Support Vector Machines (SVMs)

Short Report

1. Introduction

Support Vector Machines, or SVMs, are one of those algorithms that are both simple in idea and surprisingly powerful in practice. They're mostly used for classification — like deciding whether an email is spam or not — but can also be adapted for regression tasks. The basic goal of an SVM is to find the "best" dividing line (or plane, or hyperplane) between two groups of data. And by "best," we mean the one that keeps the largest possible gap between the two classes. This gap is what we call the *margin*, and having a wide margin usually means the model will handle new data better.

2. The Main Idea

Let's say we have two sets of points on a graph, each representing a different class. Many lines could separate them, but SVM tries to find the one that's right in the middle — farthest from the nearest points on either side. Those points that lie closest to the boundary are called *support vectors*, and they basically define where that line sits. Everything else in the dataset doesn't affect the final boundary as much, which is part of what makes SVMs efficient once trained.

3. Mathematical Formulation

We start with a dataset of pairs $(\vec{x}_i, y_i)$ where $\vec{x}_i \in \mathbb{R}^n$ represents the features of the i^{th} data point, and $y_i \in \{-1, +1\}$ is its label. We want to find a hyperplane

$$\vec{w} \cdot \vec{x} + b = 0$$

that separates the classes. For the data to be correctly classified, we need

$$y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1$$

The margin, which we're trying to make as wide as possible, is given by $\frac{2}{\|\vec{w}\|}$. So, maximizing the margin is equivalent to minimizing $\frac{1}{2}\|\vec{w}\|^2$.

That gives us the optimization problem:

$$\min_{\vec{w},b} \frac{1}{2}\|\vec{w}\|^2 \quad \text{subject to } y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1$$

This is for the ideal case where the data is perfectly separable. But in real life, data is messy — points overlap, and clean separation isn't always possible. To handle that, we introduce small error terms called *slack variables* ξ_i:

$$\min_{\vec{w},b,\xi} \frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{n} \xi_i$$

subject to

$$y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1 - \xi_i$$

Here, C is a constant that controls how strict we are. A high C means fewer mistakes but a tighter boundary; a smaller C means a wider margin with some errors allowed.

4. The Kernel Trick

When data can't be separated by a straight line, SVMs use something clever called the *kernel trick*. The idea is to map the data into a higher-dimensional space where a linear separator might actually work. Instead of doing this mapping manually, we use a kernel function $K(\vec{x}_i, \vec{x}_j)$ that measures similarity between points in that higher space:

$$K(\vec{x}_i, \vec{x}_j) = \phi(\vec{x}_i) \cdot \phi(\vec{x}_j)$$

Some popular kernels:

$$\text{Linear: } K(\vec{x}_i, \vec{x}_j) = \vec{x}_i \cdot \vec{x}_j$$

$$\text{Polynomial: } K(\vec{x}_i, \vec{x}_j) = (\vec{x}_i \cdot \vec{x}_j + 1)^d$$

$$\text{RBF (Gaussian): } K(\vec{x}_i, \vec{x}_j) = e^{-\gamma \|\vec{x}_i - \vec{x}_j\|^2}$$

The kernel trick lets SVMs handle complex, curvy decision boundaries without actually computing the high-dimensional transformation. That's part of what makes them so powerful.

5. A Simple Example

Imagine you're building a spam filter. Each email becomes a feature vector $\vec{x}$ based on word frequency or certain keywords. A linear SVM might just draw a clean line dividing spam from not-spam. But if the patterns are more complicated — say, certain phrases only matter when combined with others — an SVM with a Gaussian kernel can create a smooth, curved boundary that handles that complexity.

6. Advantages

- Works really well even when there are many features.
- The final model depends only on the support vectors, not the entire dataset.
- The kernel trick gives it flexibility for both linear and non-linear problems.
- Has a solid mathematical foundation — the optimization always finds a global solution.

7. Limitations

- Choosing the right kernel and parameters (C, γ) can take trial and error.
- Training can be slow for huge datasets.
- Doesn't directly give probabilities for its predictions.
- Can struggle when classes overlap heavily or data is very noisy.

8. Applications

SVMs have been used in a lot of different areas:

- **Image recognition:** handwriting and face detection.
- **Text classification:** spam filters, sentiment analysis, and topic sorting.
- **Bioinformatics:** analyzing gene expression data.

9. Conclusion

Support Vector Machines strike a nice balance between simplicity and mathematical elegance. They focus on the most important data points, build robust decision boundaries, and generalize well to new situations. Even though newer deep learning methods often get more attention, SVMs are still a reliable and interpretable choice — especially when the data is clean, structured, and not too large.