Support Vector Machine (SVM) Algorithm

Rafiq Islam

2024-11-05

Table of contents

## Introduction

Support Vector Machines (SVM) is a powerful non-parametric supervised machine learning algorithm used for classification and, less commonly, regression tasks. Support Vector Machines are designed to find an optimal hyperplane that best separates data points into classes. The key idea behind SVMs is to maximize the margin between data points of different classes while minimizing classification errors. This leads to a robust decision boundary that generalizes well to unseen data.

## The Mathematical Foundation of SVM

Consider a classification problem. Given a dataset where , represents the feature vector of the -th sample, and represents the class label. The goal of SVM is to find a hyperplane that maximally separates the classes.

### Hyperplane and Dicision Boundary

Definition (Hyperplane)

A hyperplane in an -dimensional space is defined by:

where:

* is the weight vector,
* is the bias term,
* is any point on the hyperplane.

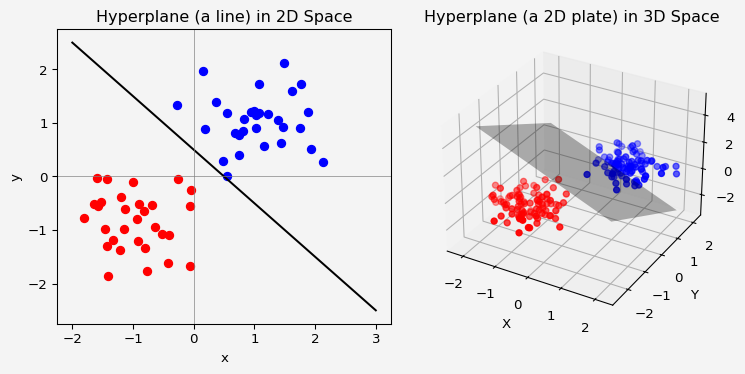
For a two-dimensional space, this hyperplane is simply a line.

$$
w^T\mathbf{x}+b=0;\hspace{4mm}\implies w\_0x+w\_1y+b=0;\hspace{4mm}\implies y=\frac{-w\_0x-b}{w\_1}
$$

and for a three-dimensional space, this hyperplane is simply a 2D plane

$$
w^T\mathbf{x}+b=0;\hspace{4mm}\implies w\_0x+w\_1y+w\_2z+b=0;\hspace{4mm}\implies z=\frac{-w\_0x-w\_1y-b}{w\_2}
$$

import numpy as np  
import matplotlib.pyplot as plt  
from mpl\_toolkits.mplot3d import Axes3D  
  
w\_2d = np.array([1,1])  
b\_2d = -0.5  
  
w\_3d = np.array([1,1,1])  
b\_3d = -1  
  
def decision\_boundary\_2d(x):  
 return (-w\_2d[0]\*x-b\_2d) / w\_2d[1]  
  
def decision\_boundary\_3d(x, y):  
 return (-w\_3d[0]\*x-w\_3d[1]\*y-b\_3d) / w\_3d[2]  
  
np.random.seed(0)  
class1x\_2d = np.random.normal(loc=[1,1],scale=0.5, size=(30,2))  
class2x\_2d = np.random.normal(loc=[-1,-1],scale=0.5, size=(30,2))  
  
class1x\_3d = np.random.normal(loc=[1,1,1],scale=0.5, size=(90,3))  
class2x\_3d = np.random.normal(loc=[-1,-1,-1],scale=0.5, size=(90,3))  
  
fig = plt.figure( figsize=(7.9,4))  
ax1 = fig.add\_subplot(121)  
x\_vals\_2d = np.linspace(-2,3,100)  
plt.plot(  
 x\_vals\_2d, decision\_boundary\_2d(x\_vals\_2d),  
 'k-', label = "Decision Boundary (Hyperplane)"  
 )  
ax1.scatter(  
 class1x\_2d[:,0], class1x\_2d[:,1], color='blue',  
 marker='o', label = 'Class +1'  
 )  
ax1.scatter(  
 class2x\_2d[:,0], class2x\_2d[:,1], color='red',  
 marker='o', label = 'Class -1'  
 )  
ax1.set\_xlabel('x')  
ax1.set\_ylabel('y')  
ax1.set\_title('Hyperplane (a line) in 2D Space')  
ax1.axhline(0, color='grey', lw = 0.5)  
ax1.axvline(0, color='grey', lw = 0.5)  
  
  
ax2 = fig.add\_subplot(122, projection = '3d')  
x\_vals\_3d = np.linspace(-2,2,30)  
y\_vals\_3d = np.linspace(-2,2,30)  
X, Y = np.meshgrid(x\_vals\_3d, y\_vals\_3d)  
Z = decision\_boundary\_3d(X, Y)  
  
ax2.plot\_surface(X, Y, Z, color='k', alpha = 0.3, rstride=100, cstride=100, edgecolor='none')  
ax2.scatter(class1x\_3d[:,0], class1x\_3d[:,1],class1x\_3d[:,2], color = 'blue', marker='o', label='Class +1')  
ax2.scatter(class2x\_3d[:,0], class2x\_3d[:,1],class2x\_3d[:,2], color = 'red', marker='o', label='Class -1')  
ax2.set\_xlabel('X')  
ax2.set\_ylabel('Y')  
ax2.set\_zlabel('Z')  
ax2.set\_title('Hyperplane (a 2D plate) in 3D Space')  
  
plt.tight\_layout()  
axes = [ax1,ax2]  
for ax in axes:  
 ax.set\_facecolor('#f4f4f4')  
plt.gcf().patch.set\_facecolor('#f4f4f4')  
plt.show()



### Margin and the Optimal Hyperplane

Definition (Margin)

The margin is the distance between the hyperplane and the nearest data points from either class. SVM aims to maximize this margin to achieve better separation, which makes the classifier more robust.

To define the margin mathematically, we impose that for all points:

For a data vector with label :

* If : we want (to be on the correct side of the hyperplane)
* If : we want (to be on the correct side of the hyperplane)

These two conditions combaine the equation mention above. That is all points must be at least a unit distance from the hyperplane on the correct side. The data points that satisfy or lie on the “support vectors,” or the points closest to the hyperplane.

import plotly.io as pio  
import plotly.graph\_objects as go  
  
A,B,C = 2,-3,5  
D1 = 5  
D2 = -3  
  
dist = abs(D1-D2)/np.sqrt(A\*\*2+B\*\*2+C\*\*2)  
x = np.linspace(-10,10,100)  
y = np.linspace(-10,10,100)  
x,y = np.meshgrid(x,y)  
  
z1 = (-A\*x-B\*y - D1)/C  
z2 = (-A\*x-B\*y - D2)/C  
  
fig = go.Figure()  
  
fig.add\_trace(go.Surface(z=z1, x=x, y=y, opacity=0.5, colorscale='Blues', name='Plane 1'))  
fig.add\_trace(go.Surface(z=z2, x=x, y=y, opacity=0.5, colorscale='Blues', name='Plane 2'))  
  
x0,y0 = 0,0  
z0 = (-A\*x0-B\*y0 - D1)/C  
z1 = (-A\*x0-B\*y0 - D2)/C  
  
fig.add\_trace(  
 go.Scatter3d(x=[x0,x0],y=[y0,y0], z=[z0,z1], mode='lines', line=dict(color='red',width=10),name='Distance')  
 )  
fig.update\_layout(  
 scene=dict(  
 xaxis\_title = 'X',  
 yaxis\_title = 'Y',  
 zaxis\_title = 'Z',  
 bgcolor = '#f4f4f4'  
 ),  
 title = "Distance Between two planes",  
 width = 600,  
 height = 600  
)  
fig.show()

Unable to display output for mime type(s): text/html

Unable to display output for mime type(s): text/html

The margin can then be expressed as:

Our objective is to maximize , or equivalently, minimize subject to the constraints .

### Optimization of the SVM

The optimization problem can be formulated as follows:

**Primal Form:**

subject to:

This is a convex optimization problem because the objective function is convex, and the constraints are linear.

### 6. The Dual Form of SVM

To solve the optimization problem, it is often more efficient to use the dual form. By introducing Lagrange multipliers , we can construct the Lagrangian:

Taking the partial derivatives of with respect to and and setting them to zero yields:

By substituting these into the Lagrangian, we arrive at the dual problem:

**Dual Form:**

subject to:

The solution to the dual form gives the values of , which are used to construct the optimal hyperplane. The final decision boundary is then:

### 7. Kernel Trick and Nonlinear SVM

When the data is not linearly separable, SVMs use the **kernel trick** to map the data into a higher-dimensional space where a linear separation is possible. Common kernels include:

* **Linear Kernel:**
* **Polynomial Kernel:**
* **Radial Basis Function (RBF) Kernel:**

Using a kernel function , we replace in the dual form with , which allows for classification in complex, non-linear spaces.

### 8. Conclusion

Support Vector Machines provide a powerful framework for classification, balancing complexity and model accuracy by maximizing the margin between classes. By using the dual form, they efficiently handle high-dimensional data and allow non-linear decision boundaries via the kernel trick. This combination of mathematical rigor and flexibility makes SVMs a popular choice in many machine learning tasks.

Understanding these foundations will give you deeper insight into why SVMs perform so well and how they achieve robust classifications even with complex datasets.

This post provides an overview for those interested in understanding SVM from both a theoretical and practical perspective. For code examples and implementations, you could explore libraries like scikit-learn, which offer out-of-the-box SVM functionality.