SVM (Support Vector Machines)

→ Used for both classification & regression

→ Mainly used for classification of numeric data
(Unlike logistic regression which acts on catagorical data)

→ Linear SVM → when data is linearly seperable
Nonlinear SVM → when data is not linearly seperable

→ Applications

Handwriting detection

Cancer diagnosis

Steganography detection

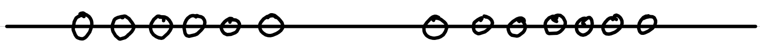
Face detection

Image classification

- Concept "Good fences makes good neighbors"

Linear SVM

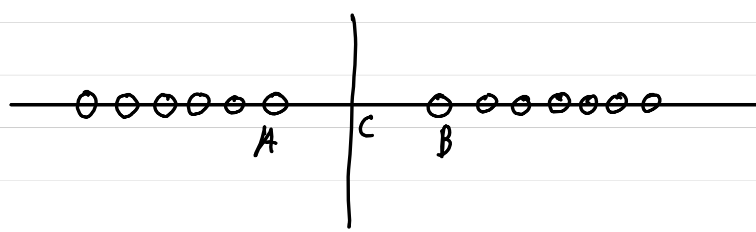
Consider classification of data points



In order to split these points, we use a threshold.

The threshold line can be the midpoint of the last points of the data

That is the point C is Midpoint of A & B



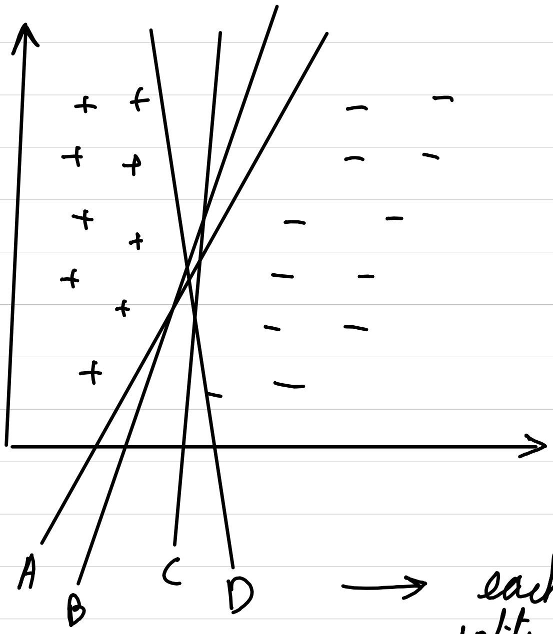
A & B are the edge points of the data

They are called as support vectors

This form of classifier is called Maximal Margin classifier

For linearly seperable data like that maximum margin hyperplane can be used.

For higher dimentions, an hyperplane is formed



Left of hyperplane is an class
Right is another class

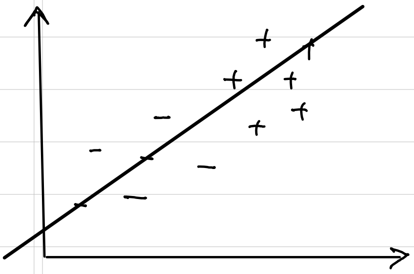
→ each hyperplane splits data into 2 parts

There are ∞ such hyperplanes

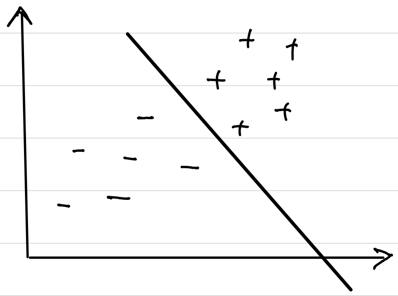
We need a hyperplane that Not only performs well on the training data but also on unseen test data

So the plane which has maximum "margin" from the datapoints is required

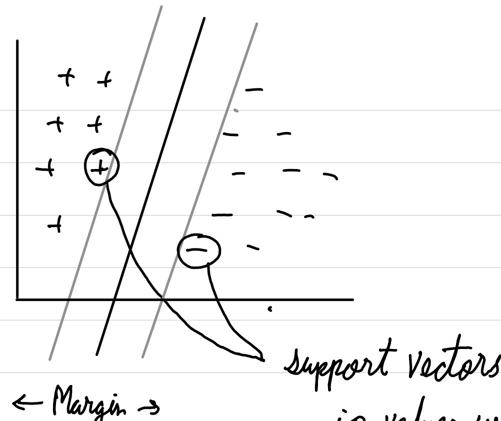
Note → This is different from linear regression



Linear regression



SVM



ie values which lie on the boundaries

Classifier that has small margin is susceptible to overfitting

Hence we need to find plane with the Maximum margin

Optimize a hyperplane such that it has highest margin

Margin is distance of Margin with the nearest points of either class.

Intutively classifier with small margin is more succeptable to model overfitting & classify with weak confidence on unseen data

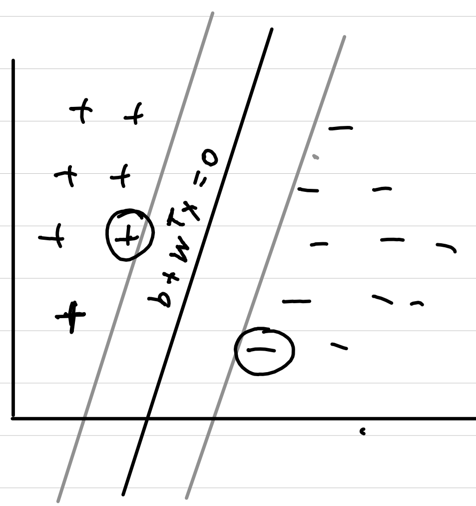
Hence SVM is maximum margin classifier

Equation of hyperplane $y = mx + c$ (2D)

$y = w_0 + w_1 x_1 + w_2 x_2 \cdots$ (xD)

$y = b + W^T X$ (matrix form)

Consider two Support Vectors X_+ & X_-



For all points above hyperplane

$Wx + b > 0$ is class +

For all points below hyperplane

$Wx + b < 0$ is class −

$$y \begin{cases} + & \text{if } wx + b > 0 \\ - & \text{if } wx - b < 0 \end{cases}$$

We rotate the axes for hyperplane such that $b + W^T x = 0$ for hyperplane

($b + W^T x \neq y$ now)

For x_+, $b + W^T x_+ = k_1$

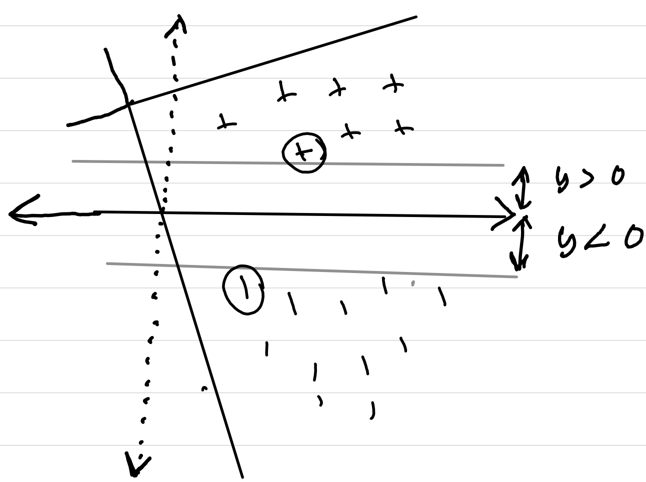
x_-, $b + W^T x_- = k_2$

Note that w & b are our numbers

We can scale them the way we want

They dont represent anything on the graph as long as the condition is met

From hyperplanes perspective the graph looks like this



In order to normalize things we will set scale as

$$wx + b = 1 \quad \text{at } x_+$$
$$wx + b = -1 \quad \text{at } x_-$$

$x_+ \rightarrow x_1 \quad x_- \rightarrow x_2$ (new terminology)

Hence we get

$$W(X_1 - X_2) = 2$$

Matrix of weights

Matrix of data points on boundary

Remember we want to maximize the margin that is $(x_1 - x_2)$

Hence we get Margin $d = (x_1 - x_2) = \frac{2}{||W||}$

In order to Maximize Margin, we can minimize this function $\frac{||W||^2}{2}$ —(1)

We made it $||W||^2$ by purpose

Also, we have condition

$Wx_i + b \geq 1$ if class is $+$

$Wx_i + b \leq -1$ if class is $-$

Let us denote class $+$ by $y_i = 1$

$-$ by $y_i = -1$

This y has no relation with any other y (of axes)

Here y is the label (class) of the training dataset

i.e. $y_i(Wx_i + b) \geq 1$ — (2)

for all datapoints in dataset

Note → Here i are datapoints $i = 1, 2, \ldots n$

So we want to minimize objective function ①
with respect to a condition ②

This is called as convex optimization problem

Here the Objective function is Quadratic $||W||^2$

Inequalities are linear. $y_i(W.x_i+b) \geq 1$

We can use the Lagrange multiplier method

LMM has got KKT constraints

So Maximize →

$$L = \frac{||W||^2}{2} - \sum_{i=1}^{n} \lambda_i (y_i (W_i X_i + b) - 1)$$

λ is lagrangian multiplier for a datapoint

The KKT constraints are computationally expensive & can be solved using a linear / quadratic programming technique to get values of λ_i

To classify a test tuple X

$$\delta(X) = WX + b$$

$$= \sum_{i=1}^{n} \lambda_i y_i X_i \; X + b$$

Matrix with
Length = Length X

$$b = \frac{b_1 + b_2}{2}$$

$b_1 = 1 - W.X_1$
$b_2 = 1 - W.X_2$

X_1 & X_2 are support vectors

For non support vectors $\lambda = 0$

From KKT condition we get

$$y_i(WX_i + b) - 1 \geq 0 \quad \text{for all datapoints } i$$

from this, we can solve and find values of λ as

$$y_i(WX_i + b) = 1 \quad \forall i$$

$$WX_i + b = y_i \quad \forall i$$ since $\|y_i\| = 1$ only sign is there

$$\left(\sum \lambda_j y_j \cancel{X_j}\right) \cdot X_i + b = y_i$$

By augmenting b as additional dimention we can remove b & get

$$\left(\sum \lambda_j y_j \cancel{X_j}\right) X_i = y_i$$

$$\sum \lambda_j y_j (X_j \cdot X_i) = y_i \quad \forall i$$

unknown

known known known

Since $j = i$, we have i unknowns & i equations

So solve to get λ

for non support vectors $\lambda = 0$ will be found

Q1 Training data

	A_1	A_2	y	λ_i
1.	0.38	0.47	+	65.52
2.	0.49	0.61	−	65.52
3.	0.92	0.41	−	0
4.	0.74	0.89	−	0
	0.5	0.5	?	

→ 1. & 2. are support vectors

by quadratic programming we have

$$W_i = \sum \lambda_i X_i y_i$$

$$= 65.52 \times \begin{pmatrix} 0.38 \\ 0.47 \end{pmatrix} \times 1$$

$$+ 65.52 \times \begin{pmatrix} 0.49 \\ 0.61 \end{pmatrix} \times -1$$

$$+ 0 + 0$$

$$W = \begin{pmatrix} -7.20 \\ -9.17 \end{pmatrix}^T$$

$$b_1 = 1 - W.X_1 = 1 - \left(\begin{pmatrix} -7.20 \\ -9.17 \end{pmatrix} . \begin{pmatrix} 0.38 \\ 0.47 \end{pmatrix} \right)$$

$$= \begin{matrix} 3.73 + \\ 5.30 \end{matrix} = 9.03$$

$$b_2 = 1 - W.X_2 = \begin{matrix} 4.528 + \\ 6.593 \end{matrix} = 11.121$$

$$b = \frac{b_1 + b_2}{2} = 10.0755$$

for 0.5, 0.5

$$\delta(x) = W.X + b$$

$$= \begin{pmatrix} -7.20 \\ -9.17 \end{pmatrix}^T \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix} + 10.0755$$

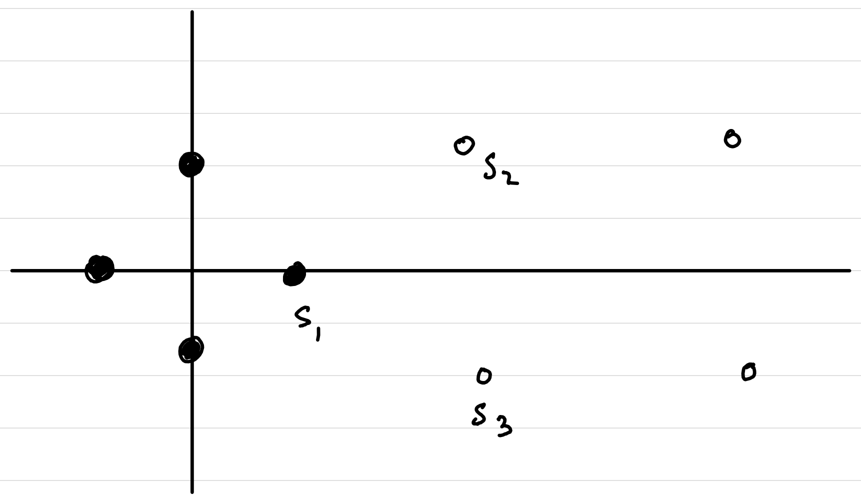
$$= 1.8905$$

$\delta(x) > 0$ i.e. class +

(2) $\left.\begin{matrix}(3,1)\\(3,-1)\\(6,1)\\(6,-1)\end{matrix}\right\}$ +ve $\quad$ $\left.\begin{matrix}(1,0)\\(0,1)\\(0,-1)\\(-1,0)\end{matrix}\right\}$ −ve

classify (4, 1)

→



Support vectors are $S_1 = (1,0)$
$S_2 = (3,1)$
$S_3 = (3,-1)$

for putting $x_0 = 1$
& finding bias

Add extra 1

$$S_1 = \begin{bmatrix}1\\0\\1\end{bmatrix} \quad S_2 = \begin{bmatrix}3\\1\\1\end{bmatrix} \quad S_3 = \begin{bmatrix}3\\-1\\1\end{bmatrix}$$

$$[\alpha_1\ \alpha_2\ \alpha_3]\begin{bmatrix}S_1\\S_2\\S_3\end{bmatrix} \cdot \begin{bmatrix}S_1\\S_2\\S_3\end{bmatrix} = y$$

ie.

$$\alpha_1 S_1 S_1 + \alpha_2 S_1 S_2 + \alpha_3 S_1 S_3 = y_1$$
$$\alpha_1 S_2 S_1 + \alpha_2 S_2 S_2 + \alpha_3 S_2 S_3 = y_2$$
$$\alpha_1 S_3 S_1 + \alpha_2 S_3 S_2 + \alpha_3 S_3 S_3 = y_3$$

$\alpha_1 \alpha_2 \alpha_3 \rightarrow$ Constants
$S_1 S_2 S_3 \rightarrow$ Matrices
Multiplication is dot product

$$\therefore \alpha_1 \begin{bmatrix}1\\0\\1\end{bmatrix}\begin{bmatrix}1\\0\\1\end{bmatrix} + \alpha_2 \begin{bmatrix}3\\1\\1\end{bmatrix}\begin{bmatrix}1\\0\\1\end{bmatrix} + \alpha_3 \begin{bmatrix}3\\-1\\1\end{bmatrix}\begin{bmatrix}1\\0\\1\end{bmatrix} = -1$$

$$\therefore \alpha_1 \begin{bmatrix}1\\0\\1\end{bmatrix}\begin{bmatrix}3\\1\\1\end{bmatrix} + \alpha_2 \ldots\ldots = 1$$

$$\therefore \alpha_2 \begin{bmatrix}1\\0\\1\end{bmatrix}\begin{bmatrix}3\\-1\\1\end{bmatrix} + \alpha_2 \ldots = 1$$

$$\therefore \quad 2\alpha_1 + 4\alpha_2 + 4\alpha_3 = -1 \quad -(1)$$
$$4\alpha_1 + 11\alpha_2 + 9\alpha_3 = 1 \quad -(2)$$
$$4\alpha_1 + 9\alpha_2 + 11\alpha_3 = 1 \quad -(3)$$

On solving (1) (2) & (3)

$$\alpha_1 = -3.5$$
$$\alpha_2 = 0.75$$
$$\alpha_3 = 0.75$$

$$W = \sum \alpha_i S_1$$
$$= -3.5\begin{bmatrix}1\\0\\1\end{bmatrix} + 0.75\begin{bmatrix}3\\1\\1\end{bmatrix} + 0.75\begin{bmatrix}3\\-1\\1\end{bmatrix}$$
$$= \begin{bmatrix}1\\0\\-2\end{bmatrix}$$

← Weights
← $b = -2$

for point (4, 1)

$$Wx + b = \begin{bmatrix}1\\0\end{bmatrix}^T \begin{bmatrix}4\\1\end{bmatrix} - 2$$
$$= 2$$

$2 > 0 \quad \therefore$ Point (4, 1) is positive class

!! Wait $W = \sum \lambda_i y_i S$ right ?
what is α?
also our equations were

$$\sum \lambda_j y_j (x_j \cdot x_i) = y_i$$

Also as per KKT, $\lambda > 0$

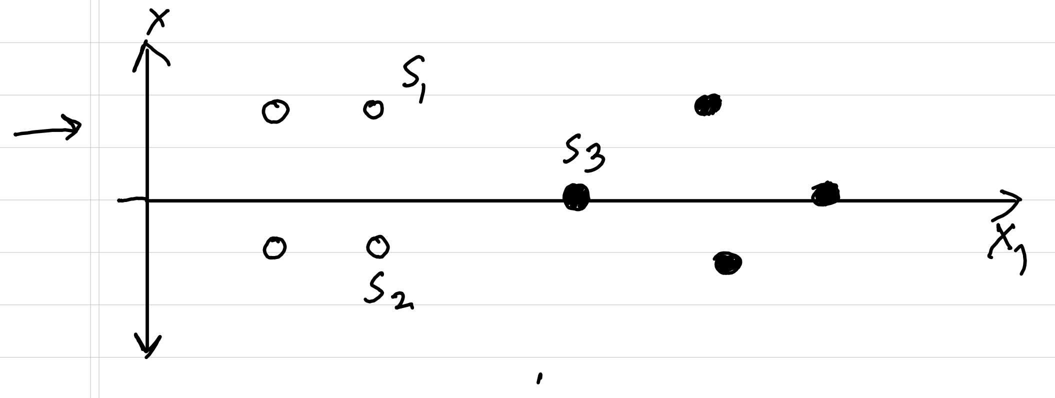
So basically we are clubbing

$$\alpha_i \rightarrow \lambda_i \cdot y_i$$

Classify the point $(6,1)$ using SVM

(3)

x_1	x_2	y
1	1	-1
2	1	-1
1	-1	-1
2	-1	-1
4	0	1
5	1	1
6	0	1
5	-1	1



$$S_1 = \begin{pmatrix}2\\1\\1\end{pmatrix} \quad S_2 = \begin{pmatrix}2\\-1\\1\end{pmatrix} \quad S_3 = \begin{pmatrix}4\\0\\1\end{pmatrix}$$

$$y_1 = -1 \qquad y_2 = -1 \qquad y_3 = 1$$

$$\alpha_1 S_1 S_1 + \alpha_2 S_1 S_2 + \alpha_3 S_1 S_3 = y_1$$
$$\alpha_1 S_2 S_1 + \alpha_2 S_2 S_2 + \alpha_3 S_2 S_3 = y_2$$
$$\alpha_1 S_3 S_1 + \alpha_2 S_3 S_2 + \alpha_3 S_3 S_3 = y_3$$

$$\alpha_1 \begin{pmatrix}2\\1\\1\end{pmatrix}\begin{pmatrix}2\\1\\1\end{pmatrix} + \alpha_2 \begin{pmatrix}2\\1\\1\end{pmatrix}\begin{pmatrix}2\\-1\\1\end{pmatrix} + \alpha_3 \begin{pmatrix}2\\1\\1\end{pmatrix}\begin{pmatrix}4\\0\\1\end{pmatrix} = -1$$

$$\alpha \begin{pmatrix}2\\-1\\1\end{pmatrix}\begin{pmatrix}2\\1\\1\end{pmatrix} + \alpha_2 \begin{pmatrix}2\\-1\\1\end{pmatrix}\begin{pmatrix}2\\-1\\1\end{pmatrix} + \alpha_3 \begin{pmatrix}2\\-1\\1\end{pmatrix}\begin{pmatrix}4\\0\\1\end{pmatrix} = -1$$

$$\alpha \begin{pmatrix}4\\0\\1\end{pmatrix}\begin{pmatrix}2\\1\\1\end{pmatrix} + \alpha_2 \begin{pmatrix}4\\0\\1\end{pmatrix}\begin{pmatrix}2\\-1\\1\end{pmatrix} + \alpha_3 \begin{pmatrix}4\\0\\1\end{pmatrix}\begin{pmatrix}4\\0\\1\end{pmatrix} = 1$$

$$6\alpha_1 + 4\alpha_2 + 9\alpha_3 = -1$$
$$4\alpha_1 + 6\alpha_3 + 9\alpha_3 = -1$$
$$9\alpha_1 + 9\alpha_2 + 17\alpha_3 = 1$$

$\alpha_1 = -3.25$
$\alpha_2 = -3.25$
$\alpha_3 = 3.5$

$$W_i = \Sigma \alpha_i S_i$$
$$= -3.25\begin{pmatrix}2\\1\\1\end{pmatrix} + -3.25\begin{pmatrix}2\\-1\\1\end{pmatrix} + 3.5\begin{pmatrix}4\\0\\1\end{pmatrix}$$

$$= \begin{pmatrix}1\\0\\-3\end{pmatrix} \} w$$

$b = -3$

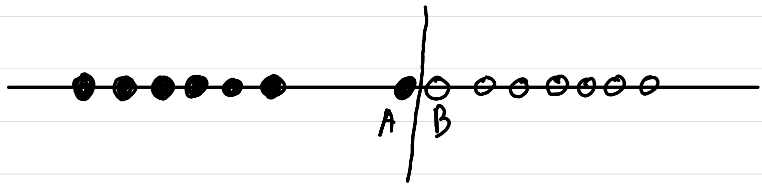
for point $(6,1)$

WX ($wx + b$ here b is added dimention)

$$\begin{pmatrix}6\\1\\1\end{pmatrix}\begin{pmatrix}1\\0\\-3\end{pmatrix} = 6 - 3 = 3$$

$3 > 0 \quad \therefore$ class +ve

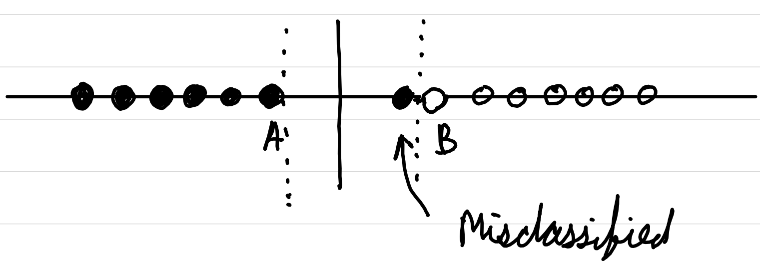
The maximum Margin classifier is the hard margin approach.



But that approach fails if we have an outlier
Hence a soft margin approach is used

Points are allowed to be Misclassified
This is called Support Vector classifier

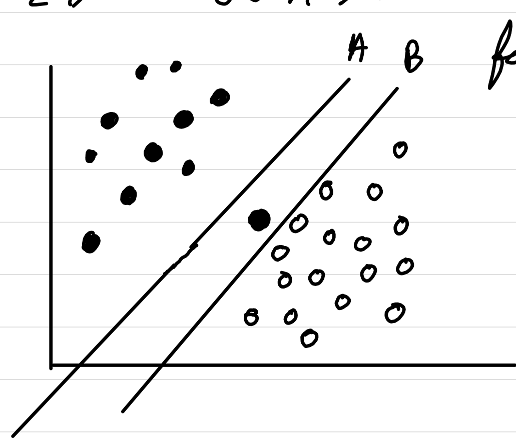
In order to provide variance to the dataset the support vectors are not the edge points, but some points chosen by cross validation that are towards the edge



Soft Margin SVM

Wider Margin is required, but also on training data

For 2D or n > 2



for binary classification

A → soft
B → Hard

For Higher dimentional data, an Hyperplane is formed

Misclassification must be allowed to reduce overfitting

Allow hyperplane to have errors

We need to penalize the misclassified points else model will break

Objective of soft Margin SVM

$$L = 1/2 \, \|w\|^2 + C \left(\ldots \text{some math} \right)$$

number of allowed mistakes

C is a hyperparameter that decides tradeoff between maximizing margin and minimize mistake

C: Small → focus more on avoiding missclassification

Large → Maximize margin

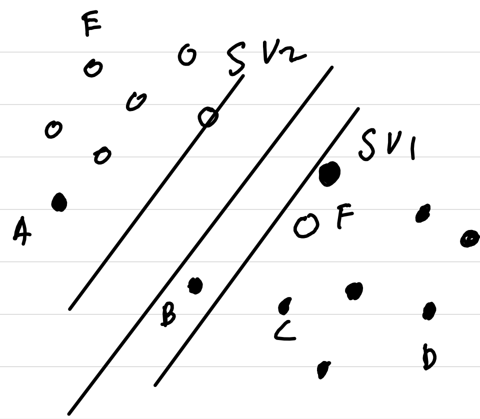
Slack Variables

To check misclassified points, we put a variable ξ_i

$\xi_i = 0$ if point correct

$0 \leq \xi_i \leq 1$ if point between Support Vectors

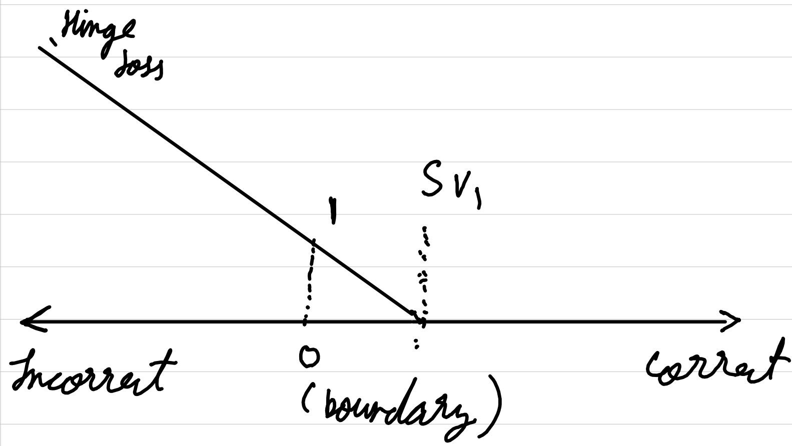
$\xi_i > 1$ if point on wrong side



		Hinge
A, F	$\xi_i > 1$	> 1
B	$0 \leq \xi_i \leq 1$	$\in (0,1)$
C, D	$\xi_i = 0$	0

Hinge Loss

Hinge loss = $\max(0, 1 - y_i \bar{y}_i)$



Example →

Actual y_i	Predicted $\bar{y}_i$	Hinge loss
+1	0.97	0.03
+1	1.20	0.00
+1	0	1.00
+1	−0.25	1.25
−1	−0.88	0.12
−1	−1.01	0.00

So we can write objective function as

$$L = \frac{1}{2} ||w||^2 + c \sum_{i=1}^{n} (\varepsilon_i)^{\phi}$$

$\phi \longrightarrow$ parameter

& each x_i needs to be

$$y_i (w \cdot x_i + b) \geq 1 - \varepsilon_i$$

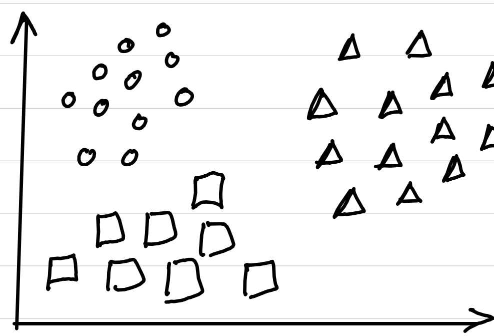
Now proceed with Lagrange multiplier

Multiclass SVM

Till now we saw 2 class (binary) classification

However even this approach has a drawback that we have restricted only to 2 classes.

What if we have 3 classes?



But SVM can only handle 2 classes at a time

Then we need to use SVM multiple times

There are two strategies for this

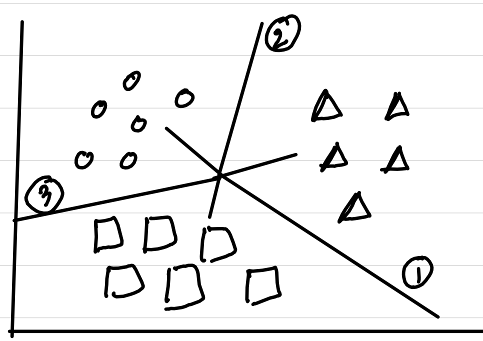
OVO One Vs One
OVA One Vs All

OVO (One vs One)

→ Consider two classes at a time & ignore the rest. Run SVM only on the two classes. Do this for all pairs of classes

→ Find MMH for <u>each pair</u> of classes.

→ We can test OVO for each class and find the correct class.

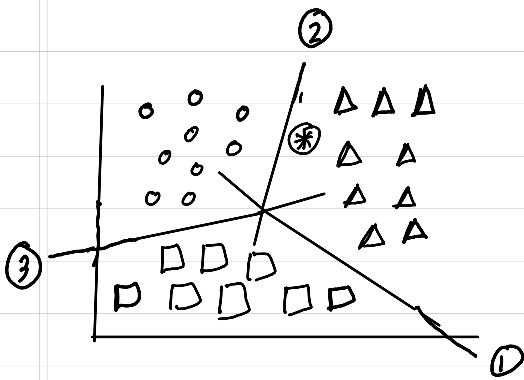


① A vs B
② B vs C
③ C vs A

→ But this is costly

→ The amount of classification will increase exponentially with no of classes.

classify point marked *



MMH

1 □ vs △
2 △ vs ○
3 □ vs ○

3 MMH formed

□ vs △ → △
□ vs ○ → ○
△ vs ○ → △

Winner △

For 4 class

A	B	B
A	C	A
A	D	D
B	C	B
B	D	B
C	D	D

B

6 MMH

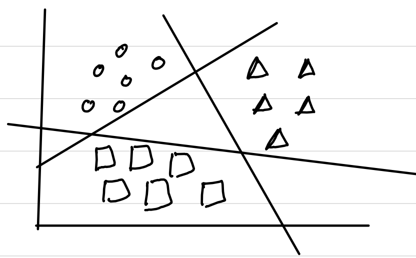
No of MMH $= \frac{n(n-1)}{2} = O(n^2)$

Too large growth

OVA (One Vs All)

Take one class and group others into another. Do this for all classes.

Instead of making MMH for every pair of classes we will make MMH for every class.



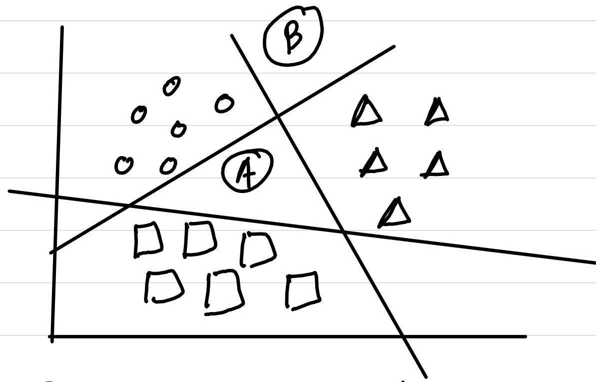
Predicted label is the maximum of all labels.

If classes are not linearly seperable then this fails

This is a faster method of classification.

Requires only n MMHs for n classes

Ambiguity in the classification



Ⓐ → Unclassified tuples

Ⓑ → In both classes ○ △

OVO & OVA can be thought as ensemble techniques where multiple models are combined and "winner take all" is used

Error Correcting Output Codes

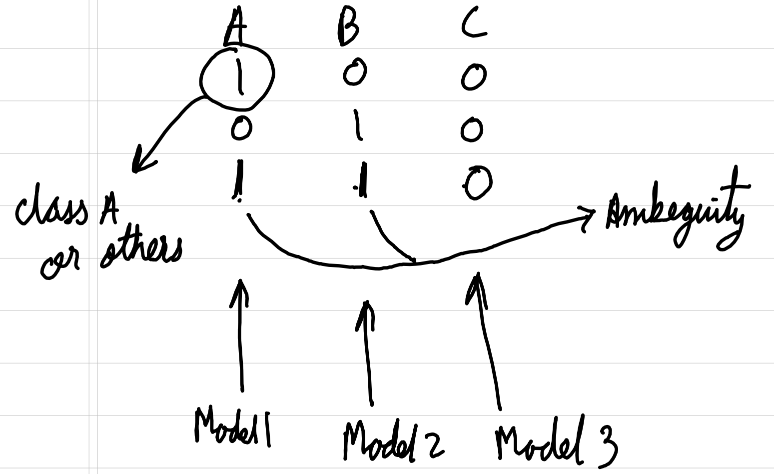
Technique that reframes multiclass classification as multiple binary classification.

A bit string (binary encoding) can be used to represent each class in the problem.

Each bit in the string can be predicted as 0 or 1 by a seperate binary classification problem.

Each model recieves the full input data and predicts one position in the output string.

OvA can be thought as one hot encoding



But ECOC is more redundant and error correcting

eg

$$\left.\begin{array}{l} A \rightarrow 0\,1\,1 \\ B \rightarrow 1\,0\,1 \\ C \rightarrow 0\,1\,0 \end{array}\right\} \text{encoding scheme}$$

Model 1 Model 2 Model 3

Pred. 011 $\therefore$ class A

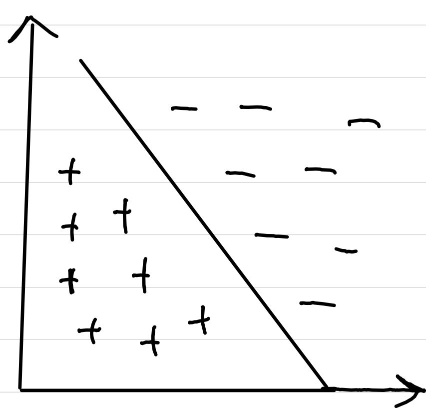
000 closest to class C

Care has to be taken that each encoded class has a very different encoded string

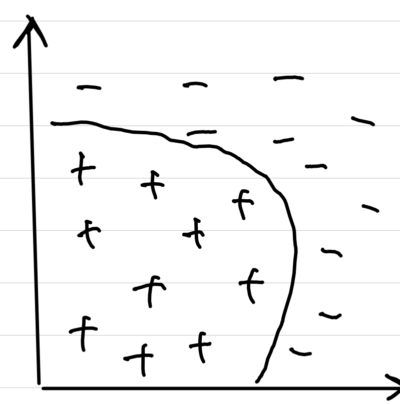
We saw MMH for linearly seperable datapoints

This type of data is not handeled by the classifier because points are not linearly seperable

In order to solve this problem Non Linear SVMs are required

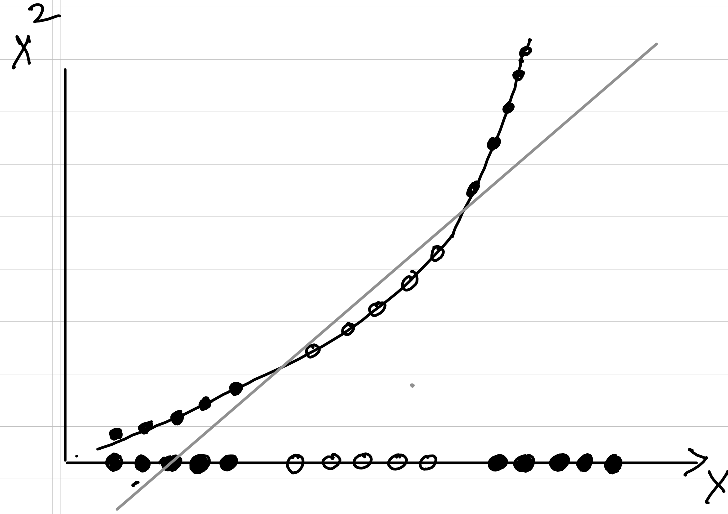


Linearly seperable Data



Non linearly Data

what will happen if we square the data?



Square the observations & make a 2D graph

x axis → observations

y axis → $(\text{observation})^2$

Now the data becomes linearly seperable!

Now apply SVC to the graph

What we have done here is to apply some function on the input data to make it to a higher dimention

1 D data which was not linearly seperable

$\Updownarrow$ converted to

2 D data which is linearly seperable

Here we can represent the function we applied as

$$\phi = \begin{pmatrix} x^2 \\ x \end{pmatrix} \quad \text{for input} \quad (x)$$

2 D plot x^2 vs x 1 D original x

Here ϕ is our transformation

This is a part of polynomial kernel

Why did we square the data and not cube it?

what if we cant draw a straight line even in 2 D?

The value of the degree to which power is raised is d

d is decided by the SVM using cross validation

So basically we dont like the input space we are in and we want to transform the input space into a space where things are more convinent

Here we have raised the input space into higher dimention to make the transformation

But there are other ways also

Different input spaces require different transformations

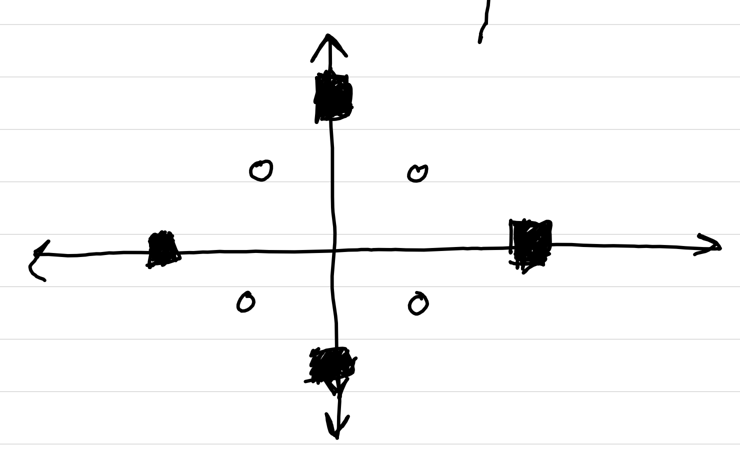
So we need a transformation ϕ

$\phi(\bar{x})$ will give transformed vector x

ϕ may or may not increase the dimentions

④ $\phi = \begin{cases} \begin{pmatrix} 6 - x_1 + (x_1 - x_2)^2 \\ 6 - x_2 + (x_1 - x_2)^2 \end{pmatrix} & \text{if } \sqrt{x_1^2 + x_2^2} \geq 2 \\ \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} & \text{o/w} \end{cases}$

Blue class vectors: $\begin{pmatrix} 1 \\ 1 \end{pmatrix} \begin{pmatrix} -1 \\ 1 \end{pmatrix} \begin{pmatrix} -1 \\ -1 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \end{pmatrix}$ | Red class: $\begin{pmatrix} 2 \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 2 \end{pmatrix} \begin{pmatrix} -2 \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ -2 \end{pmatrix}$



Apply transformation ϕ to transform the space

$\rightarrow \begin{pmatrix} 1 \\ 1 \end{pmatrix} \rightarrow \begin{pmatrix} 1 \\ 1 \end{pmatrix}$

$\begin{pmatrix} -1 \\ 1 \end{pmatrix} \rightarrow \begin{pmatrix} -1 \\ 1 \end{pmatrix}$

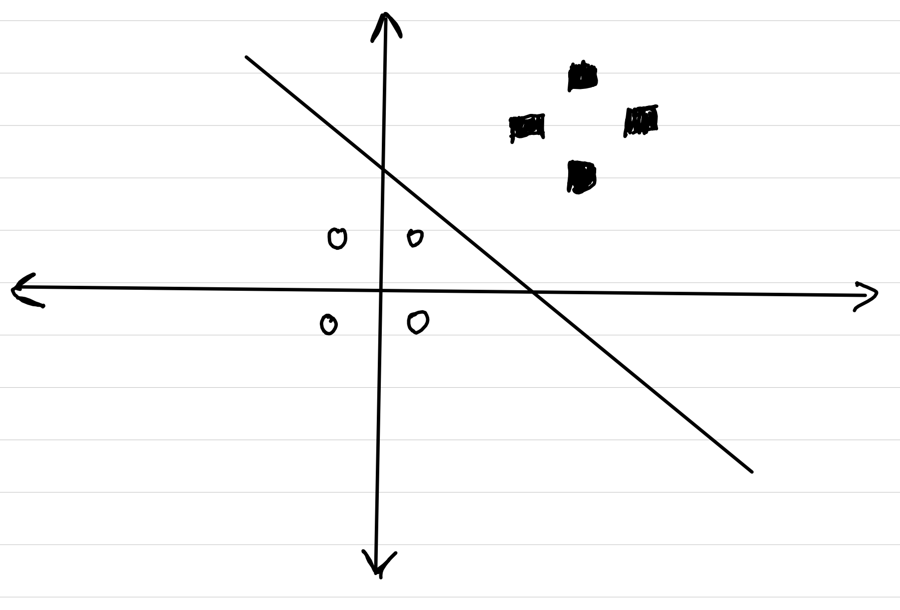
All blue will be same

$\begin{pmatrix} 2 \\ 0 \end{pmatrix} \rightarrow \begin{pmatrix} 8 \\ 10 \end{pmatrix}$

$\begin{pmatrix} 0 \\ 2 \end{pmatrix} \rightarrow \begin{pmatrix} 10 \\ 8 \end{pmatrix}$

$\begin{pmatrix} -2 \\ 0 \end{pmatrix} \rightarrow \begin{pmatrix} 12 \\ 10 \end{pmatrix}$

$\begin{pmatrix} 0 \\ -2 \end{pmatrix} \rightarrow \begin{pmatrix} 10 \\ 12 \end{pmatrix}$



Now solve as linear SVM

$\bar{s}_1 = \begin{pmatrix} 8 \\ 10 \\ 1 \end{pmatrix} \quad \bar{s}_2 = \begin{pmatrix} 10 \\ 8 \\ 1 \end{pmatrix} \quad \bar{s}_3 = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$

$\lambda_1 S_1 S_1 + \lambda_2 S_1 S_2 + \lambda_3 S_1 S_3 = y_1$
$\lambda_1 S_2 S_1 + \lambda_2 S_2 S_2 + \lambda_3 S_2 S_3 = y_2$
$\lambda_1 S_3 S_1 + \lambda_2 S_3 S_2 + \lambda_3 S_3 S_3 = y_3$

$165 \lambda_1 + 161 \lambda_2 + 19 \lambda_3 = +1$
$161 \lambda_1 + 165 \lambda_2 + 19 \lambda_3 = +1$
$19 \lambda_1 + 19 \lambda_2 + 3 \lambda_3 = -1$

$\lambda_1 = 0.0859$
$\lambda_2 = 0.0859$
$\lambda_3 = -1.421$

$w = \sum \lambda \cdot s$

$= \begin{pmatrix} 0.1252 \\ 0.1252 \\ -1.2501 \end{pmatrix}$

In order to reduce complexity, we can do

$w = w / 0.1252$

$\therefore w = \begin{pmatrix} 1 \\ 1 \\ -9.98 \end{pmatrix}$

$\therefore$ Hyperplane will be formed

The transformation here was 2D to 2D without adding any dimention

Kernel Trick

So we are going to feed the transformed points $\phi(x_i)$ to the SVM

It is found that every time applying ϕ on every point is expensive

We need to optimize the process

But when we look at the internal structure of SVM, we notice that actually what is happening to the vectors we are providing is

for a test vector u (b is augmented)

$$y = W \cdot u$$

Predicted class ← y; W ← Pre calculated weights; u ← Input

Weights are calculated at training time

What if we don't do that?

$$W = \sum \lambda_i \, y_i \, x_i$$

step 1: found by solving equations (λ_i)

step 2: find actual value of W

We skip step 2. & only calculate λ

$$\therefore y_{Pred} = \left(\sum \lambda_i \, y_i \, x_i\right) u \cdot$$

$$y = \sum \lambda_i \, y_i \, (x_i \cdot u) \quad \text{— (1)}$$

Now remember λ equations?

$$\sum \lambda_j \, y_j \, (x_j \cdot x_i) = y_i \qquad \forall i \quad \text{— (2)}$$

$$\left.\begin{matrix} \alpha_1 S_1 S_1 + \alpha_2 S_1 S_2 + \alpha_3 S_1 S_3 = y_1 \\ \alpha_1 S_2 S_1 + \alpha_2 S_2 S_2 + \alpha_3 S_2 S_3 = y_2 \\ \alpha_1 S_3 S_1 + \alpha_2 S_3 S_2 + \alpha_3 S_3 S_3 = y_3 \end{matrix}\right\} \text{example}$$

from (1) & (2) we observe that everything depends only on dot products of the vectors

If we put transformed space vectors in the model, then in end we are going to get.

$$y = \sum \lambda_i \, y_i \, \phi(x_i) \, \phi(u)$$

$$\sum \lambda_i \, y_i \, \phi(x_i) \, \phi(x_j) = y_i$$

In place of $x \cdot y \rightarrow \phi(x) \cdot \phi(y)$

So insted, can we replace the dot product with $k(x, y)$?

$$k(x,y) = \phi(x) \cdot \phi(y)$$

We now need to know k. We dont need to know ϕ anymore

$$\therefore \left.\begin{matrix} y = \sum \lambda_i \, y_i \, k(x, y) \\ \sum \lambda_i \, y_i \, k(x, y) = y_i \end{matrix}\right\} \text{New SVM}$$

So the function k is a function that provides the dot product of the vectors in another space

We dont need to know the transformation of vectors into another space

This is known as the kernel trick

Kernel Properties →

Mercer conditions must be satisfied

Addition of two kernels is a kernel

Polynomial kernel

$$K(u,v) = (u \cdot v + r)^d$$

if $r = 1/2$, $d = 2$

$$(u \cdot v + 1/2)^2 = u \cdot v + u^2 \cdot v^2 + 1/4$$

$$= (u, u^2, 1/2) \cdot (v, v^2, 1/2)$$

x axis

y axis

z axis

Since z axis is same, ignore

$$(u, u^2) \cdot (v, v^2)$$

So this was what we did earlier

ϕ vs K Time complexity

Consider the last kernel

ϕ | K

$\begin{pmatrix} x^2 \\ x \end{pmatrix} \begin{pmatrix} u^2 \\ u \end{pmatrix}$ | $(u \cdot v + 1/2)^2$

$= x^2 u^2 + x u$

1 x multiplication
1 * square

2 x squaring
2 x multiplication

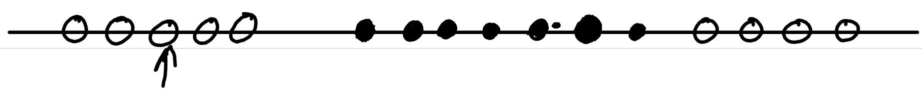
If the dimentionality was higher the growth would be massive for higher dimentions

Also kernels like radial basis operate in ∞ dimentions, so they wouldnt be possible without kernel trick

Radial Kernel

$$K(a,b) = e^{-\gamma(a-b)^2}$$

This kernel finds support vectors in ∞ dimentions
This can be found by expanding e into taylor like series



On data like these, the radial kernel acts as weighted nearest neighbor classifier to classify the points

γ is found by crossvalidation

γ controlls the influence of the points

→ Advantages

Memory efficiant

Works well for both low & high dimentional data

May Perform better than ANNs in some cases.

Works on text and images as well

→ Disadvantages

Long training time for large datasets

Doesnt work well on noise

Lesser interpretability

Choosing right kernel is difficult

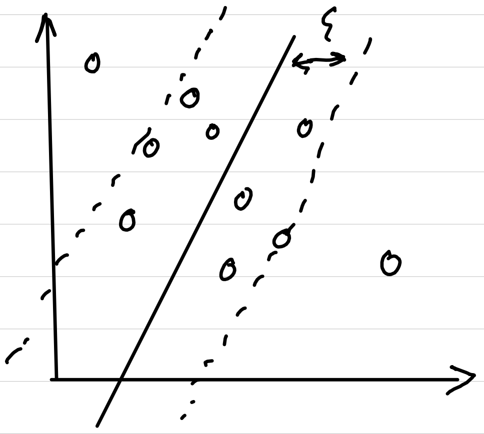
Doesnt give probability, directly the resultant class

Support Vector Regression (SVR)

Used to handle complex non-linear data through Kernels but hence slower

In SVM the goal is to find hyperplane that maximizes the margin between different classes.

But in SVR, the objective function is the one that deviates from the actual target values by a value no greater than ε



ε : tolerence value

The line must be as flat as possible

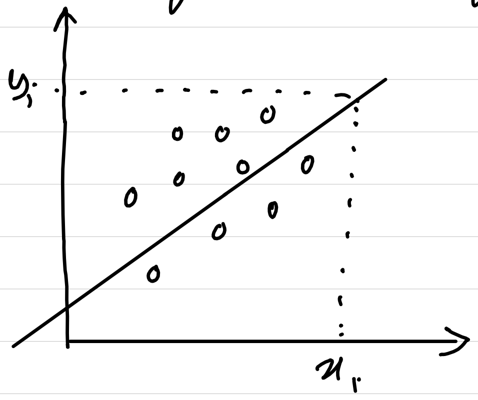
Epsillion insensative loss →

$$\min_{w,b,\xi,\xi^*} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}(\xi_i+\xi_i^*)$$

Regularization Parameter that determines the trade off between the flatness of the function & amount up to which deviations larger than ε are tolerated.

ξ_i → upper Boundary
ξ_i^* → lower

Then the regression can be performed in same way as linear regression



find y_i given x_i

Kernalized SVR can be very accurate