An in-depth guide to support vector machines (SVM)

Introduction to Support Vector Machines (SVM)

One kind of supervised learning technique that is commonly used for classification and regression problems is Support Vector Machines (SVM). SVM stands out for its ability to handle datasets that are linearly and non-linearly separable using specialised functions called kernels. SVM's primary objective is to identify the best hyperplane that divides classes by the largest margin. This margin is computed as the distance between the hyperplane and the nearest data points, or support vectors ***(Cortes & Vapnik, 1995***).

Why Use SVM?

SVM is a popular choice for several reasons:

1. **Effective in High Dimensions:** When there are a lot of features compared to samples, SVM works well. This makes it especially useful in text classification, image recognition, and bioinformatics tasks where data often has a high dimensionality ***(Schölkopf & Smola, 2002).***
2. **Versatile Kernels:** SVM can effectively use kernel functions to handle non-linearly separable data by converting it into a space with more dimensions. This adaptability allows it to solve a wide range of problems ***(Boser et al., 1992).***
3. **Robustness:** In SVM, only a subset of training data known as support vectors affects the decision boundary. The model's resilience to noise and outliers is guaranteed by these support vectors ***(Cortes & Vapnik, 1995).***
4. **Clear Mathematical Foundation:** SVM is grounded in optimization theory, ensuring that the solution obtained is mathematically sound and globally optimal (at least for the linear case) ***(Vapnik, 1998)***.

Applications of SVM:

1. **Text Classification:** SVM is commonly used for spam email detection, sentiment analysis, and document categorization ***(Hastie et al., 2009)***.
2. **Image Classification:** In computer vision, SVM is employed for tasks like face detection and handwritten digit recognition ***(Schölkopf & Smola, 2002)***.
3. **Bioinformatics:** SVM helps classify proteins, predict diseases, and analyze gene expression data ***(Platt, 1999).***

The Role of Kernels in SVM

Kernels are the core of SVM's versatility. A kernel function transforms input data into a space with greater dimensions where each class can be efficiently split by a linear hyperplane ***(Schölkopf & Smola, 2002)***. The kernels allow SVM to tackle non-linear classification problems efficiently.

Understanding Kernels

In a higher-dimensional space, a kernel calculates the dot product between two points without figuring out the transformation directly. This procedure, referred to as the kernel technique, drastically lowers the processing cost ***(Boser et al., 1992).***

Types of Kernels:

1. **Linear Kernel:**
    - **Equation:** K(x,y)=xTy$K(x,y)=xTy$
    - Directly maps the data into the original feature space.
    - Fits data that is linearly separable.
    - Example: Classifying emails as spam or not based on word frequency ***(Hastie et al., 2009).***
2. **Polynomial Kernel:**
    - **Equation:** K(x,y)=(xTy+c)d$K(x,y)=(xTy+c)d$
    - Projects data into a polynomial feature space.
    - Controlled by the degree parameter (dd), which defines the polynomial's complexity.
    - Example: Classifying patterns with moderate non-linearity, such as distinguishing handwriting styles ***(Schölkopf & Smola, 2002).***
3. **Radial Basis Function (RBF) Kernel:**
    - **Equation:** K(x,y)=exp⁡(−γ||x−y||2)$K(x,y)=exp(−γ||x−y||2)$
    - Maps data into an infinite-dimensional space.
    - Controlled by the gamma parameter (γ$γ$), which determines the influence of individual data points.
    - Example: Detecting faces in images ***(Platt, 1999).***
4. **Sigmoid Kernel:**
    - **Equation:** K(x,y)=tanh⁡(αxTy+c)$K(x,y)=tanh(αxTy+c)$

- Mimics the activation function in neural networks.
- Less commonly used due to its tendency to behave like RBF under certain conditions ***(Hastie et al., 2009)***.

Dataset Overview

The dataset used in this tutorial is the Iris dataset, a benchmark dataset for classification tasks ***(UCI Machine Learning Repository, n.d.).*** It is simple yet versatile, making it ideal for learning machine learning techniques like SVM.

Dataset Characteristics:

1. **Number of Samples:** 150
2. **Classifications:** Three species of Iris flowers:
    - Setosa
    - Versicolor
    - Virginica
3. **Characteristics:**
    - Sepal length
    - Sepal width
    - Petal length
    - Petal width
4. **Objective Variable:** Species of the Iris flower.

Objective:

Task is to classify the Iris samples into one of the three species based on the given features.

```
# Load the Iris dataset
iris = load_iris()
X = pd.DataFrame(iris.data, columns=iris.feature_names)
y = pd.Series(iris.target, name="Species")

# Dataset Overview
print("Dataset Overview:")
print(X.describe())
```

```
Dataset Overview:
       sepal length (cm)  sepal width (cm)  petal length (cm)  \
count         150.000000        150.000000         150.000000
mean            5.843333          3.057333           3.758000
std             0.828066          0.435866           1.765298
min             4.300000          2.000000           1.000000
25%             5.100000          2.800000           1.600000
50%             5.800000          3.000000           4.350000
75%             6.400000          3.300000           5.100000
max             7.900000          4.400000           6.900000

       petal width (cm)
count        150.000000
mean           1.199333
std            0.762238
min            0.100000
25%            0.300000
50%            1.300000
75%            1.800000
max            2.500000
```

Data Preprocessing

An integral part of any machine learning pipeline is data preprocessing. It is particularly crucial for SVM since the method is sensitive to the size of the input characteristics.

Why is Preprocessing Necessary?

1. **Feature Scaling:** Without scaling, features with large ranges can dominate the optimization process, skewing the results. Standardization ensures all features contribute equally to the model ***(Hastie et al., 2009).***
2. **Data Splitting:** To evaluate the model's performance, the dataset is divided into training (70%) and testing (30%) subsets ***(UCI Machine Learning Repository, n.d.).***

Exploring Different Kernels

Objective:

Compare the performance of SVM models using Linear, Polynomial, and RBF kernels.

Hyperparameter Tuning:

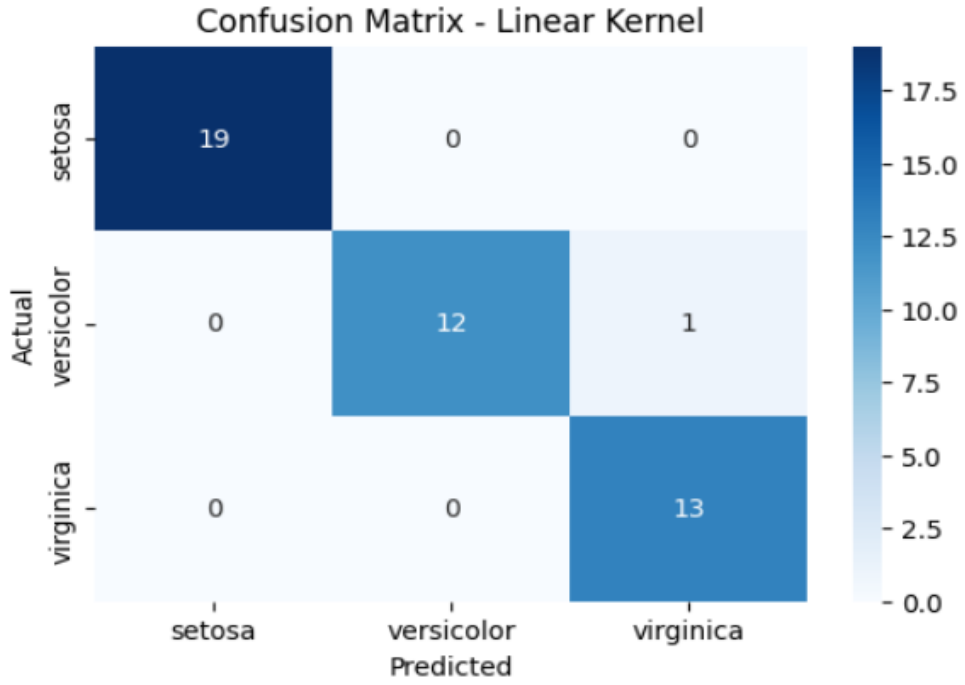
1. **C (Regularization Parameter):**
    - Strikes a balance between minimising classification error and optimising margin.
    - A high C*C* value results in a complex decision boundary, potentially overfitting.
    - A low C*C* value simplifies the decision boundary, possibly underfitting ***(Platt, 1999).***
2. **Degree:**
    - Only works with polynomial kernels.
    - Determines the complexity of the polynomial function.
3. **Gamma:**
    - Applicable for RBF and Polynomial Kernels.
    - A high γ*γ* value focuses on close neighbors, leading to overfitting.
    - A low γ*γ* value captures global patterns, resulting in underfitting ***(Hastie et al., 2009).***

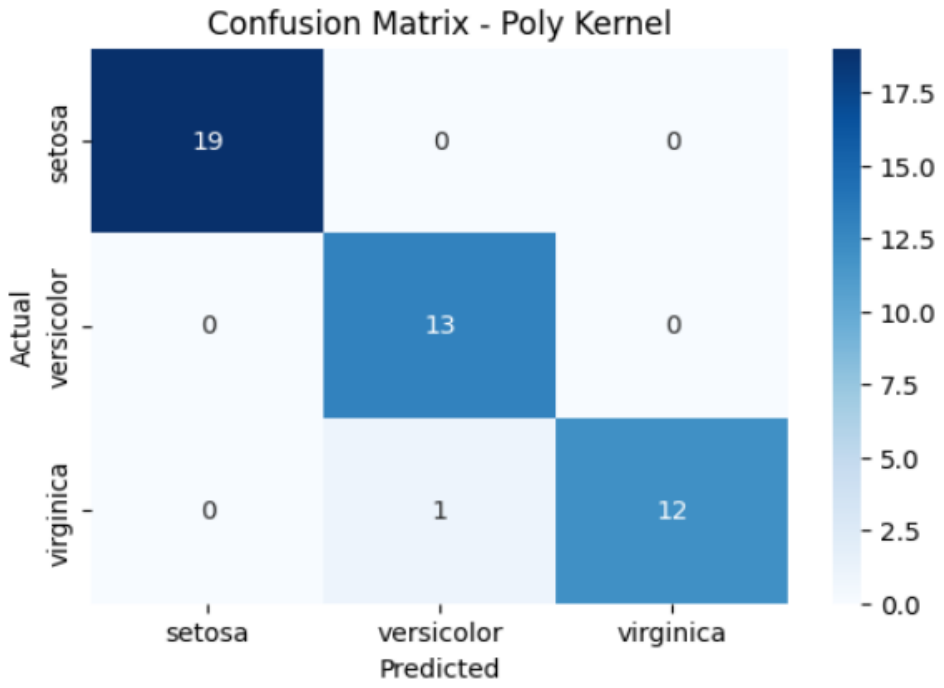
```
kernels = ['linear', 'poly', 'rbf']
results = []

for kernel in kernels:
    # Constructing the parameter grid based on the kernel
    param_grid = {'C': [0.1, 1, 10]}  # 'C' is applicable to all kernels

    if kernel == 'poly':
        param_grid['degree'] = [2, 3, 4]  # Only applicable to polynomial kernel
        param_grid['gamma'] = ['scale', 'auto']  # Also applicable to poly
    elif kernel == 'rbf':
        param_grid['gamma'] = ['scale', 'auto']  # Only applicable to RBF kernel

    # Running GridSearchCV
    grid_search = GridSearchCV(SVC(kernel=kernel), param_grid, cv=3)
    grid_search.fit(X_train, y_train)
    best_model = grid_search.best_estimator_
    y_pred = best_model.predict(X_test)

    # Collecting results
    accuracy = accuracy_score(y_test, y_pred)
    report = classification_report(y_test, y_pred, output_dict=True)
    cm = confusion_matrix(y_test, y_pred)
    results.append({
        "kernel": kernel,
        "best_params": grid_search.best_params_,
        "accuracy": accuracy,
        "report": report,
        "confusion_matrix": cm
    })
```

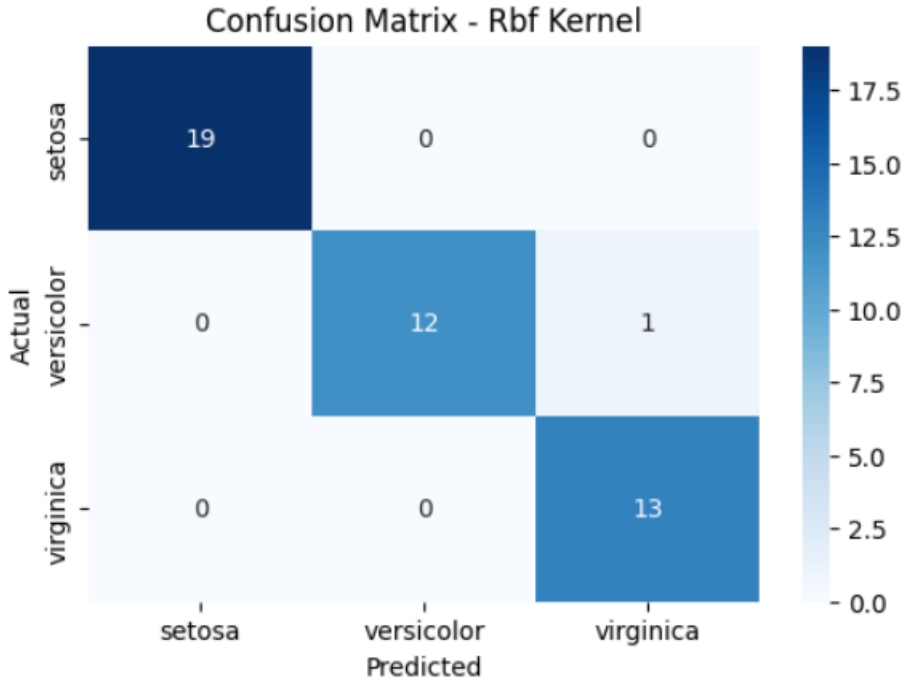
Performance Visualization

Confusion matrices reveal information about the model's performance. They display the quantity of misclassifications (off the diagonal) and accurate predictions (on the diagonal).

```
for result in results:
    kernel = result["kernel"]
    cm = result["confusion_matrix"]

    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=iris.target_names, yticklabels=iris.target_names)
    plt.title(f'Confusion Matrix - {kernel.capitalize()} Kernel')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()
```







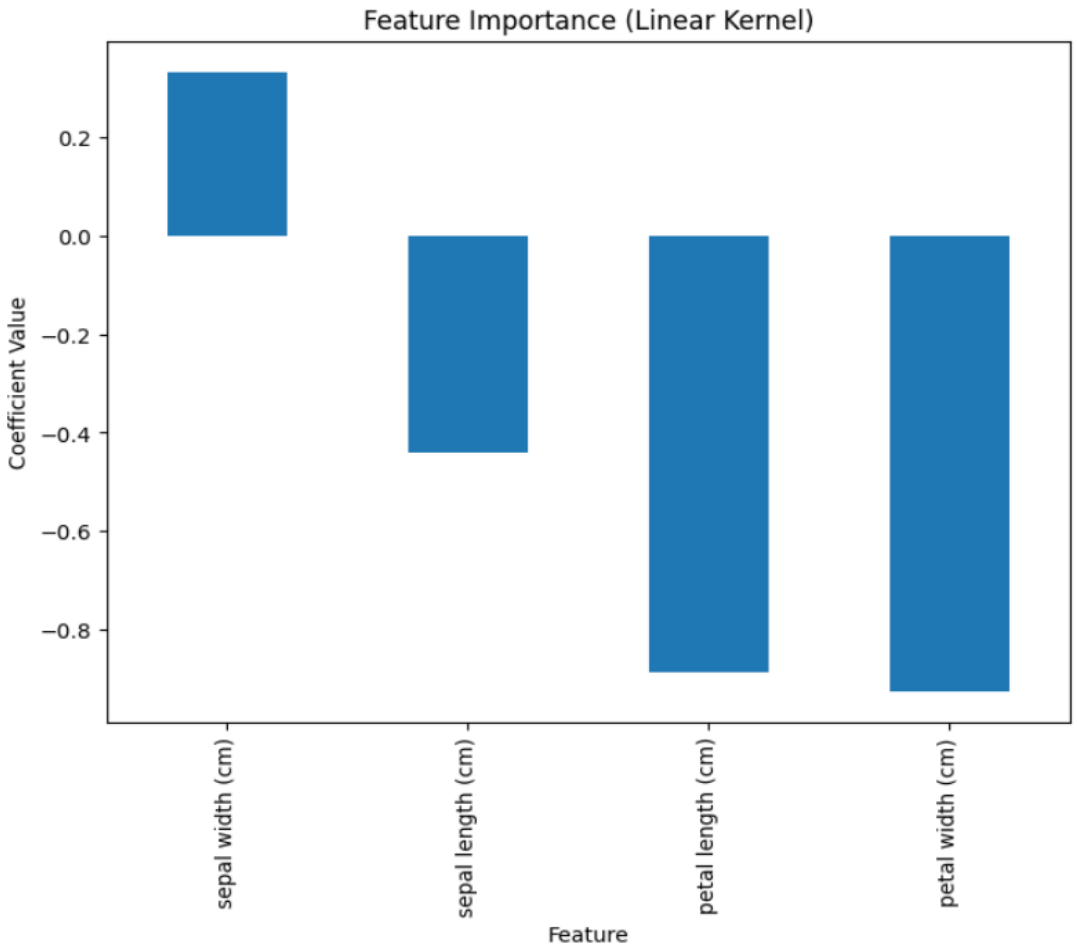
Feature Importance for Linear Kernel

For linear SVM, feature importance can be determined using the coefficients of the hyperplane.

Insights:

1. Features with higher coefficients contribute more to the decision boundary.
2. For the Iris dataset, **petal length** and **petal width** are more influential than sepal features.

[28]:
```
linear_model = SVC(kernel='linear', C=1).fit(X_train, y_train)
feature_importance = pd.Series(data=linear_model.coef_[0], index=iris.feature_names)
feature_importance.sort_values(ascending=False, inplace=True)

# Plotting feature importance
plt.figure(figsize=(8, 6))
feature_importance.plot(kind='bar')
plt.title("Feature Importance (Linear Kernel)")
plt.ylabel("Coefficient Value")
plt.xlabel("Feature")
plt.show()
```



Support Vectors and Their Visualization

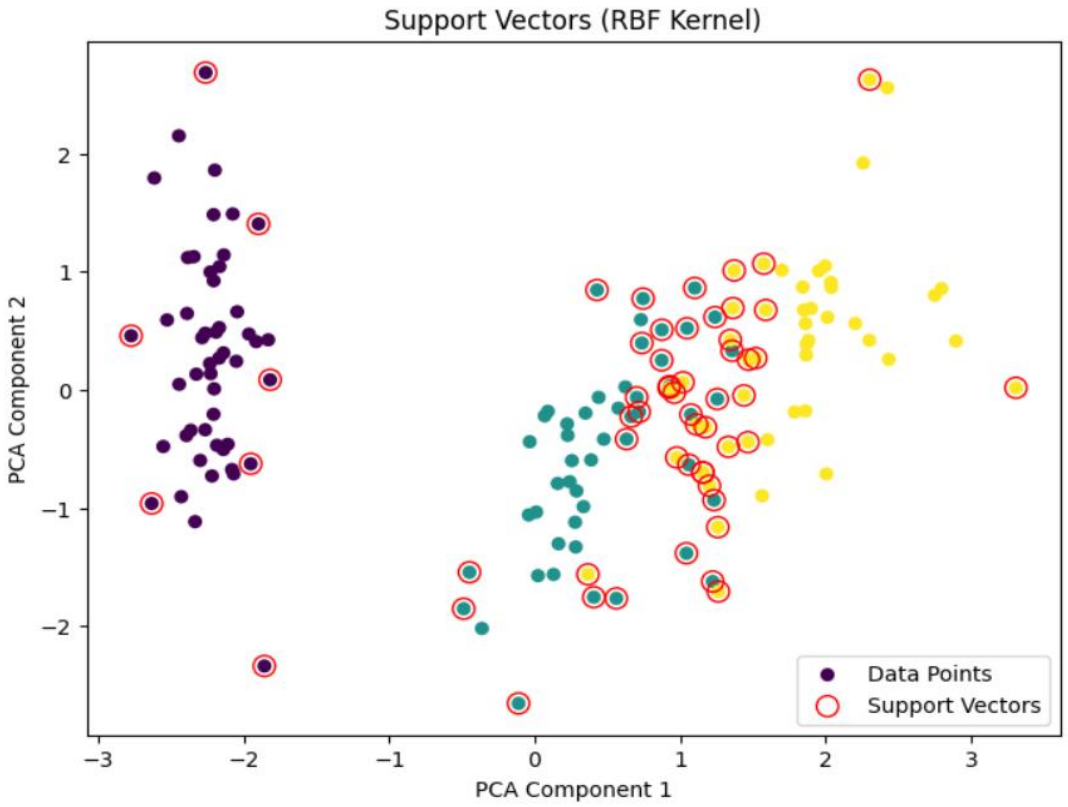
Support Vectors: What Are They?

Support vectors are the data points that are nearest to the decision border. They establish the hyperplane's position and orientation.

Visualization:

1. **Dimensionality Reduction:** PCA reduces the dataset to two dimensions for visualization.
2. **Scatter Plot:** Focusses attention to support vectors and illustrates how they affect the decision boundary.

[29]:
```
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

svm_rbf = SVC(kernel='rbf', C=1).fit(X_pca, y)

plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='viridis', s=30, label='Data Points')
plt.scatter(svm_rbf.support_vectors_[:, 0], svm_rbf.support_vectors_[:, 1],
            s=100, facecolors='none', edgecolors='r', label='Support Vectors')
plt.title("Support Vectors (RBF Kernel)")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.legend()
plt.show()
```



Sensitivity Analysis for Hyperparameters

Objective:

To study how C and γ influence the performance of the RBF kernel.

Methodology:

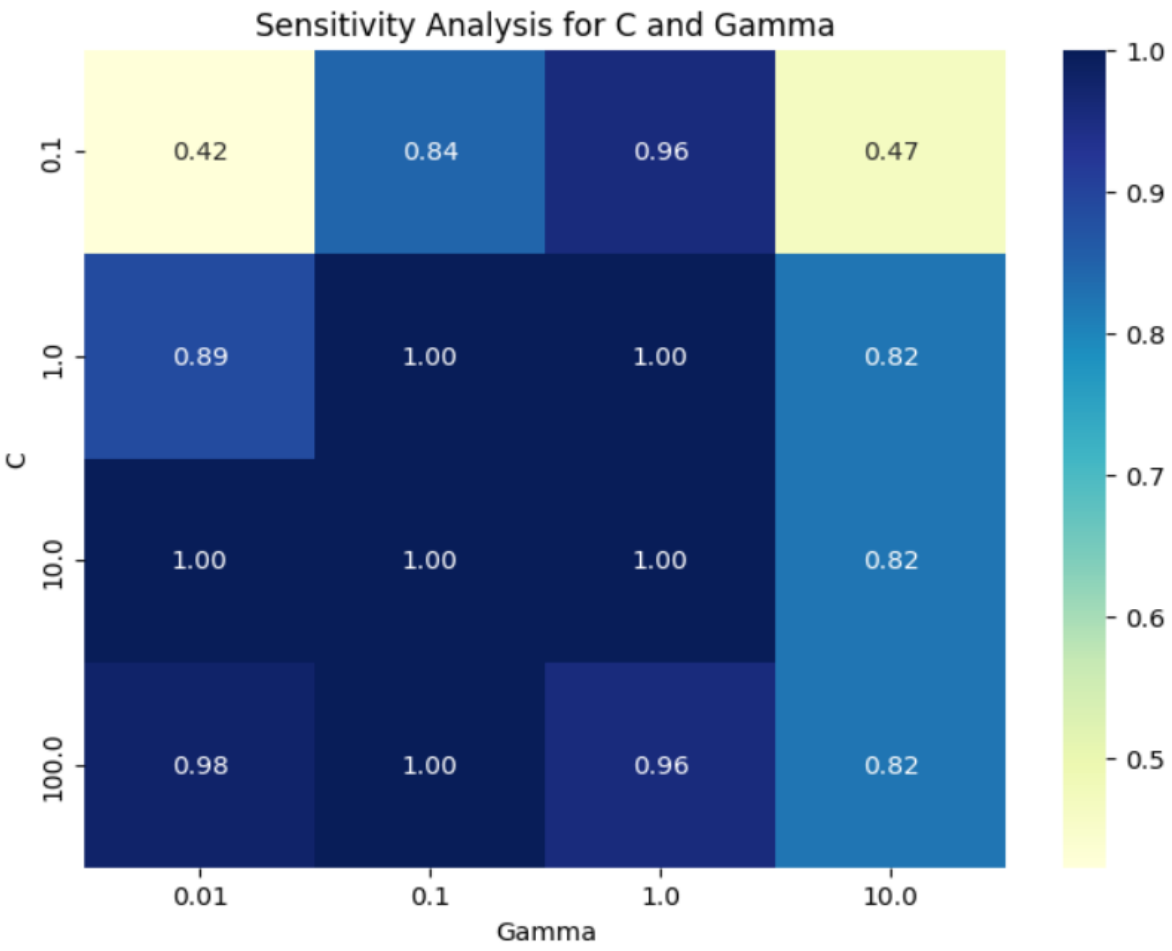
1. Evaluate the model for various combinations of C and γ.
2. Use a heatmap to visualize the accuracy.

Insights:

1. **High C:**
    - Overfits the data, leading to a complex decision boundary.
2. **Low C:**
    - Simplifies the decision boundary but risks underfitting.
3. **High γ:**
    - Focuses on individual data points, potentially overfitting.

4. **Low γγ:**
    - Captures broader patterns, resulting in smoother decision boundaries.

```
C_values = [0.1, 1, 10, 100]
gamma_values = [0.01, 0.1, 1, 10]
results_sensitivity = []

for C in C_values:
    for gamma in gamma_values:
        svm_model = SVC(kernel='rbf', C=C, gamma=gamma)
        svm_model.fit(X_train, y_train)
        y_pred = svm_model.predict(X_test)
        acc = accuracy_score(y_test, y_pred)
        results_sensitivity.append({"C": C, "Gamma": gamma, "Accuracy": acc})

# Convert results to a DataFrame
sensitivity_df = pd.DataFrame(results_sensitivity)

# Create a pivot table for heatmap visualization
pivot_table = sensitivity_df.pivot_table(index="C", columns="Gamma", values="Accuracy")

# Plotting the heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(pivot_table, annot=True, fmt=".2f", cmap="YlGnBu")
plt.title("Sensitivity Analysis for C and Gamma")
plt.xlabel("Gamma")
plt.ylabel("C")
plt.show()
```



Key Insights

1. **Kernel Comparison:**
    - Linear kernels are efficient for linearly separable data.
    - Polynomial and RBF kernels are better suited for non-linear data.
    - RBF kernels often outperform others due to their flexibility.
2. **Feature Importance:**
    - Petal features play a dominant role in classifying Iris flowers.
3. **Hyperparameter Sensitivity:**
    - Proper tuning of C and γ is essential for achieving optimal performance.
4. **Support Vectors:**
    - Only a subset of data influences the decision boundary, making SVM efficient.

Conclusion

By understanding core elements like kernels, support vectors, and hyperparameter tuning, SVM can be applied effectively to diverse datasets. This tutorial emphasizes the significance of preprocessing, kernel selection, and model evaluation, empowering users to harness SVM's potential ***(Cortes & Vapnik, 1995; Schölkopf & Smola, 2002).***

References

1. Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning, 20*(3), 273-297.
2. Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, 144-152.
3. Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Science & Business Media.
4. Schölkopf, B., & Smola, A. J. (2002). *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press.
5. Platt, J. C. (1999). Fast training of support vector machines using sequential minimal optimization. *Advances in Kernel Methods - Support Vector Learning*, MIT Press.
6. UCI Machine Learning Repository. Iris Dataset. Retrieved from https://archive.ics.uci.edu/ml/datasets/iris.