Mastering SVMs: Understanding Kernels and Visualizing Decision Boundaries

Introduction to Support Vector Machines (SVM)

For classification problems, Support Vector Machines (SVMs) are a common kind of machine learning method. Finding the ideal border (or hyperplane) to divide data points into several classes is the aim of SVM. Consider that you are attempting to divide various colored dots on a sheet of paper by drawing a line (or a curve). Finding that line as efficiently as possible while optimizing the distance (or margin) between the dots on each side of the line is the task of SVM.

SVM is unique in that it can handle both linear and non-linear data. SVM is hence capable of handling data that can be readily separated by a straight line or that calls for a more intricate curve. The usage of kernels provides this flexibility.

The Concept of Kernels in SVM

The fascinating part is that SVM can use kernels to operate in higher-dimensional domains. Similar to a magical tool, a kernel aids SVM in establishing intricate decision limits. Imagine transforming a straightforward straight line into something more flexible by bending or curling a flat sheet of paper to fit around your data points.

SVM can adjust to different kinds of data to several kernel types:

- **Linear Kernel:** The most basic type of kernel is the linear one. It makes the assumption that a plane or straight line may be used to divide the data.
- **Polynomial kernel:** When the data isn't linearly separable but still exhibits a pattern that a polynomial curve may capture, the polynomial kernel performs well.
- **Radial Basis Function (RBF) kernel:** The RBF kernel is excellent for more difficult situations since it may produce curved, flexible decision boundaries if your data is complex and untidy.

The performance of the SVM can be significantly impacted by the kernel selection. Depending on the nature of your data, each kernel has advantages. Part of the difficulty is determining which kernel is ideal for your specific issue. Finding the most effective method to segregate your data without overfitting is ultimately what matters.

Step-by-Step

Let's start by the entire procedure, beginning with the creation of the datasets.

Establishing the Datasets

To keep things simple, we chose make_moons and make_blobs, two datasets frequently used for classification tasks.

- **make_moons:** As the name implies, this dataset has a crescent moon form. It works well for testing algorithms for classification, particularly those that have to deal with non-linear bounds. It's not a simple task, and it comes with some noise to make it a little more difficult.
- **make_blobs:** This dataset is composed of points that are grouped together into blobs, in contrast to the moon-shaped data. The objective is to differentiate the classes that are represented by each blob. It is a nice place to start when testing SVM with various kernels because it is easier to use and cleaner.

Both datasets are ideal for training a classifier such as SVM since they are labeled, allowing us to determine which class each data point belongs to.

Splitting the Data

Sorting the datasets into training and test sets was the next step after obtaining them. This is important because we want to evaluate our model's performance on unseen data after training it on a portion of the data.

In our instance, 20% of the data was utilized for testing and 80% for training. In addition to providing the model with sufficient data for learning, this also sets aside some data to assess the model's generalization to novel, untested situations. After you've taught the model the content, it's like giving it a test!

Training the SVM Models

Training the model is the exciting part now! We employed three distinct kernels for the Support Vector Machine (SVM) algorithm: linear, poly (polynomial), and rbf (Radial Basis Function). Each kernel functions as follows, along with its benefits:

- **Linear Kernel:** The most basic kernel is the linear kernel. When the data can be separated by a straight line or plane, it is said to be linearly separable. Although it is quick and simple, it performs poorly with more complicated data.
- **Polynomial Kernel:** This type of kernel introduces curves to deal with data that isn't linearly separable. It works well with data that is somewhat curved but not very complicated. It gives us additional freedom by allowing the decision limit to change.
- **RBF Kernel:** The Radial Basis Function kernel, or RBF kernel, functions something like magic. Because it permits the decision border to curve in a variety of ways, it performs incredibly well on complicated datasets. It's particularly useful when you require a flexible decision boundary and the data is dispersed across the environment.

Evaluating the Results

It was time to assess the model's performance on the test data after training. Here's where precision is important.

Moons Dataset:

- **Linear Kernel:** About 90% accuracy. The linear kernel performed rather well, although it had some difficulty capturing the non-linear crescent-shaped data.
- **Polynomial Kernel:** There was a minor gain in accuracy, which increased to 91%. To better handle the data, the polynomial kernel slightly curved the decision boundary.
- **RBF Kernel:** The accuracy increased to 96.67%, demonstrating that the RBF kernel was capable of handling the moons dataset's intricate geometry.

Blobs Dataset:

- **Linear Kernel:** A linear kernel has 100% accuracy. The linear kernel has no issues at all because the blobs are simple to separate.
- **Polynomial Kernel:** 100% accuracy once more. Since the data is already straightforward, the polynomial kernel didn't make much of an improvement.

- **RBF Kernel:** 100% accuracy as well. For this straightforward dataset, the RBF kernel was overkill, but it still worked well.

In Conclusion

we observed that:

- The linear kernel performed poorly on the non-linear moons dataset but well on the simple blobs dataset.
- On the moons dataset, the polynomial kernel slightly outperformed the linear kernel.
- On the moons dataset, the RBF kernel truly excelled, managing the intricate decision boundaries with ease and offering the highest overall accuracy.

This methodical technique demonstrates how several kernels affect SVM's performance and why it's critical to select the best one depending on the features of your dataset.

Visualization of Decision Boundaries

It is quite helpful to look at the decision boundaries in order to fully understand how the Support Vector Machine (SVM) operates. The lines or curves that the SVM uses to divide data points from various classes are represented by these boundaries. Let's examine each plot's contents and how they aid in our comprehension of the model's performance using various kernels.

What the Plots Represent

In each plot, you'll observe:

- **Data point:** The tiny dots dispersed throughout the graph are known as data points. The objective is for the SVM to create a border that best divides the two classes, each of which is represented by a dot.
- **Decision boundary:** The line or curve that the SVM draws separating the two classes is known as the decision boundary. Everything on one side of the line is supposed to belong to a particular class, while everything on the other side is supposed to belong to a different class.

- **Supporting vector:** The points that are closest to the decision border are known as support vectors, and they are crucial in forming it. The margin between these support vectors and the border is what the SVM aims to optimize.

Interpreting the Plots for Each Kernel

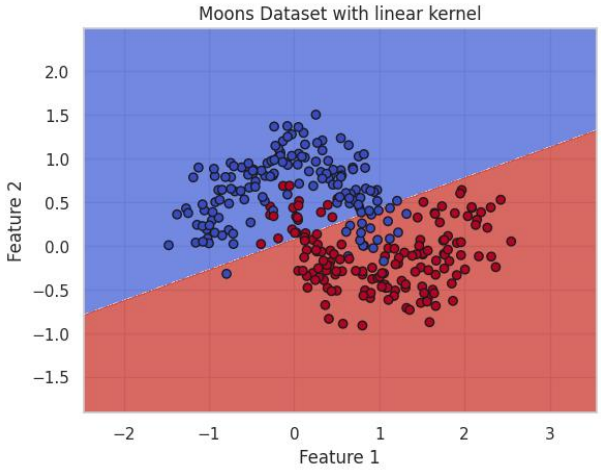
- **Linear Kernel**
    - The decision boundary in this plot is represented by a straight line. When a single, straight line can be used to segregate the data, this kernel performs best. You can observe that it performs flawlessly on the blobs dataset. A less-than-ideal categorization results from the straight line's inability to adequately represent the moons dataset's curved form.
- **Polynomial Kernel**
    - The decision border is no longer a straight line in this case. In order to better match the data, the polynomial kernel bends the border. When there is any curvature in the data, such as in the moons dataset, this is really helpful. However, given the blobs dataset is already readily separable, it might not have a significant impact.
- **RBF Kernel**
    - The magic happens in the RBF kernel. Because of its great flexibility, the decision boundary in this case may accommodate more intricate forms. It stretches and contorts to precisely suit the moons dataset. Because it can wrap around the data points and capture even the most intricate patterns, it works well with non-linear data.

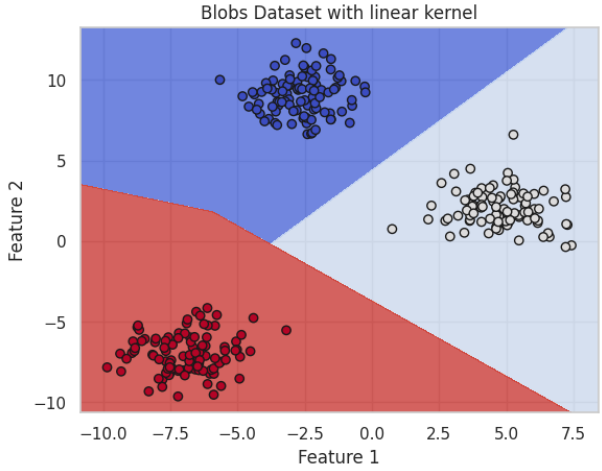
Why These Plots Matter

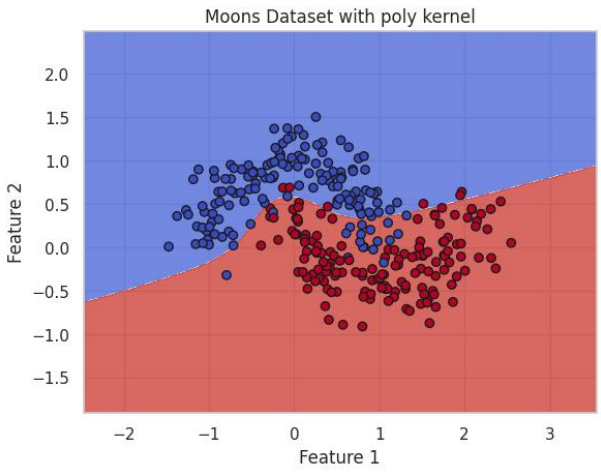
The decision boundary plots provide a visual representation of the SVM's data classification process. They help us understand:

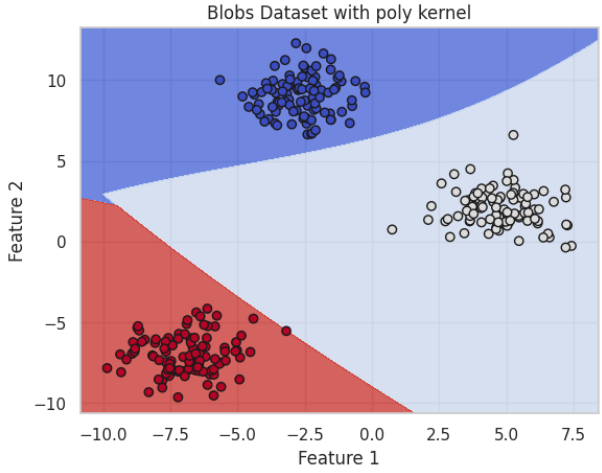
- How successfully the data is being separated by the model.
- The advantages and disadvantages of various kernels.
- Depending on how complicated the boundary is, the model may be overfitting or underfitting.

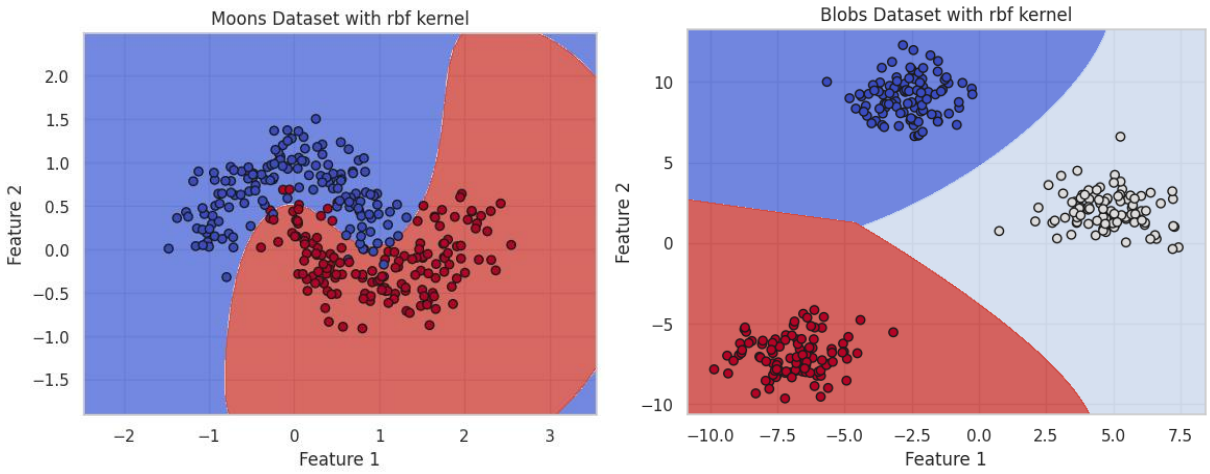
By looking at these graphs, we can see that selecting the appropriate kernel is crucial to the SVM's accuracy in data classification. The RBF kernel can handle nearly everything, whereas a linear kernel may perform brilliantly on basic datasets but falter on more complicated ones.











Results and Conclusion

We observed some intriguing outcomes for every dataset and kernel after executing our models.

With an accuracy of 96.67%, the RBF kernel outperformed the others for the moons dataset. Given that the data is non-linear and the RBF kernel excels at capturing intricate forms, this makes sense. With 91%, the polynomial kernel came in second, which is still good but less accurate. Due to its inability to manage the data's curved separation, the linear kernel performed poorly in this case, achieving an accuracy of 90%.

The outcomes on the blobs dataset were simpler to understand. Since the data was already clearly segregated using straightforward straight lines, the linear kernel performed well. The RBF and polynomial kernels also attained 100%, but since the data didn't require such intricate limits, they didn't significantly enhance performance.

Why Some Kernels Perform Better

- **Linear kernel:** For straightforward, linearly separable data, such as the blobs dataset, a linear kernel works well. It is quick, but it has trouble with more complicated data.
- **Polynomial kernel:** The polynomial kernel is less flexible than the RBF kernel, but it works well with data that has some curvature, such as the moons dataset.
- **RBF Kernel:** Performs exceptionally well in more intricate, non-linear situations. It might be overkill for simpler issues, but it adapts well to complex data, such as the Moons dataset.

In conclusion, the type of data truly determines the optimal kernel. A linear kernel is ideal for datasets that are simpler. But the RBF kernel is the best option for more complex, non-linear data.

Code snippets and visualization

Dataset Creation and Splitting

```
from sklearn.datasets import make_moons, make_blobs
from sklearn.model_selection import train_test_split
```

Creating the datasets

```
X_moons, y_moons = make_moons(n_samples=300, noise=0.2, random_state=42)
X_blobs, y_blobs = make_blobs(n_samples=300, centers=3, random_state=42, cluster_std=1.2)
```

Splitting the datasets into training and testing sets

```
X_train_moons, X_test_moons, y_train_moons, y_test_moons =
 train_test_split(X_moons, y_moons, test_size=0.3, random_state=42)
X_train_blobs, X_test_blobs, y_train_blobs, y_test_blobs =
 train_test_split(X_blobs, y_blobs, test_size=0.3, random_state=42)
```

Training the SVM with Different Kernels

```
from sklearn.svm import SVC
```

Linear Kernel SVM

1. svm_linear = SVC(kernel='linear')
2. svm_linear.fit(X_train_moons, y_train_moons)
3. linear_accuracy = svm_linear.score(X_test_moons, y_test_moons)

Polynomial Kernel SVM

4. svm_poly = SVC(kernel='poly')
5. svm_poly.fit(X_train_moons, y_train_moons)
6. poly_accuracy = svm_poly.score(X_test_moons, y_test_moons)

RBF Kernel SVM

7. svm_rbf = SVC(kernel='rbf')
8. svm_rbf.fit(X_train_moons, y_train_moons)
9. rbf_accuracy = svm_rbf.score(X_test_moons, y_test_moons)

Decision Boundary Visuals

```
import numpy as np
import matplotlib.pyplot as plt
```

```
def plot_decision_boundary(model, X, y, title):
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                         np.arange(y_min, y_max, 0.01))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, alpha=0.8, cmap=plt.cm.coolwarm)
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolor='k', cmap=plt.cm.coolwarm)
    plt.title(title)
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.show()
```

plot decision boundries

```
plot_decision_boundary(model, np.vstack((X_train, X_test)),
                       np.hstack((y_train, y_test)),
                       f'{dataset_name} with {kernel} kernel')
```

References

Scikit-learn Documentation:https://scikit-learn.org/stable/

The official documentation was used for understanding the implementation details of `SVC` and dataset generators like `make_moons` and `make_blobs`.

DataCamp: *Support Vector Machines in Python*:

https://www.datacamp.com/tutorial/svm-classification

This tutorial was helpful for explaining SVMs in a practical, Python-oriented context.

Towards Data Science: *Understanding Kernel Functions in Machine Learning*:

https://towardsdatascience.com/kernel-functions