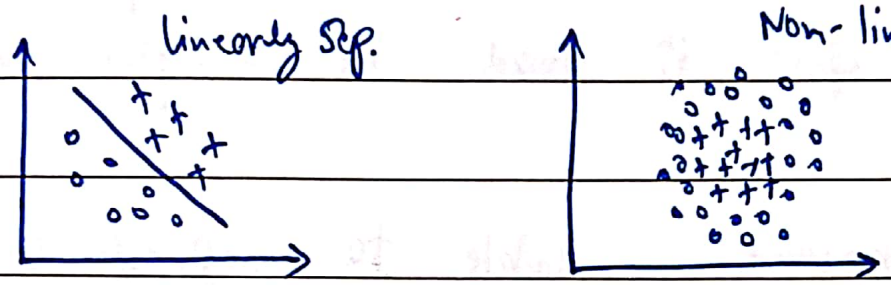
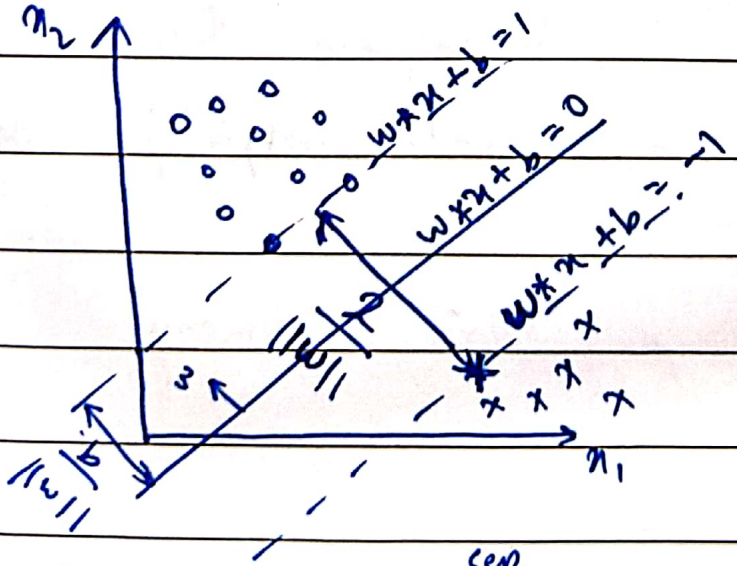
Support Vector Machines (SVM)

* 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 sep. data, it uses 'Kernel Trick'

What is hyperplane?

A 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:

$wx + b > 0$ (for data points in class 1)

$wx + b < 0$ (for data points in class 0)

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

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

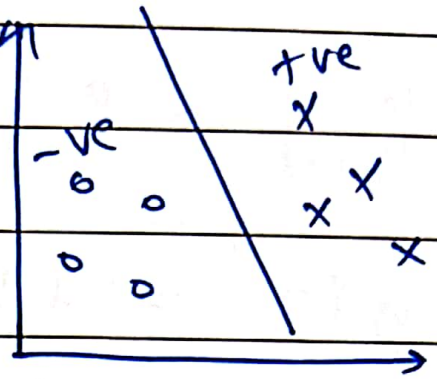
* **Example:** Say you have a binary classification.

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

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

Each x_i has n no. of features.

$y^{(i)} \in \{-1, 1\}$



Key Idea: Separate data with max. margin.

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

$$\theta_0 + \theta_1 x_1^{(1)} + \theta_2 x_2^{(2)} + \theta_3 x_1^{(3)} + \dots$$

$\theta_0 \rightarrow b$

$\theta_1 x_1^{(1)} + \theta_2 x_2^{(2)} + \theta_3 x_1^{(3)} + \dots \rightarrow w^T x$

$$w = \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

Optimal Hyperplane :

$x^{(i)} \rightarrow i^{th}$ example

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

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

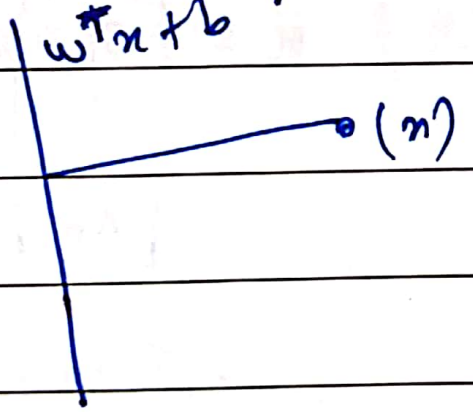
* Predictions should be correct & confident- more distance of point from hyperplane means more confident coz if we change hyperplane then large distance might not change the class.

$$y_{pred} = g(w^T x + b)$$

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

* Unlike logistic, we will not compute probabilities.

* Now to find distance



$$dis = \frac{w^T x + b}{\sqrt{w_1^2 + w_2^2 + \dots}}$$

$$= \frac{w^T x + b}{\|w\|_2}$$

(L_2 norm)

L_2 norm is Euclidian distance of point from origin.

Goal : Maximise the minimum distance of point from the hyperplane.

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

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

So, all the points should have atleast γ distance.

Optimization / sum objective :

find max value of γ $[\max_{\gamma, w, b}]$ such that $y^{(i)} * (w^T x^{(i)} + b) \geq \gamma$ for all $i = 1 \ldots m$.

(This just gives absolute distance.)

$-1 * -d = +ve$

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

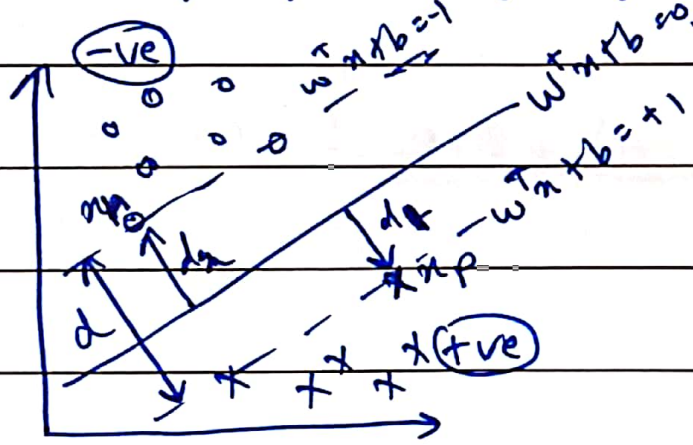
* Hence we will convert it to a simple solvable way.

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 hyper plane.

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



→ This hyperplane is equidistant from support vectors of both classes.

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

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

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

$$\boxed{d = d_1 + d_2 = \frac{2}{||w||}} \leftarrow \text{Minimize this}$$

✦ SVM objective 2:

Minimize $\frac{1}{2}||w||$ where all points have min distance of $1/||w||$.

$\Rightarrow$ min $\frac{1}{2}||w||^2$ such that $y_i \frac{(w^T x^{(i)} + b)}{||w||} \geq \frac{1}{||w||}$

Final SVM objective

Convex

$\forall i=1..m$, $y_i (w^T x^{(i)} + b) \geq 1$

Linear constraint

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

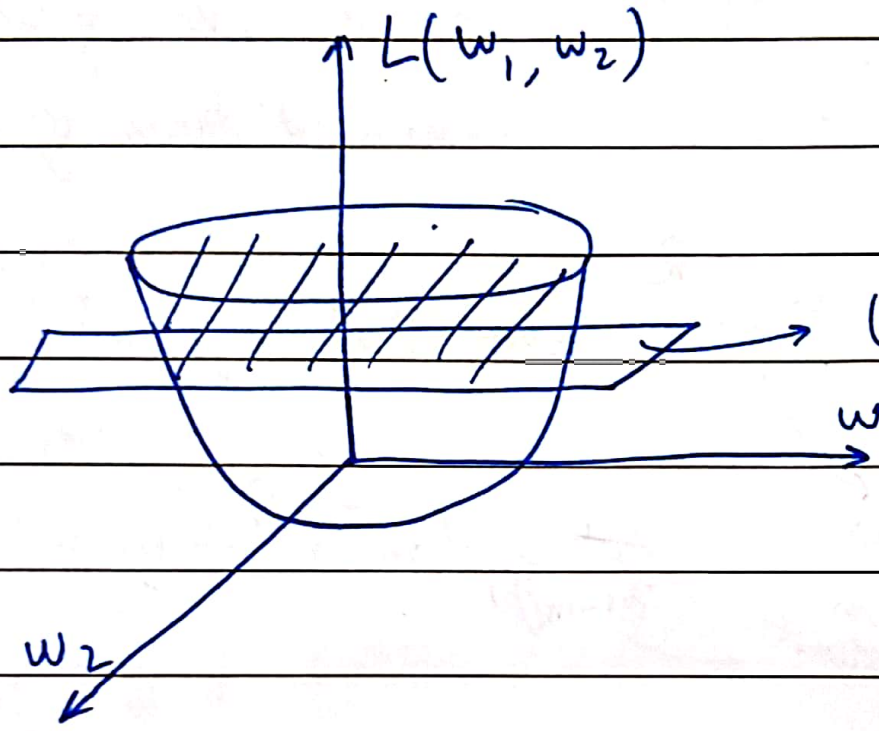
$$||w||_2^2 = w^T w$$

$$\nabla_w w^T w = w \quad \text{// Derivative}$$

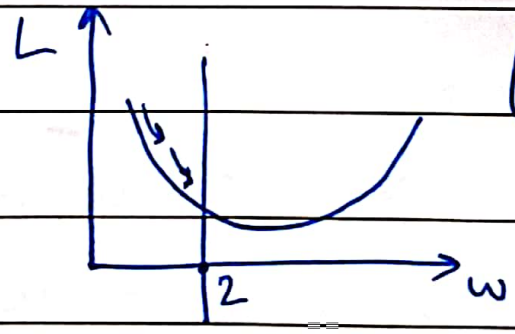
→ This can be solved by:

1. Quadratic Solvers
2. Langrajian Duality (Andrew NG)
3. Pegasos (converts constrained convex problem into unconstrained problem).

Meaning of Convex optimisation problem with some constraint.



In 1-D:

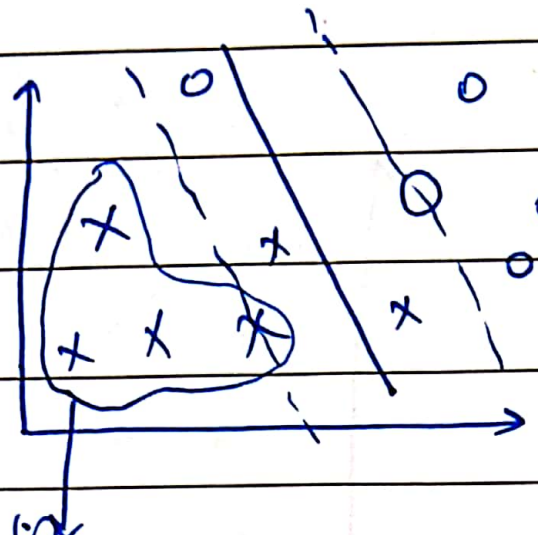


(constraint $w \leq 2$)

i.e. stop gradient descent at $w \leq 2$

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.

error $\xi^{(i)} = 0$

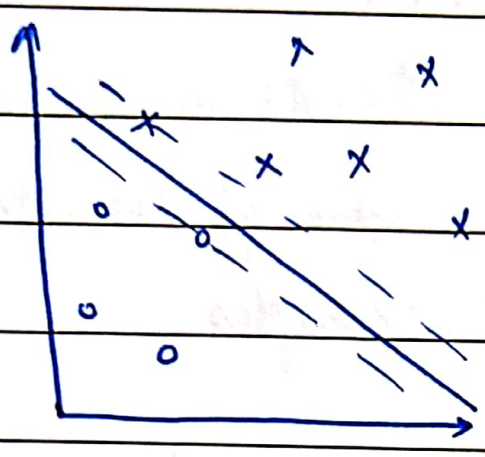
$(x_i, y_i) \rightarrow \xi^{(i)}$ // Error for each $x^{(i)}$

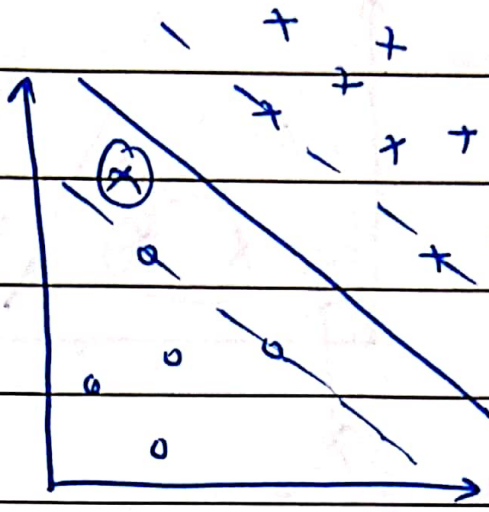
Primal objective

$$\frac{1}{2} w w^T + C \sum_{i=1}^{n} \xi^{(i)} = \text{loss}$$

$$y^{(i)} * (w^T x^{(i)} + b) \geq \underbrace{1 - \xi^{(i)}}_{\text{Allow some error.}}$$

* Some points can also have margin < 1 & even -ve (outlier on opp side).
* C controls amount of penality 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.

$C \rightarrow$ small

Some misclassifications for outliers but margin is max.

Better.

Try different values of C when coding.

Pegasos Algorithm for Unconstrained optimisation.

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

$\downarrow$

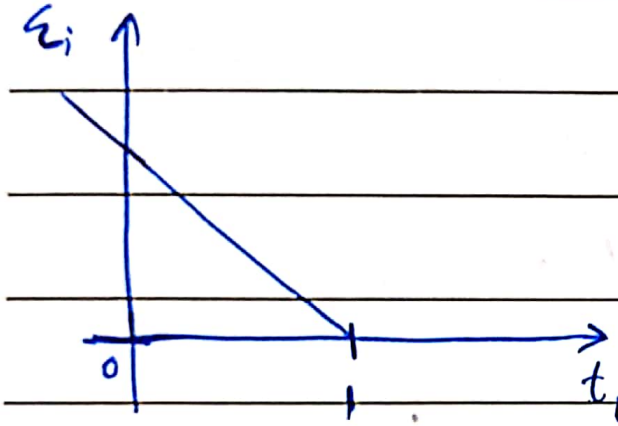
Convex with constraints + outliers

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

① $\varepsilon^{(i)} \geq 1 - \underbrace{y^{(i)}(w^T x^{(i)} + b)}_{t_i}$ // from constraint

② $\varepsilon^{(i)} \geq 0$

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



$\varepsilon_i = 1 - t_i$, if $t_i \leq 1$

$= 0$, if $t_i > 1$

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)

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

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

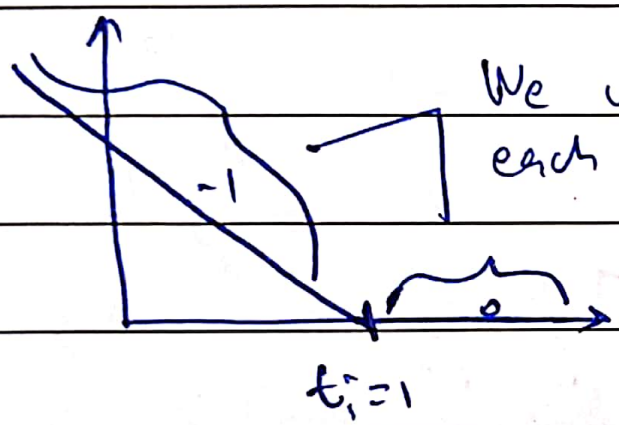
* If point lies in same class & $t_i \uparrow$, then $E = 0$.
* $t \leq 1$ when point lies behind the line & here $t_i \uparrow$ means pt. moving in opp direction hence $E \uparrow$.

Gradient Descent:

$w = w - \eta \nabla_w L$ → learning rate

$b = b - \eta \nabla_b L$

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



We will define 2 gradients for each part.

// Also called Hinge-loss

* This technique is called sub-gradient.

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

$= w + c \sum \frac{\partial f}{\partial t} \cdot \frac{dt}{\partial w}$ // chain rule

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

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

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

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

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

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

$$= y^{(i)}$$

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

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

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

// Final formula for weight update rule.