
The AI-ML and Data Science Club of IITH

# Support Vector Machine

## Contents



**Compiled by :** Srinith

# 1 What is a Support Vector Machine?

Support Vector Machine (SVM) is one of the most popular Machine Learning Classifiers. It falls under the category of Supervised learning algorithms and uses the concept of a Margin to classify between classes. The objecttive of the SVM is to find a hyper place in an N-dimentional space(N - number of features) that distinctly classifies the datapoints. We can say that this works well for linear seperable data .This classifier is mostly used for binary classification. Our objective is to find a plane that has maximum margin , i.e maximum distance from the closest points on both sides of the plane. Support vectors are the datapoints which influence the position and orientation of the hyperplane , i.e the datapoints which are closer to the hyperplane.

# 2 Mathematical Formulation

Let us consider binary-classification at the moment and it is linearly seperable . Let the classes be represented as {-1,1}. Suppose there are m given samples , as $(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)$ , where $y_i$ belongs to {-1,1}.

Now we try to find the hyperplane which separates the two classes.

$$f(x) =< w, x > +b$$

where $w \in R^n$ and $b \in R$. The hyperplane has the normal vector $w$ and the intercept $b$ ,such that ,

$$\begin{cases} < w, x_n > +b \geq 0 & \text{when } y_n = 1 \\ < w, x_n > +b \leq 0 & \text{when } y_n = -1 \end{cases}$$

But we can find many such $w$ . So, we that's why we consider the hyperplane which has the maximum margin .

## 2.1 Geometrical Margin

For a data-point $(X^i, y_i)$ , geometrical margin $\gamma^i$ is the distance of $(X^i, y^i)$ from the decision surface.

$$\gamma^i = \frac{y_i(w^T X^i + b)}{||w||}$$

And wrt to the whole dataset it is the minimum over all the datapoints.

$$\gamma = \min_i \gamma^i$$

So, now our objective is to find $w$ and $b$ which maximise the value of geometrical margin wrt to the dataset ,i.e,

$$\max_{\gamma,w,b} \gamma$$
$$\text{s.t. } y_i(w^T X^i + b) > \gamma, ||w|| = 1$$

In other words , we want the positive examples to be further than $\gamma$ from the hyperplane and the negative examples to be further than distance $\gamma$ in negative direction .

## 3 Finding the optimal hyperplane

We have seen what has to be maximised above , we can modify it to another form .

let $x_s$ be a supporting vector on the positive side on the hyper plane .Then ,

$$x_s = x'_s + \gamma\hat{w}$$

where $x'_s$ is a point on the hyper plane and $\hat{w}$ is unit vector normal to the hyperplane . And since its a supporting vector , the distance from the hyperplane will be $\gamma$.

So , now if we take margin to be 1 rather than normalising the vector , we can say as $x'_s$ lies on the hyper plane and $x_s$ lies on the plane $\langle w, x_s\rangle + b = 1$,

$$\begin{aligned}
\langle w, x'_s\rangle + b &= 0 \\
\left\langle w, x_s - \gamma\frac{w}{\|w\|}\right\rangle + b &= 0 \\
\langle w, x_s\rangle + b + \left\langle w, -\gamma\frac{w}{\|w\|}\right\rangle &= 0 \\
\left\langle w, \gamma\frac{w}{\|w\|}\right\rangle &= 1 \\
\gamma &= \frac{1}{\|w\|}
\end{aligned}$$

So , we can modify the condition to ,

$$\begin{aligned}
&\min_{w,b} \frac{1}{2}||w||^2 \\
&\text{s.t. } y_i(w^T X^i + b) > 1
\end{aligned}$$

This is a standard quadratic programming optimization problem that can be solved by using Lagrange multipliers. We can define a single loss function as,

$$L(w, b, \alpha) = \frac{1}{2}||w||^2 - \sum_{i=1}^{n} \alpha_i[y_i(\langle w, x_i\rangle + b) - 1]$$

Setting $\nabla L(w, b, \alpha) = 0$, we get

$$\begin{aligned}
\frac{dL(w,b,\alpha)}{dw} &= 0 \\
\frac{d\left(\frac{1}{2}||w||^2 - \sum_{i=1}^{n}\alpha_i[y_i(\langle w, x_i\rangle + b) - 1]\right)}{dw} &= 0 \\
w &= \sum_{i=1}^{n} \alpha_i y_i x_i
\end{aligned}$$

$$\begin{aligned}
\frac{dL(w,b,\alpha)}{db} &= 0 \\
\sum_{i=1}^{n} \alpha_i y_i &= 0
\end{aligned}$$

# 4 Soft Margin SVM

When we consider the dataset which is not linearly sepeable , we accept misclassifications.Soft margin SVM takes care of optimising this.To minimize the error, we should define a loss function. A common loss function used for soft margin is the hinge loss.

$$max\{0, 1 - y_i(\langle w, x_i \rangle + b)\}$$

A new regularization parameter C controls the trade-off between maximizing the margin and minimizing the loss.The problem for soft max gets converted to,

$$\min_{w,b} \frac{1}{2}||w||^2 + C\sum_{i=1}^{n} \eta_i$$
$$\text{s.t. } y_i(w^T X^i + b) > 1 - \eta_i$$

We change the loss function a bit , bringing the hingeloss function

$$L(w, b, \alpha) = \frac{1}{2}||w||^2 + \frac{C}{N}\left[\sum_{i=1}^{n} max(0, \alpha_i[1 - y_i(\langle w, x_i \rangle + b)])\right]$$

where $C$ is the regularisation parameter , larger $C$ gives a narrow margin and smaller $C$ results in the wider margin.

Now, as usual we can use SGD to find the optimal weights .

# 5 Questions

**Subjective :**

1. What is the significance of the threshold term b?
2. Is SVM more memory efficient that other algorithms? Give reason.
3. What is the condition on weight vector for functional and geometrical margin to be equal?
4. What are the disadvantages of using SVM?
5. Can SVMs be used for multi-class classification? If yes , how?

**Objective :**

1) Scaling W (weight vector) changes the value of geometrical margin.

   a) True
   b) False

2) Feature scaling and outlier removal is not required in SVM.

   a) True
   b) False

3) For a given feature vector x of dimension n, polynomial kernel function can calculate a polynomial of order m from x in _____ time complexity.

a) O(n)

b) O($n^2$)

c) O($n^m$)

d) O(nm)

## 6 SVM with python

SVM classifier code snippets

```
    import numpy as np
```

```
    # SVM soft margin classifier
class SVM:
    def __init__(self, C=1.0):
        self.C = C
        self.weights = None
        self.bias = None

    def train(self, X, y, learning_rate=0.01, num_epochs=1000):
        num_samples, num_features = X.shape

        # Initialize weights and bias
        self.weights = np.zeros(num_features)
        self.bias = 0

        # Gradient descent training
        for epoch in range(num_epochs):
            # Compute margin and loss
            margin = y * (np.dot(X, self.weights) + self.bias)
            loss = 1 - margin

            # Apply hinge loss function
            loss = np.maximum(0, loss)

            # Compute the gradient
            dW = self.weights - self.C * np.dot(X.T, y * (loss > 0))
            dB = -self.C * np.sum(y * (loss > 0))

            # Update weights and bias
            self.weights -= learning_rate * dW
            self.bias -= learning_rate * dB

    def predict(self, X):
        # Predict class labels
        y_pred = np.sign(np.dot(X, self.weights) + self.bias)
        return y_pred.astype(int)
```

```
    svm = SVM(C = 1)
    svm.fit(X_train, y_train)
    y_pred = svm.predict(X_test)
```