Support Vector Machines Assignment - 2

March 7, 2024

Q1. What is the relationship between polynomial functions and kernel functions in machine learning algorithms?

Polynomial functions and kernel functions are both used in machine learning algorithms to transform data into a higher-dimensional space in order to make it easier to classify or analyze.

Polynomial functions are a type of function that involve raising an input variable to various powers and multiplying by coefficients. Polynomial functions are commonly used in regression analysis, where they can be used to fit a curve to a set of data points. In machine learning, polynomial functions can be used as a basis function to transform data into a higher-dimensional space.

Kernel functions, on the other hand, are a general class of functions that take two inputs and return a scalar value. In machine learning, kernel functions are used to compute the similarity between two data points in a feature space. The kernel function essentially measures the degree of similarity between two inputs, and this similarity metric is used to determine how close or far apart the inputs are in the transformed feature space.

In many cases, polynomial functions can be used as kernel functions in machine learning algorithms. This is because polynomial functions can be expressed as a dot product of two vectors in a higher-dimensional space. Specifically, the polynomial kernel function is defined as:**K(x, y) = (x^T y + c)^d**

where x and y are input vectors, d is the degree of the polynomial, and c is a constant term. This kernel function computes the similarity between two inputs in a higher-dimensional feature space without explicitly transforming the data into that space.

In summary, while polynomial functions and kernel functions are different types of functions, they are related in that polynomial functions can be used as kernel functions in machine learning algorithms.

Q2. How can we implement an SVM with a polynomial kernel in Python using Scikit-learn?

To implement a **support vector machine (SVM)** with a **polynomial kernel** in **Python** using **Scikit-learn**, you can follow these steps:

Step 1: Import the necessary libraries

[1]:
```
from sklearn.svm import SVC
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
```

Step 2: Generate a sample dataset for classification

[2]:
```
X, y = make_classification(n_samples=100, n_features=10, n_informative=5,␣
↪n_classes=2, random_state=42)
```

Step 3: Split the data into training and testing sets

[3]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

Step 4: Create an instance of the SVM model with the polynomial kernel

[4]:
```
poly_svm = SVC(kernel='poly', degree=3, gamma='scale')
```

In the above code, we have specified the **kernel parameter** as **'poly'** to use the **polynomial kernel** and the **degree parameter as 3** to specify the **degree of the polynomial**.

Step 5: Fit the model to the training data

[5]:
```
poly_svm.fit(X_train, y_train)
```

[5]:
```
SVC(kernel='poly')
```

Step 6: Make predictions on the test data

[6]:
```
y_pred = poly_svm.predict(X_test)
```

Step 7: Evaluate the performance of the model

[7]:
```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)
```

```
Accuracy: 0.85
```

In the above code, we have used the **accuracy_score function** from the **metrics module** of **Scikit-learn** to calculate the **accuracy of the model**.

Overall, these steps show how you can implement a **support vector machine (SVM)** with a **polynomial kernel** in **Python** using **Scikit-learn**.

Q3. How does increasing the value of epsilon affect the number of support vectors in SVR?

In Support Vector Regression (SVR), the parameter epsilon is a hyperparameter that controls the trade-off between the flatness of the regression function and the degree to which errors are tolerated in the training data. Specifically, epsilon defines a margin of tolerance around the predicted value, within which errors are not penalized.

Increasing the value of epsilon in SVR tends to increase the number of support vectors. This is because as the value of epsilon increases, the flatness of the regression function increases and the degree of error tolerance also increases. As a result, more data points are likely to fall within the margin of tolerance, which in turn leads to more support vectors.

The number of support vectors in SVR has a direct impact on the complexity of the model and its ability to generalize to new data. In general, a higher number of support vectors can lead to longer training times and potentially overfitting to the training data. On the other hand, a lower number of support vectors can result in a simpler model that may not capture all of the complex relationships in the data.

Therefore, the choice of epsilon should be carefully considered to strike a balance between model complexity and accuracy. In practice, the optimal value of epsilon is often determined through a process of cross-validation or grid search over a range of hyperparameter values.

Q4. How does the choice of kernel function, C parameter, epsilon parameter, and gamma parameter affect the performance of Support Vector Regression (SVR)? Can you explain how each parameter works and provide examples of when you might want to increase or decrease its value?

The **performance** of **Support Vector Regression (SVR)** is **heavily dependent** on the choice of **hyperparameters** such as the **kernel function, C parameter, epsilon parameter, and gamma parameter**. In this answer, we will discuss each **parameter** and explain **how it works** and when you might want **to increase or decrease its value**. 1. **Kernel Function**: The choice of kernel function determines the mapping of the input data to a high-dimensional feature space. The kernel function is specified using the kernel hyperparameter in the SVR object. Scikit-learn provides several options for kernel functions, such as linear, polynomial, and radial basis function (RBF). The choice of kernel function depends on the nature of the data and the problem at hand. In general, the RBF kernel is a good default choice and works well in many scenarios. 2. **C Parameter**: The C parameter controls the trade-off between model complexity and training error. It is specified using the C hyperparameter in the SVR object. A smaller value of C will result in a wider margin and a simpler model, while a larger value of C will result in a narrower margin and a more complex model. Increasing the value of C can lead to overfitting, while decreasing the value of C can lead to underfitting. You might want to increase the value of C if the model is underfitting, and decrease the value of C if the model is overfitting. 3. **Epsilon Parameter**: The epsilon parameter determines the width of the margin around the predicted value within which errors are not penalized. It is specified using the epsilon hyperparameter in the SVR object. A larger value of epsilon will allow for more error in the model, while a smaller value of epsilon will make the model more sensitive to errors. You might want to increase the value of epsilon if the model is too sensitive to noise, and decrease the value of epsilon if the model is not sensitive enough to noise. 4. **Gamma Parameter**: The gamma parameter determines the width of the Gaussian kernel for the RBF kernel function. It is specified using the gamma hyperparameter in the SVR object. A larger value of gamma will result in a narrower kernel and a more complex model, while a smaller value of gamma will result in a wider kernel and a simpler model. Increasing the value of gamma can lead to overfitting, while decreasing the value of gamma can lead to underfitting. You might want to increase the value of gamma if the model is underfitting, and decrease the value of gamma if the model is overfitting.

In general, the choice of **hyperparameters** in **Support Vector Regression (SVR)** can have a **significant impact** on the **performance of the model**. It is **highly recommended** to test **different values** of these **hyperparameters** using **techniques** such as **cross-validation** and **grid search** to find the **optimal combination of hyperparameters** for the **given problem**.

Q5. Assignment: 1. **Import the necessary libraries and load the dataset.** 2. **Split the dataset into training and testing sets.** 3. **Preprocess the data using any technique of**

your choice. (e.g. scaling, normalization) 4. **Create an instance of the SVC classifier and train it on the training data.** 5. **Use the trained classifier to predict the labels of the testing data.** 6. **Evaluate the performance of the classifier using any metric of your choice. (e.g. accuracy, precision, recall, F1-score)** 7. **Tune the hyperparameters of the SVC classifier using GridSearchCV or RandomizedSearchCV to improve its performance.** 8. **Train the tuned classifier on the entire dataset.** 9. **Save the trained classifier to a file for future use.**

NOTE: You can use any dataset of your choice for this assignment, but make sure it is suitable for classification and has a sufficient number of features and samples.

Here's an example of how to implement these steps using the Red Wine dataset from Seaborn library:

[10]:
```
# 1. Import the necessary libraries and load the dataset
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
import joblib
df = pd.read_csv('winequality-red.csv')

# 2. Split the dataset into training and testing sets
X = df.drop('quality', axis=1)
y = df['quality']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# 3. Preprocess the data using StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 4. Create an instance of the SVC classifier and train it on the training data
svc = SVC()
svc.fit(X_train_scaled, y_train)

# 5. Use the trained classifier to predict the labels of the testing data
y_pred = svc.predict(X_test_scaled)

# 6. Evaluate the performance of the classifier using accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
print("")

# 7. Tune the hyperparameters of the SVC classifier using GridSearchCV
```

```
param_grid = {'C': [0.1, 1, 10], 'gamma': [0.1, 1, 10], 'kernel': ['linear',␣
 ↪'rbf']}
grid = GridSearchCV(SVC(), param_grid, refit=True, verbose=3)
grid.fit(X_train_scaled, y_train)

# 8. Train the tuned classifier on the entire dataset
best_svc = grid.best_estimator_
X_scaled = scaler.fit_transform(X)
best_svc.fit(X_scaled, y)

# 9. Save the trained classifier to a file for future use
joblib.dump(best_svc, 'svc_model.joblib')
```

```
Accuracy: 0.60

Fitting 5 folds for each of 18 candidates, totalling 90 fits
[CV 1/5] END …C=0.1, gamma=0.1, kernel=linear;, score=0.562 total time=   0.1s
[CV 2/5] END …C=0.1, gamma=0.1, kernel=linear;, score=0.555 total time=   0.1s
[CV 3/5] END …C=0.1, gamma=0.1, kernel=linear;, score=0.570 total time=   0.1s
[CV 4/5] END …C=0.1, gamma=0.1, kernel=linear;, score=0.613 total time=   0.1s
[CV 5/5] END …C=0.1, gamma=0.1, kernel=linear;, score=0.655 total time=   0.1s
[CV 1/5] END …C=0.1, gamma=0.1, kernel=rbf;, score=0.559 total time=   0.1s
[CV 2/5] END …C=0.1, gamma=0.1, kernel=rbf;, score=0.570 total time=   0.1s
[CV 3/5] END …C=0.1, gamma=0.1, kernel=rbf;, score=0.590 total time=   0.1s
[CV 4/5] END …C=0.1, gamma=0.1, kernel=rbf;, score=0.598 total time=   0.1s
[CV 5/5] END …C=0.1, gamma=0.1, kernel=rbf;, score=0.643 total time=   0.1s
[CV 1/5] END …C=0.1, gamma=1, kernel=linear;, score=0.562 total time=   0.1s
[CV 2/5] END …C=0.1, gamma=1, kernel=linear;, score=0.555 total time=   0.1s
[CV 3/5] END …C=0.1, gamma=1, kernel=linear;, score=0.570 total time=   0.1s
[CV 4/5] END …C=0.1, gamma=1, kernel=linear;, score=0.613 total time=   0.1s
[CV 5/5] END …C=0.1, gamma=1, kernel=linear;, score=0.655 total time=   0.1s
[CV 1/5] END …C=0.1, gamma=1, kernel=rbf;, score=0.430 total time=   0.1s
[CV 2/5] END …C=0.1, gamma=1, kernel=rbf;, score=0.434 total time=   0.1s
[CV 3/5] END …C=0.1, gamma=1, kernel=rbf;, score=0.430 total time=   0.2s
[CV 4/5] END …C=0.1, gamma=1, kernel=rbf;, score=0.430 total time=   0.2s
[CV 5/5] END …C=0.1, gamma=1, kernel=rbf;, score=0.431 total time=   0.1s
[CV 1/5] END …C=0.1, gamma=10, kernel=linear;, score=0.562 total time=   0.1s
[CV 2/5] END …C=0.1, gamma=10, kernel=linear;, score=0.555 total time=   0.1s
[CV 3/5] END …C=0.1, gamma=10, kernel=linear;, score=0.570 total time=   0.1s
[CV 4/5] END …C=0.1, gamma=10, kernel=linear;, score=0.613 total time=   0.1s
[CV 5/5] END …C=0.1, gamma=10, kernel=linear;, score=0.655 total time=   0.1s
[CV 1/5] END …C=0.1, gamma=10, kernel=rbf;, score=0.430 total time=   0.2s
[CV 2/5] END …C=0.1, gamma=10, kernel=rbf;, score=0.434 total time=   0.2s
[CV 3/5] END …C=0.1, gamma=10, kernel=rbf;, score=0.430 total time=   0.2s
[CV 4/5] END …C=0.1, gamma=10, kernel=rbf;, score=0.430 total time=   0.2s
[CV 5/5] END …C=0.1, gamma=10, kernel=rbf;, score=0.431 total time=   0.2s
[CV 1/5] END …C=1, gamma=0.1, kernel=linear;, score=0.562 total time=   0.1s
```

```
[CV 2/5] END …C=1, gamma=0.1, kernel=linear;, score=0.551 total time=   0.1s
[CV 3/5] END …C=1, gamma=0.1, kernel=linear;, score=0.559 total time=   0.1s
[CV 4/5] END …C=1, gamma=0.1, kernel=linear;, score=0.602 total time=   0.1s
[CV 5/5] END …C=1, gamma=0.1, kernel=linear;, score=0.647 total time=   0.1s
[CV 1/5] END …C=1, gamma=0.1, kernel=rbf;, score=0.605 total time=   0.1s
[CV 2/5] END …C=1, gamma=0.1, kernel=rbf;, score=0.586 total time=   0.1s
[CV 3/5] END …C=1, gamma=0.1, kernel=rbf;, score=0.609 total time=   0.1s
[CV 4/5] END …C=1, gamma=0.1, kernel=rbf;, score=0.641 total time=   0.1s
[CV 5/5] END …C=1, gamma=0.1, kernel=rbf;, score=0.686 total time=   0.1s
[CV 1/5] END …C=1, gamma=1, kernel=linear;, score=0.562 total time=   0.1s
[CV 2/5] END …C=1, gamma=1, kernel=linear;, score=0.551 total time=   0.1s
[CV 3/5] END …C=1, gamma=1, kernel=linear;, score=0.559 total time=   0.1s
[CV 4/5] END …C=1, gamma=1, kernel=linear;, score=0.602 total time=   0.1s
[CV 5/5] END …C=1, gamma=1, kernel=linear;, score=0.647 total time=   0.1s
[CV 1/5] END …C=1, gamma=1, kernel=rbf;, score=0.645 total time=   0.2s
[CV 2/5] END …C=1, gamma=1, kernel=rbf;, score=0.648 total time=   0.2s
[CV 3/5] END …C=1, gamma=1, kernel=rbf;, score=0.621 total time=   0.2s
[CV 4/5] END …C=1, gamma=1, kernel=rbf;, score=0.668 total time=   0.2s
[CV 5/5] END …C=1, gamma=1, kernel=rbf;, score=0.682 total time=   0.2s
[CV 1/5] END …C=1, gamma=10, kernel=linear;, score=0.562 total time=   0.1s
[CV 2/5] END …C=1, gamma=10, kernel=linear;, score=0.551 total time=   0.1s
[CV 3/5] END …C=1, gamma=10, kernel=linear;, score=0.559 total time=   0.1s
[CV 4/5] END …C=1, gamma=10, kernel=linear;, score=0.602 total time=   0.1s
[CV 5/5] END …C=1, gamma=10, kernel=linear;, score=0.647 total time=   0.1s
[CV 1/5] END …C=1, gamma=10, kernel=rbf;, score=0.562 total time=   0.2s
[CV 2/5] END …C=1, gamma=10, kernel=rbf;, score=0.562 total time=   0.2s
[CV 3/5] END …C=1, gamma=10, kernel=rbf;, score=0.547 total time=   0.2s
[CV 4/5] END …C=1, gamma=10, kernel=rbf;, score=0.578 total time=   0.2s
[CV 5/5] END …C=1, gamma=10, kernel=rbf;, score=0.569 total time=   0.2s
[CV 1/5] END …C=10, gamma=0.1, kernel=linear;, score=0.559 total time=   0.2s
[CV 2/5] END …C=10, gamma=0.1, kernel=linear;, score=0.543 total time=   0.2s
[CV 3/5] END …C=10, gamma=0.1, kernel=linear;, score=0.559 total time=   0.2s
[CV 4/5] END …C=10, gamma=0.1, kernel=linear;, score=0.598 total time=   0.2s
[CV 5/5] END …C=10, gamma=0.1, kernel=linear;, score=0.647 total time=   0.2s
[CV 1/5] END …C=10, gamma=0.1, kernel=rbf;, score=0.629 total time=   0.1s
[CV 2/5] END …C=10, gamma=0.1, kernel=rbf;, score=0.609 total time=   0.1s
[CV 3/5] END …C=10, gamma=0.1, kernel=rbf;, score=0.594 total time=   0.1s
[CV 4/5] END …C=10, gamma=0.1, kernel=rbf;, score=0.656 total time=   0.1s
[CV 5/5] END …C=10, gamma=0.1, kernel=rbf;, score=0.678 total time=   0.1s
[CV 1/5] END …C=10, gamma=1, kernel=linear;, score=0.559 total time=   0.2s
[CV 2/5] END …C=10, gamma=1, kernel=linear;, score=0.543 total time=   0.2s
[CV 3/5] END …C=10, gamma=1, kernel=linear;, score=0.559 total time=   0.2s
[CV 4/5] END …C=10, gamma=1, kernel=linear;, score=0.598 total time=   0.2s
[CV 5/5] END …C=10, gamma=1, kernel=linear;, score=0.647 total time=   0.2s
[CV 1/5] END …C=10, gamma=1, kernel=rbf;, score=0.590 total time=   0.2s
[CV 2/5] END …C=10, gamma=1, kernel=rbf;, score=0.648 total time=   0.2s
[CV 3/5] END …C=10, gamma=1, kernel=rbf;, score=0.633 total time=   0.2s
[CV 4/5] END …C=10, gamma=1, kernel=rbf;, score=0.641 total time=   0.2s
```

```
[CV 5/5] END …C=10, gamma=1, kernel=rbf;, score=0.671 total time=   0.2s
[CV 1/5] END …C=10, gamma=10, kernel=linear;, score=0.559 total time=   0.2s
[CV 2/5] END …C=10, gamma=10, kernel=linear;, score=0.543 total time=   0.2s
[CV 3/5] END …C=10, gamma=10, kernel=linear;, score=0.559 total time=   0.2s
[CV 4/5] END …C=10, gamma=10, kernel=linear;, score=0.598 total time=   0.2s
[CV 5/5] END …C=10, gamma=10, kernel=linear;, score=0.647 total time=   0.2s
[CV 1/5] END …C=10, gamma=10, kernel=rbf;, score=0.562 total time=   0.2s
[CV 2/5] END …C=10, gamma=10, kernel=rbf;, score=0.570 total time=   0.2s
[CV 3/5] END …C=10, gamma=10, kernel=rbf;, score=0.547 total time=   0.2s
[CV 4/5] END …C=10, gamma=10, kernel=rbf;, score=0.574 total time=   0.2s
[CV 5/5] END …C=10, gamma=10, kernel=rbf;, score=0.569 total time=   0.2s
```

[10]: `['svc_model.joblib']`