[redacted]

## Object Recognition and svm.

① The goal of recognition / classification is to identify in the image different description of the objects. we can have image tagging, object detection (for eg. find pedestrian) Activity recognition (usually on a sequence of images) and image parsing (reading the image) and automatically generation of text description from image. and image classification.

② The statistical learning framework suggests to apply a prediction function "f()" to a feature representation of the image to get the desired output.

output ← $\boxed{y = f(x)}$ → image feature.
↓
prediction function

③ Training → for training, we have a training set of labelled examples, we know, the input and output and we estimate the prediction function "f" by minimizing the prediction error on the training set.

Testing → during testing, we apply the prediction function to a new test example (eg image feature) and see the predicted value i.e. $y = f(x)$

Steps → ① we have a training set which consists of training images.
② we define some image features, with Training labels and feed it to training model.
③ the model learns the prediction function.

Testing → ① we have an image never seen before. we set the image features and learned model "f".

④ Traditional recognition pipeline:

Image pixels → [Hand-designed feature extraction] → [Trainable classifier] → object class

Hand-designed feature extraction ↓ we extract the features. they are not learned we design it.

Trainable classifier ↓ we apply generic training classifiers (svm) ← (f)

Non-linear classifiers

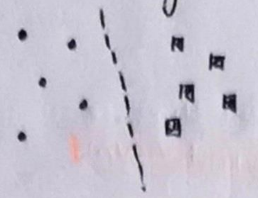
Boundaries distinguishing the classes are not defined by a hyperplane

⑤ Classifiers: Nearest Neighbour → for eg. we have a training examples of diff. class and we want to see, in which category does our test example belongs. we will simply calculate the distance function for our inputs. we will take the smallest distance and see what label does that sample has. We will assign our point with that label.

KNN classifier → Instead of looking at nearest neighbour, we define a no. "K" which signifies how much region would be covered in that point. eg. if K = 10: we will take 10 closest points around our sample points. After that we will vote for the class label and label our point with max. no. of vote.

Outlier can be defined as the points which lie outside of the major region.
KNN is more robust to outliers. KNN is non-linear

→ linear classifiers → A hyperplane distinguishing the classes.



→ ① There is a hyperplane distinguishing b/w the classes.
② we see on which side does our point lie.
③ $f(x) = \text{sign}(w \cdot x + b)$ → offset; $x$ → our point.

+ve → one side → different classes
-ve → another side.

w → vector
x → vector
b → scaler

dot product = scaler

| NN advantageous | NN disadvantages | Linear class. Adv | Linear class. disa. |
|---|---|---|---|
| ① Simple to implement ② Decision boundaries are not necessarily linear ③ Works for any no. of classes ④ Non-parametric method | ① Need good distance function. ② Slow at test time | ① Low dimensional parametric representation. ② Very fast at test time | ① Works for only 2 class ② We have to see how far from the linear function ③ What if the data is not linearly separable |

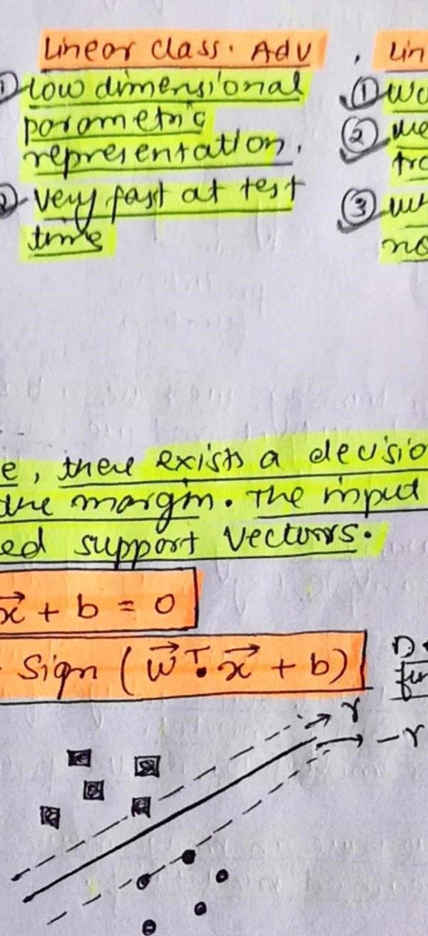
→ Support Vectors → for every decision hyperplane, there exists a decision boundary which is calculated by maximising the margin. The input points (vectors) lying on the decision boundary are called support vectors.

→ Decision hyperplane can be defined as $\boxed{\vec{w}^T \cdot \vec{x} + b = 0}$

→ for taking a decision, we will use $\boxed{D(\vec{x_i}) = \text{Sign}(\vec{w}^T \cdot \vec{x} + b)}$ Decision function. and look at the sign.

→ Margin Hyperplanes: $\vec{w}^T \cdot \vec{x} + b = \gamma$

$\vec{w}^T \cdot \vec{x} + b = -\gamma$



→ Scale invariance → $c\vec{w}^T \cdot \vec{x} + cb = 0$ This scaling does not change the decision hyperplane or the SV hyperplane. But with this, we eliminate a variable from optimization.

$\boxed{\vec{w}^{*T} \cdot \vec{x} + b^* = 1}$ $\boxed{\vec{w}^{*T} \vec{x} + b^* = -1}$

After scaling, we set the equations to 1 (Normalization).

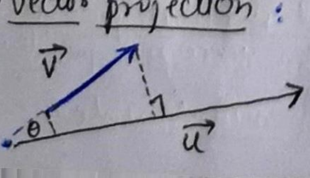
⇒ How to represent the size of the margin.

→ we represent the size of the margin in terms of w (always normalised w) by taking that there must be atleast one point that lies on each support hyperplane so that $\boxed{w^T \cdot x_1 + b = 1}$ and $\boxed{w^T \cdot (x_1 - x_2) = 2}$

$\boxed{w^T \cdot x_2 + b = -1}$

⇒ The vector W is perpendicular to the decision hyperplane i.e. if the dot product of 2 vectors (W and $x_1, x_2$) equals zero, the two vectors are perpendicular. The length of this vector (W) will be the distance b/w 2 hyperplanes.

→ Vector projection: projection of $\vec{V}$ into $\vec{u}$ $\quad \vec{V} \cdot \vec{u} = \|\vec{V}\| \|\vec{u}\| \cos\theta$



$\cos\theta = \frac{\text{adjacent}}{\text{hypotenuse}} \frac{\|\vec{V}\|}{\|goal\|}$ $\quad \cos\theta = \frac{\|goal\|}{\|\vec{V}\|}$, $\quad \frac{\|\vec{V}\| \|\vec{u}\|}{\|\vec{V}\| \|\vec{u}\|} \cos\theta = \frac{\|goal\|}{\|\vec{V}\|}$

$$\frac{\vec{v}\cdot\vec{u}}{\|\cancel{\vec{v}}\|\|\vec{u}\|} = \frac{\|\vec{goal}\|}{\|\cancel{\vec{v}}\|} \Rightarrow \boxed{\frac{\vec{v}\cdot\vec{u}}{\|\vec{u}\|} = \|\vec{goal}\|}$$

→ i.e. the margin is the projection of $x_1 - x_2$ into $W$ (the normal of the hyperplane)

$$\boxed{W^T(x_1 - x_2) = 2}$$

→ Projection → $\frac{\vec{v}\cdot\vec{u}}{\|\vec{u}\|}\vec{u} \simeq \frac{\vec{w}^T(x_1 - x_2)}{\|\vec{w}\|} \simeq$ Size of the margin $= \boxed{\frac{2}{\|\vec{w}\|}}$

→ Goal: Maximize the margin: $\boxed{\max \frac{2}{\|\vec{w}\|}}$ OR, $\boxed{\min \|\vec{w}\|}$

With this, we can have a decision boundary which separates the regions in a linear fashion where $t_i(\vec{w}^T x_i + b) \geq 1$

$\vec{w}^T x_i + b \geq 1$ if $t_i = 1$

$\vec{w}^T x_i + b \leq -1$ if $t_i = -1$

→ A problem: We want to have the best Hyperplane i.e. min $\|\vec{w}\|$ where $\underbrace{t_i(\vec{w}^T x_i + b) \geq 1}_{\text{constraint}}$. minimization

usually, the optimisation is done with "Lagrange multipliers".

Imp

$$\underbrace{L(\vec{w}, b)}_{\text{Optimize Primal}} = \frac{1}{2}\vec{w}\cdot\vec{w} - \sum_{i=0}^{N-1} \alpha_i [t_i((\vec{w}\cdot\vec{x}_i) + b) - 1]$$

Main equation

→ Lagrange multiplier + KKT condition

* If we show or optimize the Primal, we can solve the minimization problem.

→ Optimising the primal: Partial differentiation wrt b $\frac{\partial L(\vec{w}, b)}{\partial b} = 0$

$$\sum_{i=0}^{N-1} \alpha_i t_i = 0$$

Partial diff wrt w $\frac{\partial L(\vec{w}, b)}{\partial \vec{w}} = 0$

$$\vec{w} - \sum_{i=0}^{N-1} \alpha_i t_i x_i = 0 \; ; \; \boxed{\vec{w} = \sum_{i=0}^{N-1} \alpha_i t_i x_i}$$

Substituting the value of $\vec{w}$ in the Primal equation we get:

Imp

$$\boxed{W(\alpha) = \sum_{i=0}^{N-1} \alpha_i - \frac{1}{2}\sum_{i,j=0}^{N-1} \alpha_i \alpha_j t_i t_j (\vec{x}_i \cdot \vec{x}_j)} \quad \text{where } \alpha_i \geq 0$$

we solve this quadratic ~~equation~~ Programming to identify the Lagrange's multipliers

KKT condition → Constraint * Lagrange multiplier = 0

$$\boxed{\alpha_i(1 - t_i(\vec{w}^T x_i + b)) = 0}$$

if $\alpha_i \neq 0$, then $t_i(\vec{w}^T x_i + b) = 1$ i.e. only points in the decision boundary contributes to the solution.



Final decision function $D(\vec{x}) = \text{sign}(\vec{w}^T\vec{x} + b) = \text{sign}\left(\left[\sum_{i=0}^{N-1} \alpha_i t_i \vec{x}_i\right]^T \vec{x} + b\right) = \text{sign}\left(\left[\sum_{i=0}^{N-1} \alpha_i t_i (\vec{x}_i^T \cdot \vec{x})\right] + b\right)$

[illegible] svm → It is based on the idea that the original input space can always be mapped to some higher dimensional feature space where the training set is separable.

Kernel trick → It states that, instead of explicitly computing the lifting transformation $\varphi(x)$, define a kernel function $K$ such that $\boxed{K(x,y) = \varphi(x)\cdot\varphi(y)}$

→ (To be valid, the kernel function must satisfy mercer's condition)

$$[wx+b = \sum_i \alpha_i y_i x_i \cdot x + b]$$ original function linear svm.

learned weight — support vector

Kernel trick $$\boxed{\sum_i \alpha_i y_i \varphi(x_i)\cdot\varphi(x) + b = \sum_i \alpha_i y_i K(x_i, x) + b}$$ } This gives a non linear decision boundary in the original feature space.

Different kernels

① Polynomial kernels → $K(x,y) = (c + x\cdot y)^d$

② Gaussian " " → $K(x,y) = \exp\left(-\frac{1}{\sigma^2}\|x-y\|^2\right)$

Support Vector machines

Pros: ① Kernel based framework is very powerful, flexible

② Training is convex optimization, global optimal solution can be found

③ Amenable to theoretical analysis

④ Works very well, even with very small training samples.

Cons: ① There is no direct multi-class svm, must combine 2 class svm

② computation, memory.

Training Error
Testing Error } underfitting → Training and Test error are both high (model is too simple)

overfitting → Training error is low but testing error is high (model is too complex)