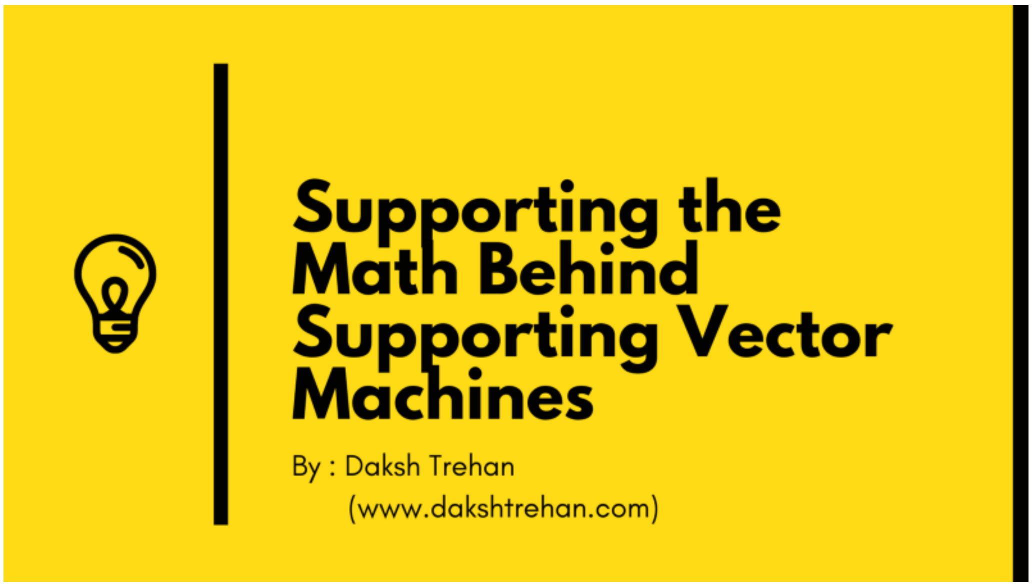
Machine Learning, Mathematics

# Supporting the Math Behind Supporting Vector Machines!

**A quick tour to SVM constituting mathematical & theoretical explanation along with from the scratch implementation.**



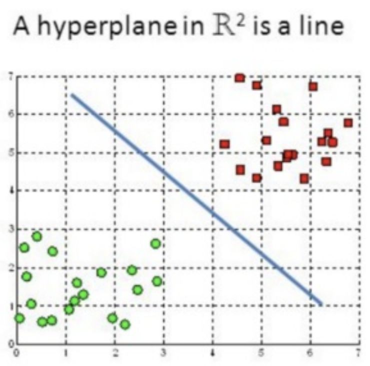
**A support vector machine** is another simple algorithm that every machine learning expert should have in his/her arsenal. The support vector machine is highly preferred by many as it produces data with remarkable accuracy demanding less computation power. SVM can be used for both **regression and classification tasks.**

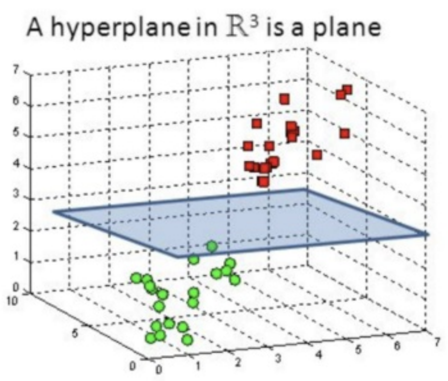
## Table of Content

### ***Introduction to Support*** **Vector Machine**

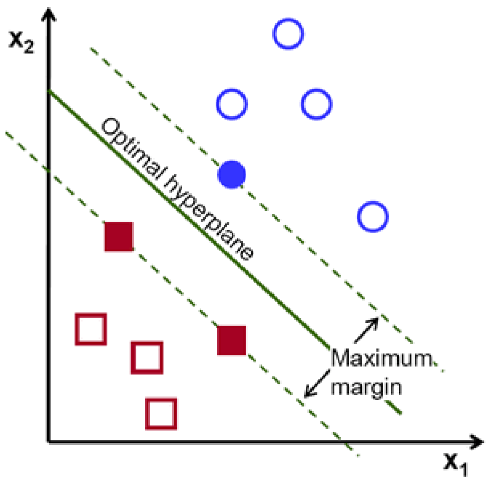
Support Vector Machine(SVM) is a powerful classifier that works with both **linear and non-linear data.**

> If you have a **n**-dimensional space, then the dimension of the hyperplane will be **(n-1).**





The **goal of SVM** is to find an **optimal hyperplane** that best separates our data so that distance from the nearest points in space to itself is maximized.

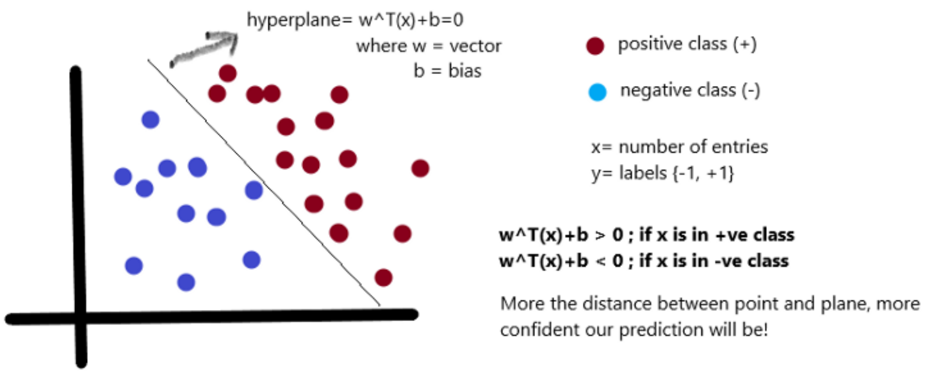


To keep it simple, consider a road, which separates the left, right-side cars, buildings, pedestrians and makes the widest lane as possible. And those cars, buildings, really close to the street are the support vectors.

This is how the Support Vector machine works.

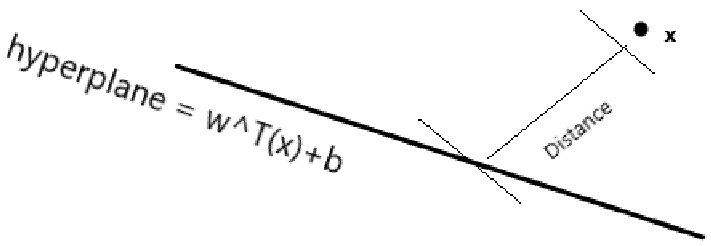
**How SVM works?**

**Goal**: To find an optimal hyperplane.


hyperplane= w^T(x)+b=0
where w = vector
b = bias
● positive class (+)
● negative class (-)
x= number of entries
y= labels {-1, +1}
w^T(x)+b > 0 ; if x is in +ve class
w^T(x)+b < 0 ; if x is in -ve class
More the distance between point and plane, more confident our prediction will be!

$$y_{pred} = g\left(w^T x^{(i)} + b\right)$$
$$g(z) = +1 \, ; \, if \; z \geqslant 0$$
$$g(z) = -1 \, ; \, if \; z < 0$$

## Computing distance of point from Supporting vector



$$Distance = \left( \frac{w^T x^{(i)} + b}{\sqrt{w_1^2 + w_2^2 + \ldots\ldots\ldots + w_n^2}} \right)$$

We want a hyperplane that can classify points in a very efficient manner i.e. allocating the classes in a precise manner.

To execute that precisely, we need to alter our goal.

**Modified Goal:** To maximize minimum distance from the hyperplane.

$$r^i = \left( \frac{w^T x^{(i)} + b}{||w||_2} \right) \quad where \; ||w||_2 = \sqrt{w_1^2 + w_2^2 + \ldots\ldots\ldots w_n^2}$$

Let us assume a **minimum threshold distance = γ**

$$\gamma \; = \; min_{i=1,2....m} \;\; r^{(i)}$$

Now all points must satisfy following two conditions:

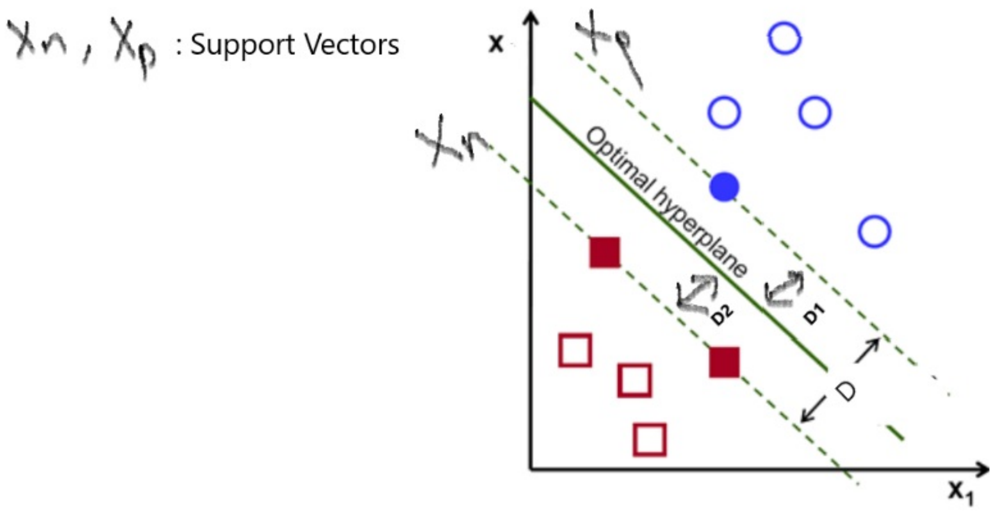
$$\begin{aligned} &1.\ minimum\ \gamma \\ &2.\ if\ w^T x + b > 0\ ;\ belongs\ to\ +ve\ class \\ &\quad\ if\ \ w^T x + b < 0\ ;\ belongs\ to\ -ve\ class \end{aligned}$$

In order to achieve minimum distance, we need to **maximize "w".**

$$\begin{aligned} &max\ _{r,w,b}\ such\ that: \\ &\quad y^{(i)}\left(w^T x + b\right)\ \geqslant\ r \\ &\qquad for\ i =\ 1, 2, 3, .... m \\ &\qquad where\ y^{(i)}\ =\ class\ label\ i.e.\ \ +/ \end{aligned}$$

This execution is a little tedious and cumbersome, therefore to reduce mathematical complexities we will reformulate our equations.

So, we will re-normalize our data in such a way that support vectors must lie on the hyperplane.



So, in order to **maximize "w", we need to maximize "D".**

D = D1+D2; so we need to maximize D1 and D2 too.

$$D1 = \left(\frac{w^T x^{(i)} + b}{||w||_2}\right) = \left(\frac{1}{||w||_2}\right); \quad w^T x^{(i)} + b = +1$$

$$D2 = \left(\frac{w^T x^{(i)} + b}{||w||_2}\right) = \left(\frac{1}{||w||_2}\right); \quad w^T x^{(i)} + b = -1$$

$$D = D1 + D2$$

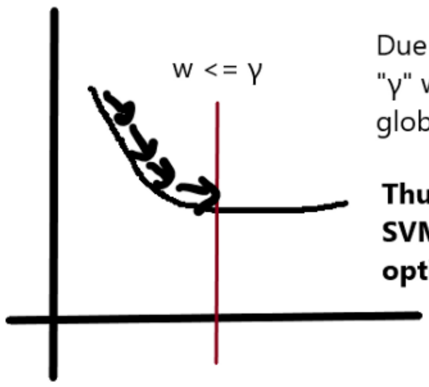
$$D = \left(\frac{2}{||w||_2}\right)$$

Now, in order **to increase "D", we must focus on decreasing "||w||"** under the condition that the **minimum distance must be 1**.

$$minimize \ \frac{1}{||w||_2} \quad where \ ||w||^2 = w^T w$$

$$such\ that : y^i\left(w^T x^{(i)} + b\right) \geqslant 1 \ \& \ \gamma = 1$$

In our new goal, we need to decrease ||w|| but we need to keep in mind that ||w|| isn't free i.e. **it depends upon γ.**



Due to interdependency of "w" on "γ" we won't be able to get to global minima.

**Thus, we can conclude SVM is constraint convex optimization problem.**

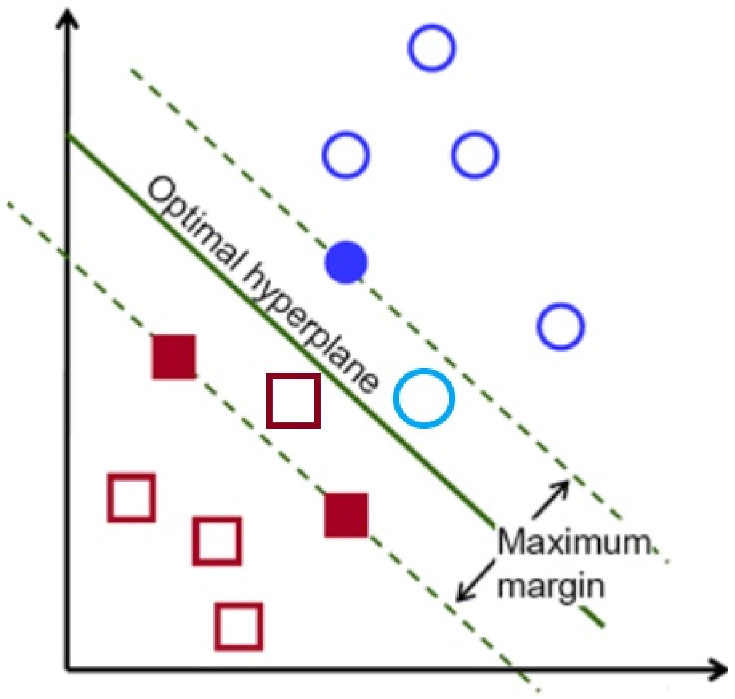
Since SVM **can't employ Gradient Descent**, due to constraint dependency, we need to look for other solutions.

The problem can be avoided using any of the following method:

- *Quadratic Solver*
- *Langragian Density*
- *PEGASOS*

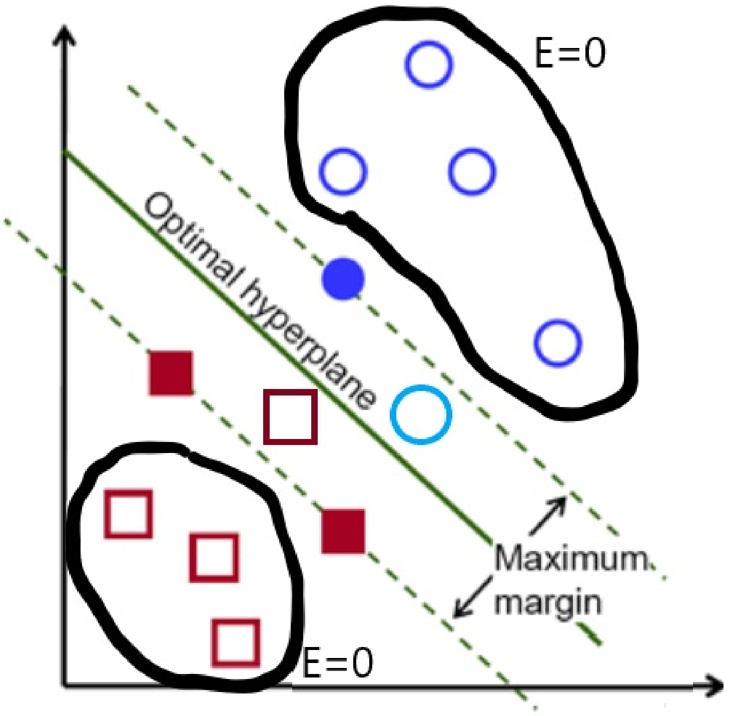
**Handling Outliers**

Sometimes for some examples **(x(i), y(i))** we might encounter error as **E(i)**.



The error will completely demolish our goal as adding error will increase the cost of loss but to develop a robust model we need to allow some error or otherwise, it will lead to *overfitting*.

But to keep the error as low as possible, we will introduce a penalty, the moment our model wrongly classify a data point it will face some **penalty(c)** that will help us to increase our accuracy.



If E=0 ; then no penalty would be fined.

$$min_{w,b} \left(\frac{1}{2}\right) w^T w + c\sum_{i=1}^{m} E_i$$
$$y^i\left(w^T x^{(i)} + b\right) \geqslant 1 - E_i$$
$$where\ E_i = error\ for\ x_i\ \&\ y_i$$
$$c = amount\ of\ penalty$$

if c is very large ; then less margin but better classification (more prone to overfitting)

if c=1 i.e. very low ; then margin is maximized but at expense of misclassification (better classifier)

**PEGASOS**

To **overcome the constraint convex optimization problem**, we employ *the PEGASOS* method.

We reformulate our equations to achieve constraint independency.

$$y^i\left(w^T x^{(i)} + b\right) \geqslant 1 - E_i$$
$$where\ E_i \geqslant 1 - y^i\left(w^T x^{(i)} + b\right)$$

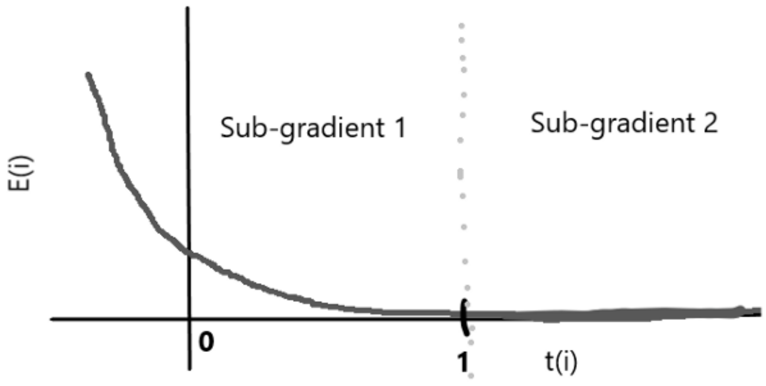
Now if we talk mathematically, the effect on *1* will be *nil* on the equation as we are adding it once and contrastingly subtracting it once.

To ease our equations, we will substitute t(i).

$$\begin{aligned} E_i &\geqslant 1 - t_i \;;\; if\ t_i \leqslant 1 \\ E_i &< 0 \qquad ;\; if\ t_i \leqslant 1 \\ &\qquad\qquad where\ 1 = w^T x^{(i)} + b \end{aligned}$$

The above equation corresponds that if point is away from hyperplane the error will be zero otherwise the error encountered will be (1-t(i)).

$$E_i = max(0, 1 - t_i)$$



The formulated function is still **not differential at t≥1** ; so we are going to divide our gradient in **Sub-gradient 1** & **Sub-gradient 2.**

Our final function that needs to be **minimize** is:

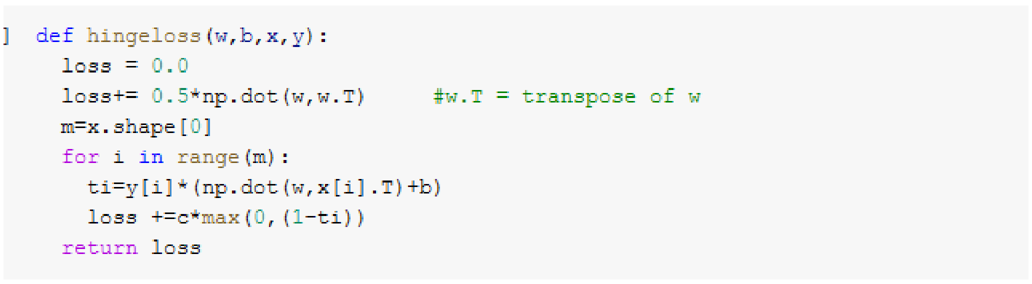
$$loss = min_{w,b} \left(\frac{1}{2}\right) w^T w + c \sum_{i=1}^{m} max(0,\ 1 - t_i)$$
$$where\ t_i = y^i \left(w^T x^{(i)} + b\right)$$

Now, since the constraints are removed and our function is independent of γ. We can employ Gradient Descent to minimize the loss.

$$w = w - \eta\delta_w loss$$
$$b = b - \eta\delta_b loss$$

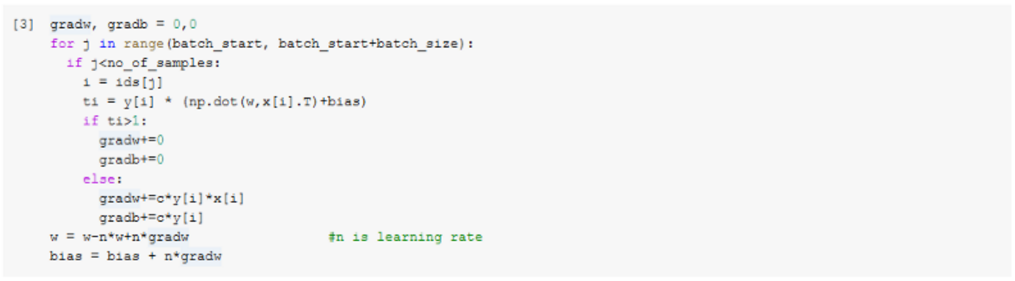
$$\delta_w loss = w + c.a.(y_i x_i)$$
$$\delta_b loss = 0 + c.a.y_i$$
$$where\ c = 0;\ if\ t_i \geqslant 1$$
$$1\ ;\ if\ t_i < 1$$

**Minimizing loss using Gradient Descent**



```
] def hingeloss(w,b,x,y):
    loss = 0.0
    loss+= 0.5*np.dot(w,w.T)     #w.T = transpose of w
    m=x.shape[0]
    for i in range(m):
      ti=y[i]*(np.dot(w,x[i].T)+b)
      loss +=c*max(0,(1-ti))
    return loss
```

Calculating loss

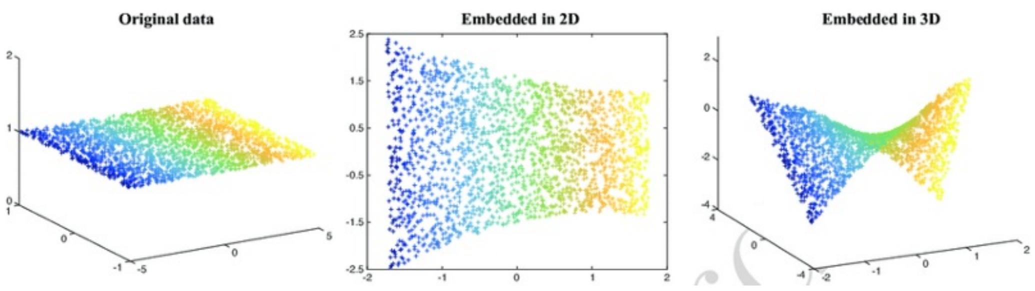


```
[3] gradw, gradb = 0,0
    for j in range(batch_start, batch_start+batch_size):
      if j<no_of_samples:
        i = ids[j]
        ti = y[i] * (np.dot(w,x[i].T)+bias)
        if ti>1:
          gradw+=0
          gradb+=0
        else:
          gradw+=c*y[i]*x[i]
          gradb+=c*y[i]
    w = w-n*w+n*gradw                    #n is learning rate
    bias = bias + n*gradw
```

Implementing Gradient Descent for SVM

**Non-Linear Classification**

To classify non-linear data using the Support vector machine, we need to project data to higher dimensions i.e. convert 2D data to 3D data by increasing its features.



Source

This increment in features will be computationally expensive.

To transform dimensions of data, we use kernels.

- *Linear Kernel*
- *RBF (Radial Basis)Kernel*

$$k(x_i, x_j) = \exp\left(-\frac{d\left(\frac{x_i}{l}, \frac{x_j}{l}\right)^2}{2}\right)$$

- *Polynomial Kernel*

$$k(\mathrm{x}_i, \mathrm{x}_j) = (\mathrm{x}_i.\mathrm{x}_j)^d$$

- *Sigmoid Kernel*

$$K(x, y) = \tanh\left(k_1 x^T y + k_2\right)$$

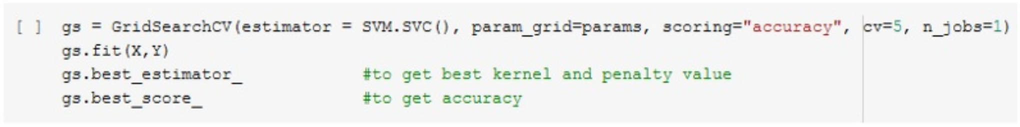
The hurdle with kernel trick is choosing the right kernel for your model, because *we never know which kernel will be best suited to our model and at what cost(penalty).*

So to auto-tune our hyperparameters, we will implement **GridSearch**.

We will specify params i.e. list consisting of dictionary with keys as kernels and penalty, we'll run our model for every kernel and get the best accuracy.

> params = [{'kernel':['linear', 'rbf', 'poly', 'sigmoid'], 'c':[0.1, 0.2, 0.5, 1.0, 2.0, 5.0]}

Now for each type of kernel and 5 types of penalty we will store accuracies and later will choose kernel and penalty(c) with best accuracy.



```
[ ]  gs = GridSearchCV(estimator = SVM.SVC(), param_grid=params, scoring="accuracy", cv=5, n_jobs=1)
     gs.fit(X,Y)
     gs.best_estimator_            #to get best kernel and penalty value
     gs.best_score_                #to get accuracy
```

*Parameters accepted:*

- **cv**: cross-validation
- **n_jobs**: number of CPU available

**Pros & Cons of Support Vector Machine**

## Pros

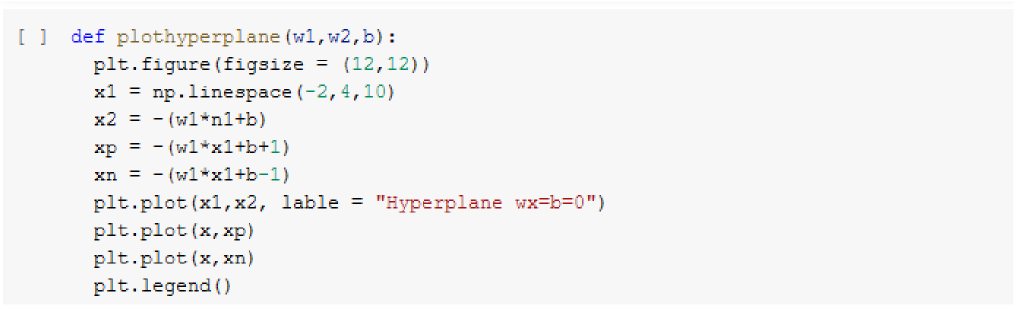
- Can easily handle large feature spaces.
- Kernel trick is real strength of SVM as it helps to find solution to even complex problems.
- Works well with both Linear and non-linear data.

- Less prone to overfitting.
- Reliable even for unstructured data.

### Cons

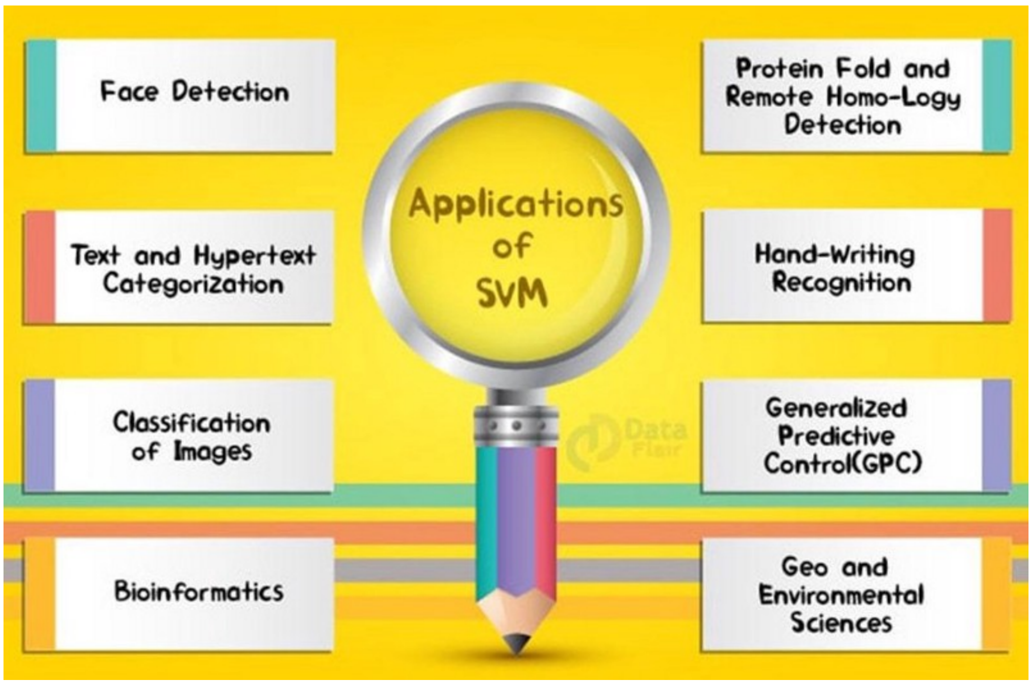
- Sensitive to noise.
- Choosing an optimal kernel is really tough.
- Long training time for Large Dataset.

**Visualizing SVM**



```
[ ]  def plothyperplane(w1,w2,b):
       plt.figure(figsize = (12,12))
       x1 = np.linespace(-2,4,10)
       x2 = -(w1*n1+b)
       xp = -(w1*x1+b+1)
       xn = -(w1*x1+b-1)
       plt.plot(x1,x2, lable = "Hyperplane wx=b=0")
       plt.plot(x,xp)
       plt.plot(x,xn)
       plt.legend()
```

**Applications of SVM**


Face Detection
Text and Hypertext Categorization
Classification of Images
Bioinformatics
Applications of SVM
Protein Fold and Remote Homo-Logy Detection
Hand-Writing Recognition
Generalized Predictive Control(GPC)
Geo and Environmental Sciences
Data Flair

Source: DataFlair

## Conclusion

Hopefully, this article will help you to understand about Support Vector Machine(SVM) in the best way possible and also assist you to its practical usage.

As always, thanks you much for reading, and please share this article if you found it useful!

Feel free to connect:

Join me at ~ www.dakshtrehan.com

*LinkedIN ~ https://www.linkedin.com/in/dakshtrehan/*

*Github ~ https://github.com/dakshtrehan*

Follow for further Machine Learning/ Deep Learning blogs.

*Medium ~ https://medium.com/@dakshtrehan*

## Want to learn more?

Detecting COVID-19 Using Deep Learning

The Inescapable AI Algorithm : TikTok

Why are YOU responsible for George Floyd's Murder and Delhi Communal Riots?

Why Random Forest and not Decision Tree?

Clustering : What it is? When to use it?

Start off your ML Journey with k-Nearest Neighbors

Naive Bayes Explained

Activation Functions Explained

Parameter Optimization Explained

Gradient Descent Explained

Logistic Regression Explained

Linear Regression Explained

Determining Perfect Fit for your ML Model

*Cheers!*

By Daksh Trehan on June 30, 2020.

Canonical link

Exported from Medium on July 15, 2020.