Theoretical

In [ ]:
```
#: What is a Support Vector Machine (SVM)?
'''
A Support Vector Machine (SVM) is a supervised machine learning algorithm used for
though it’s more commonly applied to classification problems. SVMs are particularly
cases where the number of dimensions exceeds the number of samples

How SVM Works:
Finding a Hyperplane:
The main idea of SVM is to find the best hyperplane that separates data points of d
A hyperplane is a decision boundary that divides the data space into different clas

Maximizing the Margin:
SVM selects the hyperplane that maximizes the margin between the nearest points of
The larger the margin, the better the generalization of the model.

Handling Non-Linearly Separable Data:
When data isn’t linearly separable, SVM uses the kernel trick to transform data int
hyperplane can separate the classes.
Common kernels include:
Linear Kernel: For linearly separable data.
Polynomial Kernel: For more complex relationships.
Radial Basis Function (RBF): Popular for non-linear data.
Sigmoid Kernel: Sometimes used for neural networks.

Soft Margin for Noisy Data:
SVM introduces a soft margin to allow some misclassifications in exchange for a bet
The C parameter controls the trade-off between maximizing the margin and minimizing
```

In [ ]:
```
#What is the difference between Hard Margin and Soft Margin SVM?
'''
Hard Margin SVM:
Definition:
A Hard Margin SVM assumes that the data is perfectly linearly separable and does no
to find a hyperplane that separates the classes with the maximum margin without any

Key Characteristics:
No Misclassifications: All data points must be on the correct side of the hyperplan
Strict Assumption: Suitable only for datasets that are completely linearly separabl
Sensitive to Noise: Even a small amount of noise or an outlier can prevent the mode

When to Use:
Rarely used in practice due to its inflexibility with real-world data, which often

 Soft Margin SVM:
Definition:
A Soft Margin SVM allows some misclassifications by introducing slack variables (ξi
data. It balances maximizing the margin and minimizing classification errors.

Key Characteristics:
Allows Misclassifications: Some data points can be on the wrong side of the margin
Regularization Parameter (C):
Controls the trade-off between maximizing the margin and minimizing errors.
```

```
High C: Focuses on minimizing misclassifications but may result in a narrow margin
Low C: Focuses on maximizing the margin but allows more misclassifications (better

When to Use:
Commonly used in practice due to its ability to handle noisy and non-linearly separ
```

In [ ]:
```
#What is the mathematical intuition behind SVM?
'''
The mathematical intuition behind Support Vector Machine (SVM) revolves around find
data points of different classes with the maximum margin.

The Hyperplane: Separating the Classes
In an n-dimensional space, a hyperplane is an(n-1) dimensional flat affine subspace
For a 2D space, the hyperplane is a line; for 3D, it's a plane

Mathematical representation
w - x + b = 0
w = Weight vector (normal to the hyperplane)
b = Bias term (shifts the hyperplane).
x = Feature vector (data points)

 Margin: Maximizing the Separation
Margin is the distance between the hyperplane and the closest data points from eith
The goal of SVM is to maximize this margin for better generalization.
Margin formula: 2/w

 Mathematical Intuition
 Find a hyperplane that separates classes with maximum margin
 Minimize 2/1 ‖ w ‖2 subject to constraints for classification.
Introduce slack variables for soft margin (handle misclassifications).
Transform to dual problem for non-linear separability using kernels.
Classify new data using the decision function.
```

In [ ]:
```
#What is the role of Lagrange Multipliers in SVM?
'''
In Support Vector Machines (SVM), Lagrange multipliers play a crucial role in trans
into a simpler one without them. The main idea is to maximize the margin between cl
are correctly classified. Directly solving this problem is complicated due to the c
help by incorporating these constraints into a single objective function, allowing
function. This transformation also enables the use of the dual form of SVM, which d
called support vectors.the ones closest to the decision boundary. An additional ben
it facilitates the use of the kernel trick, which allows SVM to handle non-linear d
space efficiently. Essentially, Lagrange multipliers simplify the optimization proc
linear and non-linear classification tasks.
```

In [ ]:
```
#What are Support Vectors in SVM?
'''
Support Vectors in Support Vector Machines (SVM) are the data points that lie close
These points are critical because they directly influence the position and orientat
different classes. In other words, support vectors are the most informative points
Unlike other data points that have no impact once the hyperplane is set, support ve
maximum-margin hyperplane by being either on the margin boundary or, in the case of
some misclassifications are allowed. Reducing or altering support vectors would cha
points wouldn't affect it. By focusing only on these key data points, SVM becomes e
essential for its functioning.
```

In [ ]:
```
#What is a Support Vector Classifier (SVC) ?
'''
A Support Vector Classifier (SVC) is a type of Support Vector Machine (SVM) used fo
to find the optimal hyperplane that maximally separates data points of different cl
SVC can handle both linearly separable and non-linearly separable data by using dif
data into a higher-dimensional space where a linear separation is possible.
In cases where the data is not perfectly separable, SVC allows some misclassificati
regularization parameter (C) that balances the trade-off between maximizing the mar
By focusing on the support vectors.the most critical data points near the decision
classification rule. Overall, SVC is a powerful and flexible tool for handling comp
tasks.
```

In [ ]:
```
#What is a Support Vector Regressor (SVR)?
'''
A Support Vector Regressor (SVR) is a type of Support Vector Machine (SVM) designed
classification. Unlike classification, where the goal is to separate data points in
continuous values by finding a function that fits the data with minimal error.
The core idea of SVR is to create a tube (or margin) around the regression line whe
known as epsilon (ε), are ignored. The model focuses only on data points that lie o
to find the best-fitting line. SVR uses a regularization parameter (C) to balance t
fewer support vectors) and minimizing prediction errors.Additionally, SVR can handl
kernel trick, which transforms the data into a higher-dimensional space, making it
combination of ignoring small errors within a margin and focusing on critical data
tool for regression tasks.
```

In [ ]:
```
#What is the Kernel Trick in SVM ?
'''
The Kernel Trick in Support Vector Machines (SVM) is a technique that allows SVM to
efficiently. It does this by implicitly mapping the original data into a higher-dim
can separate the classes, without having to compute the transformation explicitly.
In simple terms, the kernel trick replaces the dot product between data points with
product in a higher-dimensional space. This makes it possible to perform complex cl
the high-dimensional coordinates, which would be computationally expensive.

Common kernel functions include:
- Linear Kernel: Suitable for linearly separable data.
- Polynomial Kernel: Captures interactions between features.
- Radial Basis Function (RBF) or Gaussian Kernel: Effective for non-linear data by
- Sigmoid Kernel: Used in neural networks-like scenarios.
```

In [ ]:
```
#Compare Linear Kernel, Polynomial Kernel, and RBF Kernel.
'''
Linear Kernel: Best for large datasets with linear relationships and low computatio
```

```
Polynomial Kernel: Useful when interactions between features are relevant, but can
degrees.
RBF Kernel: Powerful for capturing complex, non-linear patterns but requires carefu
```

In [ ]:
```
#What is the effect of the C parameter in SVM?
'''
The C parameter in Support Vector Machines (SVM) is a regularization parameter that
the margin and minimizing classification errors. It determines how much you want to
training process.
Effects of C parameter
High C: Focuses on minimizing errors → Overfitting risk.
Low C: Focuses on maximizing margin → Underfitting risk
```

In [ ]:
```
#What is the role of the Gamma parameter in RBF Kernel SVM?
'''
The Gamma parameter (γ) in the Radial Basis Function (RBF) Kernel for Support Vecto
of individual training data points. It controls how far the influence of a single t
shaping the decision boundary.
Effects of the Gamma Parameter
High Gamma: Narrow influence → Overfitting risk.
Low Gamma: Wide influence → Underfitting risk.
```

In [ ]:
```
#What is the Naïve Bayes classifier, and why is it called "Naïve"?
'''
The Naïve Bayes classifier is a simple yet powerful probabilistic machine learning
It is based on Bayes' Theorem, which calculates the probability of a class given th
are independent of each other given the class label
Why is it Called "Naïve"?
The term "Naïve" refers to the assumption of independence between features. In real
Naïve Bayes classifier simplifies the computation by assuming they are independent.
efficient and easy to implement, but also "naïve" because it disregards any actual
 Key Characteristics of Naïve Bayes:
Based on Bayes' Theorem:
Uses prior probabilities of classes and the likelihood of features given the class
Independence Assumption:
Assumes that all features contribute independently to the probability of a class.
Efficient and Fast:
Works well with high-dimensional data and is computationally efficient.
Common Applications:
Text classification, spam detection, sentiment analysis, and medical diagnosis.
```

In [ ]:
```
# What is Bayes’ Theorem?
'''
Bayes' Theorem is a principle in probability theory that helps update the probabili
It allows us to reverse conditional probabilities, providing a way to infer the lik
The theorem combines three key components: the prior probability, which represents
before seeing any evidence; the likelihood, which is the probability of observing t
the evidence, which is the overall probability of the observed data. By integrating
the posterior probability.our updated belief about the hypothesis after considering
For example, if we want to determine whether an email is spam based on the presence
Theorem allows us to update our belief by combining our prior knowledge about spam
appearing in spam. This makes the theorem highly valuable in various applications,
diagnosis, machine learning, and risk assessment, by helping to make informed decis
```

In [ ]:
```
#Explain the differences between Gaussian Naïve Bayes, Multinomial Naïve Bayes, and
'''
Gaussian Naïve Bayes:
Best For: Continuous data that follows a normal (Gaussian) distribution.
How It Works: Assumes that features are normally distributed within each class. It
variance of the data.
Common Applications: Iris dataset classification, medical data analysis, and sensor
Example: Predicting whether a tumor is malignant based on continuous features like

Multinomial Naïve Bayes:
Best For: Discrete data representing counts or frequencies of events.
How It Works: Uses the frequency of features (such as word counts in text data) to
feature counts follow a multinomial distribution.
Common Applications: Text classification, spam detection, sentiment analysis.
Example: Classifying emails as spam or not based on the count of specific words

 Bernoulli Naïve Bayes:
Best For: Binary/Boolean data (presence or absence of features).
How It Works: Considers whether a feature is present (1) or absent (0) rather than
follow a Bernoulli distribution.
Common Applications: Text classification with binary features, document categorizat
Example: Classifying emails based on whether specific words appear or not
```

In [ ]:
```
#When should you use Gaussian Naïve Bayes over other variants?
'''
When to Choose Gaussian Naïve Bayes:

Continuous Data:
When your features are real-valued (e.g., height, weight, temperature) rather than

Normal Distribution Assumption:
If your data is approximately normally distributed within each class, Gaussian Naïv
effectively.

Low Dimensional Data:
Works well with a moderate number of features. With very high-dimensional data, oth

Small Datasets:
Performs well with limited training data due to its simplicity and the few paramete

Speed and Efficiency:
If you need a fast and simple classifier that can handle continuous data efficientl
```

In [ ]:
```
#What are the key assumptions made by Naïve Bayes?
'''
Key Assumptions of Naïve Bayes:

Feature Independence Assumption (Naïvety):
Assumes that all features are conditionally independent of each other given the cla
This means the presence or value of one feature does not influence another if the c
In reality, features are often correlated, making this assumption "naïve".

Class-Conditional Independence:
Assumes that the probability of observing a set of features is the product of the i
This simplifies the computation of the posterior probability significantly.
```

```
No Missing Features:
Assumes that all features are available for every instance during both training and
Missing data can degrade performance unless handled explicitly.

Correct Model Assumption:
Assumes that the data follows a specific probability distribution based on the vari

Gaussian Naïve Bayes: Assumes features are normally distributed for continuous data
Multinomial Naïve Bayes: Assumes features follow a multinomial distribution for cou
Bernoulli Naïve Bayes: Assumes features follow a Bernoulli distribution for binary

Equal Feature Importance:
Assumes that all features contribute equally and independently to the outcome.
This can be problematic if some features are more important than others.
```

In [ ]:

```
#What are the advantages and disadvantages of Naïve Bayes?
'''
Advantages: Simple, fast, handles high dimensions, and is effective for small datas
Disadvantages: Strong independence assumption, sensitive to missing data, and strug
```

In [ ]:

```
#Why is Naïve Bayes a good choice for text classification ?
'''
Naïve Bayes is a popular choice for text classification tasks due to its simplicity
Main Reasons:
Handles High-Dimensional Data Efficiently
Feature Independence Assumption Works Well
 Works Well with Sparse Data
 Fast and Scalable
 Effective with Small Datasets
 Works with Different Variants
 No Need for Feature Engineering
```

In [ ]:

```
#Compare SVM and Naïve Bayes for classification tasks
'''
SVM: Best for complex, non-linear problems with high-dimensional data but computati
Naïve Bayes: Ideal for text classification and real-time applications due to speed
assumption
```

In [ ]:

```
#How does Laplace Smoothing help in Naïve Bayes?
'''
Laplace Smoothing helps in Naïve Bayes by addressing the zero-frequency problem, wh
missing in the training data for a given class. This problem can cause the model to
during prediction, making it overly sensitive to rare or unseen features.
How Laplace Smoothing Works:
Adds a Small Positive Value
Adjusts Probability Estimates
Formula Adjustment

Benefits of Laplace Smoothing:
Prevents Zero Probabilities: Ensures that no feature leads to a zero probability fo
Improves Generalization: Makes the model better at handling unseen or rare words in
Simple and Effective: Easy to implement without significantly increasing computatio
```

Practical

In [1]:
```
#Write a Python program to train an SVM Classifier on the Iris dataset and evaluate
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

iris = datasets.load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

svm_classifier = SVC(kernel='rbf', C=1.0, gamma='scale', random_state=42)
svm_classifier.fit(X_train, y_train)

y_pred = svm_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of SVM Classifier: {accuracy * 100:.2f}%")
```

```
Accuracy of SVM Classifier: 100.00%
```

In [3]:
```
#Write a Python program to train two SVM classifiers with Linear and RBF kernels on


wine = datasets.load_wine()
X = wine.data
y = wine.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

svm_linear = SVC(kernel='linear', C=1.0, random_state=42)
svm_linear.fit(X_train, y_train)
y_pred_linear = svm_linear.predict(X_test)
accuracy_linear = accuracy_score(y_test, y_pred_linear)

svm_rbf = SVC(kernel='rbf', C=1.0, gamma='scale', random_state=42)
svm_rbf.fit(X_train, y_train)
y_pred_rbf = svm_rbf.predict(X_test)
accuracy_rbf = accuracy_score(y_test, y_pred_rbf)

print(f"Linear Kernel Accuracy: {accuracy_linear * 100:.2f}%")
print(f"RBF Kernel Accuracy: {accuracy_rbf * 100:.2f}%")
```

```
Linear Kernel Accuracy: 100.00%
RBF Kernel Accuracy: 80.56%
```

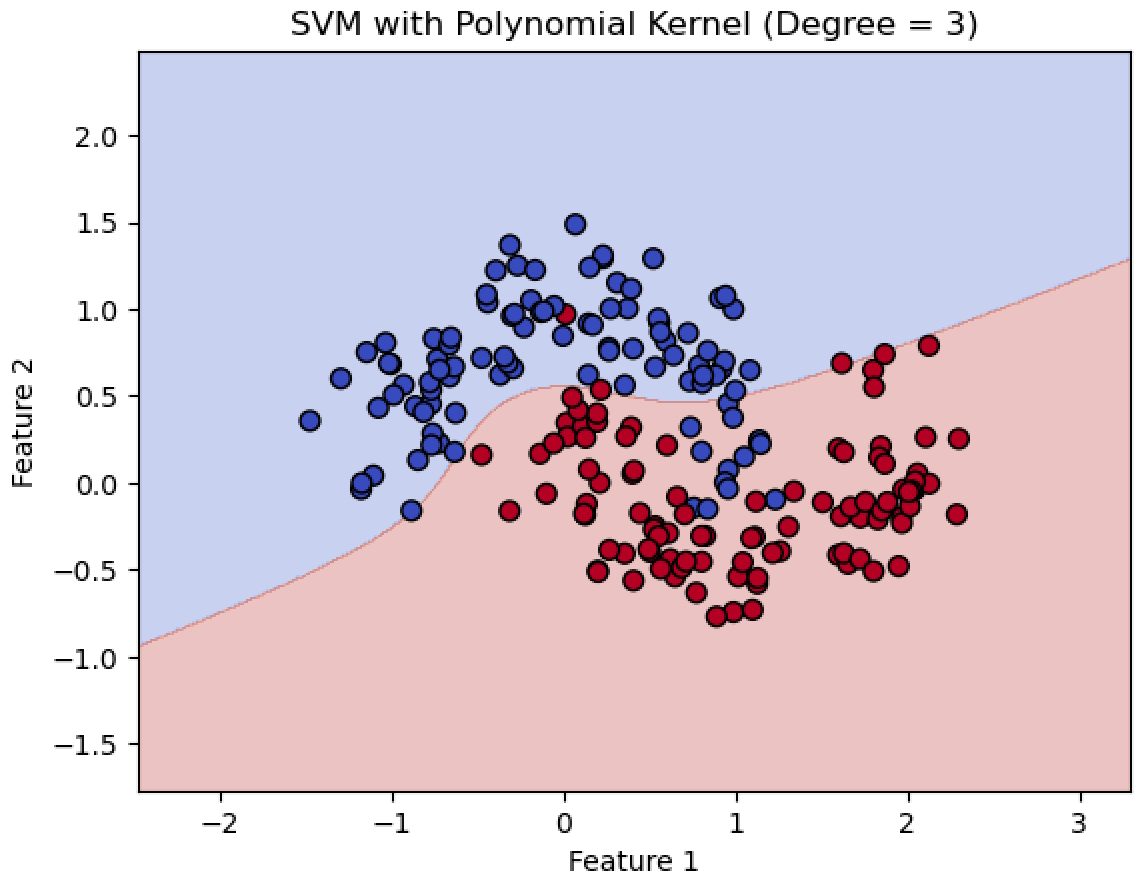
In [7]:
```
#Write a Python program to train an SVM Regressor (SVR) on a housing dataset and ev

from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error

data = datasets.load_diabetes()
```

```
X = data.data
y = data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

svr_regressor = SVR(kernel='rbf', C=100.0, gamma='scale')
svr_regressor.fit(X_train, y_train)

y_pred = svr_regressor.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error (MSE): {mse:.2f}")
```

```
Mean Squared Error (MSE): 2602.87
```

In [9]:

```
#Write a Python program to train an SVM Classifier with a Polynomial Kernel and vis
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
from sklearn.svm import SVC

X, y = make_moons(n_samples=200, noise=0.2, random_state=42)

svm_poly = SVC(kernel='poly', degree=3, C=1.0)
svm_poly.fit(X, y)

def plot_decision_boundary(model, X, y):
    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, 0.01),
                         np.arange(y_min, y_max, 0.01))

    Z = model.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)
    plt.contourf(xx, yy, Z, alpha=0.3, cmap='coolwarm')
    plt.scatter(X[:, 0], X[:, 1], c=y, marker='o', s=50, edgecolor='k', cmap='coolw
    plt.title('SVM with Polynomial Kernel (Degree = 3)')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.show()

plot_decision_boundary(svm_poly, X, y)
```



In [11]:
```
#Write a Python program to train a Gaussian Naïve Bayes classifier on the Breast Ca
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

data = load_breast_cancer()
X = data.data
y = data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

gnb = GaussianNB()
gnb.fit(X_train, y_train)

y_pred = gnb.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of Gaussian Naïve Bayes: {accuracy * 100:.2f}%")
```

```
Accuracy of Gaussian Naïve Bayes: 97.37%
```

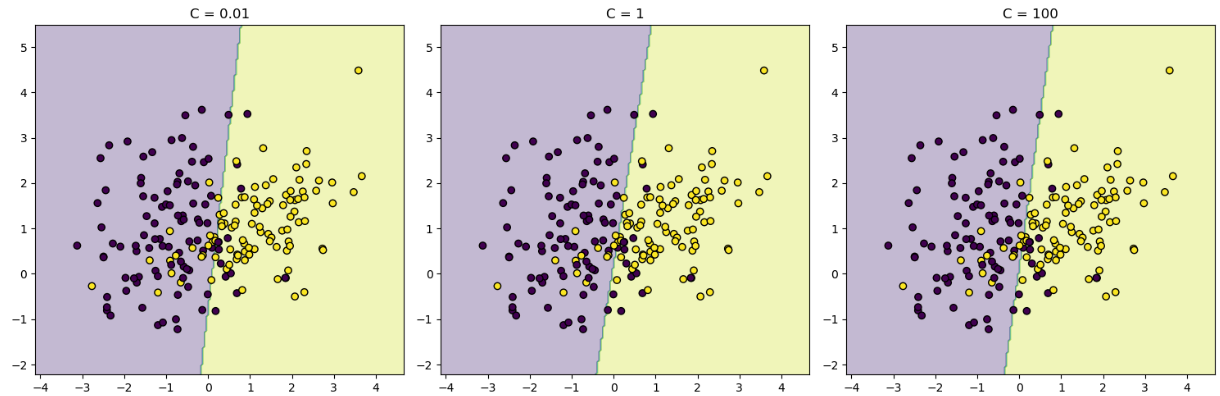
In [15]:
```
#Write a Python program to train a Multinomial Naïve Bayes classifier for text clas
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score

texts = [
```

```
    "I love programming in Python", "Java is a versatile language",
    "Python has great libraries for data science", "I prefer Java for enterprise ap
    "Machine learning is fascinating", "Deep learning requires a lot of data",
    "JavaScript is essential for web development", "React is a powerful JavaScript
]

labels = [0, 1, 0, 1, 0, 0, 1, 1]  # 0: Python/Data Science, 1: Java/JavaScript

vectorizer = CountVectorizer()
X = vectorizer.fit_transform(texts)

X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.25, rand

mnb = MultinomialNB()
mnb.fit(X_train, y_train)

y_pred = mnb.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of Multinomial Naïve Bayes: {accuracy * 100:.2f}%")
```

```
Accuracy of Multinomial Naïve Bayes: 100.00%
```

In [19]:

```
#Write a Python program to train an SVM Classifier with different C values and comp
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.svm import SVC

X, y = make_classification(n_samples=200, n_features=2, n_informative=2, n_redundan

C_values = [0.01, 1, 100]
plt.figure(figsize=(15, 5))

for i, C in enumerate(C_values, 1):
    model = SVC(kernel='linear', C=C)
    model.fit(X, y)

    plt.subplot(1, 3, i)
    plt.title(f"C = {C}")

    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.linspace(x_min, x_max, 200), np.linspace(y_min, y_max,
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)

    plt.contourf(xx, yy, Z, alpha=0.3)
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k')
    plt.xlim(x_min, x_max)
    plt.ylim(y_min, y_max)

plt.tight_layout()
plt.show()
```



In [21]:
```
#Write a Python program to train a Bernoulli Naïve Bayes classifier for binary clas
from sklearn.datasets import make_classification
from sklearn.naive_bayes import BernoulliNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

X, y = make_classification(n_samples=500, n_features=10, n_informative=5, n_classes
X = (X > 0).astype(int)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

model = BernoulliNB()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.7666666666666667
```

In [25]:
```
#Write a Python program to apply feature scaling before training an SVM model and c
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score


data = load_iris()
X = data.data
y = data.target


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta


model_unscaled = SVC(kernel='rbf', random_state=42)
model_unscaled.fit(X_train, y_train)
y_pred_unscaled = model_unscaled.predict(X_test)
accuracy_unscaled = accuracy_score(y_test, y_pred_unscaled)


scaler = StandardScaler()
```

```
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)


model_scaled = SVC(kernel='rbf', random_state=42)
model_scaled.fit(X_train_scaled, y_train)
y_pred_scaled = model_scaled.predict(X_test_scaled)
accuracy_scaled = accuracy_score(y_test, y_pred_scaled)


print("Accuracy without scaling:", accuracy_unscaled)
print("Accuracy with scaling:", accuracy_scaled)
```

```
Accuracy without scaling: 1.0
Accuracy with scaling: 1.0
```

In [27]:

```
#Write a Python program to train a Gaussian Naïve Bayes model and compare the predi

from sklearn.naive_bayes import GaussianNB


data = load_iris()
X = data.data
y = data.target


model_no_smoothing = GaussianNB(var_smoothing=0)
model_no_smoothing.fit(X, y)
y_pred_no_smoothing = model_no_smoothing.predict(X)
accuracy_no_smoothing = accuracy_score(y, y_pred_no_smoothing)


model_smoothing = GaussianNB(var_smoothing=1e-9)
model_smoothing.fit(X, y)
y_pred_smoothing = model_smoothing.predict(X)
accuracy_smoothing = accuracy_score(y, y_pred_smoothing)


print("Accuracy without Laplace Smoothing:", accuracy_no_smoothing)
print("Accuracy with Laplace Smoothing:", accuracy_smoothing)
```

```
Accuracy without Laplace Smoothing: 0.96
Accuracy with Laplace Smoothing: 0.96
```

In [29]:

```
#Write a Python program to train an SVM Classifier and use GridSearchCV to tune the

from sklearn.model_selection import GridSearchCV, train_test_split



data = load_iris()
X = data.data
y = data.target


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta
```

```
model = SVC()


param_grid = {
    'C': [0.1, 1, 10, 100],
    'gamma': ['scale', 0.01, 0.1, 1],
    'kernel': ['linear', 'rbf', 'poly']
}


grid_search = GridSearchCV(model, param_grid, cv=5, n_jobs=-1)
grid_search.fit(X_train, y_train)


best_params = grid_search.best_params_
best_model = grid_search.best_estimator_


y_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)


print("Best Parameters:", best_params)
print("Accuracy:", accuracy)
```

```
Best Parameters: {'C': 1, 'gamma': 'scale', 'kernel': 'poly'}
Accuracy: 0.9777777777777777
```

In [31]:

```
#Write a Python program to train an SVM Classifier on an imbalanced dataset and app
from sklearn.datasets import make_classification
from sklearn.metrics import classification_report, accuracy_score

X, y = make_classification(n_samples=1000, n_features=20, n_classes=2,
                           weights=[0.9, 0.1], random_state=42)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

model_unweighted = SVC(kernel='rbf', random_state=42)
model_unweighted.fit(X_train, y_train)
y_pred_unweighted = model_unweighted.predict(X_test)

accuracy_unweighted = accuracy_score(y_test, y_pred_unweighted)
print("Unweighted SVM Accuracy:", accuracy_unweighted)
print("Classification Report (Unweighted):\n", classification_report(y_test, y_pred

model_weighted = SVC(kernel='rbf', class_weight='balanced', random_state=42)
model_weighted.fit(X_train, y_train)
y_pred_weighted = model_weighted.predict(X_test)

accuracy_weighted = accuracy_score(y_test, y_pred_weighted)
print("\nWeighted SVM Accuracy:", accuracy_weighted)
print("Classification Report (Weighted):\n", classification_report(y_test, y_pred_w
```

```
Unweighted SVM Accuracy: 0.91
Classification Report (Unweighted):
              precision    recall  f1-score   support

           0       0.93      0.98      0.95       270
           1       0.60      0.30      0.40        30

    accuracy                           0.91       300
   macro avg       0.76      0.64      0.68       300
weighted avg       0.89      0.91      0.90       300


Weighted SVM Accuracy: 0.9
Classification Report (Weighted):
              precision    recall  f1-score   support

           0       0.95      0.94      0.94       270
           1       0.50      0.57      0.53        30

    accuracy                           0.90       300
   macro avg       0.73      0.75      0.74       300
weighted avg       0.91      0.90      0.90       300
```

In [35]:

```python
#Write a Python program to implement a Naïve Bayes classifier for spam detection us

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report

data = pd.DataFrame({
    'text': [
        'Congratulations, you have won a lottery!',
        'Call this number to claim your prize now',
        'Hey, are we still meeting tomorrow?',
        'Get cheap loans now',
        'Your friend sent you a photo',
        'Exclusive offer just for you',
        'Reminder for your appointment tomorrow',
        'Win cash prizes easily',
        'Can you send me the report?',
        'Limited time offer, click now!'
    ],
    'label': [1, 1, 0, 1, 0, 1, 0, 1, 0, 1]
})

X_train, X_test, y_train, y_test = train_test_split(data['text'], data['label'], te

vectorizer = CountVectorizer()
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)

model = MultinomialNB()
model.fit(X_train_vec, y_train)
```

```
y_pred = model.predict(X_test_vec)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.6666666666666666
Classification Report:
               precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.67      1.00      0.80         2

    accuracy                           0.67         3
   macro avg       0.33      0.50      0.40         3
weighted avg       0.44      0.67      0.53         3
```

```
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-packages/sklearn/metrics/
_classification.py:1469: UndefinedMetricWarning: Precision and F-score are ill-defin
ed and being set to 0.0 in labels with no predicted samples. Use `zero_division` par
ameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-packages/sklearn/metrics/
_classification.py:1469: UndefinedMetricWarning: Precision and F-score are ill-defin
ed and being set to 0.0 in labels with no predicted samples. Use `zero_division` par
ameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-packages/sklearn/metrics/
_classification.py:1469: UndefinedMetricWarning: Precision and F-score are ill-defin
ed and being set to 0.0 in labels with no predicted samples. Use `zero_division` par
ameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [37]:
```
#Write a Python program to train an SVM Classifier and a Naïve Bayes Classifier on
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

data = load_iris()
X = data.data
y = data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

svm_model = SVC()
svm_model.fit(X_train, y_train)
svm_pred = svm_model.predict(X_test)
svm_accuracy = accuracy_score(y_test, svm_pred)

nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
nb_pred = nb_model.predict(X_test)
nb_accuracy = accuracy_score(y_test, nb_pred)
```

```
print("SVM Accuracy:", svm_accuracy)
print("Naïve Bayes Accuracy:", nb_accuracy)
```

```
SVM Accuracy: 1.0
Naïve Bayes Accuracy: 0.9777777777777777
```

In [39]:

```
#Write a Python program to perform feature selection before training a Naïve Bayes
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

data = load_iris()
X = data.data
y = data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
nb_pred = nb_model.predict(X_test)
nb_accuracy = accuracy_score(y_test, nb_pred)

selector = SelectKBest(score_func=chi2, k=2)
X_train_selected = selector.fit_transform(X_train, y_train)
X_test_selected = selector.transform(X_test)

nb_model_selected = GaussianNB()
nb_model_selected.fit(X_train_selected, y_train)
nb_pred_selected = nb_model_selected.predict(X_test_selected)
nb_accuracy_selected = accuracy_score(y_test, nb_pred_selected)

print("Naïve Bayes Accuracy without Feature Selection:", nb_accuracy)
print("Naïve Bayes Accuracy with Feature Selection:", nb_accuracy_selected)
```

```
Naïve Bayes Accuracy without Feature Selection: 0.9777777777777777
Naïve Bayes Accuracy with Feature Selection: 1.0
```

In [45]:

```
#Write a Python program to train an SVM Classifier using One-vs-Rest (OvR) and One-
#dataset and compare their accuracy.
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.multiclass import OneVsRestClassifier, OneVsOneClassifier
from sklearn.metrics import accuracy_score

data = load_wine()
X = data.data
y = data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

ovr_model = OneVsRestClassifier(SVC(kernel='linear', random_state=42))
ovr_model.fit(X_train, y_train)
ovr_pred = ovr_model.predict(X_test)
```

```
ovr_accuracy = accuracy_score(y_test, ovr_pred)

ovo_model = OneVsOneClassifier(SVC(kernel='linear', random_state=42))
ovo_model.fit(X_train, y_train)
ovo_pred = ovo_model.predict(X_test)
ovo_accuracy = accuracy_score(y_test, ovo_pred)

print("OvR Accuracy:", ovr_accuracy)
print("OvO Accuracy:", ovo_accuracy)
```

```
OvR Accuracy: 0.9814814814814815
OvO Accuracy: 0.9814814814814815
```

In [47]:
```
#Write a Python program to train an SVM Classifier using Linear, Polynomial, and RB
#compare their accuracy
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

data = load_breast_cancer()
X = data.data
y = data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

kernels = ['linear', 'poly', 'rbf']
accuracies = {}

for kernel in kernels:
    model = SVC(kernel=kernel, random_state=42)
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)
    accuracies[kernel] = accuracy_score(y_test, predictions)

for kernel, accuracy in accuracies.items():
    print(f"{kernel.capitalize()} Kernel Accuracy: {accuracy}")
```

```
Linear Kernel Accuracy: 0.9649122807017544
Poly Kernel Accuracy: 0.9415204678362573
Rbf Kernel Accuracy: 0.935672514619883
```

In [49]:
```
#Write a Python program to train an SVM Classifier using Stratified K-Fold Cross-Va
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import StratifiedKFold, cross_val_score
from sklearn.svm import SVC
import numpy as np

data = load_breast_cancer()
X = data.data
y = data.target

model = SVC(kernel='linear', random_state=42)
skf = StratifiedKFold(n_splits=5)

accuracies = cross_val_score(model, X, y, cv=skf)
average_accuracy = np.mean(accuracies)
```

```
print("Accuracies for each fold:", accuracies)
print("Average Accuracy:", average_accuracy)
```

```
Accuracies for each fold: [0.94736842 0.92982456 0.97368421 0.92105263 0.95575221]
Average Accuracy: 0.9455364073901569
```

In [51]:
```
#Write a Python program to train a Naïve Bayes classifier using different prior pro
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

data = load_iris()
X = data.data
y = data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

priors_list = [None, [0.1, 0.3, 0.6], [0.3, 0.3, 0.4]]
for priors in priors_list:
    model = GaussianNB(priors=priors)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Priors: {priors}, Accuracy: {accuracy}")
```

```
Priors: None, Accuracy: 0.9777777777777777
Priors: [0.1, 0.3, 0.6], Accuracy: 0.9555555555555556
Priors: [0.3, 0.3, 0.4], Accuracy: 0.9777777777777777
```

In [53]:
```
#Write a Python program to perform Recursive Feature Elimination (RFE) before train

from sklearn.feature_selection import RFE


data = load_iris()
X = data.data
y = data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

model = SVC(kernel='linear', random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy_before = accuracy_score(y_test, y_pred)

rfe = RFE(model, n_features_to_select=2)
X_train_rfe = rfe.fit_transform(X_train, y_train)
X_test_rfe = rfe.transform(X_test)

model.fit(X_train_rfe, y_train)
y_pred_rfe = model.predict(X_test_rfe)
accuracy_after = accuracy_score(y_test, y_pred_rfe)

print(f"Accuracy before RFE: {accuracy_before}")
print(f"Accuracy after RFE: {accuracy_after}")
```

```
Accuracy before RFE: 1.0
Accuracy after RFE: 1.0
```

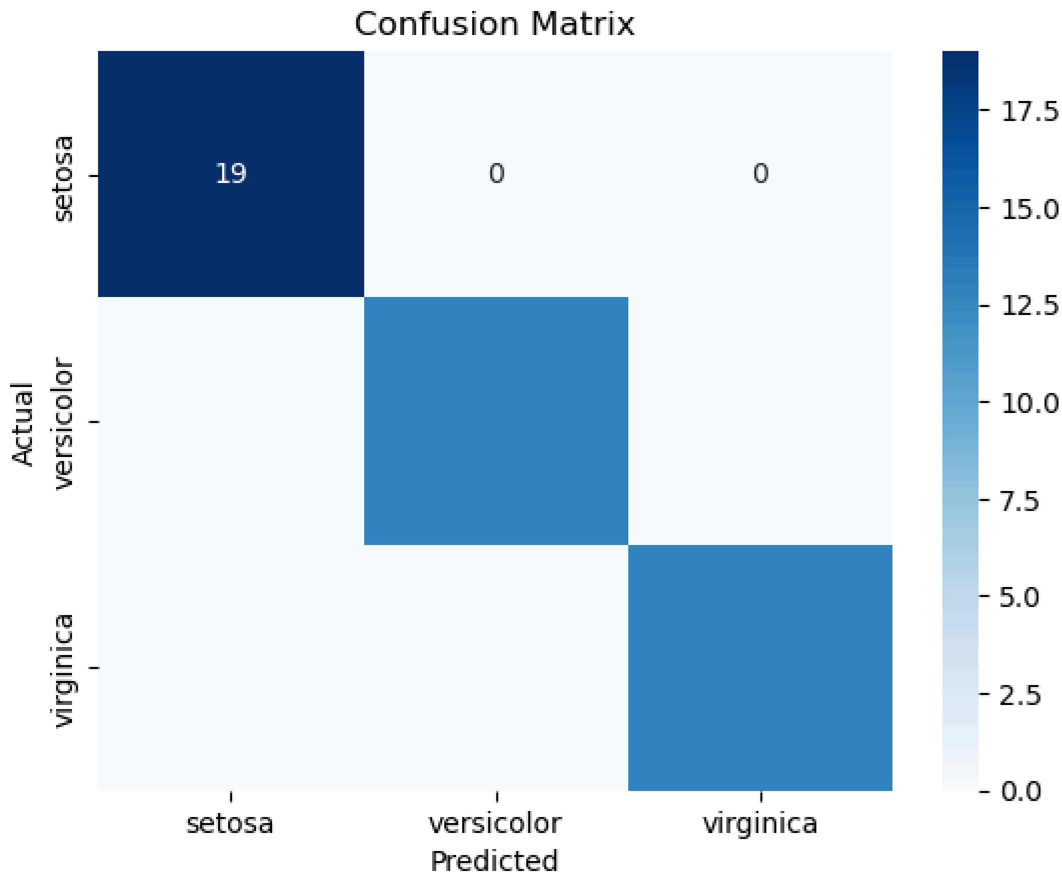
In [55]:
```
#Write a Python program to train an SVM Classifier and evaluate its performance usi
#of accuracy

from sklearn.metrics import precision_score, recall_score, f1_score

data = load_iris()
X = data.data
y = data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

model = SVC(kernel='linear', random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

precision = precision_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred, average='macro')
f1 = f1_score(y_test, y_pred, average='macro')

print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1-Score: {f1}")
```

```
Precision: 1.0
Recall: 1.0
F1-Score: 1.0
```

In [57]:
```
#Write a Python program to train a Naïve Bayes Classifier and evaluate its performa

from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import log_loss

data = load_iris()
X = data.data
y = data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

model = GaussianNB()
model.fit(X_train, y_train)
y_prob = model.predict_proba(X_test)

loss = log_loss(y_test, y_prob)

print(f"Log Loss: {loss}")
```

```
Log Loss: 0.04896447467183273
```

In [59]:
```
# Write a Python program to train an SVM Classifier and visualize the Confusion Mat
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix

data = load_iris()
X = data.data
y = data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

model = SVC(kernel='linear')
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

cm = confusion_matrix(y_test, y_pred)

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=data.target_names, y
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

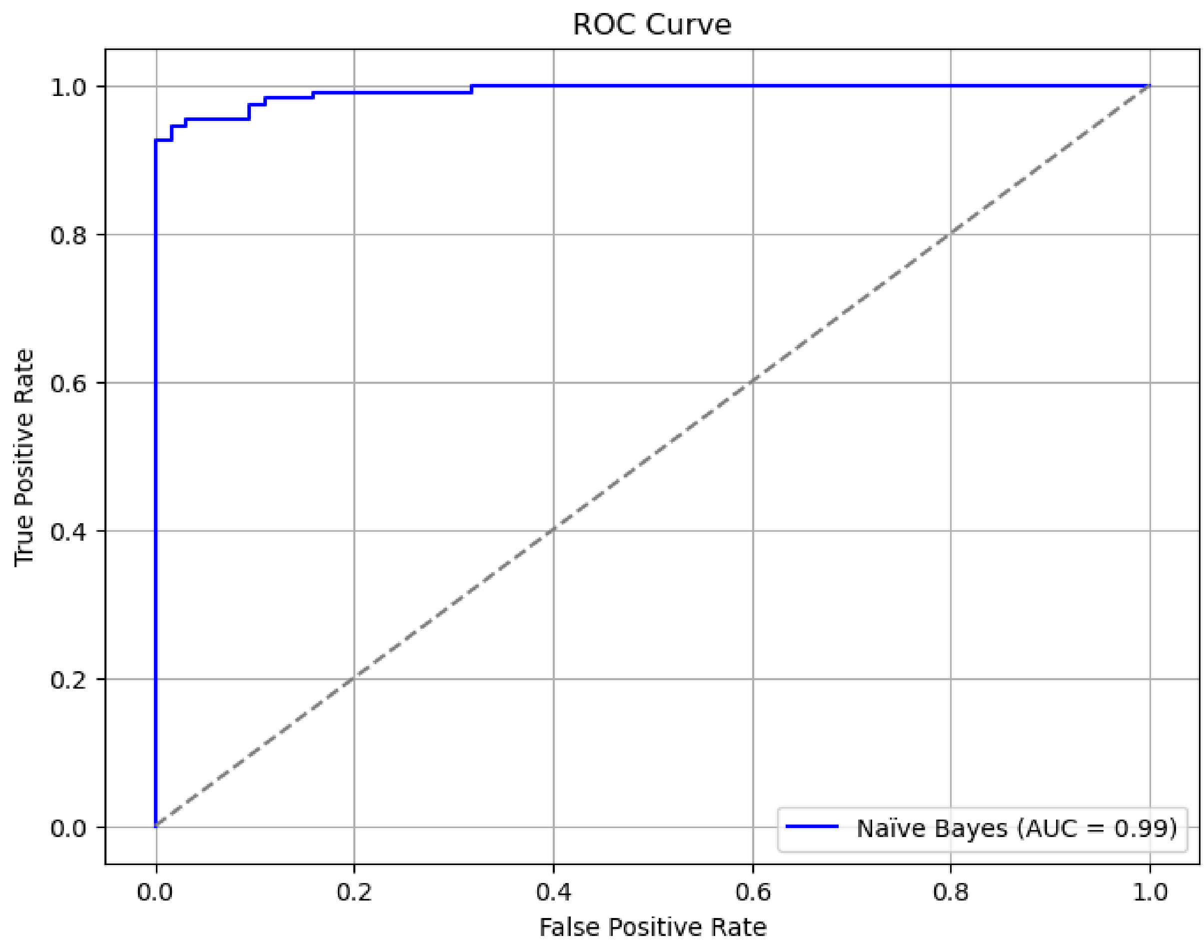


In [65]:
```
#Write a Python program to train an SVM Regressor (SVR) and evaluate its performanc
#Error (MAE) instead of MSE
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.metrics import mean_absolute_error

# Load the Diabetes dataset
data = load_diabetes()
X = data.data
y = data.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

# Train an SVM Regressor
model = SVR(kernel='rbf')
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)

# Calculate Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, y_pred)
print(f"Mean Absolute Error (MAE): {mae:.2f}")
```

```
Mean Absolute Error (MAE): 56.41
```

In [67]:
```
#Write a Python program to train a Naïve Bayes classifier and evaluate its performa
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import roc_auc_score, roc_curve
import matplotlib.pyplot as plt

# Load the Breast Cancer dataset
data = load_breast_cancer()
X = data.data
y = data.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

# Train a Gaussian Naïve Bayes classifier
model = GaussianNB()
model.fit(X_train, y_train)

# Predict probabilities for the positive class
y_proba = model.predict_proba(X_test)[:, 1]

# Calculate ROC-AUC score
roc_auc = roc_auc_score(y_test, y_proba)
print(f"ROC-AUC Score: {roc_auc:.2f}")

# Plot ROC Curve
fpr, tpr, thresholds = roc_curve(y_test, y_proba)
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label=f'Naïve Bayes (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.grid()
plt.show()
```

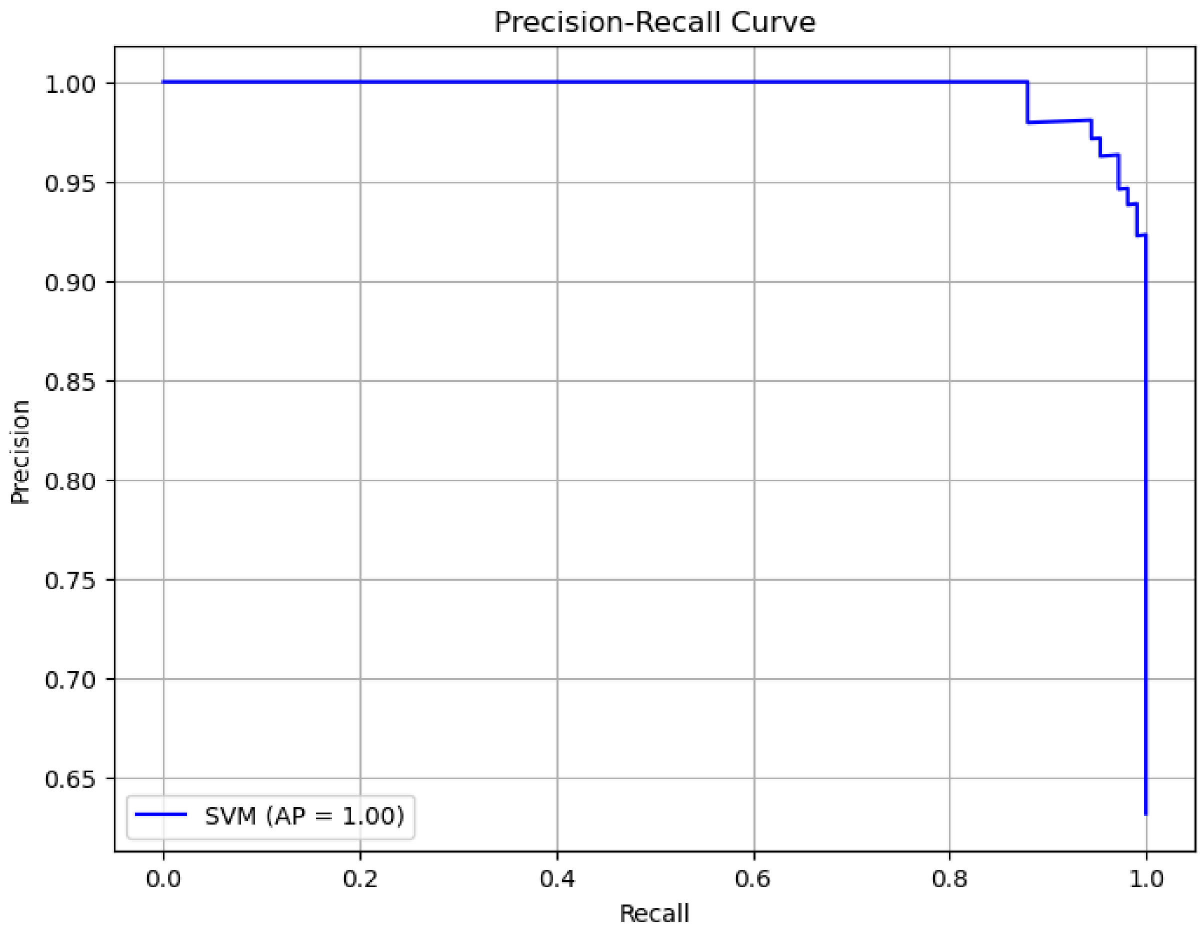
```
ROC-AUC Score: 0.99
```



In [69]:
```
#Write a Python program to train an SVM Classifier and visualize the Precision-Reca
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import precision_recall_curve, average_precision_score
import matplotlib.pyplot as plt

# Load the Breast Cancer dataset
data = load_breast_cancer()
X = data.data
y = data.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

# Train an SVM Classifier with probability estimates enabled
model = SVC(kernel='rbf', probability=True, random_state=42)
model.fit(X_train, y_train)

# Predict probabilities for the positive class
y_proba = model.predict_proba(X_test)[:, 1]

# Calculate precision, recall, and thresholds
precision, recall, thresholds = precision_recall_curve(y_test, y_proba)
avg_precision = average_precision_score(y_test, y_proba)
print(f"Average Precision Score: {avg_precision:.2f}")
```

```python
# Plot Precision-Recall Curve
plt.figure(figsize=(8, 6))
plt.plot(recall, precision, color='blue', label=f'SVM (AP = {avg_precision:.2f})')
plt.title('Precision-Recall Curve')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.legend()
plt.grid()
plt.show()
```

```
Average Precision Score: 1.00
```




```
In [ ]:
```