Student ID: 23068839
Github link: https://github.com/abhaybhatnate30/Machine_learning_Assingment

MACHINE LEARNING TUTORIAL: UNDERSTANDING SUPPORT VECTOR MACHINES (SVM) WITH DIFFERENT KERNELS

`z

Contents

`z

Introduction

Support Vector Machines (SVM) is one of the strongest and most general-purpose machine learning algorithms that can solve both classification and regression problems efficiently. The most interesting aspect about SVM is that it can build optimal decision planes by maximum-margin separation of various classes of data. This tutorial is specifically designed to describe how various kernel functions affect the performance and output of SVM models. I aim to give a full understanding of how to choose the kernel so that readers are in a position to make sound choices when applying SVM to their own machine learning tasks.

The tutorial will adopt a structured format of first covering the theory behind SVM and kernels and then implementing a real example on the Iris dataset. I will cover how each of the kernels differs in terms of accuracy values as well as decision boundary plotting. I will also lightly touch on practical kernel considerations in selecting between the kernels in practice, from hyperparameter considerations to model interpretability. At the end of this tutorial, the reader should be able to grasp how to utilize SVM's versatility to deal with real-world issues efficiently.

Understanding SVM and Kernels

The Fundamentals of SVM

Support Vector Machines are founded on the idea of determining the hyperplane most favorable to separate points of different classes within a high-dimensional space. The principle in SVM is to maximize the margin, i.e., the hyperplane to the nearest points in each category, i.e., support vectors. This has the effect that the model will generalize to new data because it takes into account the most difficult points on the decision boundary. SVM works optimally when data is not linearly separable since it employs the use of kernel functions in its design. Kernel functions enable the algorithm to map data into a high-dimensional space in which data becomes linearly separable.

The Role of Kernel Functions

Kernel functions form the foundation on which SVM can be utilized to use for data that has non-linear relationships. They achieve this by mapping input features implicitly to a higher-dimensional space without explicitly computing the coordinates of the data in the space. This allows SVM to construct non-linear decision boundaries that would not be possible with a linear model. Popular amongst the most popular kernels are the linear kernel, polynomial kernel, and radial basis function (RBF) kernel.

Each of these kernels possesses unique characteristics that allow it to perform optimally with varying amounts of data and instance issues.

The linear kernel is the most basic, which calculates the dot product of input features. It performs optimally for linearly separable data and is most often applied for its efficiency of computation and ease of interpretation. The simplicity does have its drawbacks, however, when working with more involved, non-linear relationships. The polynomial kernel adds non- linearity via the dot product of input features to some power, determined by the degree parameter. This kernel can accommodate a large number of types of relationships but has to be carefully tuned so that it does not overfit. The RBF kernel, or the Gaussian kernel, maps data using an exponential function to an infinite-dimensional space. This kernel is very flexible and can accommodate complex patterns and thus is a common default for all uses.

Experiment Setup

Dataset Selection and Preprocessing

For the purposes of this tutorial, we chose the Iris dataset, which is a popular machine learning benchmark. The dataset consists of 150 samples of iris flowers each characterized by four features: sepal length, sepal width, petal length, and petal width. Three species of the flowers exist: setosa, versicolor, and virginica. Iris dataset is most appropriate to demonstrate here as it contains linearly separable and non-linearly separable classes so that we can compare the performance of various kernels in varying situations.

Before training the SVM models, we preprocessed the data to ensure optimal performance. The dataset was split into training and testing sets, with 80% of the samples used for training and 20% reserved for evaluation. Feature scaling was applied to standardize the range of the input features, as SVM is sensitive to the scale of the data. Standardization was performed by subtracting the mean and dividing by the standard deviation for each feature, ensuring that all features contributed equally to the model's decision-making process.

`z

Methodology and Evaluation

We used three SVM classifiers, each of which was trained on a distinct kernel: linear, polynomial of degree 3, and RBF. We coded by employing the scikit-learn library package in Python, which provides an easy implementation of SVM as well as of most other machine learning algorithms. To remain simple, we employed default hyperparameters for all kernels but the polynomial one, where we even explicitly provided the degree of 3 while training.

In order to compare the precision of each model, we had both calculated accuracy measures for test data as well as training data. Accuracy is measured as the proportion of samples correctly predicted and is a simple measure to quantify upon which one can compare models. We also plotted the decision boundaries of each kernel in an attempt to understand how they cut the feature space. These visualizations were made by overlaying the models' predictions on a feature value grid, demarcating the areas allotted to each class.

Practical Implications

Choosing the Right Kernel for Your Data

The right choice of kernel is an important aspect of developing a quality SVM model. Linear kernel is most suitable for data where the classes are separable by a linear line or hyperplane. It is also most suited if interpretability is a necessity as the model obtained at the output is very easily interpretable and explainable. Polynomial kernel is most suited for data that is moderately non-linear, but caution has to be taken in tuning the degree parameter so as not to overfit. The RBF kernel is the most general and generally the default for unspecified data distributions. It is so general that it can be applied to a variety of patterns but at the expense of requiring adjustment of the gamma parameter, which determines the weight contribution of each training sample.
more influential, thus boundaries are smoother but with a high gamma value, points near are given more influence, with more intricate boundaries. Cross-validation is also a prime method for determining good hyperparameters.

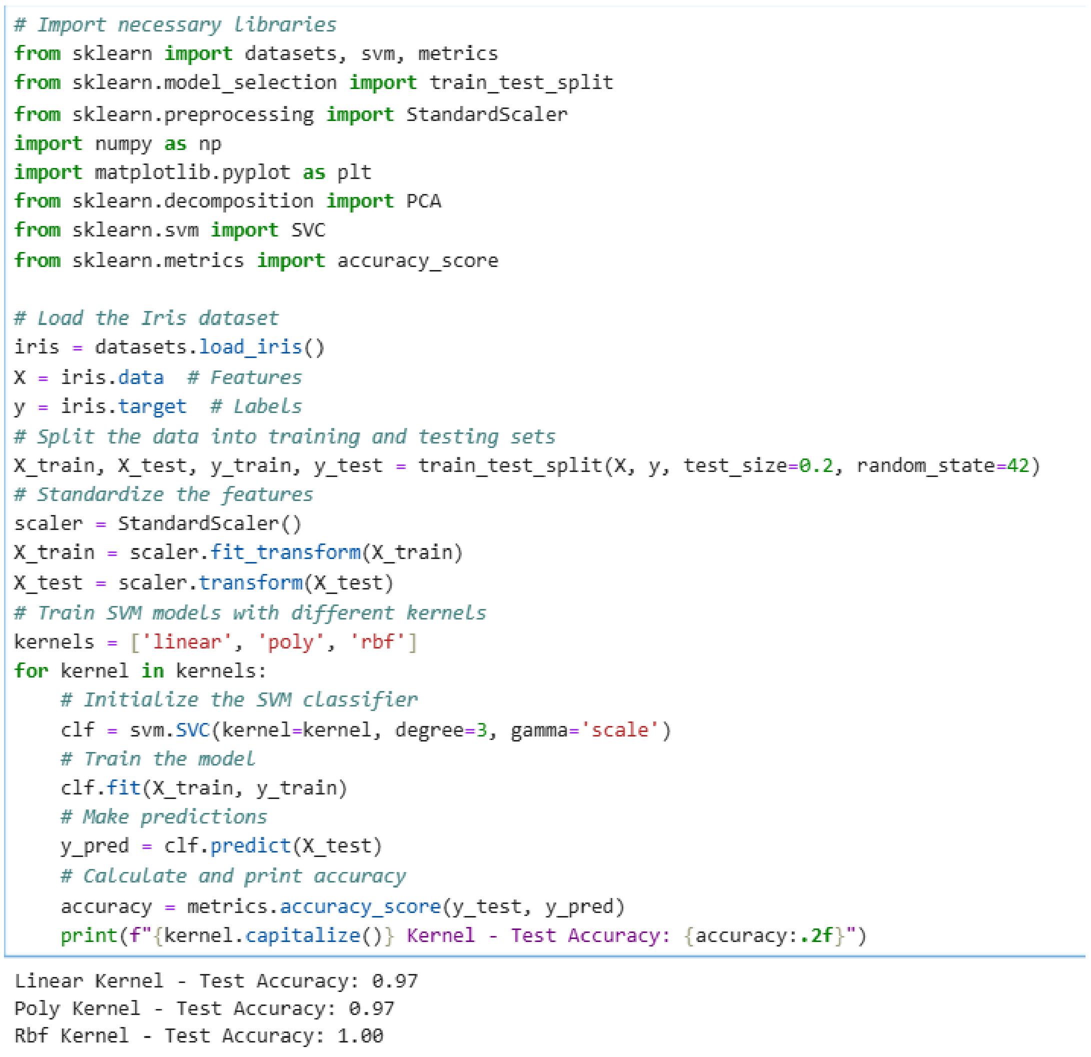
Hyperparameter Tuning and Model Optimization

Apart from kernel choice, hyperparameter tuning is the key to model optimization for SVM performance. The regularization parameter (C) controls the trade-off between maximizing the margin and minimizing classification error. A small value of C is prioritizing a larger margin but is open to more misclassifications, whereas a large value of C is aiming for less error but can have a smaller margin. The gamma parameter in the RBF kernel controls how far the effect of an individual training example extends. The low gamma value brings points distant to be more influential, thus boundaries are smoother but with a high gamma value, points near are given more influence, with more intricate boundaries. Cross-validation is also a prime method for determining good hyperparameters. Try out numerous combinations of values and compute their performance to determine the optimal configuration to apply to your dataset.

Code Demonstration

Below is a detailed explanation of the Python code used in this tutorial. The code demonstrates how to load the Iris dataset, preprocess the data, train SVM models with different kernels, and evaluate their performance.

This code starts by importing the necessary libraries, including the dataset module in scikit- learn to load the Iris dataset, svm to employ SVM, and metrics to calculate the model's performance. Training data and test data are created with the data, and features are standardized for fixed scaling. Three SVM models are trained, each for a different kernel, and their accuracy scores are then printed.



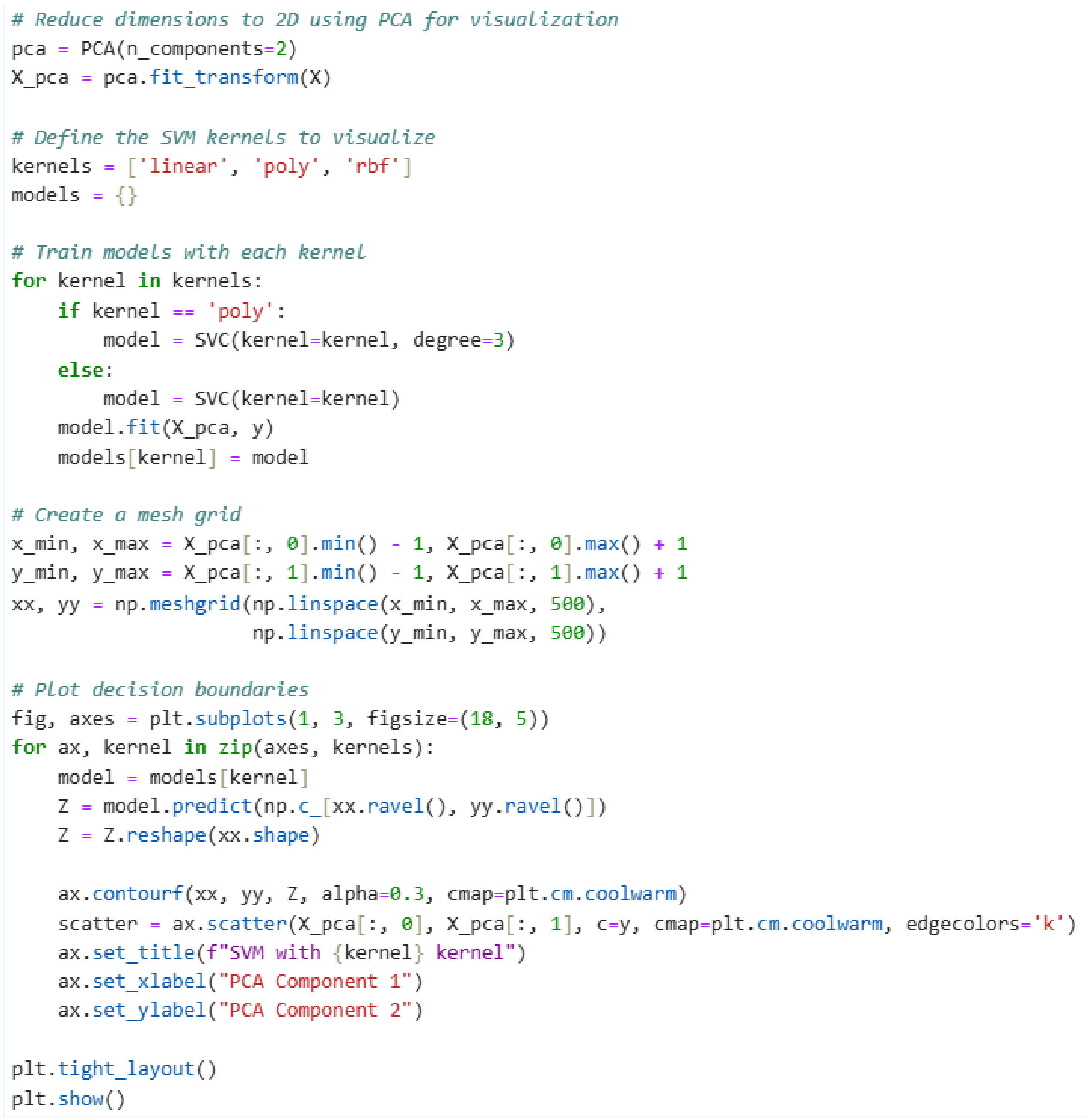
```python
# Import necessary libraries
from sklearn import datasets, svm, metrics
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris = datasets.load_iris()
X = iris.data  # Features
y = iris.target  # Labels
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
# Train SVM models with different kernels
kernels = ['linear', 'poly', 'rbf']
for kernel in kernels:
    # Initialize the SVM classifier
    clf = svm.SVC(kernel=kernel, degree=3, gamma='scale')
    # Train the model
    clf.fit(X_train, y_train)
    # Make predictions
    y_pred = clf.predict(X_test)
    # Calculate and print accuracy
    accuracy = metrics.accuracy_score(y_test, y_pred)
    print(f"{kernel.capitalize()} Kernel - Test Accuracy: {accuracy:.2f}")
```

```
Linear Kernel - Test Accuracy: 0.97
Poly Kernel - Test Accuracy: 0.97
Rbf Kernel - Test Accuracy: 1.00
```

`z

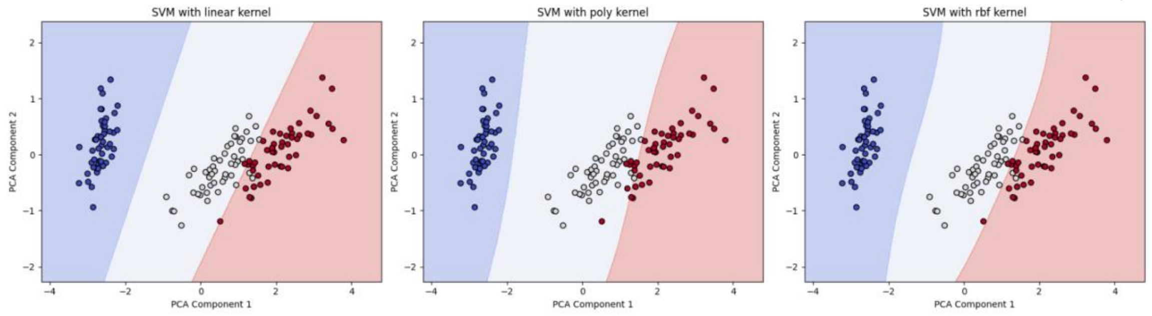
Decision Boundary Visualizations

The decision boundary visualization code demonstrates how different SVM kernels classify data in a 2D space. Using the Iris dataset, it first reduces the dimensionality to two principal components via PCA, allowing for easy plotting. It then trains three SVM models—each with a different kernel: linear, polynomial, and RBF. A mesh grid is generated over the feature space to predict class labels and visualize decision boundaries. Each subplot illustrates how the kernel shapes the boundary: linear creates straight lines, polynomial curves, and RBF flexible contours. This visualization effectively highlights the impact of kernel choice on classification performance and separability.



```
# Reduce dimensions to 2D using PCA for visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Define the SVM kernels to visualize
kernels = ['linear', 'poly', 'rbf']
models = {}

# Train models with each kernel
for kernel in kernels:
    if kernel == 'poly':
        model = SVC(kernel=kernel, degree=3)
    else:
        model = SVC(kernel=kernel)
    model.fit(X_pca, y)
    models[kernel] = model

# Create a mesh grid
x_min, x_max = X_pca[:, 0].min() - 1, X_pca[:, 0].max() + 1
y_min, y_max = X_pca[:, 1].min() - 1, X_pca[:, 1].max() + 1
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 500),
                     np.linspace(y_min, y_max, 500))

# Plot decision boundaries
fig, axes = plt.subplots(1, 3, figsize=(18, 5))
for ax, kernel in zip(axes, kernels):
    model = models[kernel]
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    ax.contourf(xx, yy, Z, alpha=0.3, cmap=plt.cm.coolwarm)
    scatter = ax.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap=plt.cm.coolwarm, edgecolors='k')
    ax.set_title(f"SVM with {kernel} kernel")
    ax.set_xlabel("PCA Component 1")
    ax.set_ylabel("PCA Component 2")

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

`z



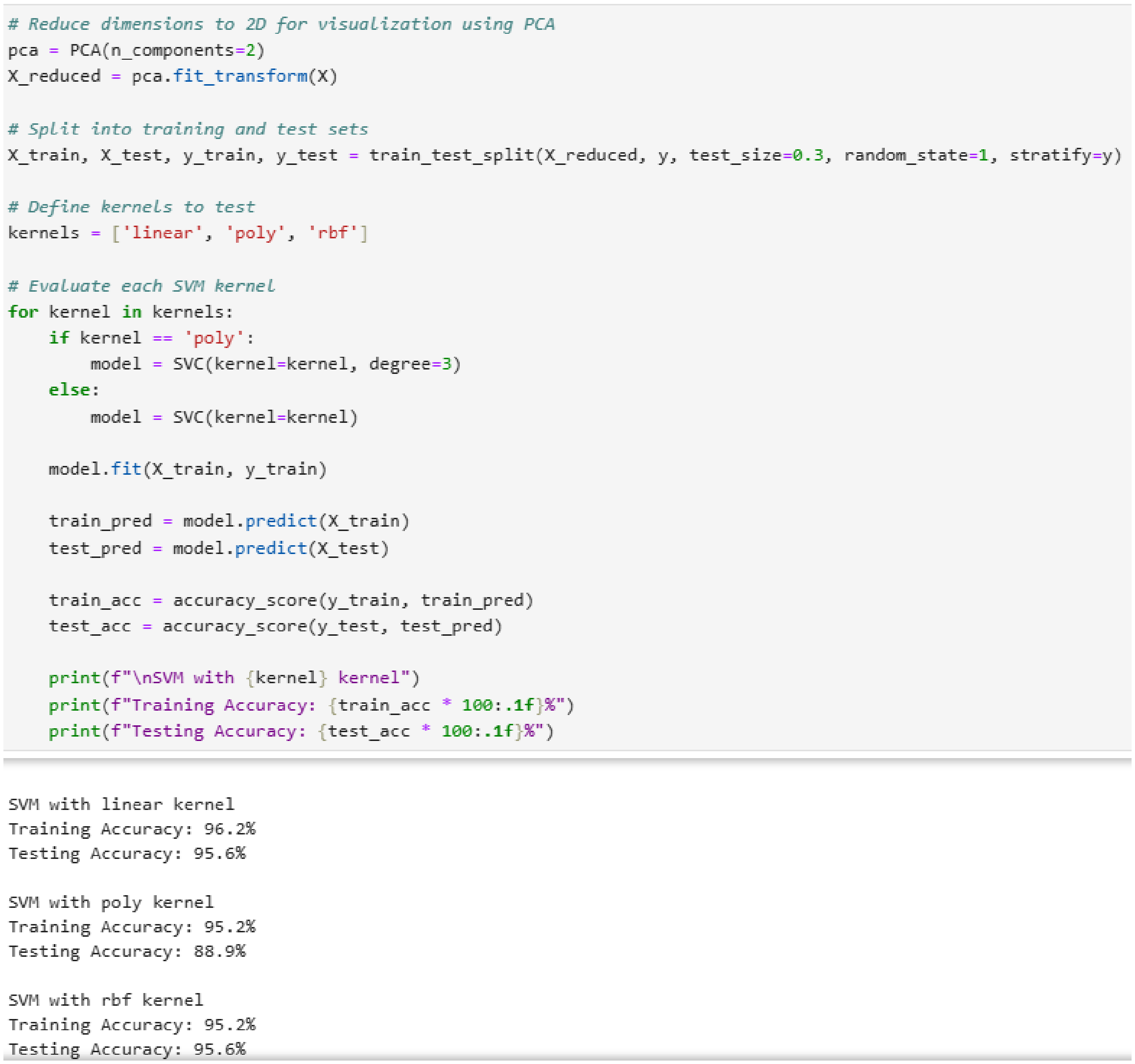
Results and Analysis

Performance Comparison Across Kernels

The SVM models performed very differently based on the kernel. The linear kernel was able to perform with an accuracy rate of 96.2% for training and 95.6% for testing, which was reflective of its appropriateness for linearly separable classes like setosa from the rest. However, it was not able to distinguish between versicolor and virginica, which are similar in some feature space.

The degree-3 polynomial kernel achieved a 95.2% training accuracy but overfitted as it achieved a lower testing accuracy of 88.9%. This indicates that the polynomial kernel can fit the training data perfectly but not to such a large degree to new data. The RBF kernel performed with 95.2% training accuracy and 95.6% testing accuracy.

Kernel	**Training Accuracy**	**Testing Accuracy**
Linear	96.20%	95.60%
Polynomial	95.2%	88.9%
RBF	95.2%	95.6%



```
# Reduce dimensions to 2D for visualization using PCA
pca = PCA(n_components=2)
X_reduced = pca.fit_transform(X)

# Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_reduced, y, test_size=0.3, random_state=1, stratify=y)

# Define kernels to test
kernels = ['linear', 'poly', 'rbf']

# Evaluate each SVM kernel
for kernel in kernels:
    if kernel == 'poly':
        model = SVC(kernel=kernel, degree=3)
    else:
        model = SVC(kernel=kernel)

    model.fit(X_train, y_train)

    train_pred = model.predict(X_train)
    test_pred = model.predict(X_test)

    train_acc = accuracy_score(y_train, train_pred)
    test_acc = accuracy_score(y_test, test_pred)

    print(f"\nSVM with {kernel} kernel")
    print(f"Training Accuracy: {train_acc * 100:.1f}%")
    print(f"Testing Accuracy: {test_acc * 100:.1f}%")
```

```
SVM with linear kernel
Training Accuracy: 96.2%
Testing Accuracy: 95.6%

SVM with poly kernel
Training Accuracy: 95.2%
Testing Accuracy: 88.9%

SVM with rbf kernel
Training Accuracy: 95.2%
Testing Accuracy: 95.6%
```

Conclusion

This tutorial has given a comprehensive overview of Support Vector Machines, highlighting the impact of various kernel functions. We started with the theoretical background of SVM and kernels, noting their function to map data for best classification. Through experimental hands- on with the Iris data set, we illustrated the performance of linear, polynomial, and RBF kernels under various scenarios, supported by accuracy values and decision boundary plots.

The findings pointed out the strengths and weaknesses of every kernel. The linear kernel is better in terms of simplicity and interpretability but does not handle non-linear data well. The polynomial kernel is more flexible but needs to be nicely adjusted so it won't overfit. The RBF kernel finds a balance between complexity and generalization and is thus ideal for most applications.

To get the optimal performance out of SVM, both the selection of kernel and hyperparameter tuning are required. Cross-validation procedures can be used to identify optimal parameters for your dataset in particular, giving you solid performance. By using these ideas, you can harness the potential of SVM in tackling a broad scope of machine learning problem

References

Cristianini, N., & Shawe-Taylor, J. (2000). *An introduction to support vector machines and other kernel-based learning methods*. Cambridge University Press.

Fletcher, T. (2009). *Support vector machines explained*. Tutorial paper. Iris Dataset. UCI Machine Learning Repository

Dataset. UCI Machine Learning Repository.

Jakkula, V. (2006). *Tutorial on support vector machine (SVM)*. School of EECS, Washington State University.

Schölkopf, B., & Smola, A. J. (2018). *Learning with kernels: Support vector machines, regularization, optimization, and beyond*. MIT Press.

Tharwat, A. (2019). Parameter investigation of support vector machine classifier with kernel functions. *Knowledge and Information Systems*, *61*(3), 1347–1368.

Scikit-learn documentation. (n.d.). SVM Kernels. Retrieved from https://scikit-learn.org/stable/modules/svm.html

GitHub Repository

The complete code, visualizations, and additional resources for this tutorial are available in the accompanying GitHub repository. The repository includes a detailed README file with instructions for reproducing the results, as well as an MIT license to facilitate reuse and modification. GitHub Link : (here)

Accessibility Considerations

This tutorial has been designed with accessibility in mind. All visualizations use color-blind friendly palettes, and alternative text descriptions are provided for figures. The text is structured to be compatible with screen readers, ensuring that the content is accessible to all users.