SVM & Naive Bayes Assignment Questions

Theoretical Questions

1. What is a Support Vector Machine (SVM)
- A Support Vector Machine is a supervised machine learning algorithm used for classification and regression tasks. It works by finding the optimal hyperplane that separates data points of different classes with the maximum margin.
1. What is the difference between Hard Margin and Soft Margin SVM
- **Hard Margin** SVM assumes that the data is linearly separable and tries to find a hyperplane that perfectly separates the classes without any misclassification.
- **Soft Margin** SVM allows for some misclassification (Softens) and introduces a penalty term to handle non-linearly separable data.
1. What is the mathematical intuition behind SVM
- The mathematical intuition behind Support Vector Machines (SVM) revolves around finding the **optimal decision boundary**—called a **hyperplane**—that best separates data points of different classes. Here's a breakdown of the key ideas:
  - Maximize the Margin: SVM seeks the hyperplane that maximizes the distance (margin) between itself and the nearest data points from each class—these are the support vectors.
  - Robustness: A larger margin implies better generalization to unseen data, reducing the risk of overfitting
- Geometric Perspective
  - In a 2D space, the hyperplane is a line; in 3D, it's a plane; in higher dimensions, it's a hyperplane.
  - The goal is to find the hyperplane that separates the classes with the widest possible gap.
- SVM solves the following convex optimization problem:
- Objective:
  - Minimize: `\frac{1}{2} \|\mathbf{w}\|^2`
  - subject to
    - `y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 \quad \text{for all } i`
  - Where:
  - \mathbf{w} is the weight vector (defines orientation of the hyperplane)
  - b is the bias term (defines position)
  - \mathbf{x}_i are the input vectors
  - y_i \in {-1, +1} are the class labels
1. What is the role of Lagrange Multipliers in SVM
- Lagrange Multipliers are used to solve the constrained optimization problem in SVM. They help convert the primal problem into a dual problem, which is easier to solve and allows the use of kernel functions for non-linear decision boundaries.
1. What are Support Vectors in SVM

- Support Vectors are the data points that lie closest to the decision boundary. They are critical in defining the position and orientation of the hyperplane.

1. What is a Support Vector Classifier (SVC)

- A Support Vector Classifier is an implementation of SVM used for classification tasks. It finds the optimal hyperplane that separates different classes in the feature space.

1. What is a Support Vector Regressor (SVR)

- Support Vector Regressor is an SVM variant used for regression tasks. It tries to fit the best line within a threshold margin and penalizes predictions that fall outside this margin.

1. What is the Kernel Trick in SVM

- The Kernel Trick allows SVM to operate in a high-dimensional space without explicitly computing the coordinates. It uses kernel functions to compute the inner products between data points in the transformed space.

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

Type	Traits
Linear Kernel	Suitable for linearly separable data, such as classifying emails based on word frequency.
Polynomial Kernel	Captures interactions between features using polynomial functions, useful for image recognition tasks.
RBF Kernel	Maps data into infinite-dimensional space and is effective for non-linear problems, like handwriting recognition

1. What is the effect of the C parameter in SVM

- The C parameter controls the trade-off between achieving a low error on the training data and maintaining a large margin. A small C allows for a wider margin with more misclassifications, while a large C tries to classify all training examples correctly. For example, in noisy datasets, a smaller C can help prevent overfitting.

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

- Gamma defines how far the influence of a single training example reaches. A low gamma means far reach, and a high gamma means close reach. It affects the shape of the decision boundary. For instance, a high gamma can lead to overfitting in a dataset with many features.

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

- Naïve Bayes is a probabilistic classifier based on Bayes' Theorem. It is called "Naïve" because it assumes that the features are conditionally independent given the class label. For example, in spam detection, it assumes that the presence of one word is independent of another.

1. What is Bayes' Theorem

- Bayes’ Theorem describes the probability of an event based on prior knowledge of conditions related to the event.

  `Formula: P(A|B) = [P(B|A) * P(A)] / P(B)`

For example, it can be used to update the probability of a disease given a positive test result.

1. Explain the differences between Gaussian Naïve Bayes, Multinomial Naïve Bayes, and Bernoulli Naïve Bayes

- Gaussian Naïve Bayes: Assumes features follow a normal distribution, suitable for continuous data like sensor readings.
- Multinomial Naïve Bayes: Suitable for discrete count data like word counts in text classification.
- Bernoulli Naïve Bayes: Works with binary/boolean features, such as presence or absence of words in a document.

1. When should you use Gaussian Naïve Bayes over other variants

- Use Gaussian Naïve Bayes when the features are continuous and approximately follow a normal distribution.
  – For example, it is ideal for classifying medical data like blood pressure and cholesterol levels.

1. What are the key assumptions made by Naïve Bayes

- Features are conditionally independent given the class label.
- All features contribute equally and independently to the outcome.

These assumptions simplify computation but may not hold in real-world data.

1. What are the advantages and disadvantages of Naïve Bayes

- Advantages:
  – Simple and fast
  – Works well with high-dimensional data
  – Effective for text classification, such as spam filtering
- Disadvantages:
  – Assumes feature independence
  – May not perform well with correlated features, like pixels in an image

1. Why is Naïve Bayes a good choice for text classification

- Naïve Bayes is effective for text classification because it handles high-dimensional data well, is computationally efficient, and performs well with sparse data.
  – For example, it is widely used in email spam detection and sentiment analysis.

1. Compare SVM and Naïve Bayes for classification tasks

- **SVM**: Effective for complex decision boundaries, robust to overfitting, but computationally intensive. Suitable for image classification.
- **Naïve Bayes**: Fast and simple, good for text data, but may struggle with correlated features. Ideal for document categorization.

1. How does Laplace Smoothing help in Naïve Bayes?

- Laplace Smoothing helps handle zero-frequency problems by adding a small constant (usually 1) to the count of each feature, ensuring that no probability is zero. This is particularly important when a feature in the test data was not observed in the training data, which would otherwise result in a zero probability and invalidate the entire prediction.

- For example, consider a spam classifier trained on emails that never contained the word "lottery." If a new email contains "lottery," the probability of spam would be zero without smoothing. Laplace Smoothing adjusts the probability to a small non-zero value, allowing the classifier to still make a reasonable prediction.

Practical Questions

```
# Imports of Libraries.
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC, SVR
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, precision_recall_curve, classification_report,
confusion_matrix, log_loss


warnings.filterwarnings('ignore')

# 21.  Write a Python program to train an SVM Classifier on the Iris
dataset and evaluate accuracy
from sklearn.datasets import load_iris

X,y = load_iris(return_X_y=True)

# Split into training and test sets (75% train, 25% test)

X_train, X_test, y_train, y_test = train_test_split(X,y,
test_size=0.25, random_state=42)

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

# Predict on test data
y_pred = svm_model.predict(X_test)

# Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of SVM classifier on Iris dataset: {accuracy:.2f}")
```

```
Accuracy of SVM classifier on Iris dataset: 1.00
```

```python
# 22. Write a Python program to train two SVM classifiers with Linear
and RBF kernels on the Wine dataset, then compare their accuracies
from sklearn.datasets import load_wine

X,y = load_wine(return_X_y=True)

# Split into training and test sets (75% train, 25% test)

X_train, X_test, y_train, y_test = train_test_split(X,y,
test_size=0.25, random_state=42)

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

# Predict on test data with linear kernel
y_pred_lin = svm_model_lin.predict(X_test)

# Train an SVM classifier using RBF
svm_model_rbf = SVC(kernel='rbf')
svm_model_rbf.fit(X_train, y_train)

# Predict on test data with rbf kernel
y_pred_rbf = svm_model_rbf.predict(X_test)


# Evaluate accuracy
accuracy_lin = accuracy_score(y_test, y_pred_lin)
print(f"Accuracy of SVM classifier with Linear Kernel on Wine Dataset:
{accuracy_lin:.2f}")

accuracy_rbf = accuracy_score(y_test, y_pred_rbf)
print(f"Accuracy of SVM classifier with RBF Kernel on Wine Dataset:
{accuracy_rbf:.2f}")
```

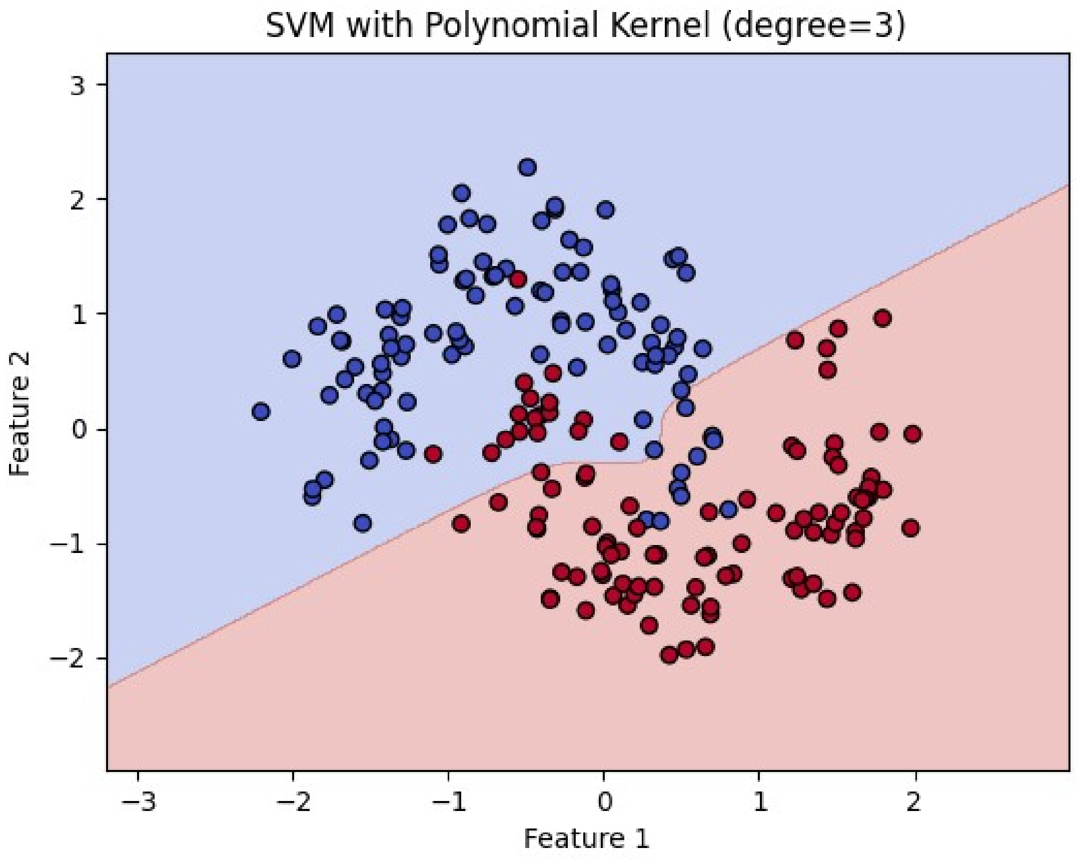
```
Accuracy of SVM classifier with Linear Kernel on Wine Dataset: 0.98
Accuracy of SVM classifier with RBF Kernel on Wine Dataset: 0.71
```

```python
# 23. Write a Python program to train an SVM Regressor (SVR) on a
housing dataset and evaluate it using Mean Squared Error (MSE)
from sklearn.datasets import fetch_california_housing
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler

# Load the California Housing dataset
housing = fetch_california_housing()
X = housing.data
y = housing.target
```

```
# Feature scaling (important for SVR)
scaler_X = StandardScaler()
scaler_y = StandardScaler()

X_scaled = scaler_X.fit_transform(X)
y_scaled = scaler_y.fit_transform(y.reshape(-1, 1)).ravel()

# Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled,
y_scaled, test_size=0.2, random_state=42)

# Train SVR model
svr_model = SVR(kernel='rbf')  # we can also try 'linear' or 'poly'
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) of SVR on housing dataset:
{mse:.4f}")
```

```
Mean Squared Error (MSE) of SVR on housing dataset: 0.2644
```

```
# 24. Write a Python program to train an SVM Classifier with a
Polynomial Kernel and visualize the decision boundary.
from sklearn.datasets import make_moons
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler

# Generate synthetic 2D dataset
X, y = make_moons(n_samples=200, noise=0.2, random_state=42)

# Feature scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train SVM with polynomial kernel
svm_poly = SVC(kernel='poly', degree=3, C=1)
svm_poly.fit(X_scaled, y)

# Function to plot decision boundary
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.linspace(x_min, x_max, 500),
                         np.linspace(y_min, y_max, 500))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.contourf(xx, yy, Z, alpha=0.3, cmap=plt.cm.coolwarm)
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm,
edgecolors='k')
    plt.title("SVM with Polynomial Kernel (degree=3)")
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
    plt.show()

# Visualize
plot_decision_boundary(svm_poly, X_scaled, y)
```



```
# 25. Write a Python program to train a Gaussian Naïve Bayes
classifier on the Breast Cancer dataset and evaluate accuracy.
from sklearn.datasets import load_breast_cancer
from sklearn.naive_bayes import GaussianNB

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

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

```python
# Train Gaussian Naïve Bayes classifier
gnb = GaussianNB()
gnb.fit(X_train, y_train)

# Predict and evaluate
y_pred = gnb.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of Gaussian Naïve Bayes on Breast Cancer dataset:
{accuracy:.2f}")
```

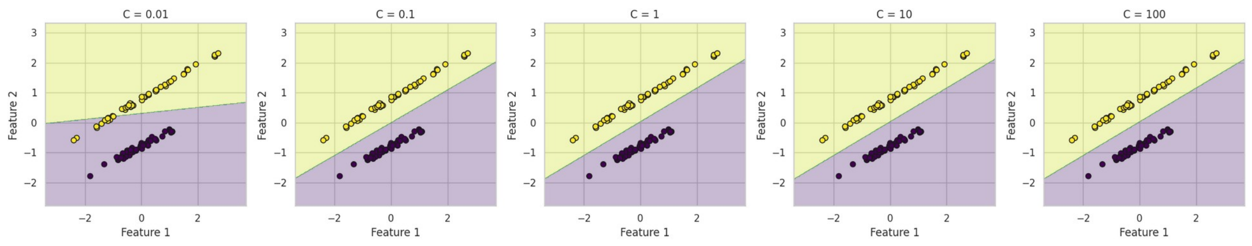
```
Accuracy of Gaussian Naïve Bayes on Breast Cancer dataset: 0.97
```

```python
# 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.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import TfidfVectorizer

# Load the 20 Newsgroups dataset (subset for speed)
categories = ['sci.space', 'rec.sport.baseball', 'comp.graphics',
'talk.politics.mideast']
newsgroups = fetch_20newsgroups(subset='all', categories=categories,
remove=('headers', 'footers', 'quotes'))

# Convert text to TF-IDF features
vectorizer = TfidfVectorizer(stop_words='english', max_features=5000)
X = vectorizer.fit_transform(newsgroups.data)
y = newsgroups.target

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

# Train Multinomial Naïve Bayes classifier
nb_model = MultinomialNB()
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"Accuracy of Multinomial Naïve Bayes on 20 Newsgroups:
{accuracy:.2f}")
```

```
Accuracy of Multinomial Naïve Bayes on 20 Newsgroups: 0.89
```

```python
# 27. Write a Python program to train an SVM Classifier with different
C values and compare the decision boundaries visually.

from sklearn import datasets
from sklearn.preprocessing import StandardScaler
```

```python
# Load synthetic 2D classification dataset
X, y = datasets.make_classification(n_features=2, n_redundant=0,
n_informative=2,
                                     n_clusters_per_class=1,
n_samples=100, random_state=42)

# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Define C values to test
C_values = [0.01, 0.1, 1, 10, 100]

# Set up plot
sns.set(style="whitegrid")
fig, axes = plt.subplots(1, len(C_values), figsize=(20, 4))

# Create mesh for decision boundary visualization
x_min, x_max = X_scaled[:, 0].min() - 1, X_scaled[:, 0].max() + 1
y_min, y_max = X_scaled[:, 1].min() - 1, X_scaled[:, 1].max() + 1
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 500),
np.linspace(y_min, y_max, 500))

for i, C in enumerate(C_values):
    clf = SVC(C=C, kernel='linear')
    clf.fit(X_scaled, y)
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    ax = axes[i]
    ax.contourf(xx, yy, Z, alpha=0.3, cmap='viridis')
    ax.scatter(X_scaled[:, 0], X_scaled[:, 1], c=y, cmap='viridis',
edgecolors='k')
    ax.set_title(f"C = {C}")
    ax.set_xlim(xx.min(), xx.max())
    ax.set_ylim(yy.min(), yy.max())
    ax.set_xlabel("Feature 1")
    ax.set_ylabel("Feature 2")

plt.tight_layout()
plt.savefig("svm_decision_boundaries.png")
plt.show()
```



```python
# 28. Write a Python program to train a Bernoulli Naïve Bayes
classifier for binary classification on a dataset with binary features
from sklearn.naive_bayes import BernoulliNB
from sklearn.datasets import make_classification

# Step 1: Generate synthetic binary feature dataset
X, y = make_classification(n_samples=1000, n_features=20,
n_informative=10,
                           n_redundant=0, n_classes=2,
random_state=42)

# Step 2: Binarize features (convert to 0/1)
X_binary = (X > 0).astype(int)

# Step 3: Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_binary, y,
test_size=0.3, random_state=42)

# Step 4: Train Bernoulli Naïve Bayes classifier
bnb = BernoulliNB()
bnb.fit(X_train, y_train)

# Step 5: Predict and evaluate
y_pred = bnb.predict(X_test)

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

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

print(f"\n Accuracy: {accuracy_score(y_test, y_pred):.4f}")
```

```
Confusion Matrix:
[[112  34]
 [ 46 108]]

 Classification Report:
              precision    recall  f1-score   support

           0       0.71      0.77      0.74       146
           1       0.76      0.70      0.73       154

    accuracy                           0.73       300
   macro avg       0.73      0.73      0.73       300
weighted avg       0.74      0.73      0.73       300


 Accuracy: 0.7333
```

```python
# 29. Write a Python program to apply feature scaling before training
an SVM model and compare results with unscaled data.

from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import StandardScaler

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

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# 1. SVM without scaling
clf_unscaled = SVC(kernel='rbf')
clf_unscaled.fit(X_train, y_train)
y_pred_unscaled = clf_unscaled.predict(X_test)
acc_unscaled = accuracy_score(y_test, y_pred_unscaled)

# 2. SVM with scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

clf_scaled = SVC(kernel='rbf')
clf_scaled.fit(X_train_scaled, y_train)
y_pred_scaled = clf_scaled.predict(X_test_scaled)
acc_scaled = accuracy_score(y_test, y_pred_scaled)

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

```
Accuracy without scaling: 0.9357
Accuracy with scaling:    0.9766
```

```python
# 30. Write a Python program to train a Gaussian Naïve Bayes model and
compare the predictions before and after Laplace Smoothing.
from sklearn.naive_bayes import GaussianNB

# Generate synthetic dataset
np.random.seed(42)
X = np.random.randn(1000, 5)
y = np.random.choice([0, 1], size=1000)

# Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)
```

```
# Train Gaussian Naive Bayes without Laplace smoothing
model_no_smoothing = GaussianNB()
model_no_smoothing.fit(X_train, y_train)
y_pred_no_smoothing = model_no_smoothing.predict(X_test)

# Train Gaussian Naive Bayes with Laplace smoothing (via
var_smoothing)
model_with_smoothing = GaussianNB(var_smoothing=1e-2)
model_with_smoothing.fit(X_train, y_train)
y_pred_with_smoothing = model_with_smoothing.predict(X_test)

# Evaluate both models
accuracy_no_smoothing = accuracy_score(y_test, y_pred_no_smoothing)
accuracy_with_smoothing = accuracy_score(y_test,
y_pred_with_smoothing)

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

print("\nClassification Report without Laplace Smoothing:")
print(classification_report(y_test, y_pred_no_smoothing))

print("\nClassification Report with Laplace Smoothing:")
print(classification_report(y_test, y_pred_with_smoothing))
```

```
Accuracy without Laplace Smoothing: 0.49
Accuracy with Laplace Smoothing: 0.48333333333333334

Classification Report without Laplace Smoothing:
              precision    recall  f1-score   support

           0       0.50      0.38      0.43       153
           1       0.48      0.61      0.54       147

    accuracy                           0.49       300
   macro avg       0.49      0.49      0.48       300
weighted avg       0.49      0.49      0.48       300


Classification Report with Laplace Smoothing:
              precision    recall  f1-score   support

           0       0.49      0.37      0.42       153
           1       0.48      0.61      0.53       147

    accuracy                           0.48       300
   macro avg       0.48      0.49      0.48       300
weighted avg       0.48      0.48      0.48       300
```

```python
# 31.  Write a Python program to train an SVM Classifier and use
GridSearchCV to tune the hyperparameters (C, gamma, kernel)
from sklearn.model_selection import GridSearchCV


# Load dataset (Iris for demonstration)
digits = datasets.load_digits()
X = digits.data
y = digits.target

# Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# Define parameter grid for GridSearchCV
param_grid = {
    'C': [0.1, 1, 10, 100],
    'gamma': [1, 0.1, 0.01, 0.001],
    'kernel': ['linear', 'rbf']
}

# Initialize SVM classifier
svc = SVC()

# Initialize GridSearchCV
grid_search = GridSearchCV(svc, param_grid, cv=5, verbose=1)

# Fit the model
grid_search.fit(X_train, y_train)

# Predict on test data
y_pred = grid_search.predict(X_test)

# Print results
print("Best Parameters:", grid_search.best_params_)
print("Best Cross-validation Score:", grid_search.best_score_)
print("Test Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test,
y_pred))
```

```
Fitting 5 folds for each of 32 candidates, totalling 160 fits
Best Parameters: {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
Best Cross-validation Score: 0.9920508442420793
Test Accuracy: 0.9907407407407407
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        53
           1       1.00      1.00      1.00        50
           2       1.00      1.00      1.00        47
           3       0.98      0.96      0.97        54
           4       1.00      1.00      1.00        60
           5       0.99      1.00      0.99        66
           6       1.00      1.00      1.00        53
           7       0.98      0.98      0.98        55
           8       0.98      1.00      0.99        43
           9       0.98      0.97      0.97        59

    accuracy                           0.99       540
   macro avg       0.99      0.99      0.99       540
weighted avg       0.99      0.99      0.99       540


#32. Write a Python program to train an SVM Classifier on an
imbalanced dataset and apply class weighting and check it improve
accuracy.
from sklearn.datasets import make_classification

# Generate an imbalanced dataset
X, y = make_classification(n_samples=1000, n_features=20,
n_informative=2,
                           n_redundant=10, n_clusters_per_class=1,
                           weights=[0.9, 0.1], flip_y=0,
random_state=42)

# Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# Train SVM without class weights
svm_no_weights = SVC(kernel='linear', class_weight=None)
svm_no_weights.fit(X_train, y_train)
y_pred_no_weights = svm_no_weights.predict(X_test)
print("Without Class Weights:\n")
print(classification_report(y_test, y_pred_no_weights))

# Train SVM with class weights
svm_with_weights = SVC(kernel='linear', class_weight='balanced')
svm_with_weights.fit(X_train, y_train)
y_pred_with_weights = svm_with_weights.predict(X_test)
print("With Class Weights:\n")
print(classification_report(y_test, y_pred_with_weights))

Without Class Weights:

              precision    recall  f1-score   support

           0       0.99      1.00      0.99       276
           1       0.95      0.83      0.89        24

    accuracy                           0.98       300
   macro avg       0.97      0.91      0.94       300
weighted avg       0.98      0.98      0.98       300

With Class Weights:

              precision    recall  f1-score   support

           0       0.99      0.98      0.98       276
           1       0.80      0.83      0.82        24

    accuracy                           0.97       300
   macro avg       0.89      0.91      0.90       300
weighted avg       0.97      0.97      0.97       300


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

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB, BernoulliNB, GaussianNB

# Load dataset directly from GitHub
df = pd.read_csv("https://raw.githubusercontent.com/dusamatej/Spam-
Email-Classification-using-Naive-Bayes/main/emails.csv")

# Split data
X_train, X_test, y_train, y_test = train_test_split(df['text'],
df['label'], test_size=0.2, random_state=42)

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

# Train Naïve Bayes classifier
model = MultinomialNB() # For Discrete counts (e.g., word frequencies)

model.fit(X_train_vec, y_train)

# Predict and evaluate
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       0.67      1.00      0.80         2
           1       0.00      0.00      0.00         1
    accuracy                           0.67         3
   macro avg       0.33      0.50      0.40         3
weighted avg       0.44      0.67      0.53         3
```

```
#34. Write a Python program to train an SVM Classifier and a Naïve
Bayes Classifier on the same dataset and compare their accuracy.
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB, BernoulliNB, GaussianNB

# Load dataset directly from GitHub
df = pd.read_csv("https://raw.githubusercontent.com/dusamatej/Spam-
Email-Classification-using-Naive-Bayes/main/emails.csv")

X,y = df['text'], df['label']

# Vectorize text
vectorizer = TfidfVectorizer(stop_words='english', max_features=5000)
X_vec = vectorizer.fit_transform(X)

# Split data
X_train, X_test, y_train, y_test = train_test_split(X_vec, y,
test_size=0.2, random_state=42)

# Train SVC
svcmodel = SVC(kernel='linear')
svcmodel.fit(X_train, y_train)
y_pred_svc = svcmodel.predict(X_test)
acc_svc = accuracy_score(y_test, y_pred_svc)

# Train Naïve Bayes classifier
model = MultinomialNB() # For Discrete counts (e.g., word frequencies)
model.fit(X_train, y_train)

# Predict and evaluate
y_pred = model.predict(X_test)
acc_NB = accuracy_score(y_test, y_pred)
#Evaluation Metrics

print("Accuracy for SVM Classifier(linear kernel):", acc_svc)
print("Accuracy for Naive Bayes Classifier:", acc_NB)

print("Classification_report: \n", classification_report(y_test,
y_pred_svc))
print("Classification_report: \n", classification_report(y_test,
y_pred))
```

```
Accuracy for SVM Classifier(linear kernel): 0.3333333333333333
Accuracy for Naive Bayes Classifier: 0.3333333333333333
Classification_report:
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         2
           1       0.33      1.00      0.50         1

    accuracy                           0.33         3
   macro avg       0.17      0.50      0.25         3
weighted avg       0.11      0.33      0.17         3

Classification_report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         2
           1       0.33      1.00      0.50         1

    accuracy                           0.33         3
   macro avg       0.17      0.50      0.25         3
weighted avg       0.11      0.33      0.17         3
```

```python
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.naive_bayes import MultinomialNB

# Step 1: Load binary subset of 20 Newsgroups
categories = ['sci.space', 'rec.sport.hockey']
data = fetch_20newsgroups(subset='all', categories=categories,
remove=('headers', 'footers', 'quotes'))

X_raw = data.data
y = data.target  # 0 = sci.space, 1 = rec.sport.hockey

# Step 2: TF-IDF Vectorization
vectorizer = TfidfVectorizer(stop_words='english', max_features=5000)
X = vectorizer.fit_transform(X_raw)

# Step 3: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=42)

# Step 4: Naïve Bayes WITHOUT feature selection
nb_full = MultinomialNB()
nb_full.fit(X_train, y_train)
pred_full = nb_full.predict(X_test)
acc_full = accuracy_score(y_test, pred_full)

# Step 5: Feature Selection (Chi-Squared)
selector = SelectKBest(chi2, k=1000)
X_train_sel = selector.fit_transform(X_train, y_train)
X_test_sel = selector.transform(X_test)
```

```
# Step 6: Naïve Bayes WITH feature selection
nb_sel = MultinomialNB()
nb_sel.fit(X_train_sel, y_train)
pred_sel = nb_sel.predict(X_test_sel)
acc_sel = accuracy_score(y_test, pred_sel)

# Step 7: Compare Results
print(f"Accuracy without feature selection: {acc_full:.4f}")
print(f"Accuracy with feature selection (top 1000): {acc_sel:.4f}")
```

```
Accuracy without feature selection: 0.9537
Accuracy with feature selection (top 1000): 0.9477
```

```
# 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.
from sklearn.datasets import load_wine
from sklearn.preprocessing import StandardScaler
from sklearn.multiclass import OneVsRestClassifier, OneVsOneClassifier

# Step 1: Load Wine dataset
data = load_wine()
X = data.data
y = data.target  # 3 classes: 0, 1, 2

# Step 2: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Step 3: Feature Scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Step 4: Train SVM with One-vs-Rest (OvR)
svm_ovr = OneVsRestClassifier(SVC(kernel='linear'))
svm_ovr.fit(X_train, y_train)
pred_ovr = svm_ovr.predict(X_test)
acc_ovr = accuracy_score(y_test, pred_ovr)

# Step 5: Train SVM with One-vs-One (OvO)
svm_ovo = OneVsOneClassifier(SVC(kernel='linear'))
svm_ovo.fit(X_train, y_train)
pred_ovo = svm_ovo.predict(X_test)
acc_ovo = accuracy_score(y_test, pred_ovo)

# Step 6: Compare Results
print(f"Accuracy using One-vs-Rest (OvR): {acc_ovr:.4f}")
print(f"Accuracy using One-vs-One (OvO): {acc_ovo:.4f}")
```

```
Accuracy using One-vs-Rest (OvR): 1.0000
Accuracy using One-vs-One (OvO): 0.9722
```

```python
# 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.

from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import StandardScaler

# Step 1: Load Breast Cancer dataset
data = load_breast_cancer()
X = data.data
y = data.target  # 0 = malignant, 1 = benign

# Step 2: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Step 3: Feature Scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Step 4: Train SVM with Linear Kernel
svm_linear = SVC(kernel='linear')
svm_linear.fit(X_train, y_train)
pred_linear = svm_linear.predict(X_test)
acc_linear = accuracy_score(y_test, pred_linear)

# Step 5: Train SVM with Polynomial Kernel
svm_poly = SVC(kernel='poly', degree=3)
svm_poly.fit(X_train, y_train)
pred_poly = svm_poly.predict(X_test)
acc_poly = accuracy_score(y_test, pred_poly)

# Step 6: Train SVM with RBF Kernel
svm_rbf = SVC(kernel='rbf')
svm_rbf.fit(X_train, y_train)
pred_rbf = svm_rbf.predict(X_test)
acc_rbf = accuracy_score(y_test, pred_rbf)

# Step 7: Compare Results
print(f"Accuracy with Linear Kernel:     {acc_linear:.4f}")
print(f"Accuracy with Polynomial Kernel: {acc_poly:.4f}")
print(f"Accuracy with RBF Kernel:        {acc_rbf:.4f}")
```

```
Accuracy with Linear Kernel:     0.9561
Accuracy with Polynomial Kernel: 0.8684
Accuracy with RBF Kernel:        0.9825
```

```python
# 38. Write a Python program to train an SVM Classifier using
Stratified K-Fold Cross-Validation and compute the average accuracy.

from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import StandardScaler

# Step 1: Load Breast Cancer dataset
data = load_breast_cancer()
X = data.data
y = data.target

# Step 2: Initialize Stratified K-Fold
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Step 3: Track accuracy across folds
accuracies = []

for train_index, test_index in skf.split(X, y):
    # Split data
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Scale features
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # Train SVM
    model = SVC(kernel='linear')
    model.fit(X_train, y_train)
    preds = model.predict(X_test)

    # Compute accuracy
    acc = accuracy_score(y_test, preds)
    accuracies.append(acc)

# Step 4: Report results
print("Fold Accuracies:", [f"{a:.4f}" for a in accuracies])
print(f"Average Accuracy: {np.mean(accuracies):.4f}")
```

```
Fold Accuracies: ['0.9912', '0.9386', '0.9561', '0.9912', '0.9912']
Average Accuracy: 0.9737
```

```python
# 39. Write a Python program to train a Naïve Bayes classifier using
different prior probabilities and compare performance.

from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

# Step 1: Load Wine dataset
data = load_wine()
X = data.data
y = data.target  # Classes: 0, 1, 2

# Step 2: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# Step 3: Feature Scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Step 4: Train Naïve Bayes with default priors
nb_default = GaussianNB()
nb_default.fit(X_train, y_train)
pred_default = nb_default.predict(X_test)
acc_default = accuracy_score(y_test, pred_default)

# Step 5: Train Naïve Bayes with custom priors
custom_priors = [0.2, 0.5, 0.3]  # Must sum to 1
nb_custom = GaussianNB(priors=custom_priors)
nb_custom.fit(X_train, y_train)
pred_custom = nb_custom.predict(X_test)
acc_custom = accuracy_score(y_test, pred_custom)

# Step 6: Compare Results
print(f"Accuracy with default priors: {acc_default:.4f}")
print(f"Accuracy with custom priors {custom_priors}:
{acc_custom:.4f}")
print("\nGaussianNB uses class priors to compute posterior
probabilities via Bayes’ theorem.")
print("\nIf the dataset is balanced, changing priors may not affect
accuracy.")
print("\nOn imbalanced data, custom priors can shift decision
boundaries.")
```

```
Accuracy with default priors: 1.0000
Accuracy with custom priors [0.2, 0.5, 0.3]: 1.0000

GaussianNB uses class priors to compute posterior probabilities via
Bayes’ theorem.

If the dataset is balanced, changing priors may not affect accuracy.

On imbalanced data, custom priors can shift decision boundaries.
```

```python
# 40. Write a Python program to perform Recursive Feature Elimination
(RFE) before training an SVM Classifier and compare accuracy.
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.feature_selection import RFE
from sklearn.metrics import accuracy_score

# Step 1: Load dataset
data = load_breast_cancer()
X = data.data
y = data.target

# Step 2: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# Step 3: Feature Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Step 4: Train SVM without RFE
svm_full = SVC(kernel='linear', random_state=42)
svm_full.fit(X_train_scaled, y_train)
pred_full = svm_full.predict(X_test_scaled)
acc_full = accuracy_score(y_test, pred_full)

# Step 5: Apply RFE with SVM as estimator
rfe = RFE(estimator=SVC(kernel='linear'), n_features_to_select=10)
rfe.fit(X_train_scaled, y_train)

# Step 6: Train SVM on selected features
X_train_rfe = rfe.transform(X_train_scaled)
X_test_rfe = rfe.transform(X_test_scaled)

svm_rfe = SVC(kernel='linear', random_state=42)
svm_rfe.fit(X_train_rfe, y_train)
pred_rfe = svm_rfe.predict(X_test_rfe)
acc_rfe = accuracy_score(y_test, pred_rfe)

# Step 7: Compare Results
print(f"Accuracy with all features: {acc_full:.4f}")
print(f"Accuracy with RFE-selected features (10): {acc_rfe:.4f}")
```

```
Accuracy with all features: 0.9766
Accuracy with RFE-selected features (10): 0.9649
```

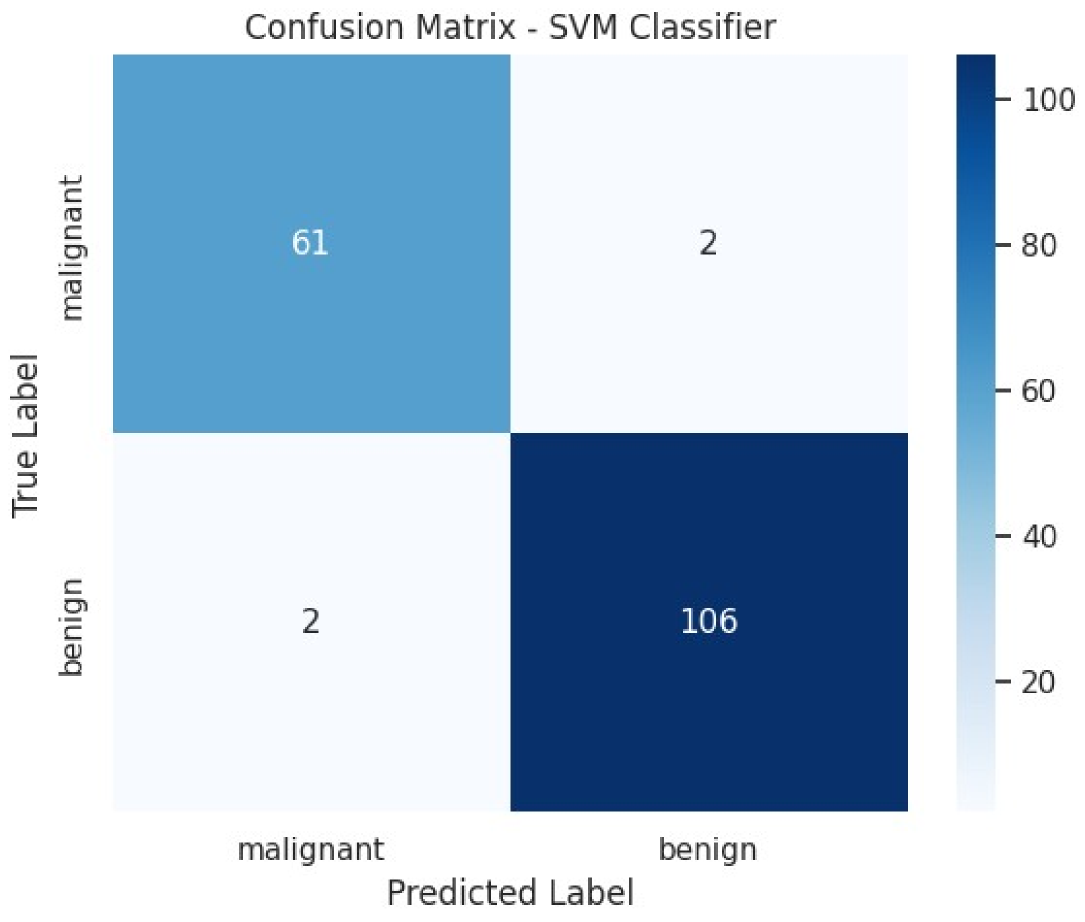
```python
# 41. Write a Python program to train an SVM Classifier and evaluate
its performance using Precision, Recall, and F1-Score instead of
accuracy.

from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import precision_score, recall_score, f1_score,
classification_report

# Step 1: Load dataset
data = load_breast_cancer()
X = data.data
y = data.target

# Step 2: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# Step 3: Feature Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Step 4: Train SVM Classifier
svm = SVC(kernel='linear', random_state=42)
svm.fit(X_train_scaled, y_train)
y_pred = svm.predict(X_test_scaled)

# Step 5: Evaluate using Precision, Recall, F1
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Step 6: Display Results
print(f"Precision: {precision:.4f}")
print(f"Recall:    {recall:.4f}")
print(f"F1-Score:  {f1:.4f}")

# Optional: Full classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred,
target_names=data.target_names))
```

```
Precision: 0.9815
Recall:    0.9815
F1-Score:  0.9815

Classification Report:
```

```
              precision    recall  f1-score   support

   malignant       0.97      0.97      0.97        63
      benign       0.98      0.98      0.98       108

    accuracy                           0.98       171
   macro avg       0.97      0.97      0.97       171
weighted avg       0.98      0.98      0.98       171


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

from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import log_loss

# Step 1: Load dataset
data = load_breast_cancer()
X = data.data
y = data.target

# Step 2: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# Step 3: Feature Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Step 4: Train Naïve Bayes Classifier
nb = GaussianNB()
nb.fit(X_train_scaled, y_train)

# Step 5: Predict Probabilities
y_proba = nb.predict_proba(X_test_scaled)

# Step 6: Evaluate using Log Loss
loss = log_loss(y_test, y_proba)

# Step 7: Display Result
print(f"Log Loss (Cross-Entropy): {loss:.4f}")

# Notes:
# - Log Loss penalizes confident wrong predictions more than uncertain
ones.
# - Lower log loss = better calibrated probabilities.
# - predict_proba() is essential — it returns class probabilities, not
labels.
```

```
Log Loss (Cross-Entropy): 0.4545
```

```
# 43. Write a Python program to train an SVM Classifier and visualize
the Confusion Matrix using seaborn.
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Step 1: Load dataset
data = load_breast_cancer()
X = data.data
y = data.target
class_names = data.target_names  # ['malignant', 'benign']

# Step 2: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# Step 3: Feature Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Step 4: Train SVM Classifier
svm = SVC(kernel='linear', random_state=42)
svm.fit(X_train_scaled, y_train)
y_pred = svm.predict(X_test_scaled)

# Step 5: Compute Confusion Matrix
cm = confusion_matrix(y_test, y_pred)

# Step 6: Visualize using seaborn
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names, yticklabels=class_names)
plt.title("Confusion Matrix - SVM Classifier")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.tight_layout()
plt.show()
```



```
# 44. Write a Python program to train an SVM Regressor (SVR) and
evaluate its performance using Mean Absolute Error (MAE) instead of
MSE.
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR
from sklearn.metrics import mean_absolute_error

# Step 1: Load dataset
data = fetch_california_housing()
X = data.data
y = data.target

# Step 2: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# Step 3: Feature Scaling
scaler_X = StandardScaler()
scaler_y = StandardScaler()
```

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

# Optional: Scale target for SVR performance
y_train_scaled = scaler_y.fit_transform(y_train.reshape(-1,
1)).ravel()
y_test_scaled = scaler_y.transform(y_test.reshape(-1, 1)).ravel()

# Step 4: Train SVR
svr = SVR(kernel='rbf', C=10, epsilon=0.1)
svr.fit(X_train_scaled, y_train_scaled)

# Step 5: Predict and inverse scale
y_pred_scaled = svr.predict(X_test_scaled)
y_pred = scaler_y.inverse_transform(y_pred_scaled.reshape(-1,
1)).ravel()

# Step 6: Evaluate using MAE
mae = mean_absolute_error(y_test, y_pred)

# Step 7: Display Result
print(f"Mean Absolute Error (MAE): {mae:.4f}")
```

```
Mean Absolute Error (MAE): 0.3770
```

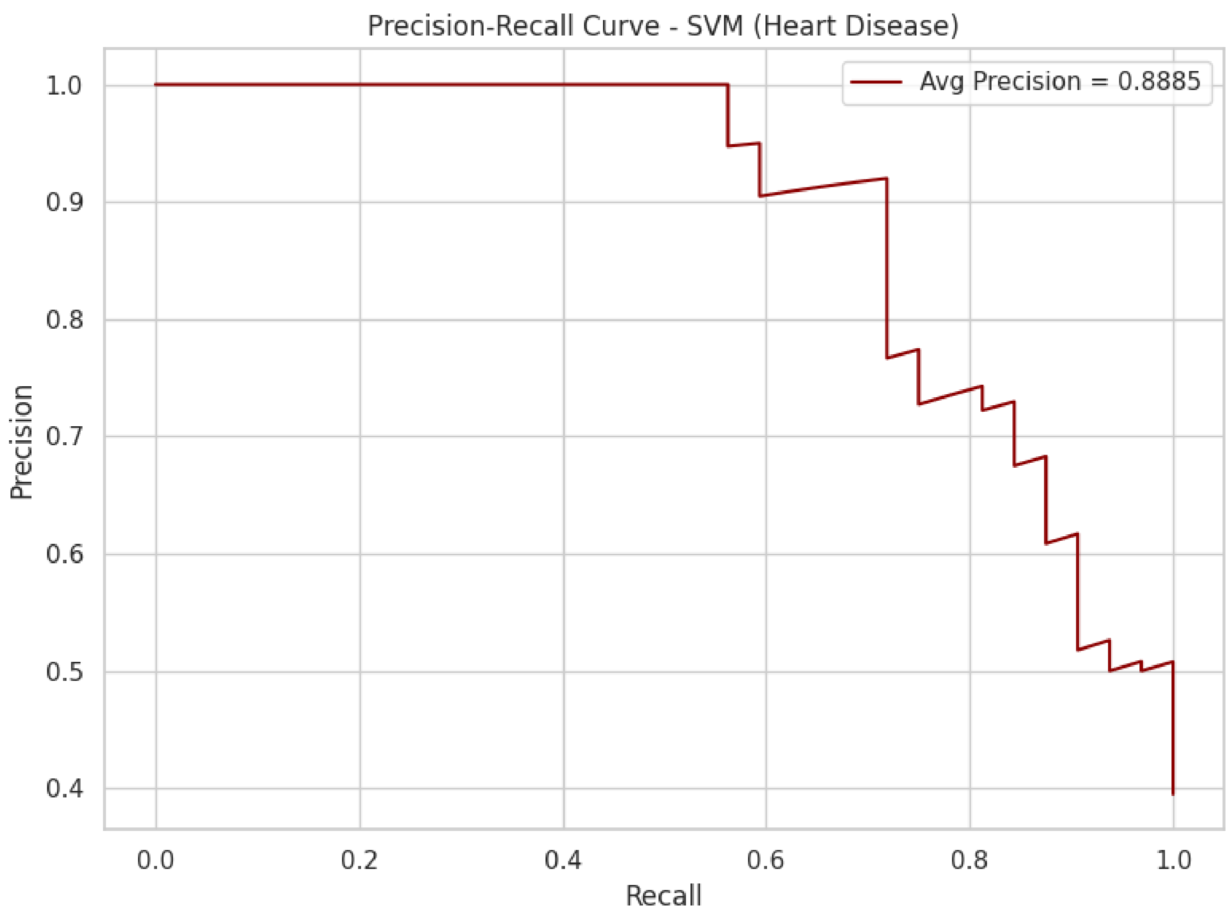
```
# 45. Write a Python program to train a Naïve Bayes classifier and
evaluate its performance using the ROC-AUC score.

from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import roc_auc_score

# Step 1: Load dataset
data = load_breast_cancer()
X = data.data
y = data.target

# Step 2: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# Step 3: Feature Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Step 4: Train Naïve Bayes Classifier
nb = GaussianNB()
```

```
nb.fit(X_train_scaled, y_train)

# Step 5: Predict Probabilities for ROC-AUC
y_proba = nb.predict_proba(X_test_scaled)[:, 1]  # Probability of
class 1

# Step 6: Evaluate using ROC-AUC
roc_auc = roc_auc_score(y_test, y_proba)

# Step 7: Display Result
print(f"ROC-AUC Score: {roc_auc:.4f}")
```

```
ROC-AUC Score: 0.9927
```

```
from enum import auto
# 46. Write a Python program to train an SVM Classifier and visualize
the Precision-Recall Curve.
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import precision_recall_curve,
average_precision_score

# Step 1: Load Heart Disease dataset from OpenML
heart = fetch_openml(name='heart', version=1, as_frame=False)
X = heart.data
y = heart.target.astype(int)  # Convert target to integer (0 or 1)

# Step 2: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# Step 3: Feature Scaling
scaler = StandardScaler(with_mean=False)
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Step 4: Train SVM with probability estimates
svm = SVC(kernel='rbf', probability=True, random_state=42)
svm.fit(X_train_scaled, y_train)

# Step 5: Predict probabilities
y_scores = svm.predict_proba(X_test_scaled)[:, 1]

# Step 6: Compute Precision-Recall values
precision, recall, thresholds = precision_recall_curve(y_test,
y_scores)
avg_precision = average_precision_score(y_test, y_scores)
```

```
# Step 7: Plot Precision-Recall Curve
plt.figure(figsize=(8, 6))
plt.plot(recall, precision, label=f'Avg Precision =
{avg_precision:.4f}', color='darkred')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve - SVM (Heart Disease)')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# 1. Trade-off Between Precision and Recall
# - Precision: Of all predicted positives, how many are truly
positive?
# - Recall: Of all actual positives, how many did we correctly
identify?
# - The curve shows how these metrics change as the decision threshold
varies.
# - A steep curve near the top-right indicates high precision and
recall — ideal!

# 2. Average Precision Score
# - This is the area under the PR curve (similar to ROC-AUC but for
PR).
```


Precision-Recall Curve - SVM (Heart Disease)
Avg Precision = 0.8885
Precision
Recall
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.0
0.2
0.4
0.6
0.8
1.0