Naive Bayes Classifier Algorithm

What is the Naive Bayes Classifier Algorithm?

The Naive Bayes classifier is a probabilistic machine learning model used for classification tasks. It is based on Bayes' theorem and the assumption of independence among features. Despite its simplicity, it can perform surprisingly well for various classification problems.

What is Naïve Bayes Classifier Used For?

Naive Bayes classifiers are used for:

1. **Text classification** (e.g., spam detection, sentiment analysis).
2. **Document categorization**.
3. **Medical diagnosis**.
4. **Real-time prediction systems**.
5. **Recommendation systems**.

How Does the Naïve Bayes Classifier Work?

The Naive Bayes classifier works as follows:

1. **Training Phase**:
    - Calculate the prior probability for each class.
    - Calculate the likelihood of each feature given each class.
    - Apply Bayes' theorem to calculate the posterior probability for each class given a new instance.
2. **Prediction Phase**:
    - For a given instance, calculate the posterior probability for each class.
    - Assign the class with the highest posterior probability to the instance.

What is "Naïve" in Naïve Bayes Classifier?

The term "naïve" refers to the assumption that the features are independent of each other given the class. In reality, this assumption is often violated, but the model can still perform well in practice despite this simplification.

Maths Behind Naïve Bayes Classifier

Bayes' theorem is the foundation of the Naive Bayes classifier:

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

Where:

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

Under the naïve assumption, the likelihood $P(X|C)$ can be decomposed into the product of individual feature probabilities:

$$P(X|C)=P(x_1|C)\cdot P(x_2|C)\cdot \ldots \cdot P(x_n|C)$$

Thus, the classifier becomes:

$$P(C|X)\propto P(C)\cdot \prod_{i=1}^{n} P(x_i|C)$$

Advantages & Disadvantages of Naive Bayes Classifier

Advantages

1. **Simplicity**: Easy to implement and understand.
2. **Speed**: Fast to train and predict.
3. **Scalability**: Works well with large datasets.
4. **Performance**: Performs well with high-dimensional data, especially in text classification.
5. **Robustness**: Handles missing data and irrelevant features well.

Disadvantages

1. **Independence Assumption**: The strong independence assumption often doesn't hold in real-world data.
2. **Zero Probability**: If a feature was not present in the training data, it can result in zero probability during prediction. This is often handled with techniques like Laplace smoothing.
3. **Limited Output Types**: Only provides probability estimates, not useful for all types of prediction tasks.

Code

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.naive_bayes import GaussianNB

from sklearn.metrics import accuracy_score, confusion_matrix

import seaborn as sns


# Load the Iris dataset

iris = load_iris()

X = iris.data

y = iris.target


# Split the dataset into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)


# Initialize the Gaussian Naive Bayes model

model = GaussianNB()


# Train the model

model.fit(X_train, y_train)


# Make predictions
```

```
y_pred = model.predict(X_test)


# Evaluate the model

accuracy = accuracy_score(y_test, y_pred)

print(f'Accuracy: {accuracy}')


# Confusion matrix

conf_matrix = confusion_matrix(y_test, y_pred)


# Visualize the confusion matrix

plt.figure(figsize=(8, 6))

sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=iris.target_names,
yticklabels=iris.target_names)

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()


# Visualization of the decision boundary

def plot_decision_boundaries(X, y, model, ax, title):

    h = .02  # step size in the mesh

    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1

    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))


    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])

    Z = Z.reshape(xx.shape)

    ax.contourf(xx, yy, Z, alpha=0.3, cmap=plt.cm.RdYlBu)
    scatter = ax.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', cmap=plt.cm.RdYlBu)
    legend = ax.legend(*scatter.legend_elements(), title="Classes")
    ax.add_artist(legend)
    ax.set_title(title)

# Reduce dimensions for visualization (only consider the first two features)
X_vis = X[:, :2]
X_train_vis = X_train[:, :2]
X_test_vis = X_test[:, :2]

# Retrain the model with the reduced features
model.fit(X_train_vis, y_train)

# Plot decision boundaries
fig, ax = plt.subplots(1, 1, figsize=(10, 8))
plot_decision_boundaries(X_test_vis, y_test, model, ax, "Naive Bayes Decision Boundary (Iris Dataset)")
plt.show()
```

Explanation

- **numpy**: A library for numerical operations.
- **matplotlib.pyplot**: A library for plotting and visualization.
- **sklearn.datasets**: Provides various datasets including the Iris dataset.
- **sklearn.model_selection**: Contains tools for splitting data into training and testing sets.
- **sklearn.naive_bayes**: Contains the Naive Bayes classifier.
- **sklearn.metrics**: Provides functions to evaluate model performance.
- **seaborn**: A visualization library built on top of matplotlib for creating more attractive plots.
- **load_iris()**: Loads the Iris dataset.
- **X**: The feature matrix (sepal length, sepal width, petal length, petal width).
- **y**: The target vector (species of the iris flower).
- **train_test_split**: Splits the dataset into training and testing sets.
- **test_size=0.3**: 30% of the data is used for testing, 70% for training.

- **random_state=42**: Ensures reproducibility by setting a seed for random number generation.
- **GaussianNB()**: Initializes a Gaussian Naive Bayes model.
- **model.fit(X_train, y_train)**: Trains the model using the training data.
- **model.predict(X_test)**: Uses the trained model to predict the labels for the test set.
- **accuracy_score(y_test, y_pred)**: Calculates the accuracy of the model.
- **print(f'Accuracy: {accuracy}')**: Prints the accuracy.
- **confusion_matrix(y_test, y_pred)**: Generates a confusion matrix to evaluate the accuracy of a classification.
- **plt.figure(figsize=(8, 6))**: Creates a figure with the specified size.
- **sns.heatmap(...)**: Plots the confusion matrix as a heatmap.
    - **conf_matrix**: The confusion matrix data.
    - **annot=True**: Annotates each cell with the numeric value.
    - **fmt='d'**: Formats the annotations as integers.
    - **cmap='Blues'**: Sets the colormap to blues.
    - **xticklabels=iris.target_names**: Sets the x-axis labels to the target names.
    - **yticklabels=iris.target_names**: Sets the y-axis labels to the target names.
- **plt.xlabel('Predicted')**: Labels the x-axis.
- **plt.ylabel('Actual')**: Labels the y-axis.
- **plt.title('Confusion Matrix')**: Sets the title of the plot.
- **plt.show()**: Displays the plot.
- **plot_decision_boundaries(...)**: Defines a function to plot decision boundaries.
    - **h = .02**: Step size for the mesh grid.
    - **x_min, x_max, y_min, y_max**: Defines the range of the grid.
    - **np.meshgrid(...)**: Creates a mesh grid.
    - **model.predict(...)**: Predicts for each point in the mesh grid.
    - **ax.contourf(...)**: Plots the decision boundary.
    - **ax.scatter(...)**: Plots the data points.
    - **ax.legend(...)**: Adds a legend.
    - **ax.set_title(title)**: Sets the title of the plot.
- **X_vis**: Uses only the first two features for visualization.
- **X_train_vis**: Training data with reduced features.

- **X_test_vis**: Test data with reduced features.
- **model.fit(X_train_vis, y_train)**: Retrains the model using only the first two features.
- **plt.subplots(1, 1, figsize=(10, 8))**: Creates a subplot.
- **plot_decision_boundaries(...)**: Plots the decision boundaries.
- **plt.show()**: Displays the plot.
-