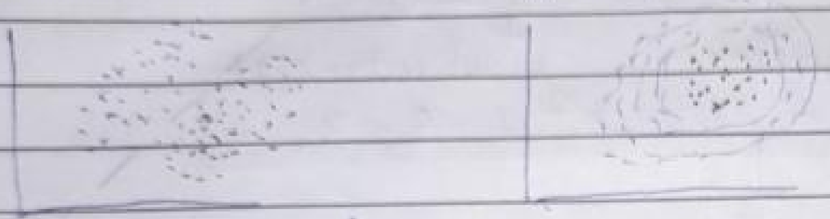
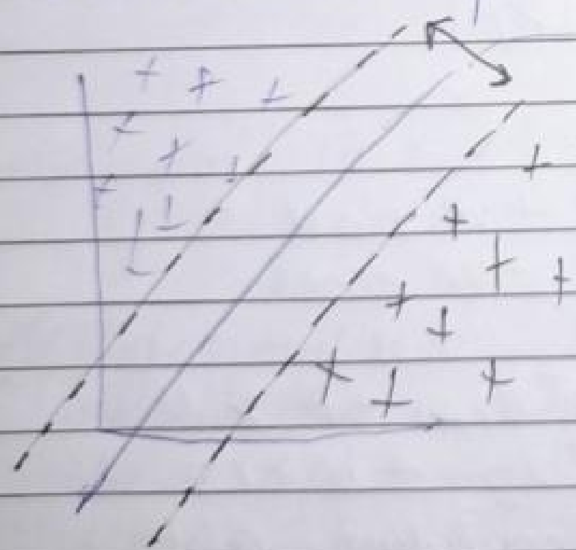
Support Vector Machine

- It's a classifier that works both on linearly and 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



goal is to maximize this distance

$ax + by + c = 0$

dist (x, y)

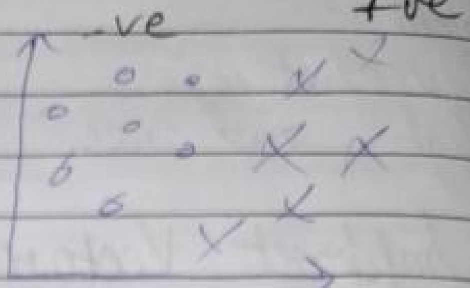
$$\frac{ax + by + c}{\sqrt{a^2 + b^2}} = \text{dist}$$

A hyperplane is a plane of $n-1$ dimension in n dimensional feature space that separated the two class. 2d → line 3d → plane

Max Margin hyperplane :- The optimal hyperplane best separates our data so that the distance / margin from nearest point (called support vectors) in space to itself is maximized

SVM - Mathematically

$X = \{x_1, x_2 \dots x_m\}$
$y = \{y_1, y_2 \dots y_m\}$
$y_i \in \{-1, +1\}$



Key idea = separate data with binary classification maximum margin

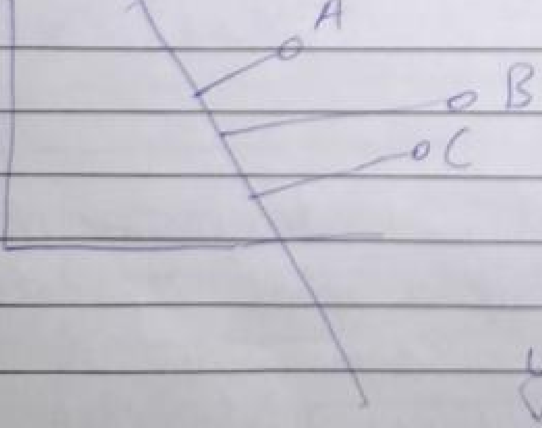
Hyperplane $\underset{\text{vector}}{w^T} x + \underset{\text{bias/ intercept}}{b} = 0$

$\underbrace{\theta_0}_{b} + \underbrace{\theta_1 x_1^1 + \theta_2 x_1^2 + \theta_3 x_1^3 \dots}_{w^T x}$

$w = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_n \end{bmatrix} \quad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$

Optimal Hyperplane

$w^T x + b \geq 0$ if $x^1 \leftarrow$ +ve class
$w^T x + b < 0$ if $x^1 \leftarrow$ -ve class

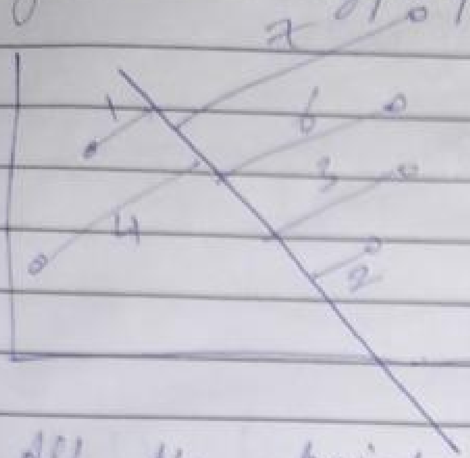


$d_1 = w^T x_{(A)} + b > 0$
$d_2 = w^T x_{(B)} + b >> 0$
more confident abt prediction at B

$y_{pred} = g(w^T x + b)$
$g(z) = +1$ if $z \geq 0$
$g(z) = -1$ if $z < 0$

Date :

goal :- To maximize the minimum distance of points from the hyperplane



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

min dis

$$r = \min_{i=1 \dots m} (r^i)$$

All the points should have atleast r distance

$w^T x + b > 0$ if x +ve
$w^T x + b < 0$ if x -ve
} Non convex

Optimization / SVM objective :-

$$\max_{r, w, b} \; r$$

such that $y^i (w^T x^i + b) \geq r$ for all $i = 1 \dots m$

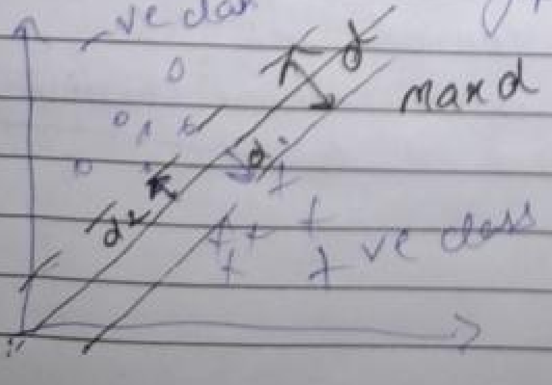
y^i → class

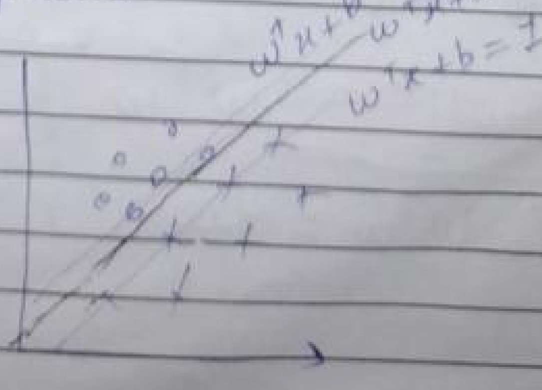
r → absolute dist of any pt from hyperplane

$$\max(r) = \min_{i=1 \dots m} \left(\frac{w^T x^i + b}{\|w\|} \right)$$

Reformulation ⇒

we want to normalise our dataset such that the support vectors (nearest pts) should always lie on the hyperplane





Support vectors should be equidistant
goal is to maximise distance d such that all pts are correctly classified

goal:- $y_i (w^T x^i + b) \geq \gamma$

$d_1 = \frac{|w^T x_{+ve} + b|}{||w||} = \frac{1}{||w||}$

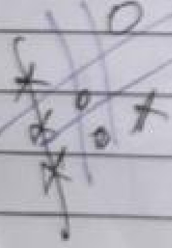
$d_2 = \frac{|w^T x_{-ve} + b|}{||w||} = \frac{1}{||w||}$

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

SVM objective minimize $\frac{1}{2}||w||$ inversion

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

$$y_i \frac{(w^T x^i + b)}{||w||} \geq \frac{1}{||w||}$$ normalization term

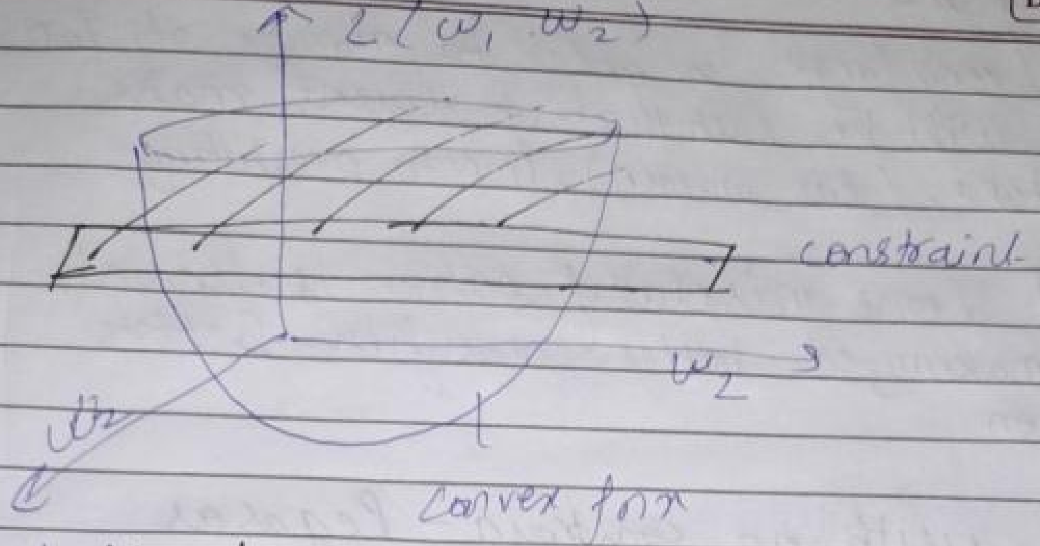


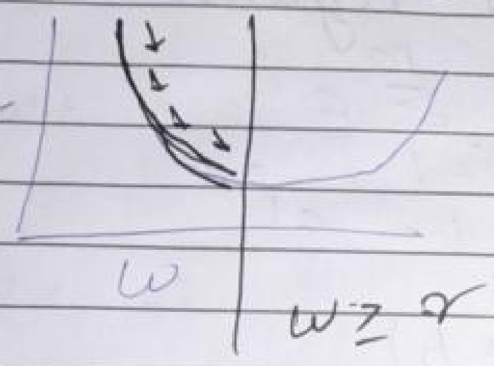
minimize $\frac{1}{2}||w||_2 = \sqrt{\sum w_i^2}$

st $y_i (w^T x^i + b) \geq 1$
$\forall i \in (1..n)$

$$\left[\begin{aligned} ||w||_2^2 &= w^T w \\ \nabla_w w^T w &= w \end{aligned} \right]$$

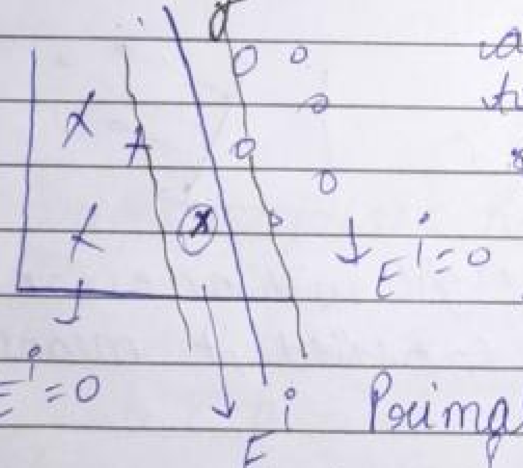
convex fnx
with linear constraint





gradient descent with constraint

Handling Outliers in SVM



allow our algorithm (objective) to do some misclassification on some training example.

$(x_i, y_i) \longrightarrow E^i$

add penalty to SVM objective.

Primal Objective :-

Convex with constraint + outlier

$$\begin{cases} \frac{1}{2} w.w^T = loss + C \sum_{i=1}^{m} \varepsilon^i \\ y^i(w^T x^i + b) \geq 1 - \varepsilon^i \end{cases}$$

New formulation

$$\begin{cases} \frac{1}{2} w.w^T + C \sum_{i=1}^{m} \varepsilon_i \\ y^i(w^T x^i + b) \geq 1 - \varepsilon^i \end{cases} \quad \boxed{\varepsilon^i = 0}$$

C → hyperparameter

If c → ∞ / very large in order to minimize obj fn optimization will be such that it doesnt make any mistake. (less margin). Prone to overfitting

c → small some misclassification is allowed, margin is maximized. Better classifier. Better generalization

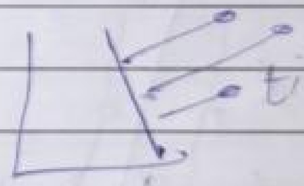
Convex with no constrain - Pegasas

Primal
$$\begin{cases} \min \frac{1}{2} w^T w + c \sum_{i=1}^{m} \varepsilon^i \\ y^i (w^T x + b) \geq 1 - \varepsilon^i \end{cases}$$

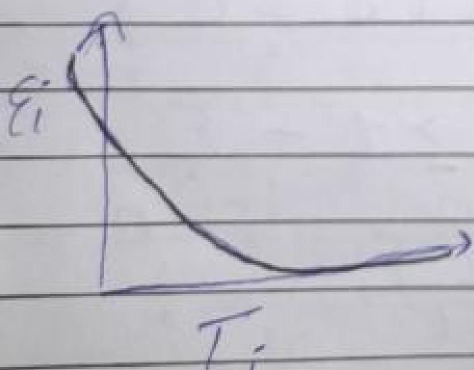
(1) $\varepsilon^i \geq 1 - \underbrace{y^i (w^T x^i + b)}_{t_i}$

(2) $\varepsilon^i \geq 0$

$\varepsilon_i \geq 1 - t_i \qquad \varepsilon_i \geq 0$



if $t_i \geq 1$, pt is lying on right with no error no since $\varepsilon_i \geq 1 - t_i$ is not satisfied it means it has no error



$\varepsilon_i = 1 - t_i \quad \text{if } T_i \leq 1$
$\varepsilon_i = 0 \quad \text{if } t_i \geq 1$

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

Date :

Loss / objective $\rightarrow$

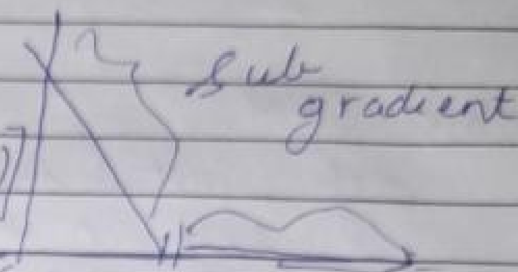
$$\min_{w,b} \frac{1}{2} w^T w + c \sum_{i=1}^{m} \max(0, 1-t_i)$$

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

Applying gradient descent

$w = init()$

$w = w - \eta \nabla_w L$

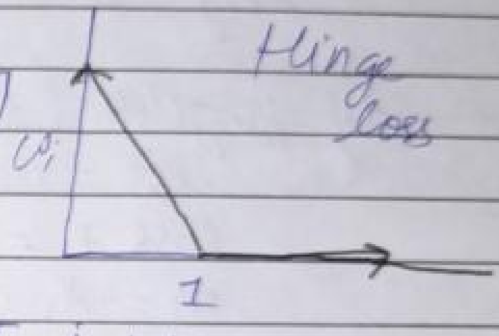


$$\nabla_w L = w + c \sum_{i=1}^{m} \nabla_w [\max(0, 1-t_i)]$$

$$= w + c \sum_{i=1}^{m} \frac{\partial y}{\partial t} \cdot \frac{\partial t}{\partial w}$$

(chain rule)

$$= w + c \sum_{i=1}^{m} \begin{bmatrix} 0 & \text{if } t_i \geq 1 \\ -1 & \text{if } t_i < 1 \end{bmatrix} y^i x^i$$



$t_i = y_i (w^T x^i + b)$ t_i

$\nabla t_i = y_i x^i$

weight update Rule

$w = w - \eta \nabla_w L$

$b = b - \eta \nabla_b L$

$$\nabla_w L = w + c \begin{bmatrix} 0 & t_i \geq 1 \\ -1 & t_i < 1 \end{bmatrix} y_i x_i$$

$$\nabla_b L = 0 + c \begin{bmatrix} 0 & t_i \geq 1 \\ -1 & t_i < 1 \end{bmatrix} \nabla_b t_i$$

$$= c \begin{bmatrix} 0 & t_i \geq 1 \\ -1 & t_i < 1 \end{bmatrix} y_i$$

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

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

```
class SVM:
    def _init_(self, c=1.0):
        self.c = c
        self.w = 0
        self.b = 0

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

    def fit(self, x, y, batch_size=100, lr=0.001):
        no_of_feature = X.shape[1]
        no_of_sample = X.shape[0]
        n = learning-rate
        c = self.c
        # Init the model parameters
        w = np.zeros((1, no_of_feature))
        bias = 0
        print(self.higeloss(w, bias, X, y))
```

$\frac{1}{2} w w^T + c \sum_{i=0}^{n} \max(0, 1-t_i)$

$w_1 x_1 + w_2 x_2 \cdots$

Date :

```
# weight and bias update rule
losses = []
for i in range(max Itr):
    l = self.hingeloss (w, bias, X, y)

    # Batch Gradient Descend with random shuffle
    for batch_start in range(0, no_of_sample
                                batch_size)
        # assume 0 gradient for the batch
        gradw = 0
        gradb = 0

        # iterate over all ex in mini batch
        for j in range( batch_start, batch_start
                                     + batchsize):
          if j < no. of samples:
            i = ids[j]
            ti = y[i] * np.dot (w, X[i].T) +
                                          bias))
            if ti > 1:
               gradw += 0
   gradient    gradb += 0
   calc else
               gradw += c * y[i] * X[i]
               gradb += c * y[i]
      # updation
      w = w - lr * w + lr * gradw
      b = b + lr * gradb

return w, bias, losses
```

Date :

SVM Kernel

- apply SVM to non-linearly separable data

For linearly separable

$$\min \begin{cases} \frac{1}{2} w^T w + c \sum_i \varepsilon_i \\ y_i (w^T x_i + b) \geq 1 - \varepsilon_i \end{cases}$$

$$x_i \longrightarrow \phi(x_i)$$

computationally expensive

Another formulation :- Based on Lagrangian

$$\max \left(\sum_i \alpha_i - \frac{1}{2} \{ \alpha_i \alpha_j y_i y_j \underbrace{\phi(x_i)^T \phi(x_j)}_{\text{computation}} \right)$$

$$\sum_i \alpha_i y_i = 0$$

Kernel Trick :-

A Kernel is a fnx which has following prop

$$K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$$

projected vector of i^{th} vector

1. RBF Kernel
2. Polynomial Kernal
3. Sigmoid Kernel
4. Linear Kernel

Page No.
Date :

① RBF Kernel - Radial Basis Kernel

$K(x_i, x_j) = e^{-\gamma |x_i - x_j|^2} \rightarrow (x_i - x_j)(x_i - x_j)^T$

γ = amplitude of fn

② Polynomial Kernel

$K(x_i, x_j) = (\gamma x_i^T \cdot x_j + r)^d \rightarrow$ degree of polynomial

③ Sigmoid Kernel

$$K(x_i, x_j) = \frac{1 - e^{-2(\gamma x_i x_j + r)}}{1 + e^{-2(\gamma x_i x_j + r)}}$$

Why Kernel? $x = (1, 2, 3)$ $y = (4, 5, 6)$

$f(x) = (1, 2, 3, 2, 4, 6, 3, 6, 9)$

$f(y) = (16, 20, 24, 20, 25, 30, 24, 30, 36)$

$\langle f(x) f(y) \rangle = 16 + 40 + 72 + 40 + 100 + 180 + 72 + 180 + 324 = 1024$

So long!

using Kernel : $K(x, y) = [(1, 2, 3) \cdot (4, 5, 6)]^2$

$= [4 + 10 + 18]^2$

$K(x_1, x_2) = (\langle x, y \rangle)^2$ $= 1024$

done so simple