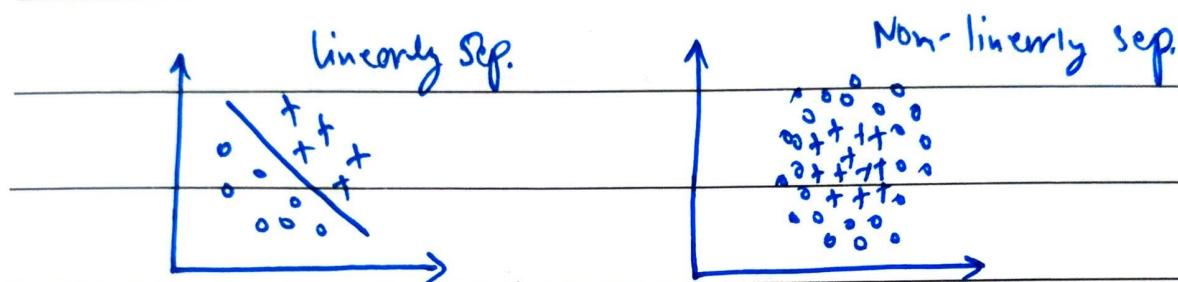
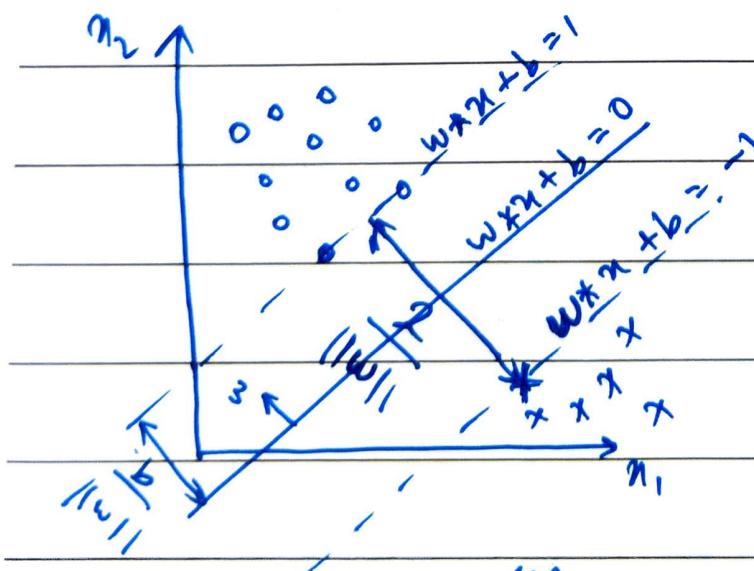


## Support Vector Machines (SVM)

- \* SVM is a powerful classifier that works both on linearly & non-linearly separable data.



- \* finds an optimal hyperplane, that best separates our data so that the distance from nearest points in space to itself (also called margin) is maximized.
- \* These nearest points are called support vectors.



- \* for non-linearly <sup>sep.</sup> data, it uses 'Kernel Trick'

what is hyperplane?

It Hyperplane is plane of  $n-1$  dimensions in  $n$  dimensional feature space, that separates the two classes. For a 2-D feature space, it would be a line & for a 3-D feature space, it would be a plane and so on.

A hyperplane is able to separate classes if for all points:

$$w_n + b > 0 \quad (\text{for data points in class 1})$$

$$w_n + b < 0 \quad (\text{for data points in class 0})$$

(Note:  $w_n + b$  is distance of a pt. from plane)

Goal of SVM: find hyperplane that separates data with max. margin from closest points.

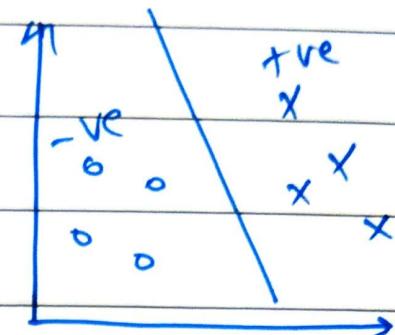
\* Example's Say you have a binary classification.

$$X = \{x_1, x_2, \dots, x_m\}$$

$$y = \{y_1, y_2, \dots, y_m\}$$

Each  $x_i$  has  $n$  no. of features.

$$y_{(i)} \in \{-1, 1\}$$



Key Idea: Separate data with max. margin.

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

$$\left[ \theta_0 + \underbrace{\theta_1 n_1^{(1)} + \theta_2 n_1^{(2)} + \theta_3 n_1^{(3)} + \dots}_{w^T x} \right]$$

$$w = \begin{bmatrix} w_0 \\ \vdots \\ w_n \end{bmatrix}, \quad x = \begin{bmatrix} x_0 \\ \vdots \\ x_n \end{bmatrix}$$

Optimal Hyperplane:

\*  $w^T x + b > 0$  if  $x^{(i)} \in$  +ve class

$w^T x + b < 0$  if  $x^{(i)} \in$  -ve class

\* Predictions should be wrong & confident.

More distance of point from hyperplane

means more confident why if we change

hyperplane then large distance might not

change the class.

$$y_{\text{pred}} = g(w^T n + b)$$

$$g(z) = \begin{cases} +1 & \text{if } z \geq 0 \\ -1 & \text{if } z < 0 \end{cases}$$

\* Unlike logistic, we ~~will~~ will not compute probabilities.

\* How to find distance

$$\text{dis} = \frac{w^T n + b}{\sqrt{w_1^2 + w_2^2 + \dots}}$$
$$= \frac{w^T n + b}{\|w\|_2}$$
$$(L_2 \text{ norm})$$

$L_2$  norm is Euclidean distance of point from origin.

goal : Maximize the minimum distance of point from the hyperplane.

$$\text{Distance, } \gamma^{(i)} = \frac{\mathbf{w}^T \mathbf{x}^{(i)} + b}{\|\mathbf{w}\|_2}$$

$$\text{Min distance, } \gamma = \min_{i=1 \dots m} \gamma^{(i)}$$

So, all the points should have atleast  $\gamma$  distance.

optimization / sum objective :

find max value of  $\gamma$  [  $\max_{\gamma, \mathbf{w}, b}$  ]

such that  $y^{(i)} * (\mathbf{w}^T \mathbf{x}^{(i)} + b) \geq \gamma$  for all

$i = 1 \dots m$ .

(This just gives absolute distance.)

$-1 \neq d = \text{true}$

→ This is very hard to solve (non-convex func)

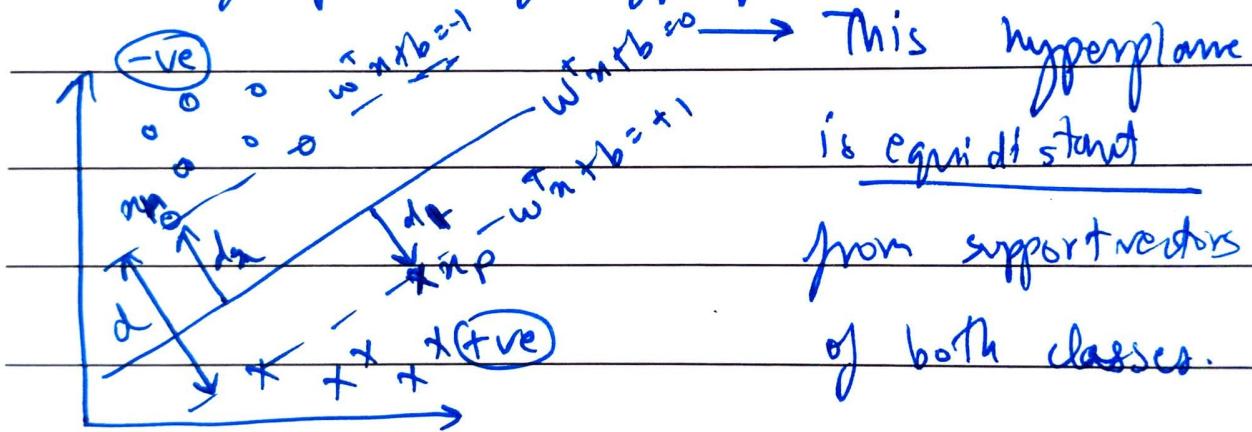
∴ Hence we will convert it to a simple solvable way.

function has many local minima/ maxima, hence cannot find global answer.

## Objective as Constrained :

Reformation : We re-normalise our data set such that the support vectors (points closest to the hyperplane) should always lie on the hyperplane.

Using support vectors we maximize the margin of the classifier. Deleting support vectors will change position of hyperplane.



Goal : Maximize  $d$  such that  $y_i \cdot w^T n^{(i)} + b \geq \gamma$

$$d_1 = \frac{|w^T n_p + b|}{\|w\|}, \quad d_2 = \frac{|w^T n_n + b|}{\|w\|}$$

$$d_1 = \frac{1}{\|w\|}, \quad d_2 = \frac{1 - 1}{\|w\|} = \frac{1}{\|w\|}$$

$$d = d_1 + d_2 = \frac{2}{\|w\|}$$

minimize this

## svm objective 2:

$$\text{Minimize} \quad \frac{1}{2} \|w\|$$

where all points have  
min distance of  $\frac{1}{\|w\|}$ .

$$\Rightarrow \min \left\{ \frac{1}{2} \|w\|^2 \right\}$$

such that  $y_i \frac{(w^T n^{(i)}) + b}{\|w\|} \geq 1$

*Final sum objective*

*Convex*

*Linear constraint*

$$\sum_{i=1 \dots m} y_i (w^T n^{(i)} + b) \geq 1$$

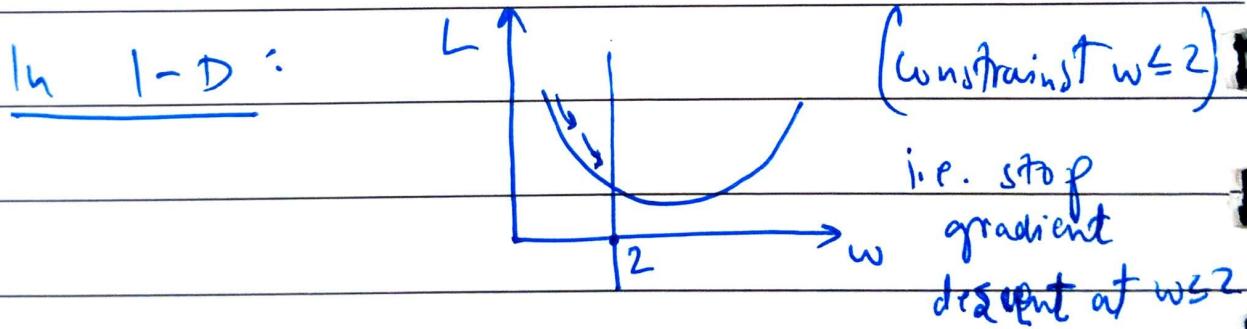
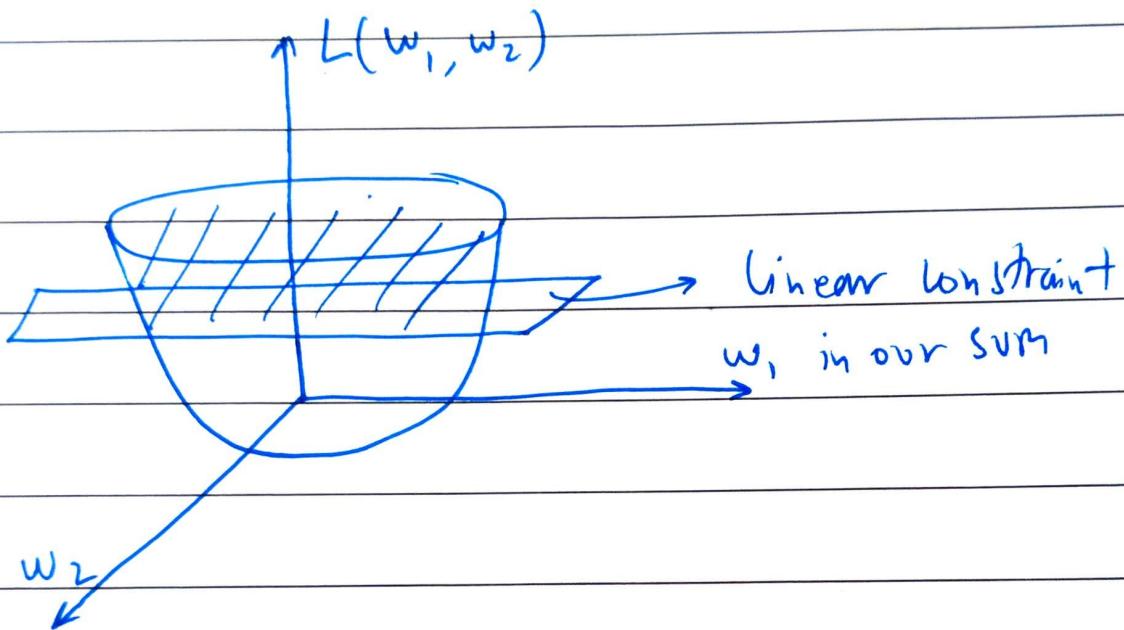
To simplify, we can take square of  $\|w\|_2$ .

$$\left. \begin{aligned} \|w\|_2^2 &= w^T w \\ \nabla_w w^T w &= w \quad // \text{Derivative} \end{aligned} \right\}$$

→ This can be solved by:

- ① Quadratic Solvers
  - ② Lagrangian Duality (Andrew Ng)
  - ③ Pegasos (converts constrained convex problem  
into unconstrained problem).

Meaning of convex optimisation problem with some constraint.



### Handling outliers in SVM:

- \* It is possible for dataset not to be perfectly linearly separable due to presence of outliers.



Allow our objective  
to do some error on  
some of the training  
examples.

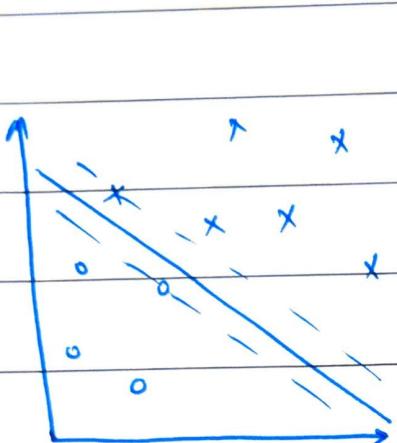
$$(n_i, y_i) \rightarrow \varepsilon^{(i)} \quad // \text{error for each } n^{(i)}$$

Primal  
objective

$$\left\{ \begin{array}{l} \frac{1}{2} w w^T + c \sum_{i=1}^m \varepsilon^{(i)} = \text{loss} \\ y^{(i)} + (w^T n^{(i)} + b) \geq 1 - \varepsilon^{(i)} \end{array} \right.$$

Allow some  
error.

- \* Some points can also have margin ~~< 1~~ < 1  
↳ even -ve (outlier on opp side).
- \*  $c$  controls amount of penalty to give when error occurs.

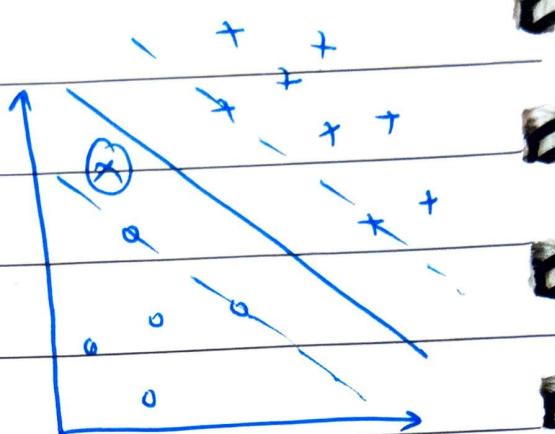


$C \rightarrow \infty$  (very large)

No misclassification as  
for loss to be minimum,  
all  $\xi^{(i)}$  should be 0 in  
this case. [ $\because C \xi^{(i)}$ ]

But margin is less.

\* More prone to overfitting.



$C \rightarrow \text{small}$

Some misclassification  
for outliers but  
margin is max.  
Better.

Try different values  
of  $C$  when working.

## Pegasos Algorithm for Unconstrained optimisation.

We did: Non-convex  $\rightarrow$  convex with constraints

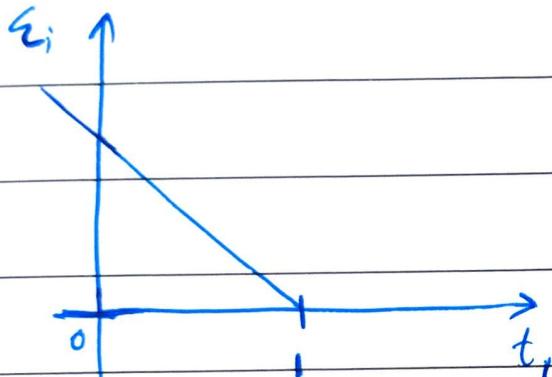
↓  
convex with constraints +  
outliers

Now, we convert it is unconstrained convex  
optimization so that we can apply gradient  
descent, using pegasos.

$$\textcircled{1} \quad \varepsilon^{(i)} \geq 1 - y^{(i)} (\mathbf{w}^T \mathbf{x}^{(i)} + b) \quad // \text{from constraint}$$

$$\textcircled{2} \quad \varepsilon^{(i)} \geq 0$$

$$\boxed{\varepsilon^{(i)} = \max(0, 1 - t_i)}$$



$$\begin{aligned} \varepsilon_i &= 1 - t_i, \text{ if } t_i \leq 1 \\ &= 0, \text{ if } t_i > 1 \end{aligned}$$

If you move away from a (say) positive hyperplane to -ve hyperplane,

Your error distance will increase, & error will be -ve.

Loss / Sum objective: (Now unconstrained)

$$L = \min_{\mathbf{w}, b} \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{i=1}^m \max(0, 1 - t_i)$$

$$\text{where, } t_i = y_i * (\mathbf{w}^T \mathbf{x}^{(i)} + b)$$

\* If point lies in same class &  $t_i \geq 1$ , then  $\varepsilon = 0$ .

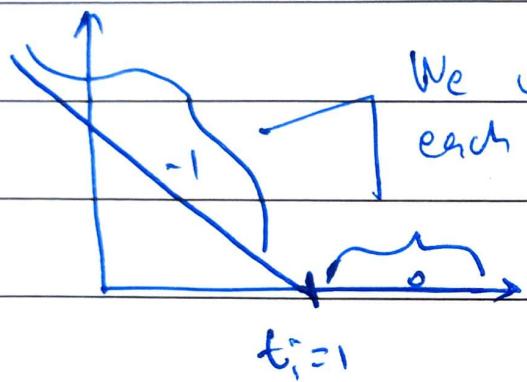
\*  $t \leq 1$  when point lies behind the line & here  $t_i$  means pt. moving in opp direction hence  $\varepsilon \uparrow$ .

## Gradient Descent:

$$w = w - n \nabla_w L \quad \rightarrow \text{learning rate}$$

~~$b = b - n \nabla_b L$~~

\* Func is not differentiable at  $t_i=1$ .



We will define 2 gradients for each point.

// Also called Hinge-loss

\* This technique is called sub-gradient.

$$\nabla_w L = w + c \leq \nabla_w [\max(0, 1-t_i)]$$

$$= w + c \leq \frac{\partial f}{\partial t} \cdot \frac{\partial t}{\partial w} \quad // \text{chain rule}$$

$f = \max(0, 1-t_i)$

$$\nabla_w L = w + c \leq \begin{bmatrix} 0 & \text{if } t_i \geq 1 \\ -1 & \text{if } t_i < 1 \end{bmatrix} \quad \Rightarrow y^{(i)} \in \mathbb{R}$$

$$t_i = y^{(i)}(w^T x^{(i)} + b)$$

$$\nabla_w t_i = y^{(i)} x^{(i)}$$

$$\nabla_b L = 0 + c \begin{bmatrix} 0 & \text{if } t_i \geq 1 \\ -1 & \text{if } t_i < 1 \end{bmatrix} y^{(i)}_*$$

$$\begin{aligned} \nabla_b t_i &= \nabla_b y^{(i)} \cdot (w^T n^{(i)} + b) \\ &= y^{(i)*} \end{aligned}$$

$$w = w - \eta \left[ w + c \begin{bmatrix} 0 & t_i \geq 1 \\ -1 & t_i < 1 \end{bmatrix} n_i y_i \right]$$

$$w = w - \eta w + \begin{bmatrix} 0, & t_i \geq 1 \\ \eta c(y_i; n_i), & t_i < 1 \end{bmatrix}$$

$$b = b - \eta \left[ c \begin{bmatrix} 0, & t_i \geq 1 \\ -1, & t_i < 1 \end{bmatrix} y_i \right]$$

II. Final formula for weight update rule.