	45

From this matrix:

- TP = 40 (correctly predicted positives)
- TN = 45 (correctly predicted negatives)
- FP = 5 (incorrectly predicted positives)
- FN = 10 (incorrectly predicted negatives)

1.7.3 Summary:

To summarize:

- **True Positives (TP)** are found in the actual positive row under predicted positives.
- **True Negatives (TN)** are found in the actual negative row under predicted negatives.
- **False Positives (FP)** are found in the actual negative row under predicted positives.
- **False Negatives (FN)** are found in the actual positive row under predicted negatives.

These metrics are essential for evaluating classification models and calculating performance measures such as accuracy, precision, recall, and F1 score.

1.7.4 Metrics Derived from the Confusion Matrix:

- **Accuracy:** Accuracy is the measure of how good our model is at predicting correct categories (classes or labels).
$$\left(\frac{TP+TN}{TP+TN+FP+FN}\right) \tag{5}$$
- **Precision:** Precision is the measure of how much of our samples were correctly classified as positive out of all positive samples.
$$\left(\frac{TP}{TP+FP}\right) \tag{6}$$
- **Recall (Sensitivity):** Sensitivity (also called as recall) of a classifier is the ratio between how much were correctly identified as positive to how much were actually positive.
$$\left(\frac{TP}{TP+FN}\right) \tag{7}$$
- **Specificity:** Specificity of a classifier is the ratio between how much were correctly classified as negative to how much was actually negative.
$$Specificity = \left(\frac{TN}{FP+TN}\right) \tag{8}$$
- **F1 Score:**
$$\left(2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}\right) \tag{9}$$

Code for Program:

```
import pandas as pd

from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score, confusion_matrix,
↪  ConfusionMatrixDisplay

from sklearn.preprocessing import LabelEncoder

import matplotlib.pyplot as plt

from matplotlib.lines import Line2D

import numpy as np

'''

The pandas library is imported as pd to handle data manipulation and analysis.

The GaussianNB class from the sklearn.naive_bayes module is imported to create a
↪  Gaussian Naive Bayes classifier.

The train_test_split function from the sklearn.model_selection module is imported
↪  to split the dataset into training and testing sets.

The accuracy_score function from the sklearn.metrics module is imported to
↪  calculate the accuracy of the classifier

The confusion_matrix function from the sklearn.metrics is imported to computer
↪  the confusion matrix of the classifier

The ConfusionMatrixDisplay function from the sklearn.metrics is imported to
↪  display the computed confusion matrix

The matplotlib.pyplot library is imported to plot graphs for doing visual
↪  repersentation of our data

'''

#data = pd.read_csv('your_dataset.csv')

# Create a simple dataset

data = pd.DataFrame({
    'Day': ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday',
↪   'Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday',
↪   'Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday',
↪   'Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday',
↪   'Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday',
↪   'Sunday'],

    'Discount': [0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1,
↪   0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1],

    'Free Delivery': [1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
↪   1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1],

    'Purchase': ['No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes',
↪   'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No',
↪   'Yes', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes',
↪   'No', 'Yes', 'No', 'Yes']

})



'''

A pandas DataFrame named data is created to store the dataset.

The dataset consists of four columns: 'Day', 'Discount', 'Free Delivery', and
↪   'Purchase'.

Each column contains corresponding values for the days of the week, discount
↪   availability, free delivery availability, and purchase status.

'''



# Encoding Categorical Data

le = LabelEncoder()

data['Day'] = le.fit_transform(data['Day'])

'''

An instance of the LabelEncoder class is created and assigned to the variable le.

The 'Day' column of the data DataFrame is transformed using the fit_transform
↪   method of the LabelEncoder object.
```

```
This step is necessary because the 'Day' column contains categorical data, and
↪  the Naive Bayes algorithm requires numerical inputs.

'''

# Split the dataset into features and target variable

X = data[['Day', 'Discount', 'Free Delivery']]

y = data['Purchase']

'''

The features are extracted from the data DataFrame and assigned to the variable
↪  X.

The target variable is extracted from the data DataFrame and assigned to the
↪  variable y.

The features are the 'Day', 'Discount', and 'Free Delivery' columns, while the
↪  target variable is the 'Purchase' column.

'''

# Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
↪  random_state=2)

'''

The train_test_split function is used to split the features (X) and target
↪  variable (y) into training and testing sets.

The training set will contain 80% of the data, and the testing set will contain
↪  20% of the data.

The random_state parameter is set to 42 to ensure reproducibility of the split.

'''

# Create a Gaussian Naive Bayes classifier

classifier = GaussianNB()

'''
```

```
An instance of the GaussianNB class is created and assigned to the variable
↪  classifier.

This classifier will be trained on the training data to learn the patterns in the
↪  features and their corresponding target values.

'''



# Train the classifier

classifier.fit(X_train, y_train)

'''

The fit method of the classifier object is called to train the classifier on the
↪  training data.

The X_train parameter represents the features of the training set, and the
↪  y_train parameter represents the corresponding target values.

'''



# Make predictions on the test set

y_pred = classifier.predict(X_test)

'''

The predict method of the classifier object is called to make predictions on the
↪  test set.

The X_test parameter represents the features of the test set, and the predicted
↪  target values are stored in the y_pred variable.

'''



# Calculate the accuracy of the classifier

accuracy = accuracy_score(y_test, y_pred)

print('Accuracy:', accuracy)
```

```
'''

The accuracy_score function is used to calculate the accuracy of the classifier's
↪  predictions.

The y_test parameter represents the true target values from the test set, and the
↪  y_pred parameter represents the predicted target values.

The calculated accuracy is stored in the accuracy variable.

The print function is used to display the accuracy of the classifier on the
↪  console.

'''

# Example to test the trained model

sample_input = pd.DataFrame({'Day': ['Monday'], 'Discount': [0], 'Free Delivery':
↪  [1]})

sample_input['Day'] = le.transform(sample_input['Day']) # Transform 'Day' column
↪  using LabelEncoder

prediction = classifier.predict(sample_input)

print('Prediction:', prediction)



'''

A sample input is created as a pandas DataFrame named sample_input.

The 'Day' column of the sample_input DataFrame is transformed using the transform
↪  method of the LabelEncoder object (le).

The predict method of the classifier object is called to make a prediction on the
↪  sample_input.

The predicted target value is stored in the prediction variable.

The print function is used to display the prediction on the console.

'''

# Confusion Matrix Visualization
cm = confusion_matrix(y_test,
                      y_pred,labels=['Yes','No'])
```

```
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.title('Confusion Matrix')
plt.show()

# Decision Boundary Visualization (for three features)
def plot_decision_boundary_3d(X, y, model):
    # Create a mesh grid for Day and Discount (we'll fix Free Delivery for
    ↪  visualization)
    x_range = np.linspace(X['Day'].min(), X['Day'].max(), num=50)
    y_range = np.linspace(X['Discount'].min(), X['Discount'].max(), num=50)
    X_grid, Y_grid = np.meshgrid(x_range, y_range)

    # Fix Free Delivery at its mean value for visualization purposes
    Z_fixed = X['Free Delivery'].mean()

    # Prepare input for model prediction as a DataFrame with valid feature names
    grid_points = pd.DataFrame(np.c_[X_grid.ravel(), Y_grid.ravel(),
    ↪  np.full(X_grid.ravel().shape[0], Z_fixed)],
                               columns=['Day', 'Discount', 'Free Delivery'])

    # Predict using the model
    Z_pred = model.predict(grid_points).reshape(X_grid.shape)

    # Plotting the decision boundary in 3D with overlaid legend
    fig = plt.figure(figsize=(12, 8))
    ax = fig.add_subplot(111, projection='3d')

    # Plot decision surface
    ax.plot_surface(X_grid, Y_grid, Z_pred == 'Yes', alpha=0.5, color='#FF0000')

    # Scatter plot of actual data points
    ax.scatter(X['Day'], X['Discount'], X['Free Delivery'], c=y.map({'Yes':
    ↪  '#FF0000', 'No': '#0000FF'}), edgecolor='k')

    # Create legend handles
    legend_handles = [
        Line2D([0], [0], marker='o', color='w', markerfacecolor='#FF0000',
        ↪  label='Predicted Yes'),
        Line2D([0], [0], marker='o', color='w', markerfacecolor='#0000FF',
        ↪  label='Actual Yes'),
        Line2D([0], [0], marker='o', color='w', markerfacecolor='#0000FF',
        ↪  alpha=0.5, label='Actual No')
    ]

    # Add legend
    ax.legend(handles=legend_handles, loc='upper left')
    ax.set_xlabel('Day')
    ax.set_ylabel('Discount')
    ax.set_zlabel('Free Delivery')

    # Adjusting viewing angle for better visibility
    ax.view_init(elev=20., azim=30)

    ax.set_title('Naive Bayes Decision Boundary in 3D')

plot_decision_boundary_3d(X,y,classifier)
plt.tight_layout()
plt.show()
```