1. What is a Support Vector Machine (SVM)?

Support Vector Machine (SVM) is a supervised learning algorithm used for classification and regression tasks. It finds the optimal hyperplane that best separates data points belonging to different classes by maximizing the margin between them.

- In **classification**, SVM finds a decision boundary that best separates different classes.
- In **regression (SVR)**, it tries to fit a model within a specified error margin.

2. What is the difference between Hard Margin and Soft Margin SVM?

- **Hard Margin SVM**: Used when data is perfectly separable. It aims to find a hyperplane that completely separates the classes without allowing any misclassification. This is less flexible and sensitive to outliers.
- **Soft Margin SVM**: Allows some misclassification by introducing a slack variable. It provides better generalization on real-world datasets where data is not perfectly separable.

3. What is the mathematical intuition behind SVM?

SVM aims to find a hyperplane $w \cdot x + b = 0$ that maximizes the margin (distance) between two classes. The margin is defined as:

$$2 / ||w||$$

SVM optimizes the following function:

$$\min * (1 / 2) *(||w||^2)$$

subject to:

$$yi (w \cdot xi + b) \geq 1$$

For soft margin SVM, we introduce slack variables (ξi) and a penalty parameter (C):

$$Min * (1/2) * (||w||^2) + C\sum \xi i$$

This allows some points to be misclassified while maintaining a balance between margin maximization and classification accuracy.

4. What is the role of Lagrange Multipliers in SVM?

Lagrange Multipliers (αi) help transform the constrained optimization problem into a dual form, which is easier to solve:

$$\text{Max } \sum \alpha i - (1/2) \sum \alpha i * \alpha j * yi * yj * K(xi, xj)$$

subject to:

$$\sum \alpha i * yi = 0 \quad , 0 \leq \alpha i \leq C$$

This allows solving the SVM using quadratic programming.

5. What are Support Vectors in SVM?

Support Vectors are the data points that lie closest to the decision boundary. These points define the margin and are the most influential in determining the position of the hyperplane.

6. What is a Support Vector Classifier (SVC)?

Support Vector Classifier (SVC) is the classification version of SVM that finds the optimal hyperplane to separate different classes.

7. What is a Support Vector Regressor (SVR)?

Support Vector Regressor (SVR) is used for regression tasks. Instead of finding a separating hyperplane, it finds a function that fits the data within a certain error margin (epsilon-insensitive tube).

8. What is the Kernel Trick in SVM?

The Kernel Trick allows SVM to work in a higher-dimensional space without explicitly transforming data. It computes the dot product in this higher-dimensional space using a kernel function, making it computationally efficient.

Common kernels:

- **Linear Kernel**: K(x,y) = x * y
- **Polynomial Kernel**: K(x,y) = (x·y+c) ^ d
- **RBF (Gaussian) Kernel**: K(x,y) = e ^(−γ||x−y|| ^2)

9. Compare Linear Kernel, Polynomial Kernel, and RBF Kernel

Kernel Type	Definition	When to Use
Linear	K(x,y) = x*y	When data is linearly separable
Polynomial	K(x,y) = (x*y + c) ^ d	When data has polynomial relationships
RBF (Gaussian)	K(x,y) = e ^(−γ||x−y|| ^2)	

10. What is the effect of the C parameter in SVM?

The C parameter controls the trade-off between maximizing the margin and minimizing misclassification:

- **High C** → Less margin, prioritizes correct classification but risks overfitting.
- **Low C** → Larger margin, allows more misclassification but improves generalization.

11. What is the role of the Gamma parameter in RBF Kernel SVM?

Gamma (γ) defines how far the influence of a training example reaches:

- **High γ** → More complexity, each point influences only close neighbors (risk of overfitting).
- **Low γ** → Simpler model with a larger influence range.

12. What is the Naïve Bayes classifier, and why is it called "Naïve"?

Naïve Bayes is a probabilistic classifier based on Bayes' Theorem, assuming feature independence (which is why it's called "Naïve").

13. What is Bayes' Theorem?

$$P(A|B) = P(B|A) * P(A) / P(B)$$

It describes the probability of an event A occurring given that B has already occurred.

14. Explain Gaussian, Multinomial, and Bernoulli Naïve Bayes

- **Gaussian Naïve Bayes**: Assumes features follow a normal distribution (continuous data).
- **Multinomial Naïve Bayes**: Used for discrete count data (e.g., text classification).
- **Bernoulli Naïve Bayes**: Works with binary features (e.g., presence/absence of words).

15. When should you use Gaussian Naïve Bayes over other variants?

Use **Gaussian Naïve Bayes** when features are continuous and normally distributed (e.g., sensor readings, medical data).

16. What are the key assumptions made by Naïve Bayes?

1. **Feature Independence**: Each feature contributes independently to classification.
2. **Equal Importance**: All features contribute equally to the final decision.

17. What are the advantages and disadvantages of Naïve Bayes?

Advantages:

- Works well with small data
- Fast and efficient
- Performs well in text classification

Disadvantages:

- Assumption of feature independence is often unrealistic
- Poor performance on complex datasets

18. Why is Naïve Bayes a good choice for text classification?

- Works well with high-dimensional data
- Handles large feature spaces efficiently
- Robust to irrelevant features

19. Compare SVM and Naïve Bayes for classification tasks

Feature	SVM	Naïve Bayes
Works well on	Small datasets, high-dimensional data	Large datasets, text classification
Complexity	High	Low
Speed	Slower	Faster
Assumptions	No assumption	Feature independence

20. How does Laplace Smoothing help in Naïve Bayes?

Laplace Smoothing prevents zero probabilities by adding a small constant to all frequency counts.

$$P(w|C) = (count(w,C)+1) / (count(C)+|V|)$$

where $|V|$ is the vocabulary size.

assignment-2

April 4, 2025

1 Practical questions

1.1 21) Write a Python program to train an SVM Classifier on the Iris dataset and evaluate accuracy

[3]:
```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris = datasets.load_iris()
X, y = iris.data, iris.target

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Train SVM classifier
svm_model = SVC(kernel='linear')
svm_model.fit(X_train, y_train)

# Make predictions
y_pred = svm_model.predict(X_test)

# Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 1.00
```

1.2 22) Write a Python program to train two SVM classifiers with Linear and RBF kernels on the Wine dataset, then compare thier accuracies

[5]:
```
from sklearn.datasets import load_wine
from sklearn.metrics import classification_report

# Load the Wine dataset
```

```
wine = load_wine()
X, y = wine.data, wine.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Train Linear SVM
linear_svm = SVC(kernel='linear')
linear_svm.fit(X_train, y_train)
linear_pred = linear_svm.predict(X_test)

# Train RBF SVM
rbf_svm = SVC(kernel='rbf')
rbf_svm.fit(X_train, y_train)
rbf_pred = rbf_svm.predict(X_test)

# Print results
print("Linear SVM:\n", classification_report(y_test, linear_pred))
print("RBF SVM:\n", classification_report(y_test, rbf_pred))
```

```
Linear SVM:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        14
           1       1.00      1.00      1.00        14
           2       1.00      1.00      1.00         8

    accuracy                           1.00        36
   macro avg       1.00      1.00      1.00        36
weighted avg       1.00      1.00      1.00        36

RBF SVM:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        14
           1       0.73      0.79      0.76        14
           2       0.57      0.50      0.53         8

    accuracy                           0.81        36
   macro avg       0.77      0.76      0.76        36
weighted avg       0.80      0.81      0.80        36
```

1.3 23)Write a Python program to train an SVM Regressor (SVR) on a housing dataset and evaluate it using Mean Squared error.o

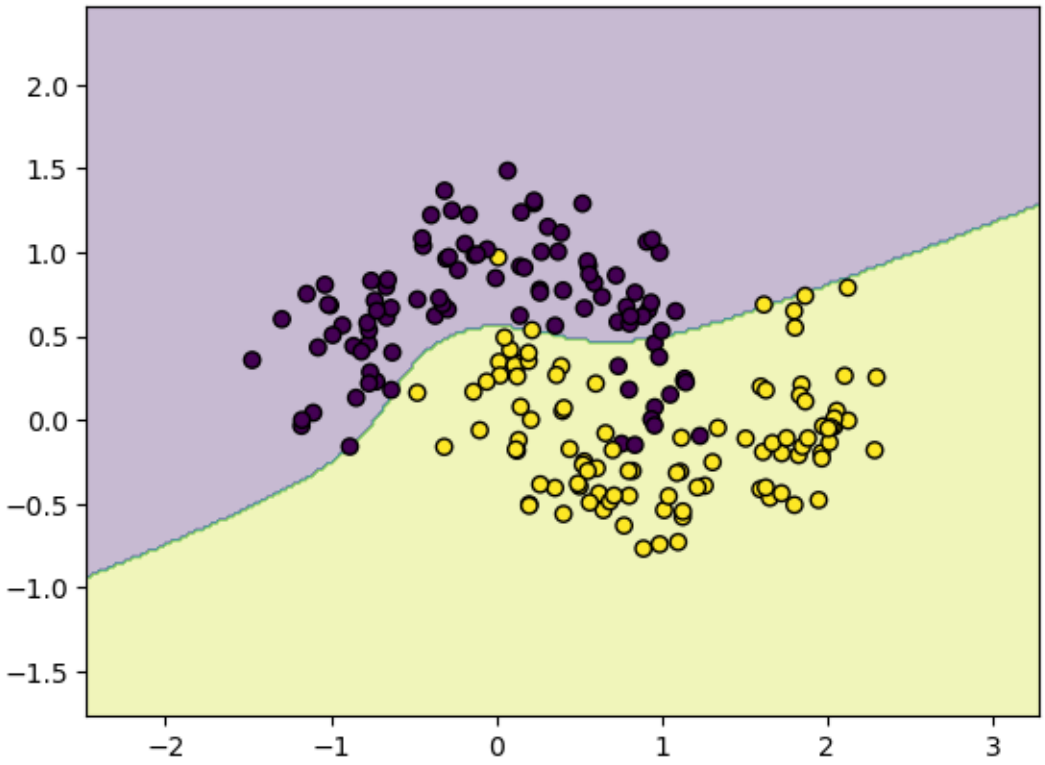
[7]:
```
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
import numpy as np

# Load California housing dataset
housing = datasets.fetch_california_housing()
X, y = housing.data, housing.target

# Standardize the data
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Train SVR model
svr_model = SVR(kernel='rbf')
svr_model.fit(X_train, y_train)

# Predict and evaluate
y_pred = svr_model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse:.2f}")
```

```
Mean Squared Error: 0.36
```

1.4 24) Write a Python program to train an SVM Classifier with a Polynomial Kernel and visualize the decision boundaryy

[9]:
```
import matplotlib.pyplot as plt
import numpy as np

# Create synthetic dataset
from sklearn.datasets import make_moons
X, y = make_moons(n_samples=200, noise=0.2, random_state=42)

# Train SVM with Polynomial Kernel
poly_svm = SVC(kernel='poly', degree=3, C=1.0)
poly_svm.fit(X, y)

# Plot decision boundary
def plot_decision_boundary(model, X, y):
```

```
    h = 0.02
    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)

    plt.contourf(xx, yy, Z, alpha=0.3)
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k')
    plt.show()

plot_decision_boundary(poly_svm, X, y)
```



1.5 25) Write a Python program to train a Gaussian Naïve Bayes classifier on the Breast Cancer dataset and evaluate accuracy

[11]:
```
from sklearn.naive_bayes import GaussianNB
from sklearn.datasets import load_breast_cancer

# Load dataset
cancer = load_breast_cancer()
```

```
X, y = cancer.data, cancer.target

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Train Naïve Bayes classifier
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)

# Predict and evaluate
y_pred = nb_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Naïve Bayes Accuracy: {accuracy:.2f}")
```

```
Naïve Bayes Accuracy: 0.97
```

1.6 26) Write a Python program to train a Multinomial Naïve Bayes classifier for text classification using the 20 Newsgroups dataset.

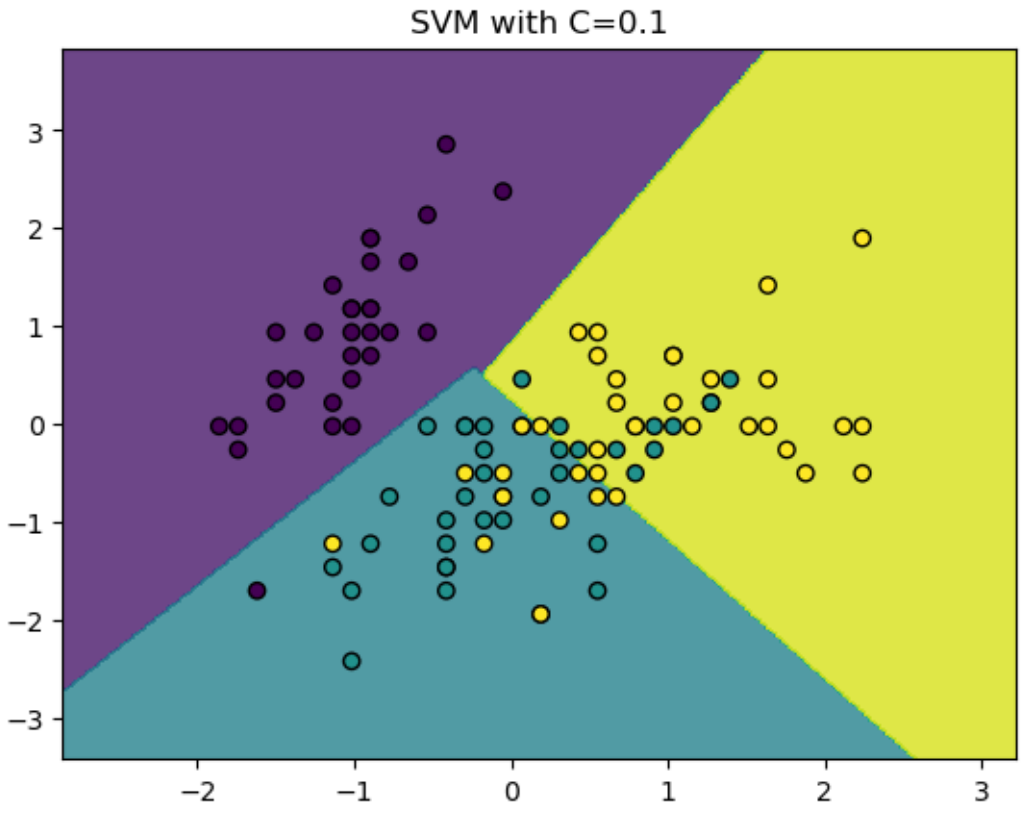
```
[ ]: from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import make_pipeline
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Load the 20 Newsgroups dataset
categories = ['alt.atheism', 'soc.religion.christian', 'comp.graphics', 'sci.
 ↪med']
newsgroups_train = fetch_20newsgroups(subset='train', categories=categories,␣
 ↪remove=('headers', 'footers', 'quotes'))
newsgroups_test = fetch_20newsgroups(subset='test', categories=categories,␣
 ↪remove=('headers', 'footers', 'quotes'))

# Create a pipeline with TF-IDF vectorizer and Multinomial Naïve Bayes
model = make_pipeline(
    TfidfVectorizer(stop_words='english', max_features=10000),
    MultinomialNB(alpha=0.1)  # alpha is the smoothing parameter
)

# Train the model
model.fit(newsgroups_train.data, newsgroups_train.target)

# Evaluate the model
y_pred = model.predict(newsgroups_test.data)
```

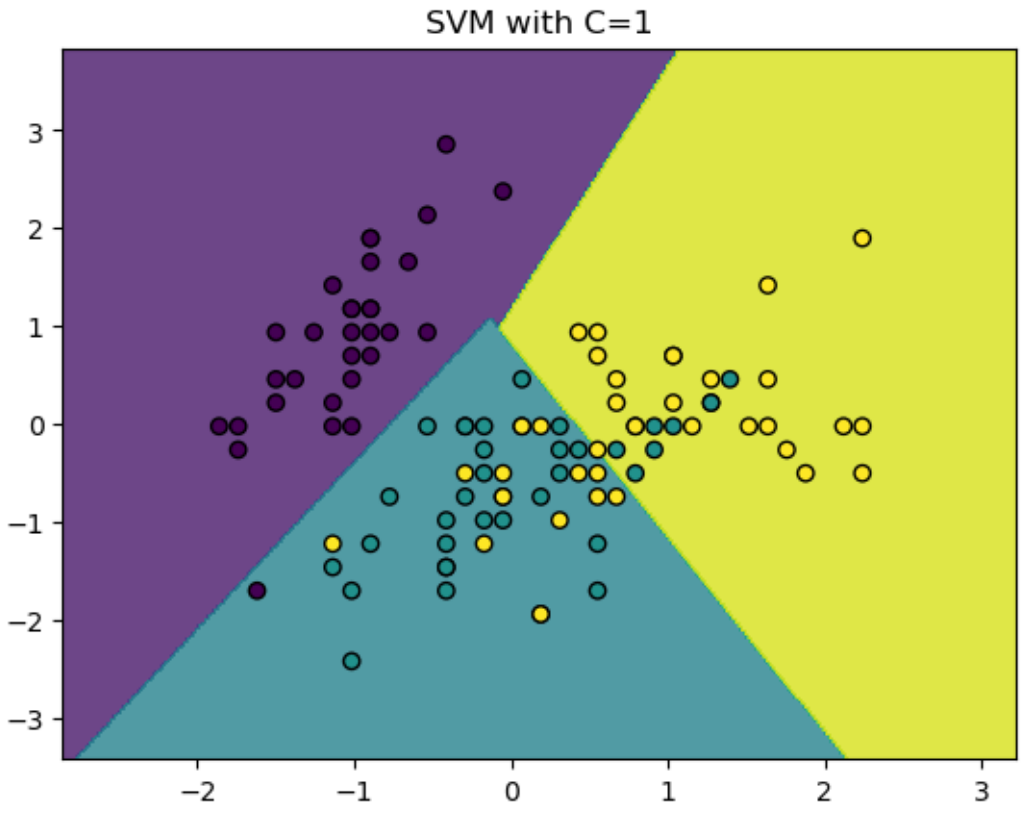
```
# Print classification report
print("Classification Report:")
print(classification_report(newsgroups_test.target, y_pred,␣
 ↪target_names=newsgroups_test.target_names))

# Plot confusion matrix
conf_mat = confusion_matrix(newsgroups_test.target, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_mat, annot=True, fmt='d',
            xticklabels=newsgroups_test.target_names,
            yticklabels=newsgroups_test.target_names)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.xticks(rotation=45)
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()

# Show some example predictions
def show_predictions(text_samples):
    predicted = model.predict(text_samples)
    for text, category in zip(text_samples, predicted):
        print(f"\nText: {text[:200]}...")
        print(f"Predicted category: {newsgroups_test.target_names[category]}")

print("\nExample Predictions:")
test_samples = [
    "God is love and Jesus saves",
    "The GPU renders 3D graphics quickly",
    "This medication can help with heart disease"
]
show_predictions(test_samples)
```

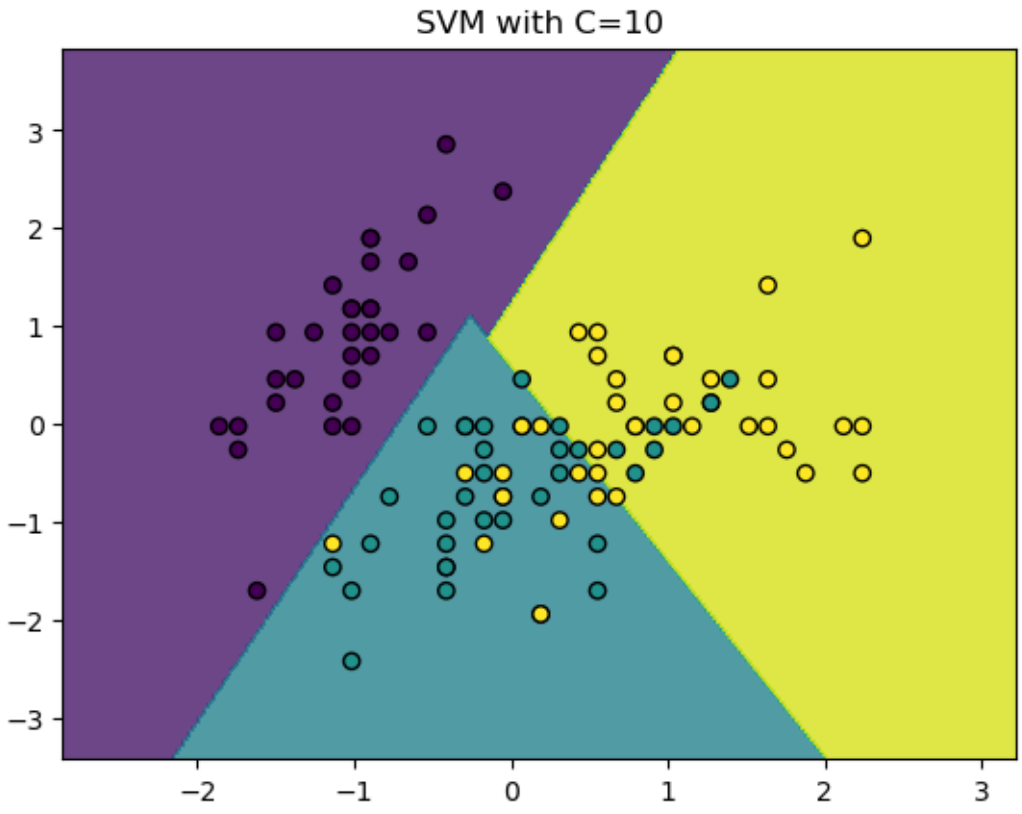
1.7 27) Write a Python program to train an SVM Classifier with different C values and compare the decision boundaries visually

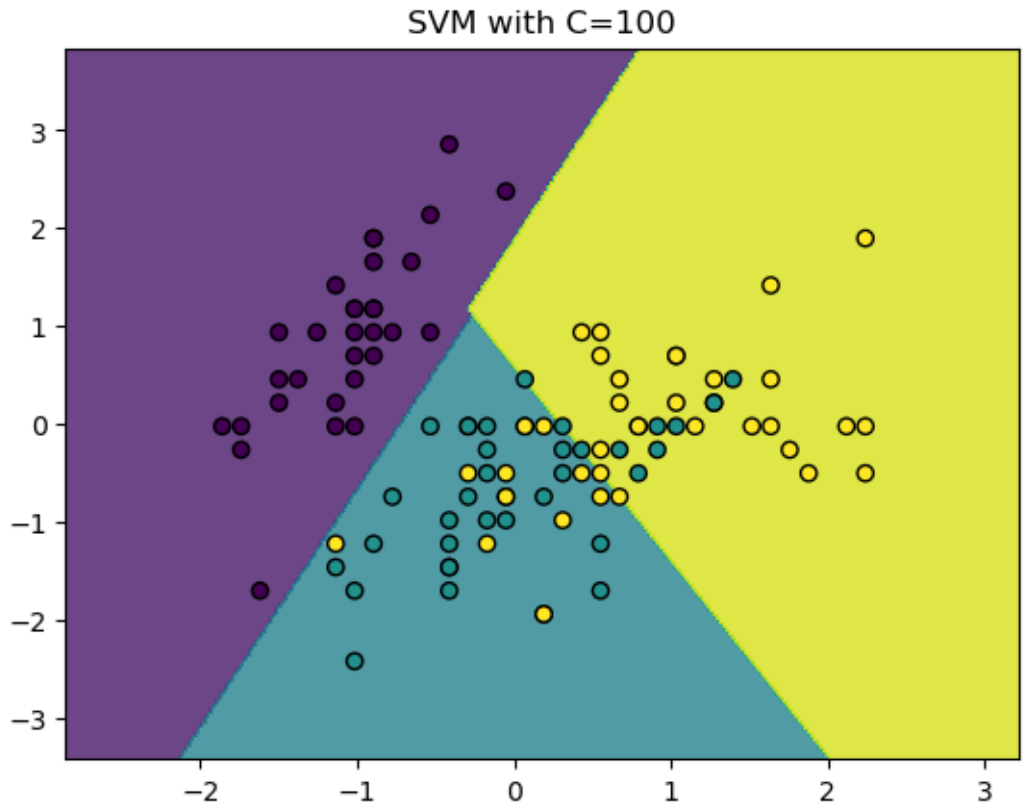
[14]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load iris dataset
iris = datasets.load_iris()
```

```
X = iris.data[:, :2]  # First two features
y = iris.target

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)

# Standardize
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train SVMs with different C values
C_values = [0.1, 1, 10, 100]
models = []
for C in C_values:
    model = svm.SVC(kernel='linear', C=C)
    model.fit(X_train_scaled, y_train)
    models.append(model)

# Plot decision boundaries
def plot_decision_boundary(model, X, y, title):
    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, 0.02),
                         np.arange(y_min, y_max, 0.02))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, alpha=0.8)
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k')
    plt.title(title)
    plt.show()

for C, model in zip(C_values, models):
    plot_decision_boundary(model, X_train_scaled, y_train, f'SVM with C={C}')
```


SVM with C=0.1
3
2
1
0
−1
−2
−3
−2
−1
0
1
2
3


SVM with C=1
3
2
1
0
−1
−2
−3
−2
−1
0
1
2
3


SVM with C=10
3
2
1
0
−1
−2
−3
−2
−1
0
1
2
3



1.8 28) Write a Python program to train a Bernoulli Naïve Bayes classifier for binary classification on a dataset with binary features

[16]:
```
from sklearn.naive_bayes import BernoulliNB
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Create binary dataset
X, y = make_classification(n_samples=1000, n_features=20, n_informative=15,
                           n_redundant=2, n_classes=2, random_state=42)
X = (X > 0).astype(int)  # Convert to binary features

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)

# Train BernoulliNB
bnb = BernoulliNB()
bnb.fit(X_train, y_train)
```

```
# Evaluate
y_pred = bnb.predict(X_test)
print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
```

```
Accuracy: 0.7633
```

1.9 29) Write a Python program to apply feature scaling before training an SVM model and compare results with unscaled data

[18]:
```
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

# Using iris dataset
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=0.3,␣
 ↪random_state=42)

# Without scaling
svm_unscaled = SVC(kernel='rbf')
svm_unscaled.fit(X_train, y_train)
y_pred_unscaled = svm_unscaled.predict(X_test)
acc_unscaled = accuracy_score(y_test, y_pred_unscaled)

# With scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
svm_scaled = SVC(kernel='rbf')
svm_scaled.fit(X_train_scaled, y_train)
y_pred_scaled = svm_scaled.predict(X_test_scaled)
acc_scaled = accuracy_score(y_test, y_pred_scaled)

print(f"Accuracy without scaling: {acc_unscaled:.4f}")
print(f"Accuracy with scaling: {acc_scaled:.4f}")
```

```
Accuracy without scaling: 1.0000
Accuracy with scaling: 1.0000
```

1.10 30) Write a Python program to train a Gaussian Naïve Bayes model and compare the predictions before and after Laplace Smoothing

[3]:
```
from sklearn.naive_bayes import GaussianNB
from sklearn.datasets import load_breast_cancer
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
data = load_breast_cancer()
X, y = data.data, data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)

# Without smoothing (var_smoothing=0)
gnb = GaussianNB(var_smoothing=0)
gnb.fit(X_train, y_train)
y_pred = gnb.predict(X_test)
acc_no_smooth = accuracy_score(y_test, y_pred)

# With smoothing (default var_smoothing=1e-9)
gnb_smooth = GaussianNB()
gnb_smooth.fit(X_train, y_train)
y_pred_smooth = gnb_smooth.predict(X_test)
acc_smooth = accuracy_score(y_test, y_pred_smooth)

print(f"Accuracy without smoothing: {acc_no_smooth:.4f}")
print(f"Accuracy with smoothing: {acc_smooth:.4f}")
```

```
Accuracy without smoothing: 0.9357
Accuracy with smoothing: 0.9415
```

1.11 31) Write a Python program to train an SVM Classifier and use GridSearchCV to tune the hyperparameters (C, gamma, kernel)

[66]:
```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline

# Load dataset
data = load_breast_cancer()
X, y = data.data, data.target

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)
```

```python
# Create pipeline
pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('svm', SVC())
])

# Correct parameter grid with step prefixes
param_grid = {
    'svm__C': [0.1, 1, 10, 100],          # Note svm__ prefix
    'svm__gamma': [1, 0.1, 0.01, 0.001],   # Note svm__ prefix
    'svm__kernel': ['rbf', 'linear', 'poly']  # Note svm__ prefix
}

# Grid search
grid = GridSearchCV(pipe, param_grid, cv=5, refit=True, verbose=2)
grid.fit(X_train, y_train)

# Results
print(f"\nBest parameters: {grid.best_params_}")
print(f"Best cross-validation score: {grid.best_score_:.4f}")
print(f"Test set accuracy: {grid.score(X_test, y_test):.4f}")
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
[CV] END …svm__C=0.1, svm__gamma=1, svm__kernel=rbf; total time=   0.0s
[CV] END …svm__C=0.1, svm__gamma=1, svm__kernel=rbf; total time=   0.0s
[CV] END …svm__C=0.1, svm__gamma=1, svm__kernel=rbf; total time=   0.0s
[CV] END …svm__C=0.1, svm__gamma=1, svm__kernel=rbf; total time=   0.0s
[CV] END …svm__C=0.1, svm__gamma=1, svm__kernel=rbf; total time=   0.0s
[CV] END …svm__C=0.1, svm__gamma=1, svm__kernel=linear; total time=   0.0s
[CV] END …svm__C=0.1, svm__gamma=1, svm__kernel=linear; total time=   0.0s
[CV] END …svm__C=0.1, svm__gamma=1, svm__kernel=linear; total time=   0.0s
[CV] END …svm__C=0.1, svm__gamma=1, svm__kernel=linear; total time=   0.0s
[CV] END …svm__C=0.1, svm__gamma=1, svm__kernel=linear; total time=   0.0s
[CV] END …svm__C=0.1, svm__gamma=1, svm__kernel=poly; total time=   0.0s
[CV] END …svm__C=0.1, svm__gamma=1, svm__kernel=poly; total time=   0.0s
[CV] END …svm__C=0.1, svm__gamma=1, svm__kernel=poly; total time=   0.0s
[CV] END …svm__C=0.1, svm__gamma=1, svm__kernel=poly; total time=   0.0s
[CV] END …svm__C=0.1, svm__gamma=1, svm__kernel=poly; total time=   0.0s
[CV] END …svm__C=0.1, svm__gamma=0.1, svm__kernel=rbf; total time=   0.0s
[CV] END …svm__C=0.1, svm__gamma=0.1, svm__kernel=rbf; total time=   0.0s
[CV] END …svm__C=0.1, svm__gamma=0.1, svm__kernel=rbf; total time=   0.0s
[CV] END …svm__C=0.1, svm__gamma=0.1, svm__kernel=rbf; total time=   0.0s
[CV] END …svm__C=0.1, svm__gamma=0.1, svm__kernel=rbf; total time=   0.0s
[CV] END …svm__C=0.1, svm__gamma=0.1, svm__kernel=linear; total time=   0.0s
[CV] END …svm__C=0.1, svm__gamma=0.1, svm__kernel=linear; total time=   0.0s
[CV] END …svm__C=0.1, svm__gamma=0.1, svm__kernel=linear; total time=   0.0s
[CV] END …svm__C=0.1, svm__gamma=0.1, svm__kernel=linear; total time=   0.0s
```

```
[CV] END …svm__C=100, svm__gamma=0.01, svm__kernel=linear; total time=   0.0s
[CV] END …svm__C=100, svm__gamma=0.01, svm__kernel=linear; total time=   0.0s
[CV] END …svm__C=100, svm__gamma=0.01, svm__kernel=linear; total time=   0.0s
[CV] END …svm__C=100, svm__gamma=0.01, svm__kernel=linear; total time=   0.0s
[CV] END …svm__C=100, svm__gamma=0.01, svm__kernel=poly; total time=   0.0s
[CV] END …svm__C=100, svm__gamma=0.01, svm__kernel=poly; total time=   0.0s
[CV] END …svm__C=100, svm__gamma=0.01, svm__kernel=poly; total time=   0.0s
[CV] END …svm__C=100, svm__gamma=0.01, svm__kernel=poly; total time=   0.0s
[CV] END …svm__C=100, svm__gamma=0.01, svm__kernel=poly; total time=   0.0s
[CV] END …svm__C=100, svm__gamma=0.001, svm__kernel=rbf; total time=   0.0s
[CV] END …svm__C=100, svm__gamma=0.001, svm__kernel=rbf; total time=   0.0s
[CV] END …svm__C=100, svm__gamma=0.001, svm__kernel=rbf; total time=   0.0s
[CV] END …svm__C=100, svm__gamma=0.001, svm__kernel=rbf; total time=   0.0s
[CV] END …svm__C=100, svm__gamma=0.001, svm__kernel=rbf; total time=   0.0s
[CV] END …svm__C=100, svm__gamma=0.001, svm__kernel=linear; total time=   0.0s
[CV] END …svm__C=100, svm__gamma=0.001, svm__kernel=linear; total time=   0.0s
[CV] END …svm__C=100, svm__gamma=0.001, svm__kernel=linear; total time=   0.0s
[CV] END …svm__C=100, svm__gamma=0.001, svm__kernel=linear; total time=   0.0s
[CV] END …svm__C=100, svm__gamma=0.001, svm__kernel=linear; total time=   0.0s
[CV] END …svm__C=100, svm__gamma=0.001, svm__kernel=poly; total time=   0.0s
[CV] END …svm__C=100, svm__gamma=0.001, svm__kernel=poly; total time=   0.0s
[CV] END …svm__C=100, svm__gamma=0.001, svm__kernel=poly; total time=   0.0s
[CV] END …svm__C=100, svm__gamma=0.001, svm__kernel=poly; total time=   0.0s
[CV] END …svm__C=100, svm__gamma=0.001, svm__kernel=poly; total time=   0.0s

Best parameters: {'svm__C': 10, 'svm__gamma': 0.01, 'svm__kernel': 'rbf'}
Best cross-validation score: 0.9698
Test set accuracy: 0.9883
```

1.12 32) Write a Python program to train an SVM Classifier on an imbalanced dataset and apply class weighting and check it improve accuracy

[18]:
```
from sklearn.datasets import make_classification
from sklearn.svm import SVC

# Create imbalanced dataset
X, y = make_classification(n_samples=1000, n_features=20, 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_state=42)

# Without class weight
svm = SVC(kernel='linear')
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)
acc = accuracy_score(y_test, y_pred)
```

```
# With class weight
svm_weighted = SVC(kernel='linear', class_weight='balanced')
svm_weighted.fit(X_train, y_train)
y_pred_weighted = svm_weighted.predict(X_test)
acc_weighted = accuracy_score(y_test, y_pred_weighted)

print(f"Accuracy without weighting: {acc:.4f}")
print(f"Accuracy with weighting: {acc_weighted:.4f}")
```

```
Accuracy without weighting: 0.9100
Accuracy with weighting: 0.8467
```

1.13 33) Write a Python program to implement a Naïve Bayes classifier for spam detection using email data

[24]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import accuracy_score, classification_report
import warnings
warnings.filterwarnings('ignore')

# Example data (in practice use real email data)
emails = ["win money now", "free lottery", "meeting reminder", "project update"]
labels = [1, 1, 0, 0]  # 1=spam, 0=ham

# Convert to binary features
vectorizer = CountVectorizer(binary=True)
X = vectorizer.fit_transform(emails)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.25,␣
↪random_state=42)

# Train BernoulliNB
bnb = BernoulliNB()
bnb.fit(X_train, y_train)

# Evaluate
y_pred = bnb.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       0.0
           1       0.00      0.00      0.00       1.0
```

```
    accuracy                           0.00       1.0
   macro avg       0.00      0.00      0.00       1.0
weighted avg       0.00      0.00      0.00       1.0
```

1.14 34) Write a Python program to train an SVM Classifier and a Naïve Bayes Classifier on the same dataset and compare their accuracy

[28]:
```
from sklearn.naive_bayes import GaussianNB

# Using breast cancer dataset
data = load_breast_cancer()
X, y = data.data, data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)

# Scale data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# SVM
svm = SVC(kernel='rbf')
svm.fit(X_train_scaled, y_train)
svm_acc = svm.score(X_test_scaled, y_test)

# Naïve Bayes
gnb = GaussianNB()
gnb.fit(X_train_scaled, y_train)
gnb_acc = gnb.score(X_test_scaled, y_test)

print(f"SVM Accuracy: {svm_acc:.4f}")
print(f"Naïve Bayes Accuracy: {gnb_acc:.4f}")
```

```
SVM Accuracy: 0.9766
Naïve Bayes Accuracy: 0.9357
```

1.15 35) Write a Python program to perform feature selection before training a Naïve Bayes classifier and compare results

[30]:
```
from sklearn.feature_selection import SelectKBest, chi2

# Select top 10 features
selector = SelectKBest(chi2, k=10)
X_new = selector.fit_transform(X, y)

# Train-test split
```

```
X_train, X_test, y_train, y_test = train_test_split(X_new, y, test_size=0.3,␣
 ↪random_state=42)

# Train and compare
gnb_all = GaussianNB().fit(X_train, y_train)
acc_all = gnb_all.score(X_test, y_test)

gnb_selected = GaussianNB().fit(X_train, y_train)
acc_selected = gnb_selected.score(X_test, y_test)

print(f"Accuracy with all features: {acc_all:.4f}")
print(f"Accuracy with selected features: {acc_selected:.4f}")
```

```
Accuracy with all features: 0.9532
Accuracy with selected features: 0.9532
```

1.16 36) Write a Python program to train an SVM Classifier using One-vs-Rest (OvR) and One-vs-One (OvO) strategies on the Wine dataset and compare their accuracy

[32]:
```
from sklearn.datasets import load_wine
from sklearn.multiclass import OneVsRestClassifier, OneVsOneClassifier

data = load_wine()
X, y = data.data, data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)

# Scale data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# OvR
ovr = OneVsRestClassifier(SVC(kernel='rbf'))
ovr.fit(X_train_scaled, y_train)
ovr_acc = ovr.score(X_test_scaled, y_test)

# OvO
ovo = OneVsOneClassifier(SVC(kernel='rbf'))
ovo.fit(X_train_scaled, y_train)
ovo_acc = ovo.score(X_test_scaled, y_test)

print(f"OvR Accuracy: {ovr_acc:.4f}")
print(f"OvO Accuracy: {ovo_acc:.4f}")
```

```
OvR Accuracy: 0.9815
```

```
OvO Accuracy: 0.9815
```

1.17 37) Write a Python program to train an SVM Classifier using Linear, Polynomial, and RBF kernels on the Breast Cancer dataset and compare their accuracy

[34]:
```
data = load_breast_cancer()
X, y = data.data, data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)

# Scale data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

kernels = ['linear', 'poly', 'rbf']
for kernel in kernels:
    svm = SVC(kernel=kernel)
    svm.fit(X_train_scaled, y_train)
    acc = svm.score(X_test_scaled, y_test)
    print(f"{kernel} kernel accuracy: {acc:.4f}")
```

```
linear kernel accuracy: 0.9766
poly kernel accuracy: 0.8947
rbf kernel accuracy: 0.9766
```

1.18 38) Write a Python program to train an SVM Classifier using Stratified K-Fold Cross-Validation and compute the average accuracy

[36]:
```
from sklearn.model_selection import StratifiedKFold, cross_val_score

svm = SVC(kernel='rbf')
skf = StratifiedKFold(n_splits=5)
scores = cross_val_score(svm, X_train_scaled, y_train, cv=skf)

print(f"Average accuracy: {scores.mean():.4f} (±{scores.std():.4f})")
```

```
Average accuracy: 0.9673 (±0.0172)
```

1.19 39) Write a Python program to train a Naïve Bayes classifier using different prior probabilities and compare performance

[38]:
```
priors = [None, [0.8, 0.2], [0.5, 0.5], [0.2, 0.8]]
for prior in priors:
    gnb = GaussianNB(priors=prior)
    gnb.fit(X_train, y_train)
```

```
    acc = gnb.score(X_test, y_test)
    print(f"Accuracy with prior {prior}: {acc:.4f}")
```

```
Accuracy with prior None: 0.9415
Accuracy with prior [0.8, 0.2]: 0.9357
Accuracy with prior [0.5, 0.5]: 0.9415
Accuracy with prior [0.2, 0.8]: 0.9415
```

1.20 40) Write a Python program to perform Recursive Feature Elimination (RFE) before training an SVM Classifier and compare accuracy

[40]:
```
from sklearn.feature_selection import RFE

svm = SVC(kernel='linear')
rfe = RFE(estimator=svm, n_features_to_select=10)
X_rfe = rfe.fit_transform(X_train, y_train)

svm.fit(X_rfe, y_train)
X_test_rfe = rfe.transform(X_test)
acc_rfe = svm.score(X_test_rfe, y_test)

print(f"Accuracy with RFE: {acc_rfe:.4f}")
```

```
Accuracy with RFE: 0.9298
```

1.21 41) Write a Python program to train an SVM Classifier and evaluate its performance using Precision, Recall, and F1-Score instead of accuracy

[50]:
```
from sklearn.metrics import precision_score, recall_score, f1_score
# If you already have a fitted SelectKBest:
selector = SelectKBest(chi2, k=10).fit(X_train, y_train)
X_train_selected = selector.transform(X_train)
X_test_selected = selector.transform(X_test)  # CRITICAL: use same selector

svm = SVC().fit(X_train_selected, y_train)
y_pred = svm.predict(X_test_selected)
print(f"Precision: {precision_score(y_test, y_pred):.4f}")
print(f"Recall: {recall_score(y_test, y_pred):.4f}")
print(f"F1 Score: {f1_score(y_test, y_pred):.4f}")
```

```
Precision: 0.9224
Recall: 0.9907
F1 Score: 0.9554
```

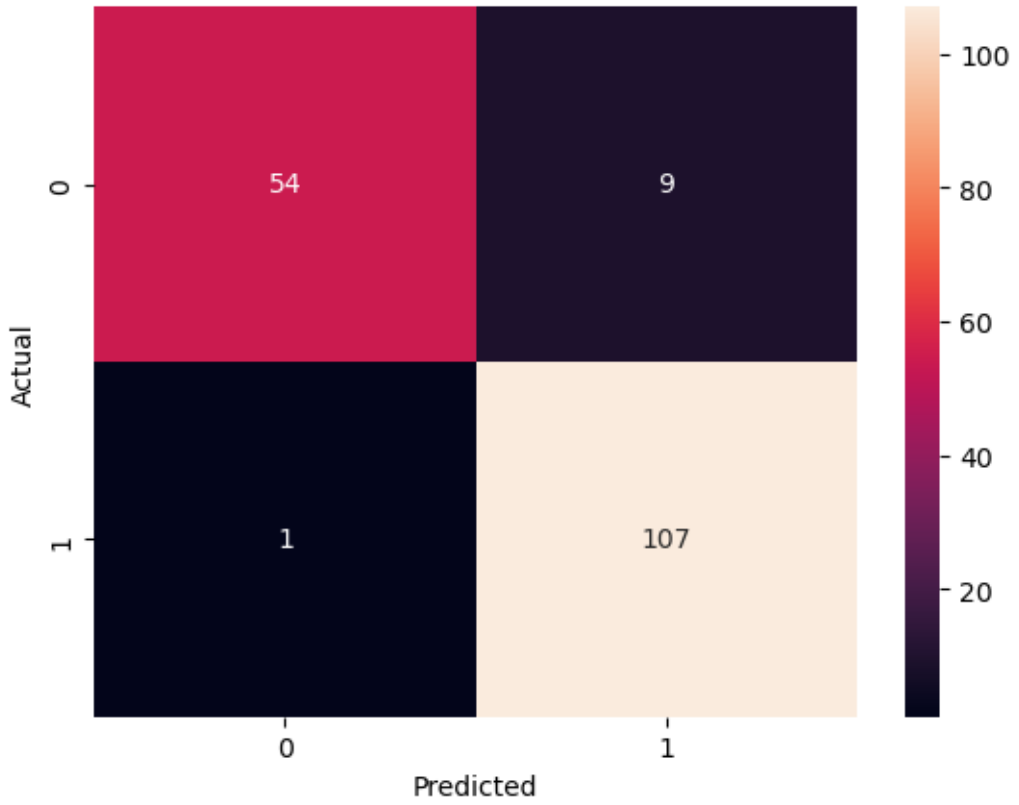
1.22 42) Write a Python program to train a Naïve Bayes Classifier and evaluate its performance using Log Loss (Cross-Entropy Loss)

[44]:
```
from sklearn.metrics import log_loss

y_pred_proba = gnb.predict_proba(X_test)
print(f"Log Loss: {log_loss(y_test, y_pred_proba):.4f}")
```

```
Log Loss: 0.5043
```

1.23 43) Write a Python program to train an SVM Classifier and visualize the Confusion Matrix using seaborn

[54]:
```
import seaborn as sns
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



1.24 44) Write a Python program to train an SVM Regressor (SVR) and evaluate its performance using Mean Absolute Error (MAE) instead of MSE

[64]:
```
from sklearn.svm import SVR
from sklearn.metrics import mean_absolute_error
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline

# Load California housing dataset
housing = fetch_california_housing()
X, y = housing.data, housing.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)


svr = SVR()
svr.fit(X_train, y_train)
y_pred = svr.predict(X_test)
print(f"MAE: {mean_absolute_error(y_test, y_pred):.4f}")
```

```
MAE: 0.8665
```

1.25 45) Write a Python program to train a Naïve Bayes classifier and evaluate its performance using the ROC-AUC score

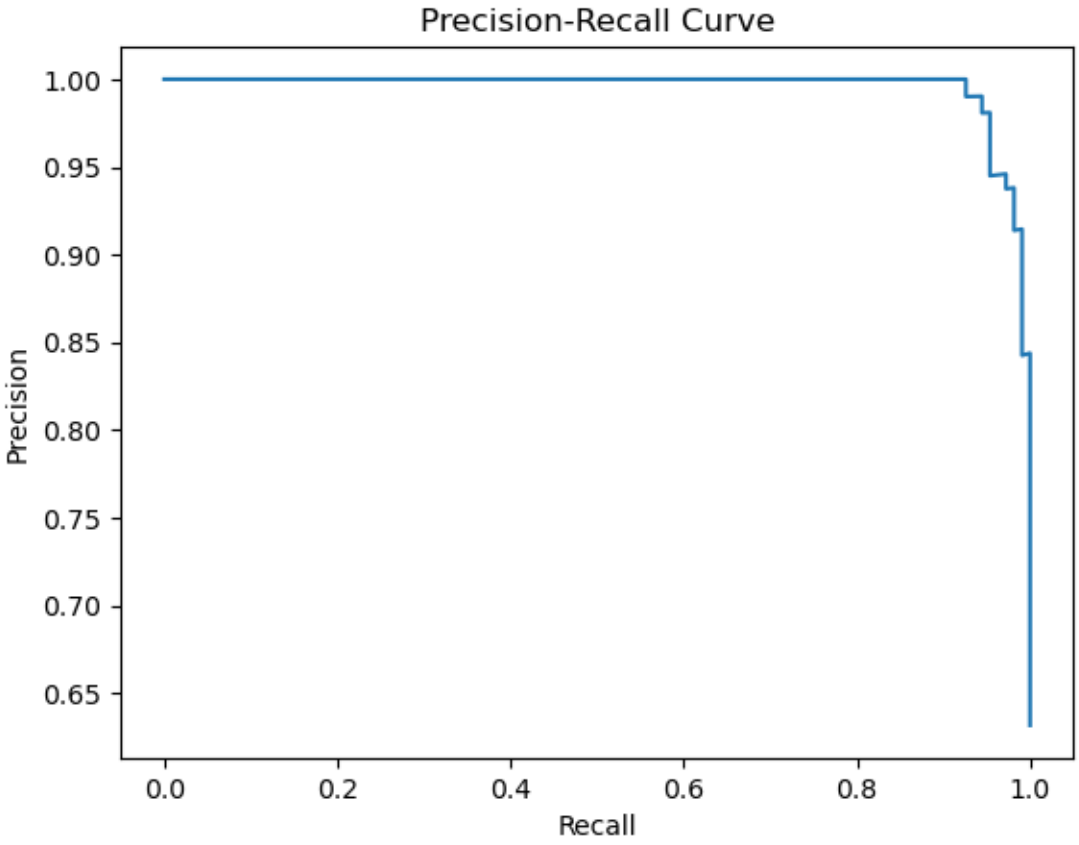
[58]:
```
from sklearn.metrics import roc_auc_score

y_score = gnb.predict_proba(X_test)[:, 1]
print(f"ROC-AUC: {roc_auc_score(y_test, y_score):.4f}")
```

```
ROC-AUC: 0.9922
```

1.26 46) Write a Python program to train an SVM Classifier and visualize the Precision-Recall Curve.

[62]:
```
from sklearn.metrics import precision_recall_curve
import matplotlib.pyplot as plt
# If you already have a fitted SelectKBest:
selector = SelectKBest(chi2, k=10).fit(X_train, y_train)
X_train_selected = selector.transform(X_train)
X_test_selected = selector.transform(X_test)  # CRITICAL: use same selector

svm = SVC().fit(X_train_selected, y_train)
```

```
y_pred = svm.predict(X_test_selected)
precision, recall, _ = precision_recall_curve(y_test, y_score)

plt.plot(recall, precision)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.show()
```



[ ]: