SVM and PCA

Mohammed Luqmaan Akhtar

October 2025

SVM

Support Vector Machine (SVM) is a supervised machine learning algorithm used for classification and regression tasks. It tries to find the best boundary known as hyperplane that separates different classes in the data. It is useful when you want to do binary classification like spam vs. not spam or cat vs. dog.
The main goal of SVM is to maximize the margin between the two classes. The larger the margin the better the model performs on new and unseen data.
Now, how does SVM work? Before diving straight into SVM let us look into what **Maximal Margin Classifier** is. It is a threshold that gives us the largest *margin* to make classification. But what is a *margin*? It is the distance between the threshold and the data points used to calculate the threshold. There is a more technical definition but before that I need to define two more terms.

- **Hyperplane:** A decision boundary separating different classes in feature space and is represented by the equation $wx + b = 0$ in linear classification
- **Support Vectors:** The closest data points to the hyperplane, crucial for determining the hyperplane and margin in SVM.

Thus, **margin** is the distance between the hyperplane and the support vectors.
SVM aims to maximize this margin for better classification performance. Now, we know what maximal margin classifier does, but it is not a good way to classify data. The margin defined by this method is **hard** i.e. it allows no misclassifcations and this often leads to errors. Instead we try to use a method which accounts for a certain number of outliers in the data called a **Soft Margin Classifier**. Support Vector Classifiers can easily differentiate between *linearly separable* data but struggles in its counter part. To fix this issue we use a *kernel* and the entire method is called **Support Vector Machine**
Basically this is how a **SVM** works:

1. Start with lower dimension data.
2. Transform the data to higher dimension or atleast pretend to do so.
3. Find the Support Vector Classifier threshold of the data.

A **Kernel** is a function that maps data to a higher-dimensional space enabling SVM to handle non-linearly separable data. But, the kernel only calculates the relationships between every pair of points as if they are in higher dimension, it does not actually transform the data so.
There are different types of kernel. Some main ones are:

- Polynomial Kernel
- Radial Kernel

Now to move on to the **mathematics** of SVM.
Let us consider a binary classification problem with labels $y_i \in \{-1, +1\}$ and feature vectors $x_i \in \mathbb{R}^n$, SVM aims to construct a linear classifier of the form:

$$w^T x + b = 0$$

where:

- w is the normal vector to the hyperplane (it determines the orientation),
- b is the bias term (it shifts the hyperplane).

The decision rule for a new data point x is given by:

$$f(x) = \text{sign}(w^T x + b)$$

For linearly separable data, we can express the two boundary hyperplanes as:

$$w^T x + b = +1 \quad \text{and} \quad w^T x + b = -1$$

The data points that lie exactly on these planes are called **support vectors**. For correctly classified data points, we have the constraint:

$$y_i(w^T x_i + b) \geq 1$$

This is so that there is no data points within the two margins. If we strictly followed this condition it would be a hard margin but practically we do not do so because it leads to alot of inconsistencies and errors.

The distance between two parallel hyperplanes

$$w^T x + b = +1 \quad \text{and} \quad w^T x + b = -1$$

is given by the formula for the distance between parallel planes:

$$\text{Distance} = \frac{|b_1 - b_2|}{\|w\|}$$

Substituting $b_1 = +1$ and $b_2 = -1$:

$$\text{Margin} = \frac{|(+1) - (-1)|}{\|w\|} = \frac{2}{\|w\|}$$

Thus, maximizing the margin is equivalent to minimizing $\|w\|$. To make the optimization mathematically smoother, we minimize $\frac{1}{2}\|w\|^2$ instead of $\|w\|$.

We can now formulate the optimization problem for linearly separable data as:

$$\begin{aligned} &\min_{w,b} \quad \frac{1}{2}\|w\|^2 \\ &\text{subject to:} \quad y_i(w^T x_i + b) \geq 1, \quad \forall i \end{aligned}$$

Here:

- The objective term $\frac{1}{2}\|w\|^2$ maximizes the margin.
- The constraints ensure that all points are correctly classified and lie outside or on the margin boundaries.

In real-world data, perfect separation is often impossible due to noise or overlapping classes. Hence, we introduce **slack variables** $\xi_i \geq 0$ that allow some violations:

$$y_i(w^T x_i + b) \geq 1 - \xi_i$$

Here:

- $\xi_i = 0$: point is correctly classified and outside the margin.
- $0 < \xi_i < 1$: point is inside the margin but correctly classified.
- $\xi_i > 1$: point is misclassified.

The optimization problem becomes:

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i \\ \text{subject to:} \quad & y_i(w^T x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad \forall i \end{aligned}$$

The constant $C > 0$ is a **regularization parameter** that controls the trade-off between maximizing the margin and minimizing classification error:

- Large C: higher penalty for misclassifications (harder margin).
- Small C: more tolerance for misclassifications (softer margin).

This is the basic overview of the concept and mathematics of **SVM**.

PCA

PCA reduces dimensionality of data and tells us which feature is most valueable for the data.
PCA can be achieved by two methods:

- Singular Value Decomposition
- Covariance Matrix

We take the following steps to do PCA through **SVD**:

1. Take a 2D data sample. Calculate the data center and shift the origin to it.
2. Now we try to fit a line passing through the origin to the data.
3. We start with a random line and project the data points onto the line, then, we either minimize distances from the point onto the projection or maximize distances from the the projection to the origin. Both of these conditions are the same and this can be easily proved using pythogarean theorem.
4. We find the line slope of this line. We get the proportions of each axis using the slope and then scale it so that their hypotenuse becomes one. Thus we find a unit vector. This unit vector is the eigen vector. This vector is PC1. The eigen value of this PC1 is the mean of the sum of the squared distances of the projected points about the origin.
5. PC2 is just the line perpendicular to PC1 for 2D data.
6. Now we take PC1 and PC2 as the axes and replot the points using the intersection of the perpendiculars from projections.

We take the following steps to do PCA through **Covariance Matrix**:

1. Given a dataset X with n samples and d features:

$$X = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1d} \\ x_{21} & x_{22} & \dots & x_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{nd} \end{bmatrix}$$

   First, we standardize each feature so that it has zero mean and unit variance:

$$X_{\text{std}} = \frac{X - \mu}{\sigma}$$

   where μ and σ are the mean and standard deviation of each feature.

2. Now, we calculate the covariance matrix. The diagonal elements are the variance and the non diagonal elements are the covariance. The covariance matrix captures how features vary together:

$$\frac{1}{n-1} X_{\text{std}}^T X_{\text{std}}$$

   Each element $_{ij}$ represents the covariance between feature i and feature j.

3. Find the eigen vectors and eigen values using the covariance matrix. Choose the eigen vectors with the highest eigen values to represent the major variation of the data.

4. Trasform the original data set by project them onto the eigen vectors.

$$X_{\text{PCA}} = X_{\text{std}} W$$

   where W is the set of eigen vectors which have the highest eigen values