UNIT – II CLASSIFICATION

Unit II Classification

- Binary and Multiclass Classification: Assessing Classification Performance, Handling more than two classes, Multiclass Classification-One vs One, One vs Rest Linear Models: Perceptron, Support Vector Machines (SVM), Soft Margin SVM, Kernel methods for non-linearity

Some Important Definitions

- Instance
  - Objects of Interests
- Instance space
  - Set of all possible instances
- Label Space
- Output space
- Model
  - Mapping from the instance space to output space

Classification

- A *classifier is a mapping*

$$\hat{c} : \mathcal{X} \to \mathcal{C}$$

- *where C =* $\{C_1 , C_2 , \ldots , C_K\}$ *is a finite and usually small set of class labels*

Classification

- Binary
- Multiclass

Binary Classification

- Also called as concept Learning if positive class can be meaningfully called concept.

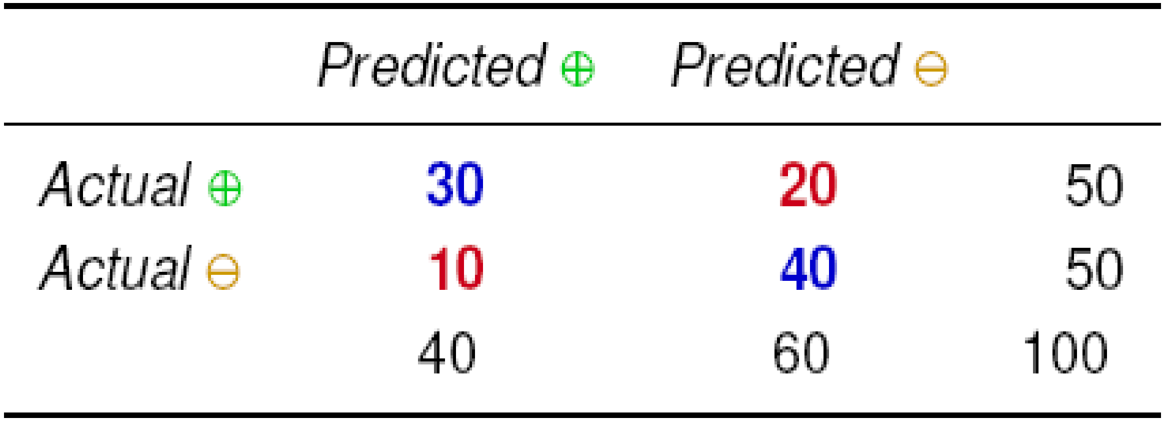
- Simplest classifier with two classes
  - Positive ⊕ or +1
  - Negative ⊖ or −1

Performance of classification

- Confusion Matrix or Contingency Table

Performance of classification

- Confusion Matrix or Contingency Table



	Predicted ⊕	*Predicted* ⊖	
Actual ⊕	30	20	50
Actual ⊖	10	40	50
	40	60	100

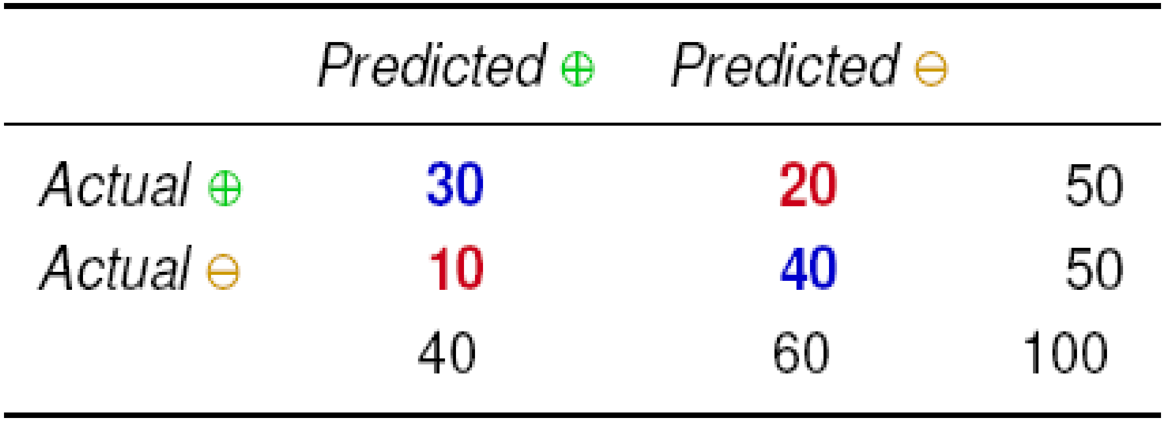
- True Positives
  - Correctly classified positives
- True Negatives
  - Correctly classified negatives

- False Negatives
  - Misclassified negatives
  - (incorrectly predicted as Negative)
- False Positives
  - Misclassified positives
  - (incorrectly predicted as Positive)

- True positive rate(tpr)
  - Proportion of positives correctly placed
  - Also called as sensitivity
- True negative rate(tnr)
  - Proportion of negatives correctly placed
  - Also called as specificity
- False positive rate(fpr)
- False negative rate(fnr)

Performance of classification

- Confusion Matrix or Contingency Table



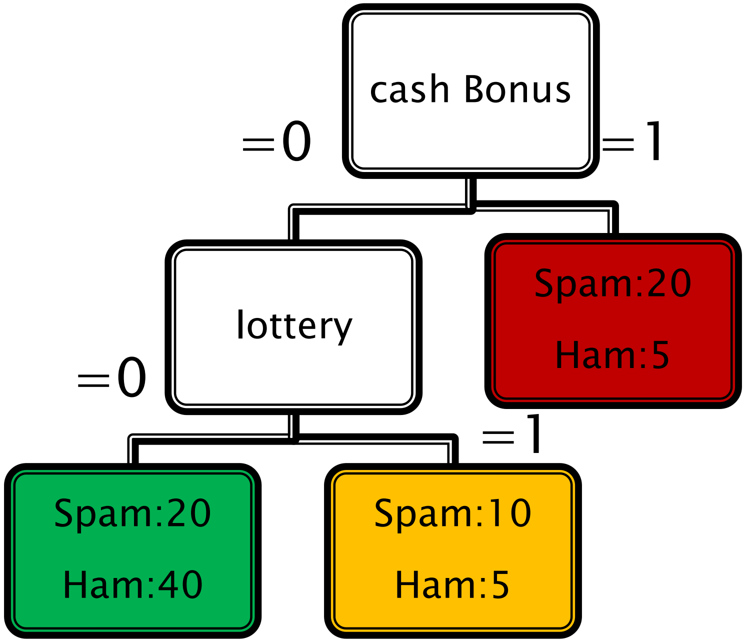
	Predicted ⊕	Predicted ⊖	
Actual ⊕	30	20	50
Actual ⊖	10	40	50
	40	60	100

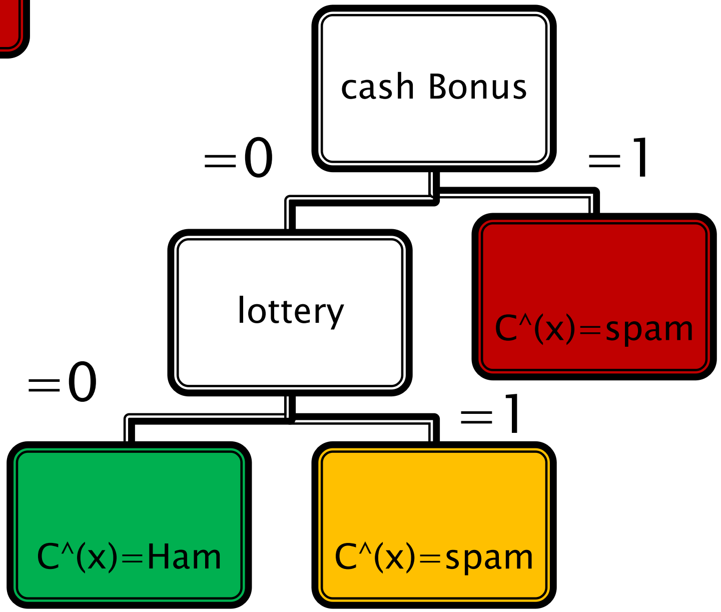
tpr =60%
tnr = 80%
fnr= 40%
fpr = 20%

Precision and Recall

- Recall : sometimes called as sensitivity or specificity (TPR)
- Precision is a counterpart to true positive rate in the following sense:
  - precision is the proportion of actual positives among the predicted positives.

Classification





	Yes	No	
Yes	100	5	105
No	10	50	60
	110	55	165

165 patients were tested for the presence of the disease. Out of those 165 cases, the classifier predicted "yes" 110 times, and "no" 55 times.

In reality, 105 patients in the sample have the disease, and 60 patients do not.

	Yes	No	
Yes	100	5	105
No	10	50	60
	110	55	165

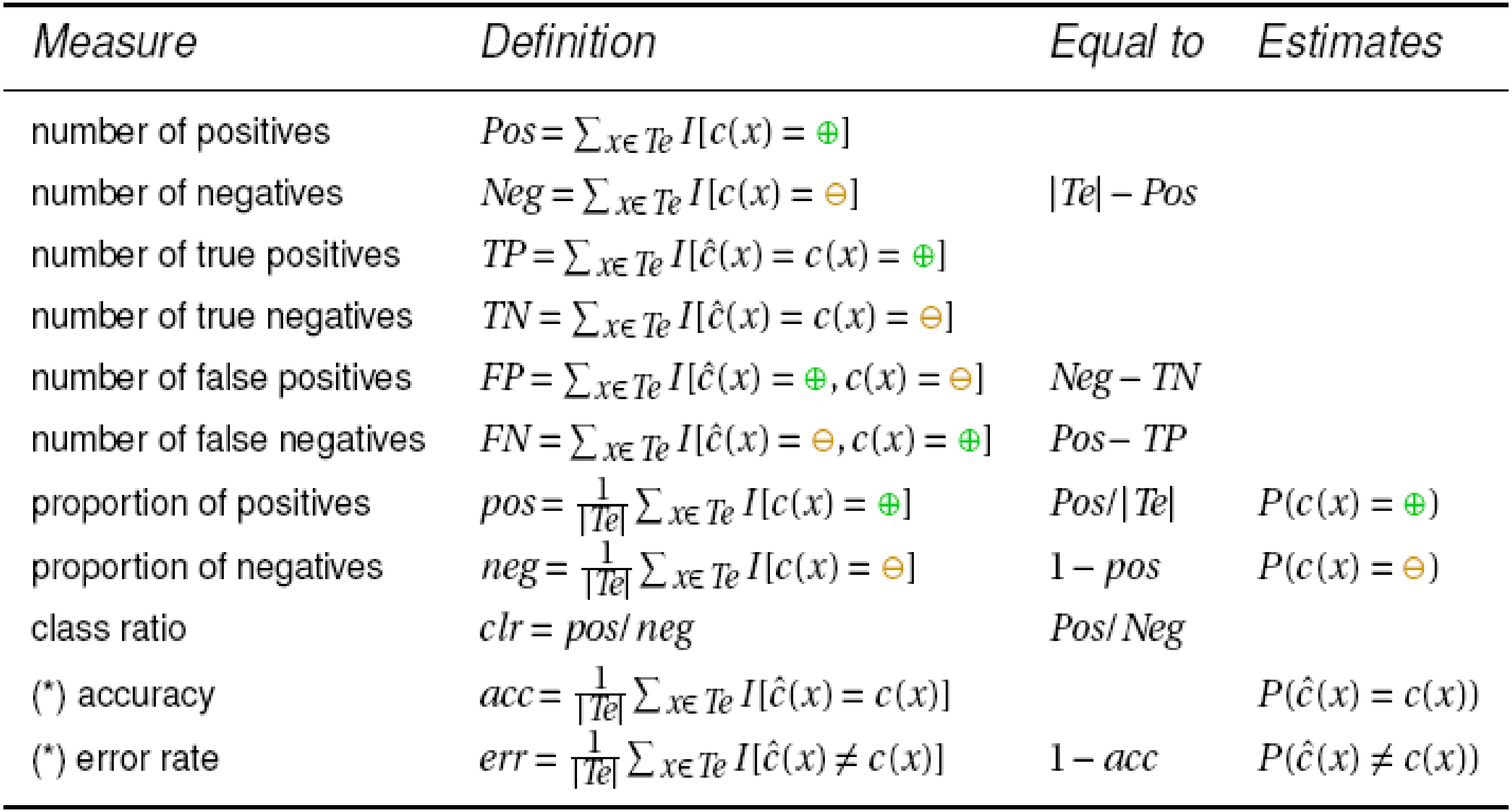
Calculate

tpr, tnr,fpr,fnr,pos,neg, accuracy, error rate precision and recall

- **pos= 105/165 = 0.64**
- **neg= 60/165=.36**
- **tpr= 100/105 =0.95**
- **tnr = 50/60 = 0.83**
- **fpr= 10/60 = 0.17**
- **fnr= 5/105 = 0.05**
- **Accracy** = (TP+TN)/total = (100+50)/165 = 0.91
- **or acc= pos . tpr + neg . tnr = 0.64 * 0.95 + 0.36 * 0.83 = 0.91**
- **Error rate**
  - 1- acc = 0.09

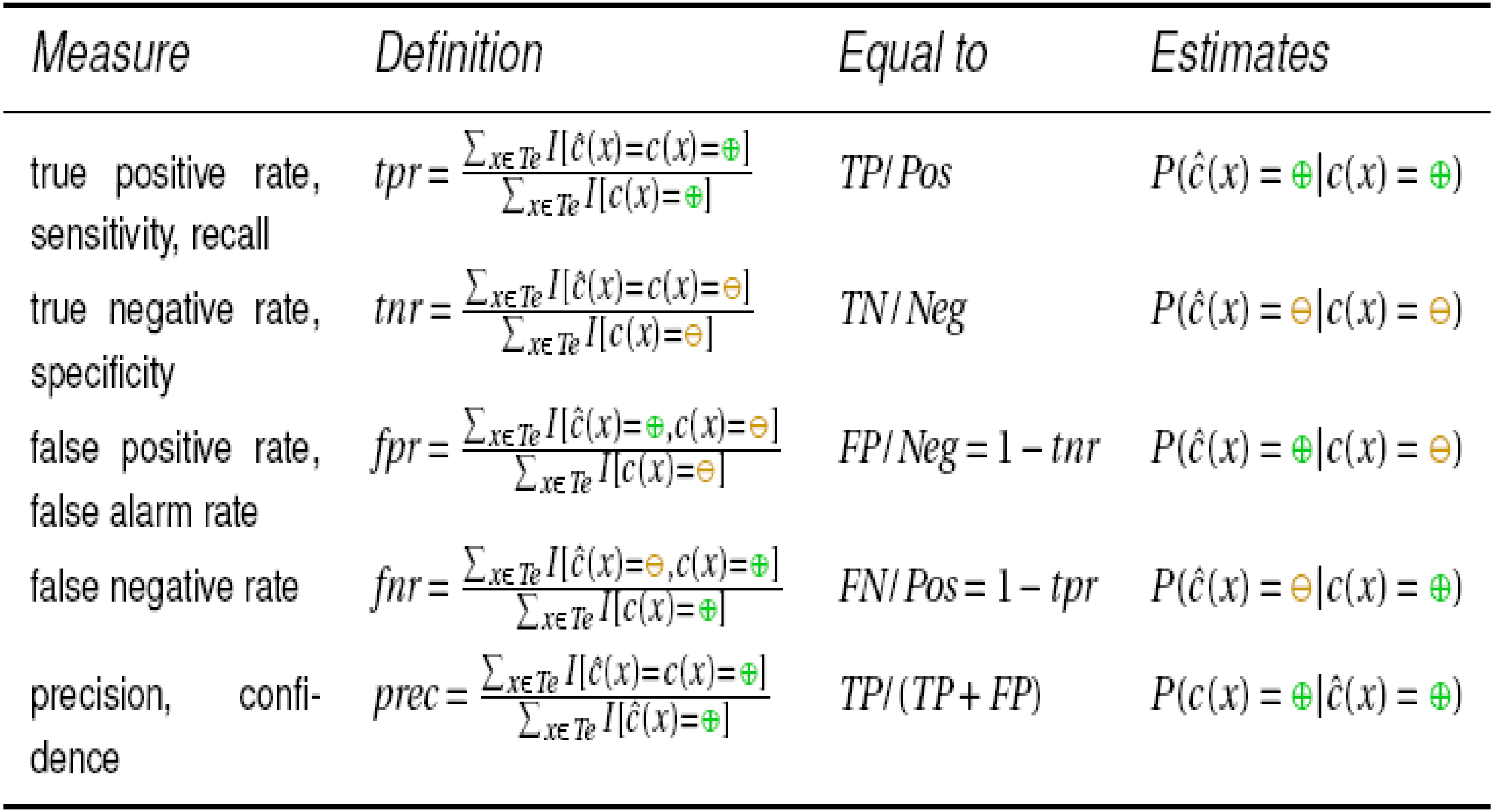
  Or
  - (FP+FN)/total = (10+5)/165 = 0.09
- **Precision:**
  - TP/predicted yes = 100/110 = 0.91

Performance Measure



Measure	*Definition*	*Equal to*	*Estimates*
number of positives	$Pos = \sum_{x \in Te} I[c(x) = \oplus]$		
number of negatives	$Neg = \sum_{x \in Te} I[c(x) = \ominus]$	$\lvert Te \rvert - Pos$	
number of true positives	$TP = \sum_{x \in Te} I[\hat{c}(x) = c(x) = \oplus]$		
number of true negatives	$TN = \sum_{x \in Te} I[\hat{c}(x) = c(x) = \ominus]$		
number of false positives	$FP = \sum_{x \in Te} I[\hat{c}(x) = \oplus, c(x) = \ominus]$	$Neg - TN$	
number of false negatives	$FN = \sum_{x \in Te} I[\hat{c}(x) = \ominus, c(x) = \oplus]$	$Pos - TP$	
proportion of positives	$pos = \frac{1}{\lvert Te \rvert} \sum_{x \in Te} I[c(x) = \oplus]$	$Pos / \lvert Te \rvert$	$P(c(x) = \oplus)$
proportion of negatives	$neg = \frac{1}{\lvert Te \rvert} \sum_{x \in Te} I[c(x) = \ominus]$	$1 - pos$	$P(c(x) = \ominus)$
class ratio	$clr = pos / neg$	Pos / Neg	
(*) accuracy	$acc = \frac{1}{\lvert Te \rvert} \sum_{x \in Te} I[\hat{c}(x) = c(x)]$		$P(\hat{c}(x) = c(x))$
(*) error rate	$err = \frac{1}{\lvert Te \rvert} \sum_{x \in Te} I[\hat{c}(x) \neq c(x)]$	$1 - acc$	$P(\hat{c}(x) \neq c(x))$

Performance Measure

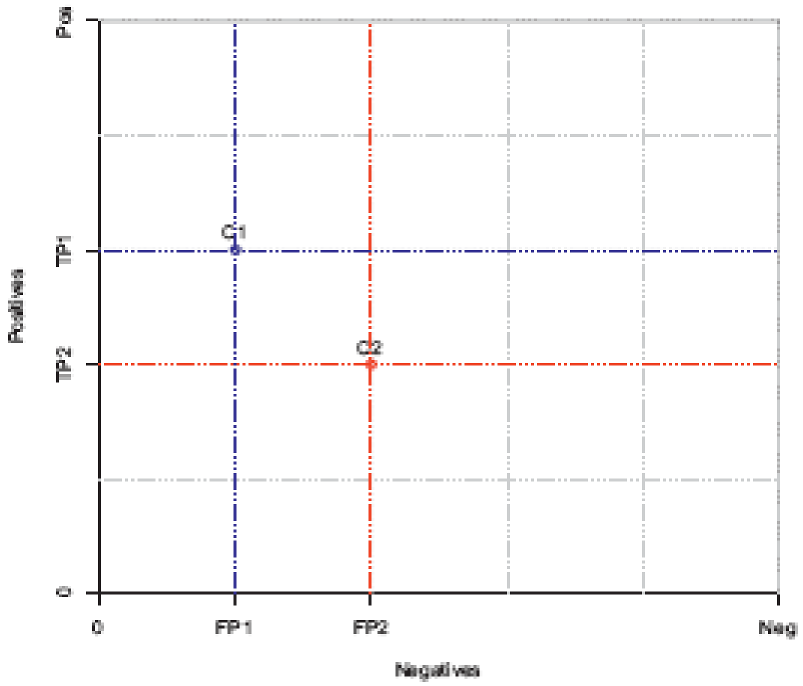


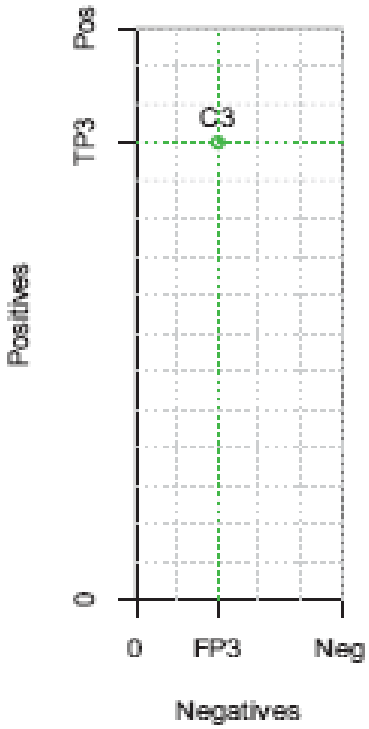
Measure	*Definition*	*Equal to*	*Estimates*
true positive rate, sensitivity, recall	$tpr = \frac{\sum_{x \in Te} I[\hat{c}(x) = c(x) = \oplus]}{\sum_{x \in Te} I[c(x) = \oplus]}$	TP/Pos	$P(\hat{c}(x) = \oplus \mid c(x) = \oplus)$
true negative rate, specificity	$tnr = \frac{\sum_{x \in Te} I[\hat{c}(x) = c(x) = \ominus]}{\sum_{x \in Te} I[c(x) = \ominus]}$	TN/Neg	$P(\hat{c}(x) = \ominus \mid c(x) = \ominus)$
false positive rate, false alarm rate	$fpr = \frac{\sum_{x \in Te} I[\hat{c}(x) = \oplus, c(x) = \ominus]}{\sum_{x \in Te} I[c(x) = \ominus]}$	$FP/Neg = 1 - tnr$	$P(\hat{c}(x) = \oplus \mid c(x) = \ominus)$
false negative rate	$fnr = \frac{\sum_{x \in Te} I[\hat{c}(x) = \ominus, c(x) = \oplus]}{\sum_{x \in Te} I[c(x) = \oplus]}$	$FN/Pos = 1 - tpr$	$P(\hat{c}(x) = \ominus \mid c(x) = \oplus)$
precision, confidence	$prec = \frac{\sum_{x \in Te} I[\hat{c}(x) = c(x) = \oplus]}{\sum_{x \in Te} I[\hat{c}(x) = \oplus]}$	$TP/(TP + FP)$	$P(c(x) = \oplus \mid \hat{c}(x) = \oplus)$

Coverage plot

- The number of positives *Pos,*
- *The number of negatives Neg, t*
- *The number of true positives TP*
- *The number of false positives FP.*
- *A coverage* plot visualises these four numbers by means of a rectangular coordinate system and a point

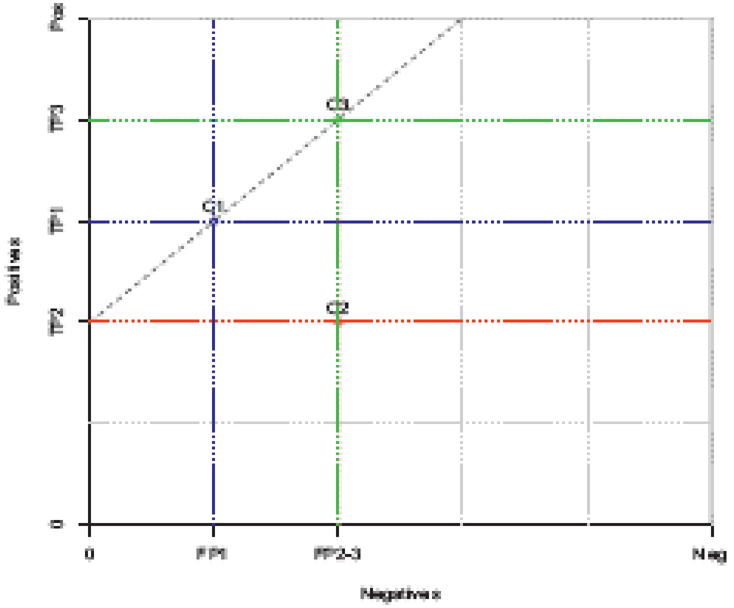
Coverage plot

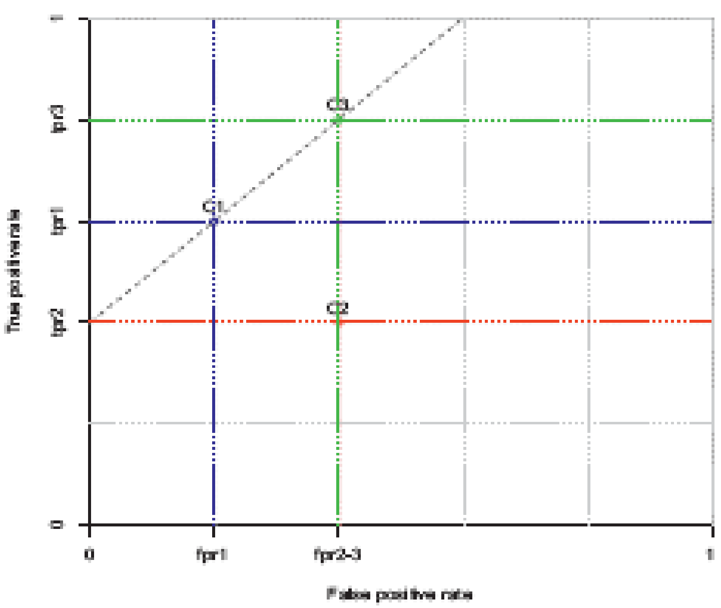




- C1 has both more true positives and fewer false positives than C2, and so is better in both respects
- If one classifier outperforms another classifier on all classes, the first one is said to *dominate the second*

Coverage plot





- Consider a third classifier C3, better than C1 on the positives but worse on the negatives
- Although both C1 and C3 dominate C2, neither of them dominates the other. Which one we prefer depends on whether we put more emphasis on the positives or on the negatives.

Coverage plot

- Notice that the line segment connecting C1 and C3 has a slope of 1
- whenever we gain a true positive, we also lose a true negative (or gain a false positive)
- This doesn’t affect the sum of true positives and true negatives, and hence the accuracy is the same wherever we are on the line.
- It follows that C1 and C3 have the same accuracy.
- **In a coverage plot, classifiers with the same accuracy are connected by line segments with slope 1.**

- consider the average of the true positive rate and the true negative rate, which we will call *average recall, denoted avg-rec.*
- On a line with slope 1 we have

avg-rec =(tpr +tnr)/2 =(tpr +1−fpr)/2 =(1+y0)/2,

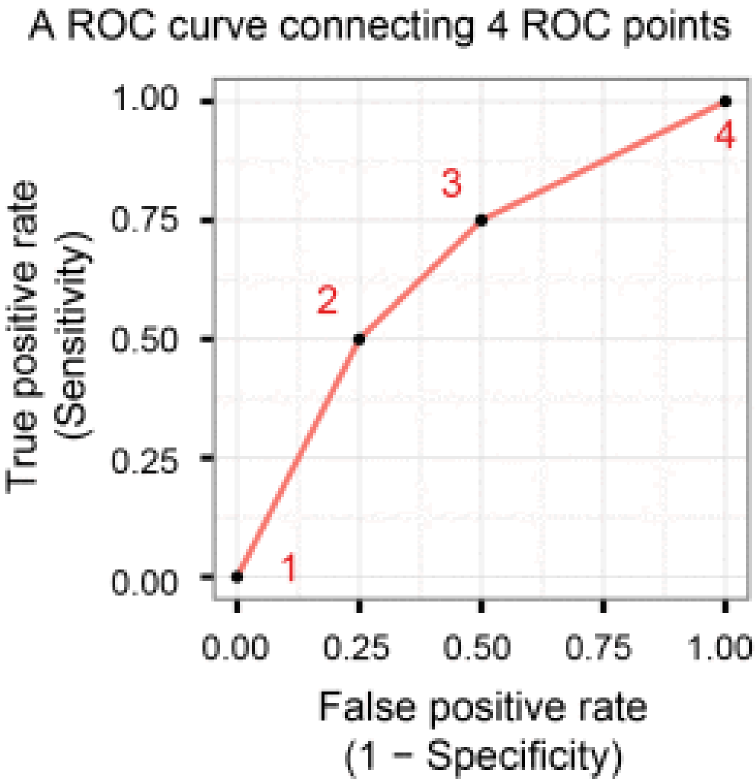
which is a constant

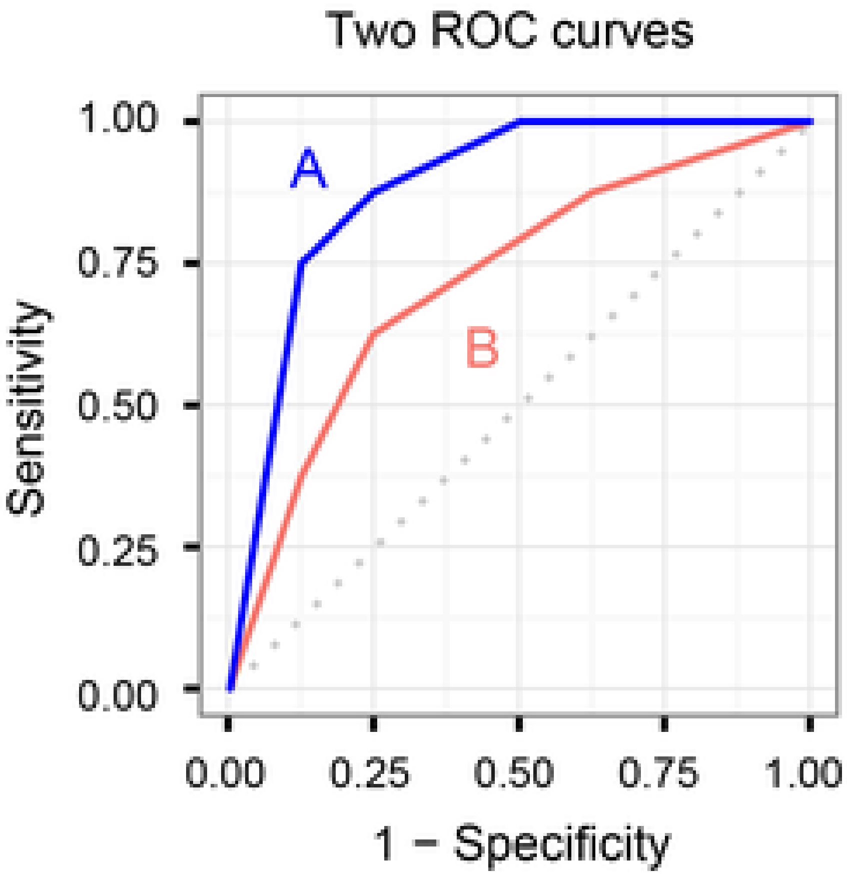
- In a normalised coverage plot, line segments with slope 1 connect classifiers with the same average recall.

ROC Curve

- Receiver operating Characteristics plot
- popular measure for evaluating classifier performance
- based on two basic evaluation measures – specificity and sensitivity.
- It is a plot of TPR (sensitivity) against FPR(1 – specificity)
- The ROC plot uses FPR on the x-axis and TPR on the y-axis.

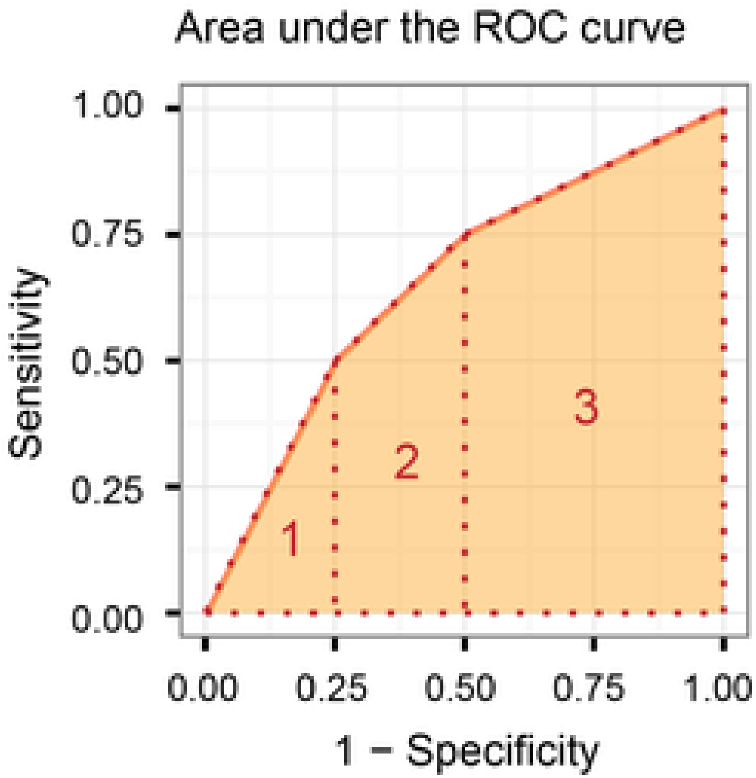
Threshold	Sensitivity	Specificity	1 – specificity
1	0.0	1.0	0.0
2	0.5	0.75	0.25
3	0.75	0.5	0.5
4	1.0	0.0	1.0

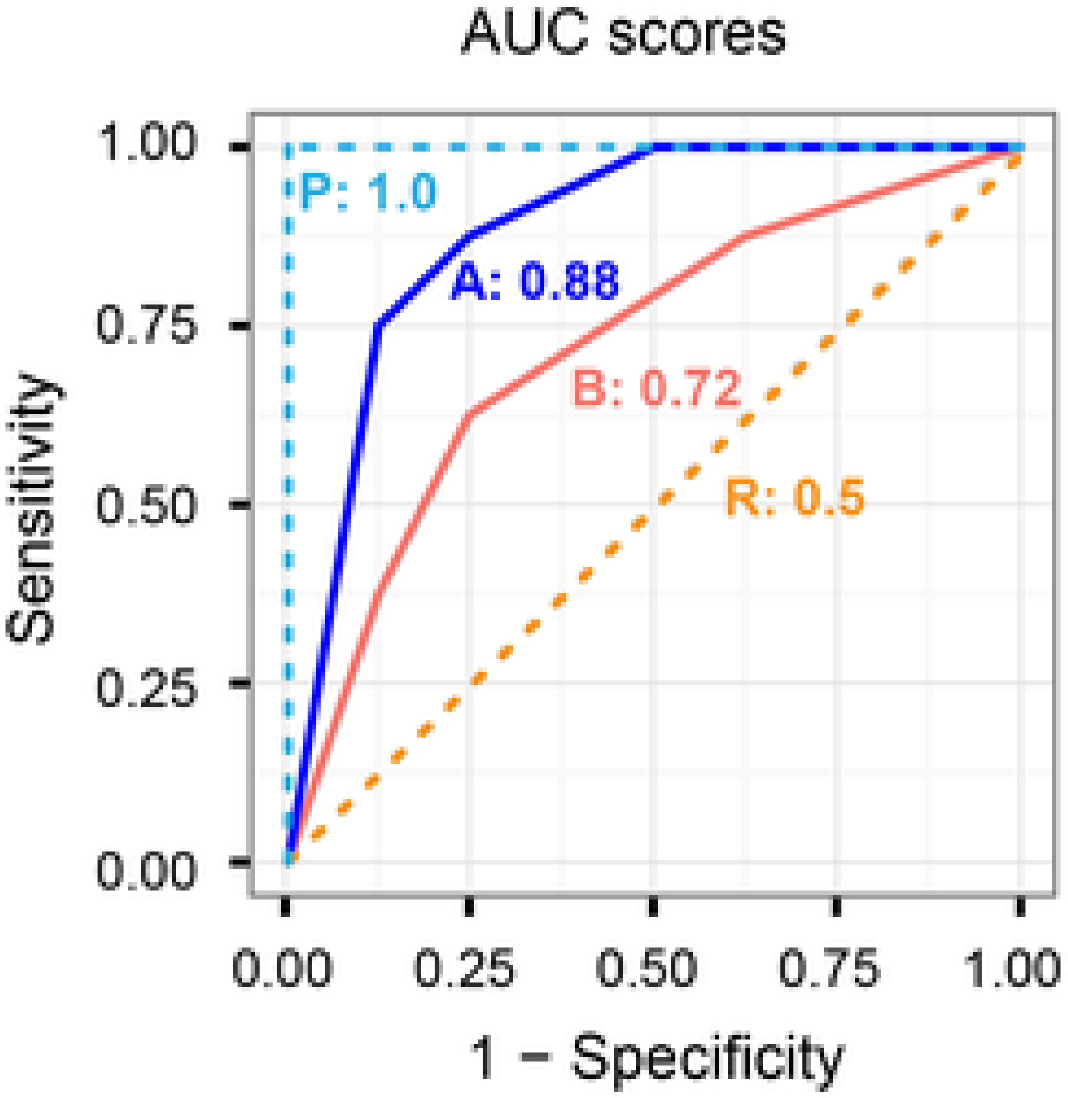


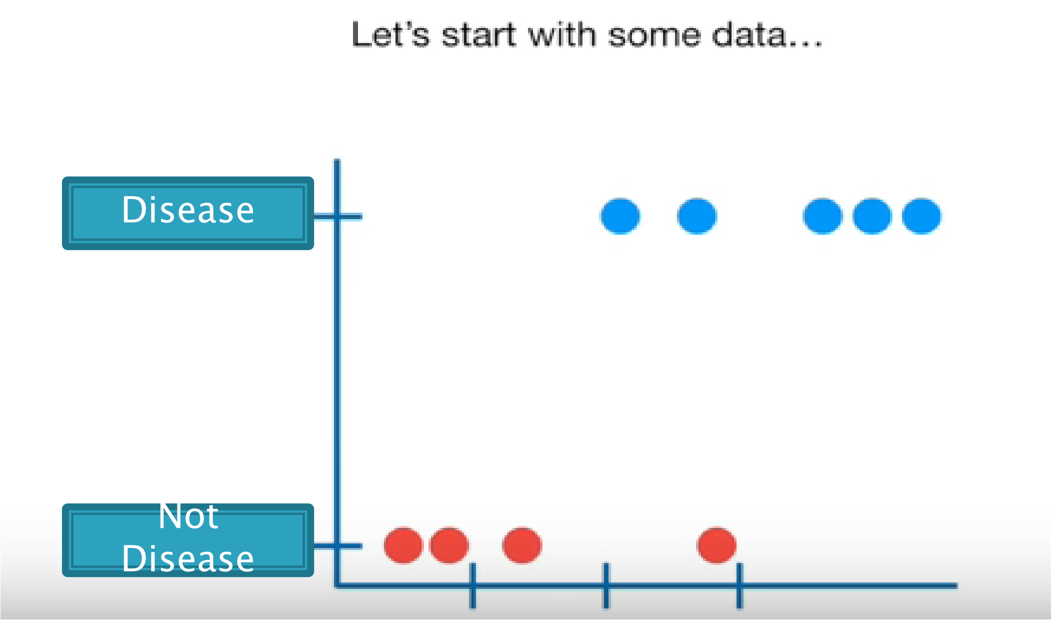

Two ROC curves
A
B
Sensitivity
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
1 − Specificity

- Area under the curve (AUC)
- If AUC is greater then that classifier is better
- If a classifier is perfect the AUC is 1
- If classifier makes all random guesses ,its AUC is 0.5

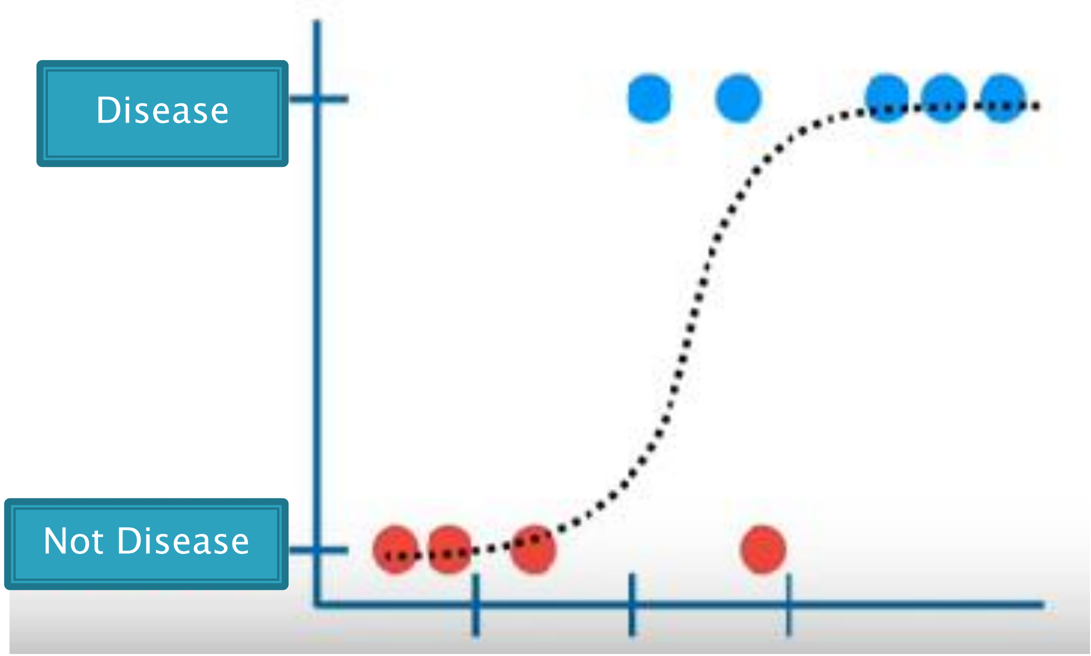
- The AUC score can be calculated by the trapezoidal rule, which is adding up all trapezoids under the curve. The areas of the three trapezoids 1, 2, 3 are 0.0625, 0.15625, and 0.4375. The AUC score is then 0.65625.

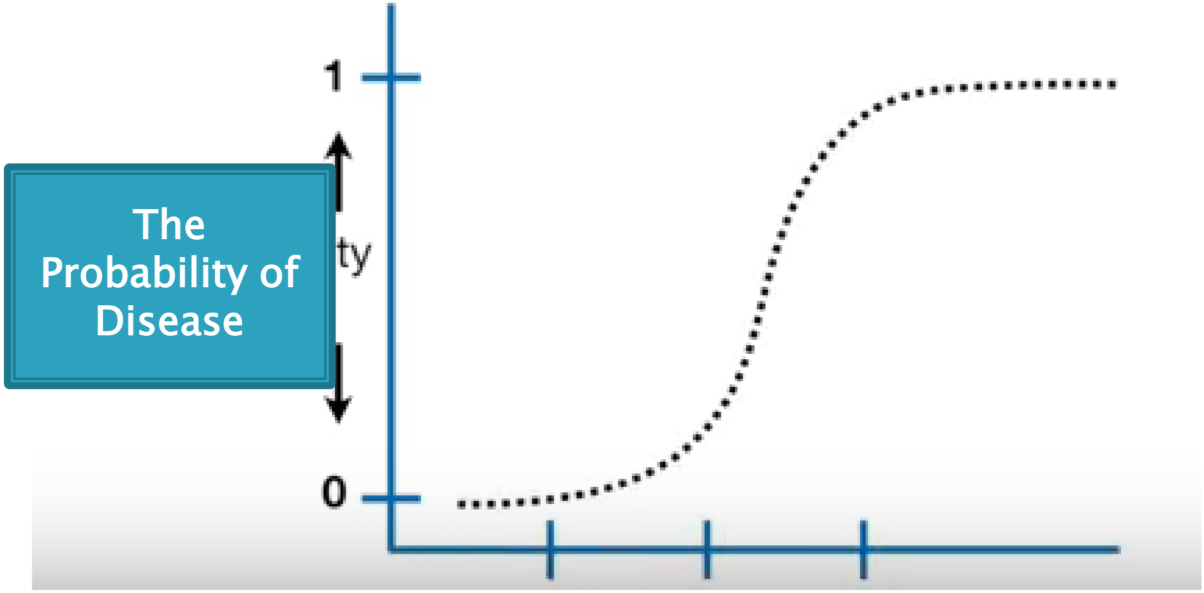



AUC scores
P: 1.0
A: 0.88
B: 0.72
R: 0.5
Sensitivity
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
1 − Specificity

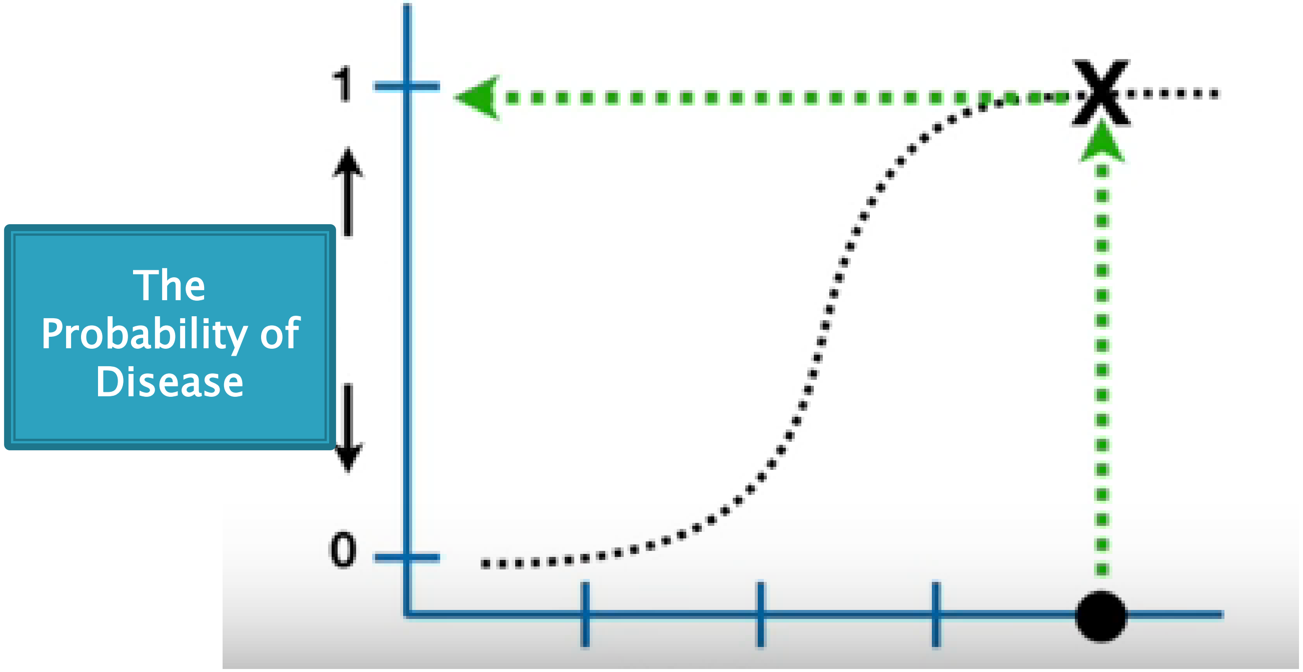


Let’s start with some data…

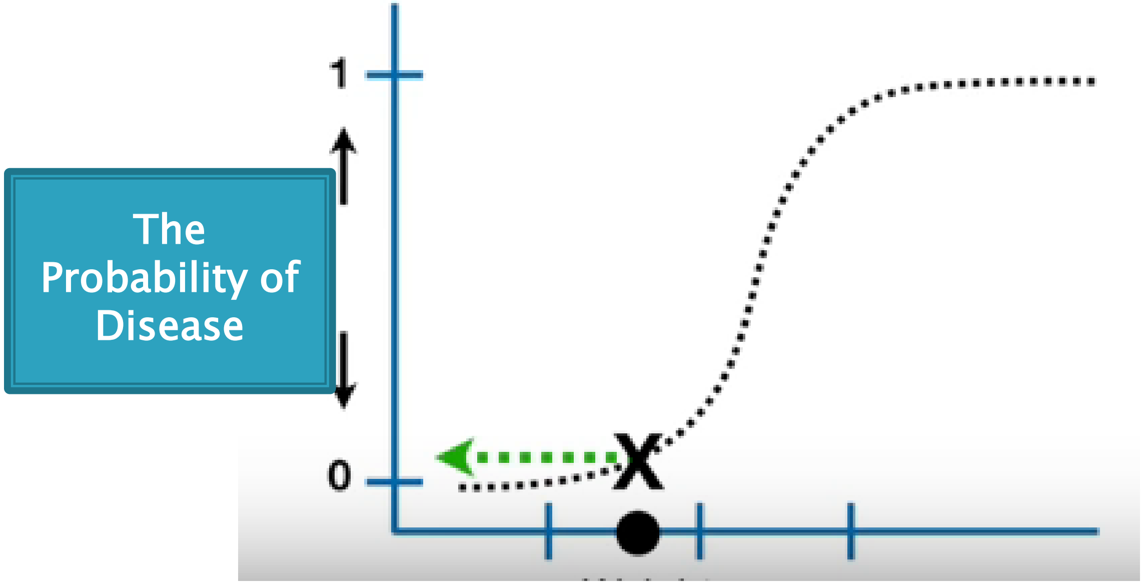

Disease
Not Disease



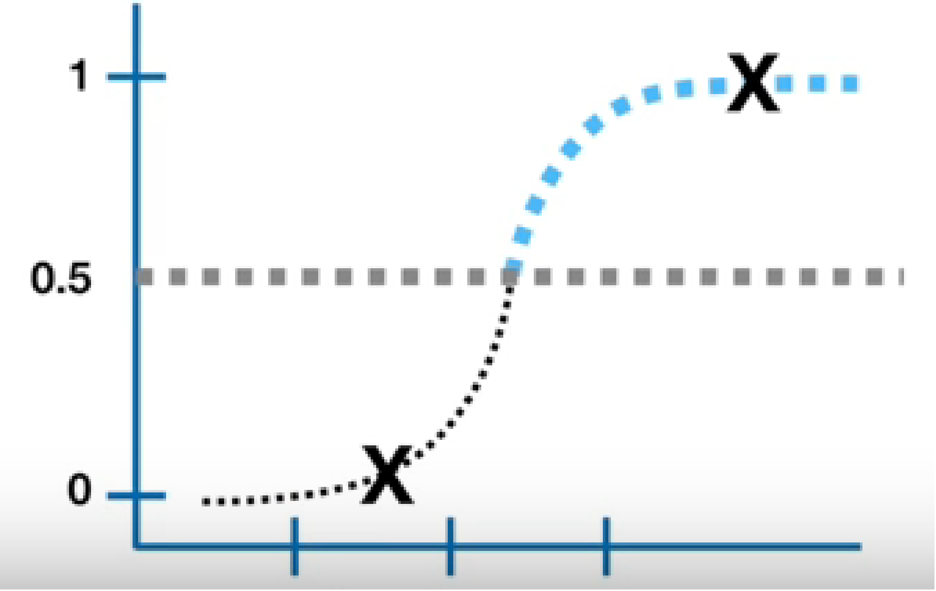
The Probability of Disease

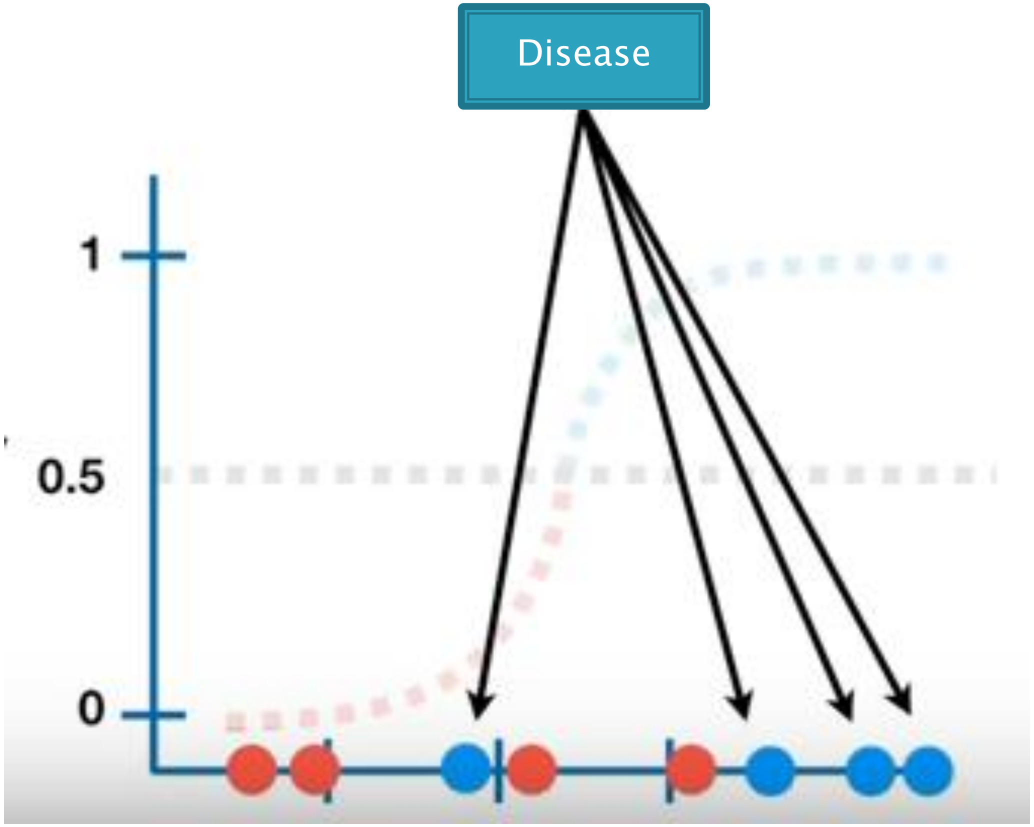


The Probability of Disease



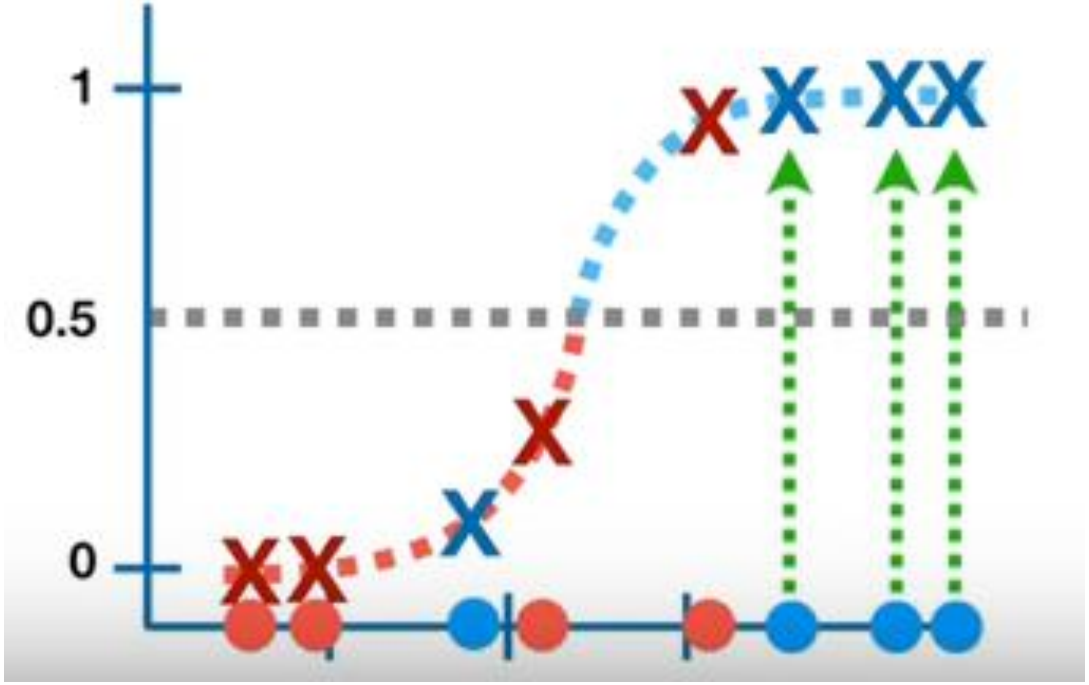
The Probability of Disease


1
0.5
0
X
X



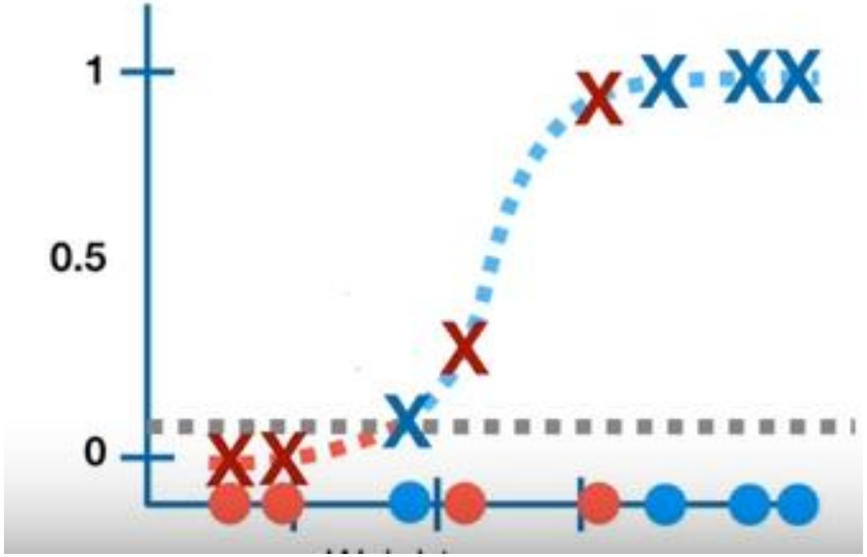
Disease

Threshold 0.5



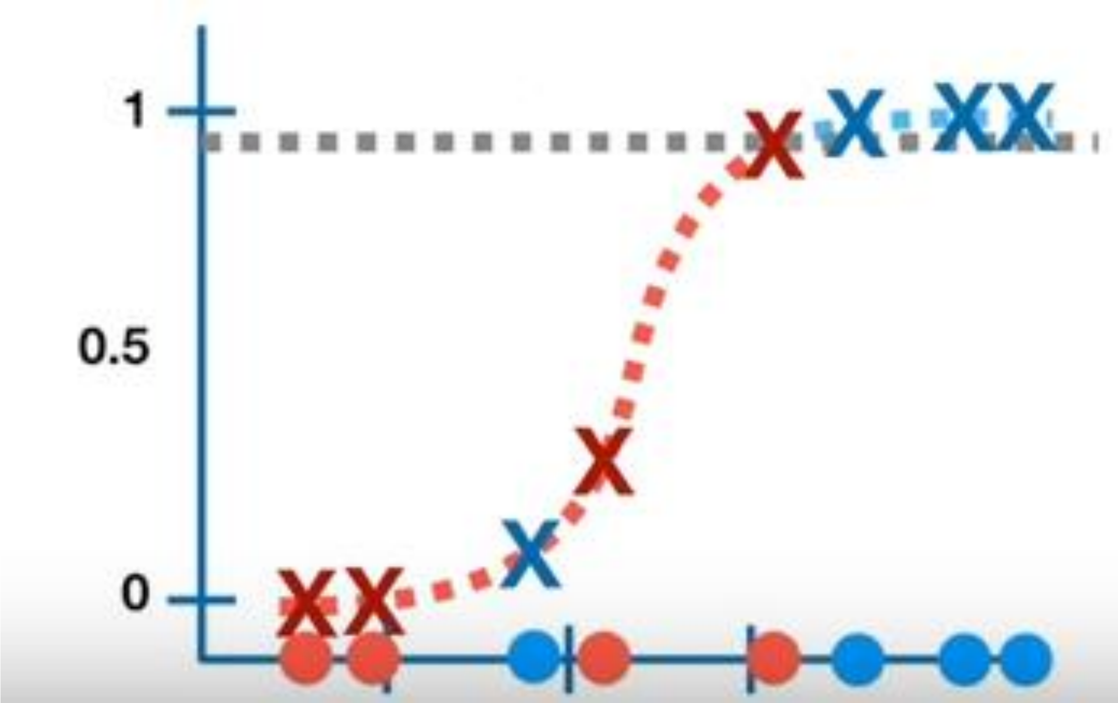
	Is having disease	Is not having disease
Is having disease	3	1
Is not having disease	1	3

Threshold 0.1

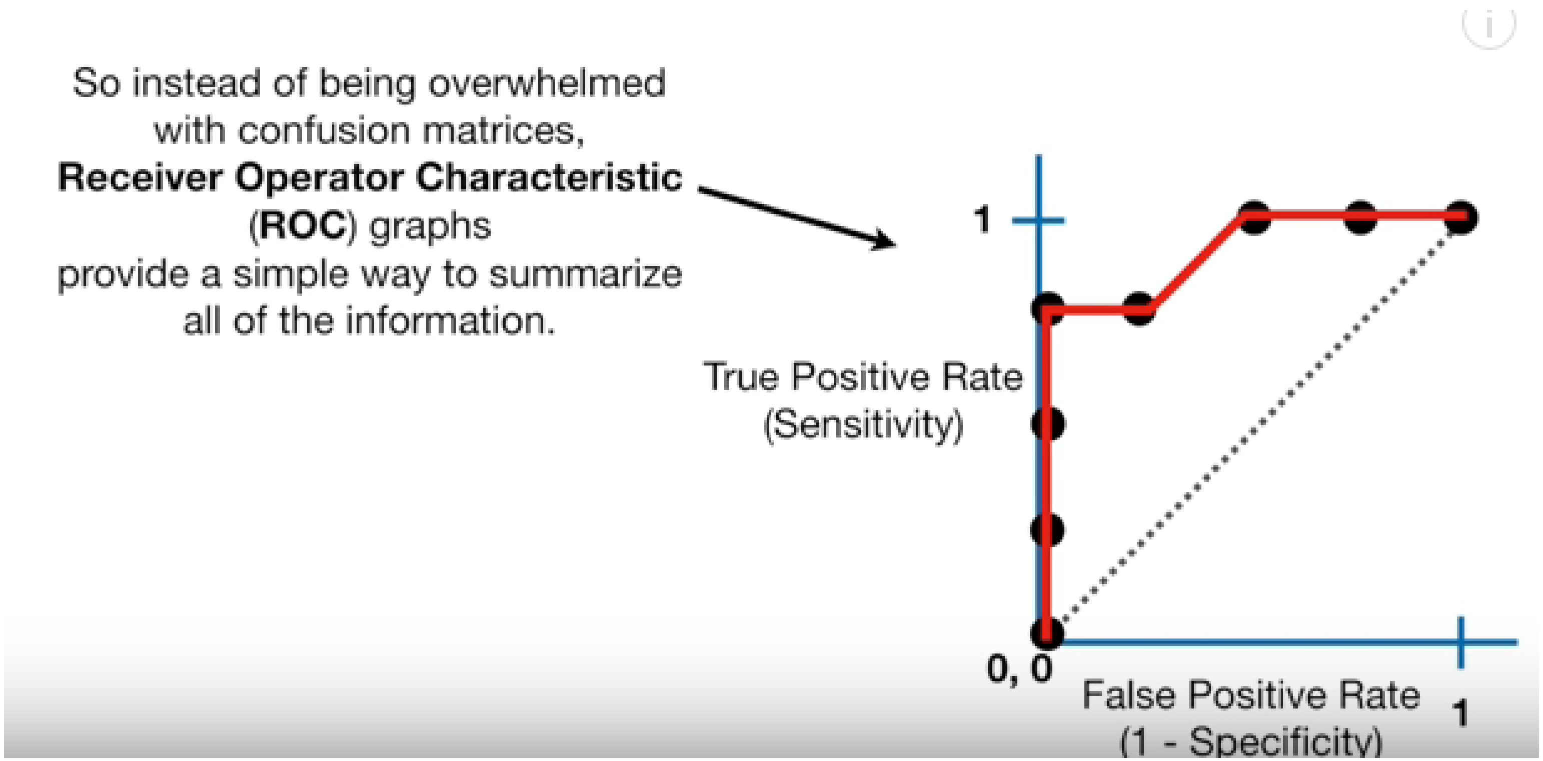


	Is having disease	Is not having disease
Is having disease	4	0
Is not having disease	2	2

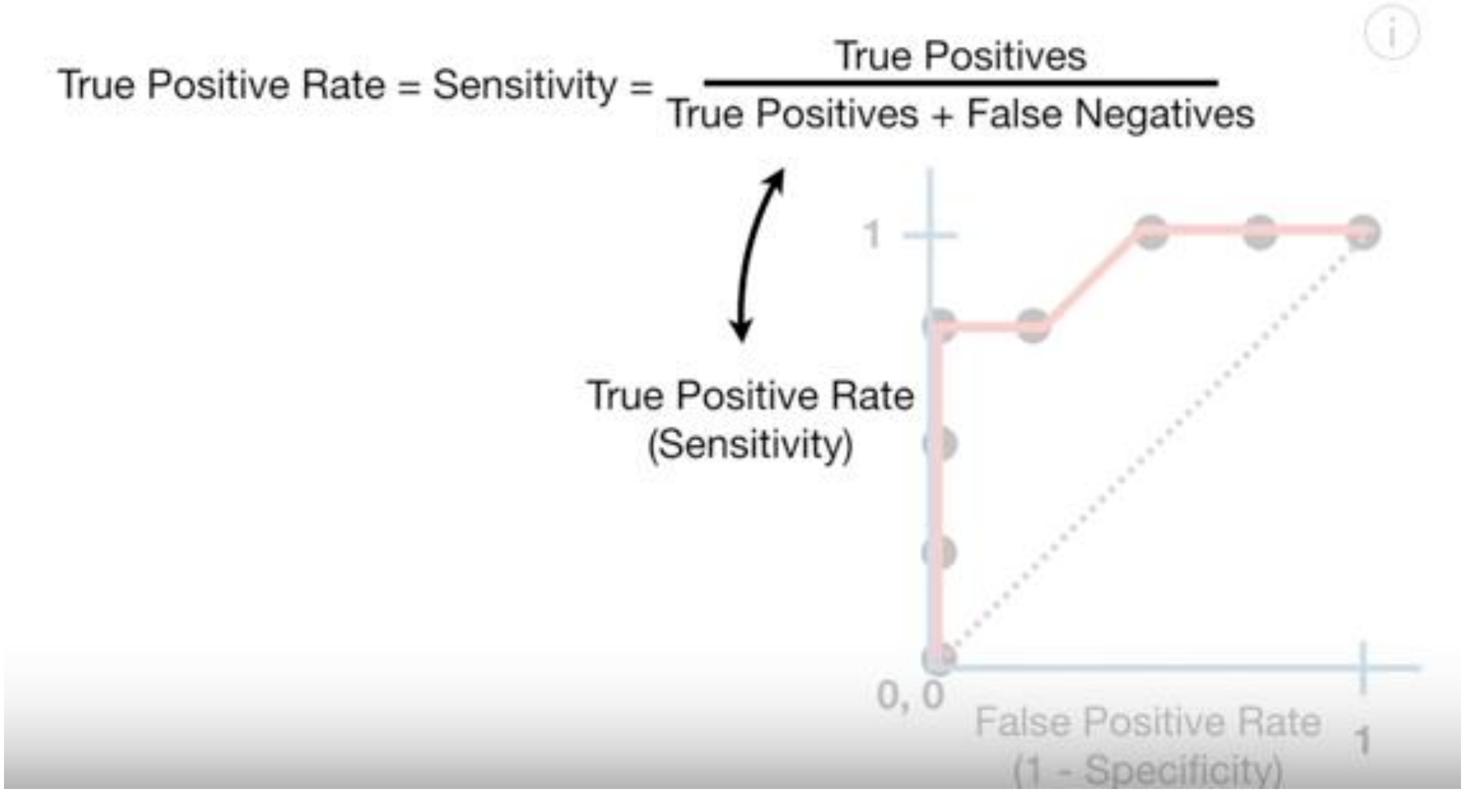
Threshold 0.9



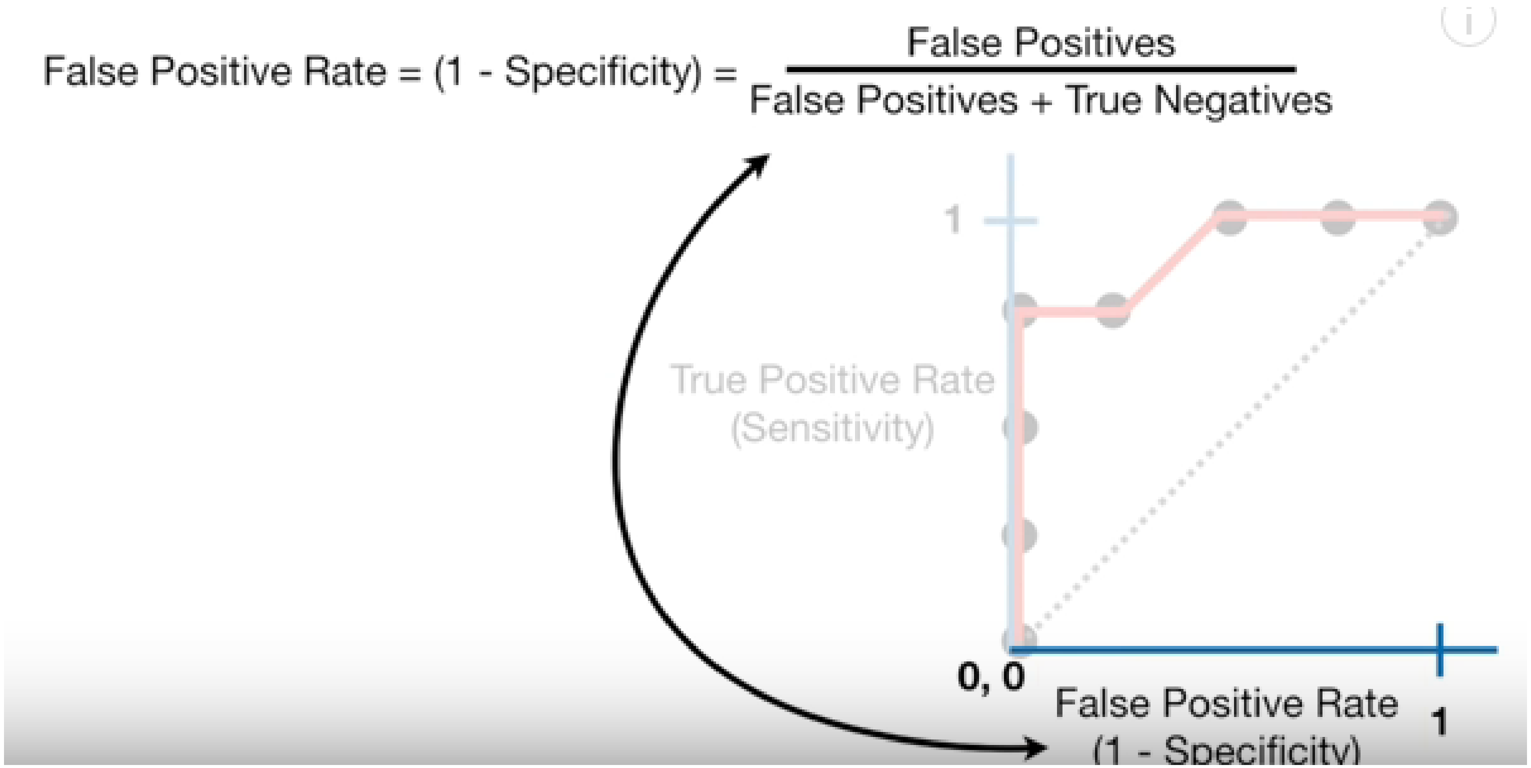
	Is having disease	Is not having disease
Is having disease	3	1
Is not having disease	0	4



So instead of being overwhelmed with confusion matrices, **Receiver Operator Characteristic** (**ROC**) graphs provide a simple way to summarize all of the information.



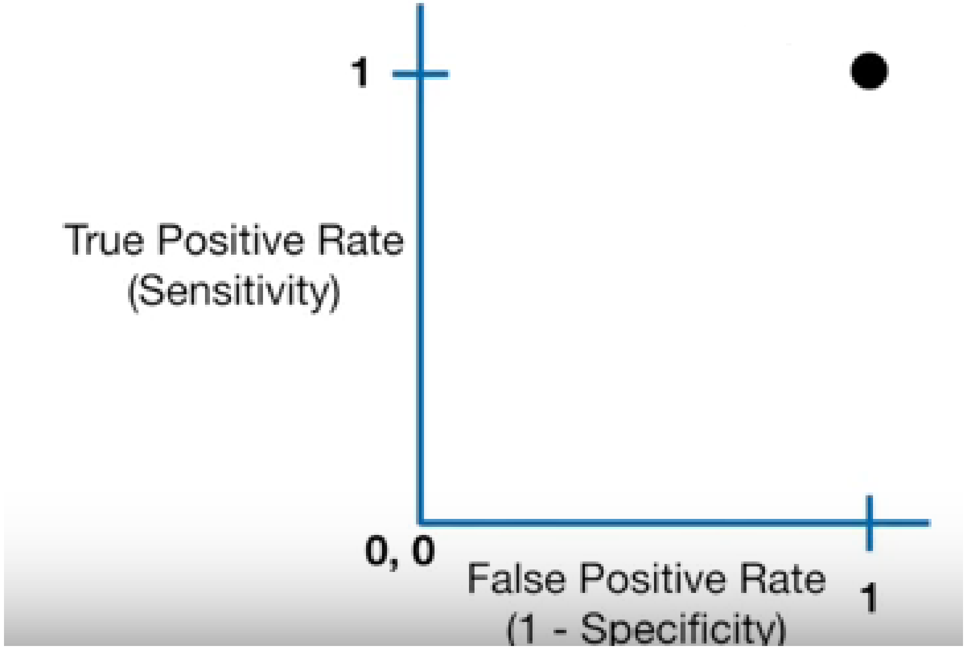
$$\text{True Positive Rate} = \text{Sensitivity} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$



$$\text{False Positive Rate} = (1 - \text{Specificity}) = \frac{\text{False Positives}}{\text{False Positives} + \text{True Negatives}}$$

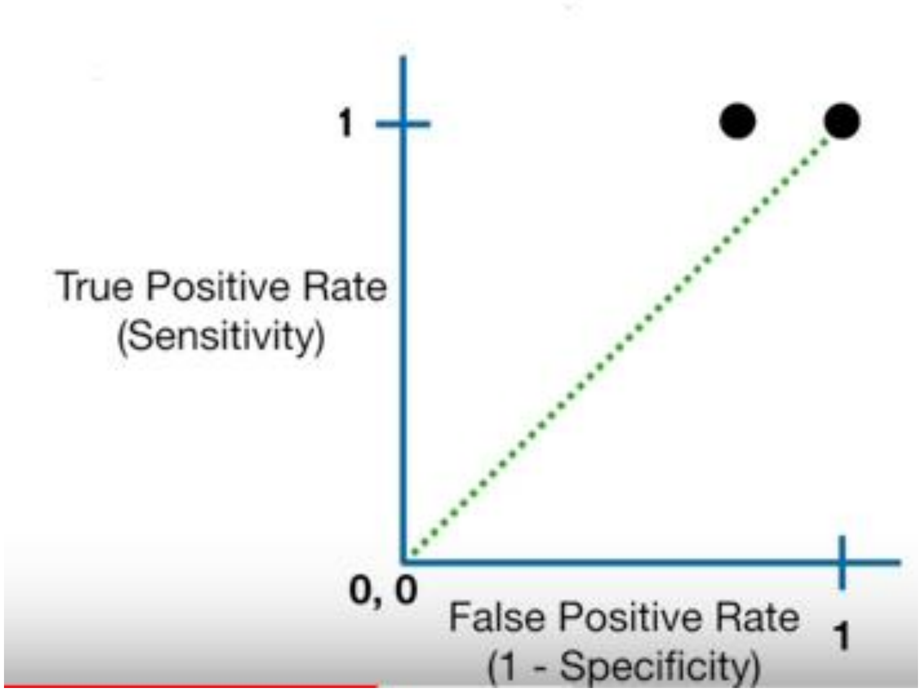
	Is having disease	Is not having disease
Is having disease	4	0
Is not having	4	0

Tpr =sensitivity = 1
Fpr = 1-specificity = 1



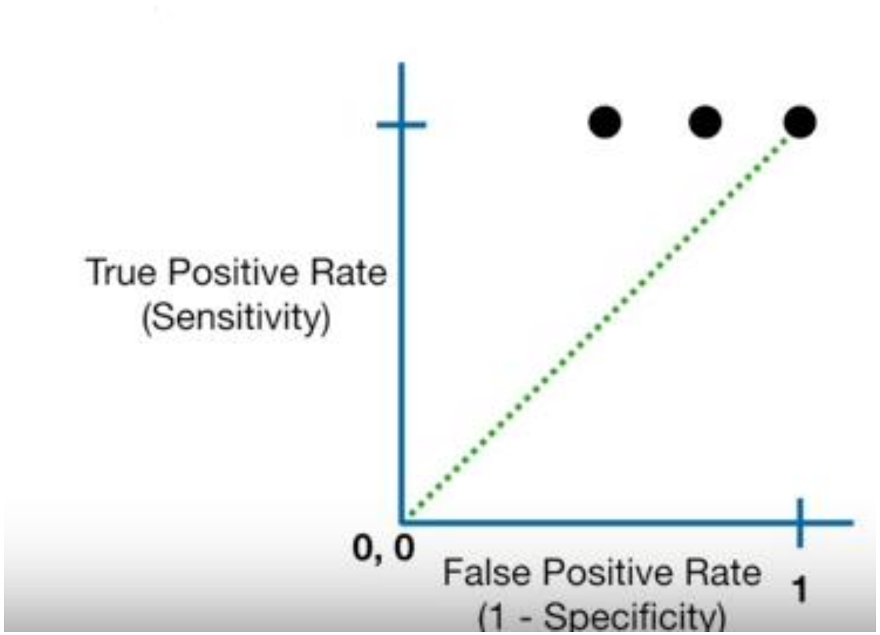
	Is having disease	Is not having disease
Is having disease	4	0
Is not having	3	1

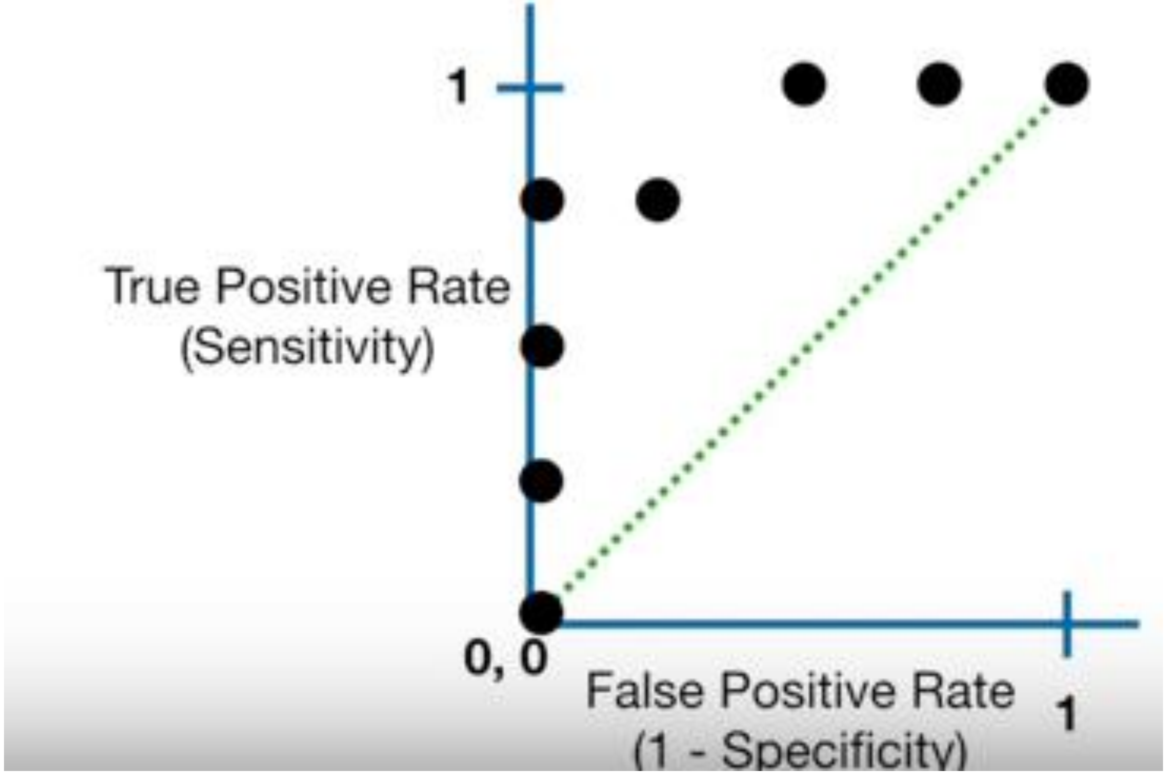
Tpr =sensitivity = 1
Fpr = 1-specificity = 0.75

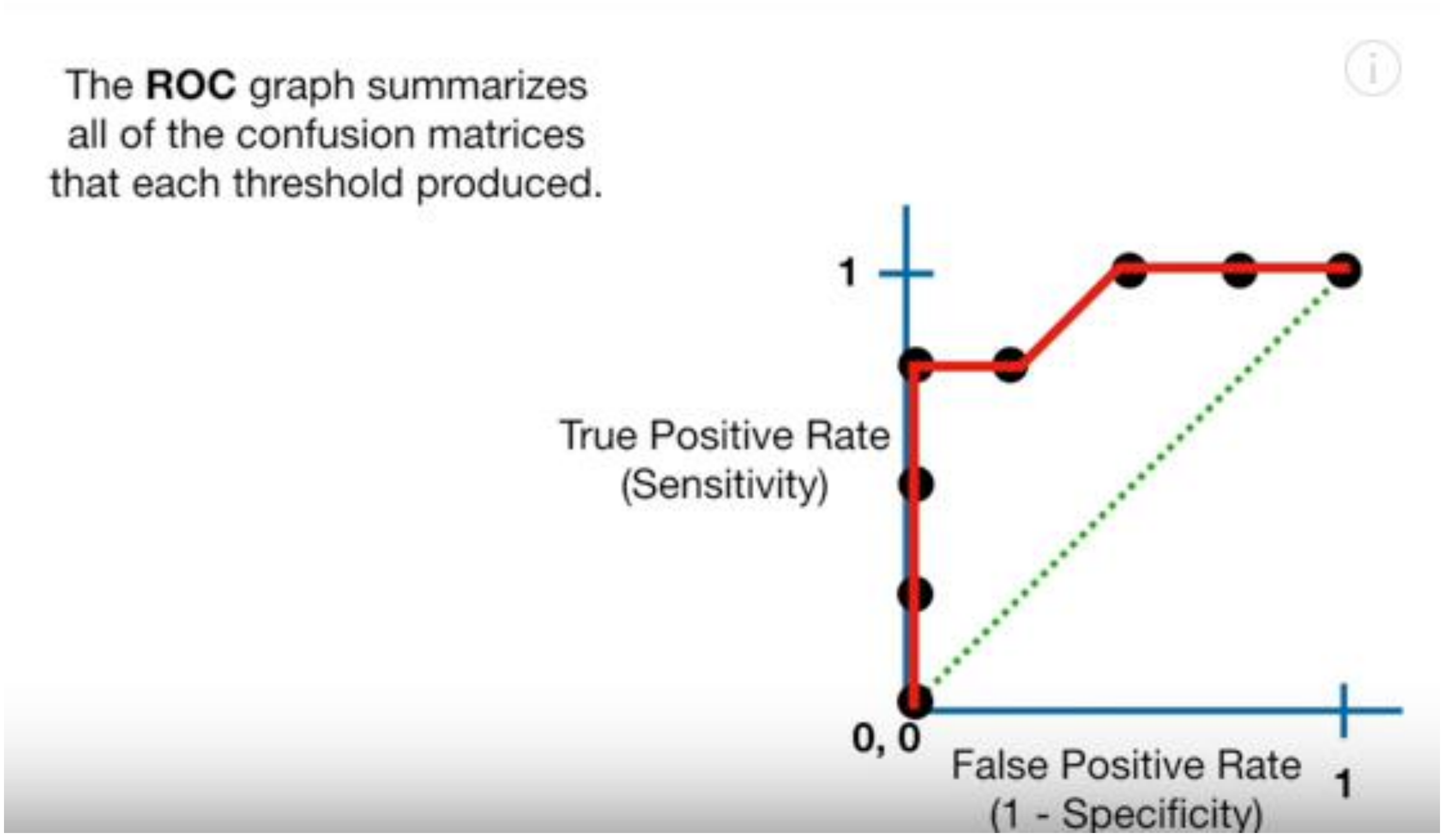


	Is having disease	Is not having disease
Is having disease	4	0
Is not having	2	2

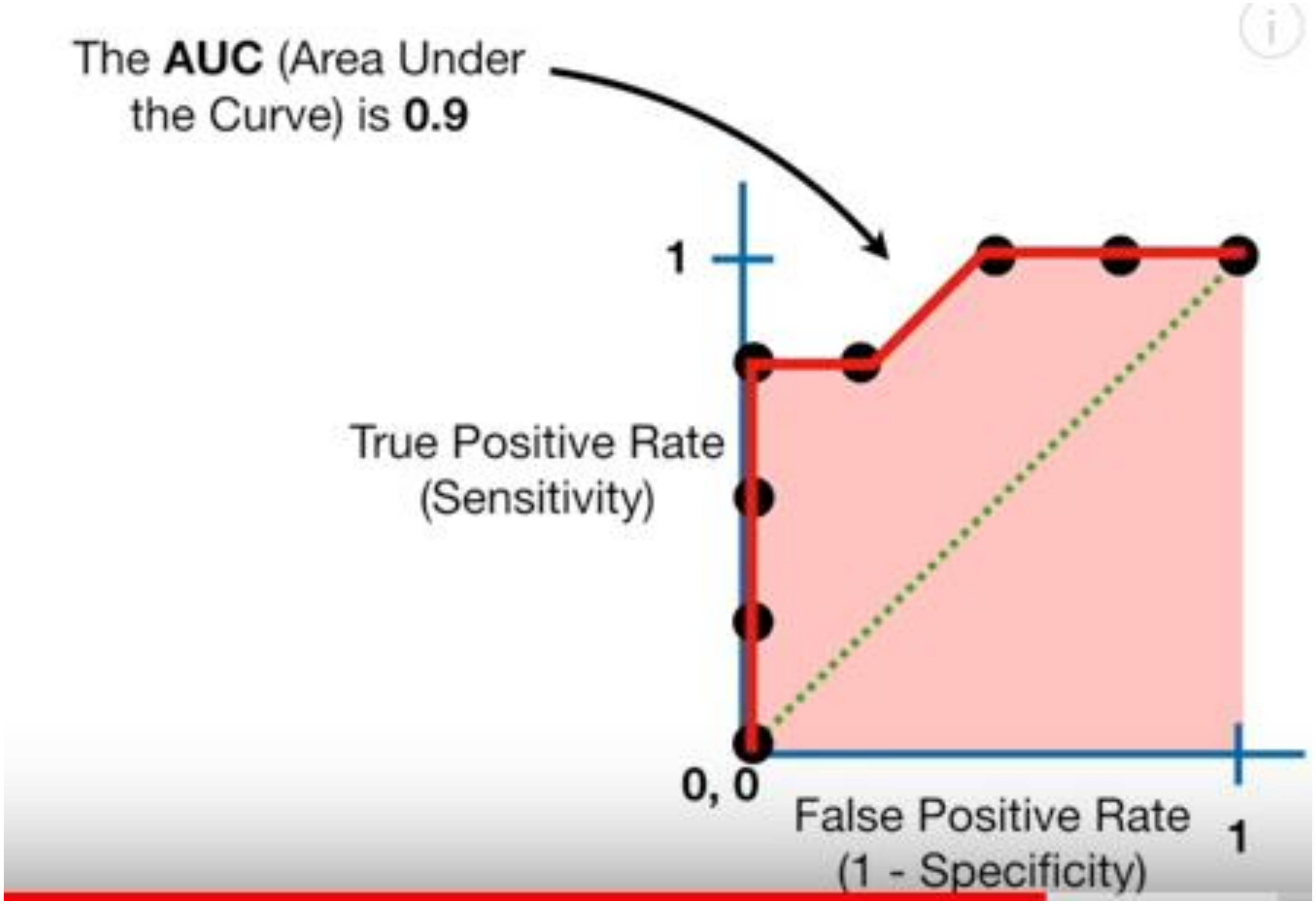
Tpr =sensitivity = 1
Fpr = 1-specificity = 0.5




1
True Positive Rate
(Sensitivity)
0, 0
False Positive Rate
(1 - Specificity)
1

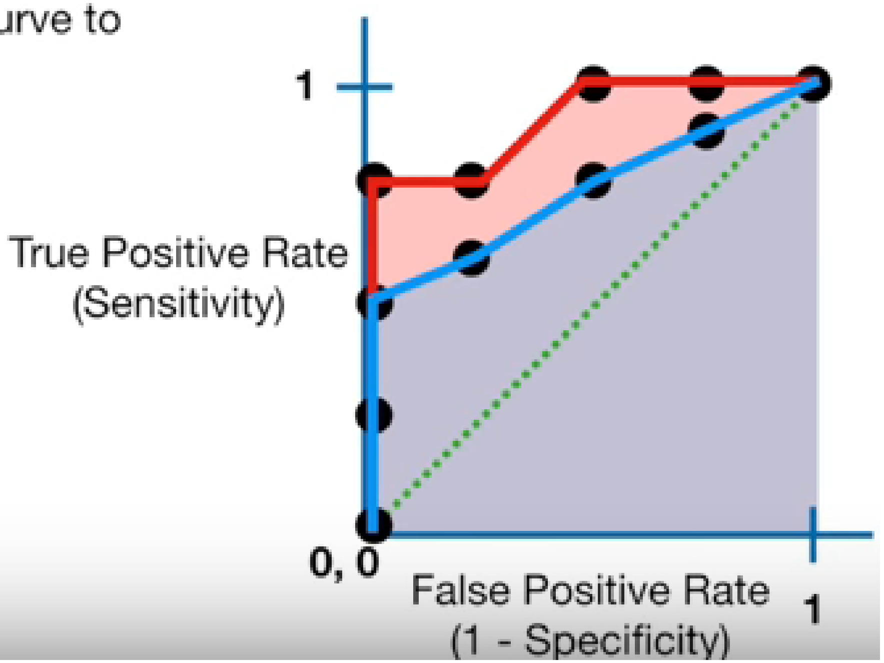


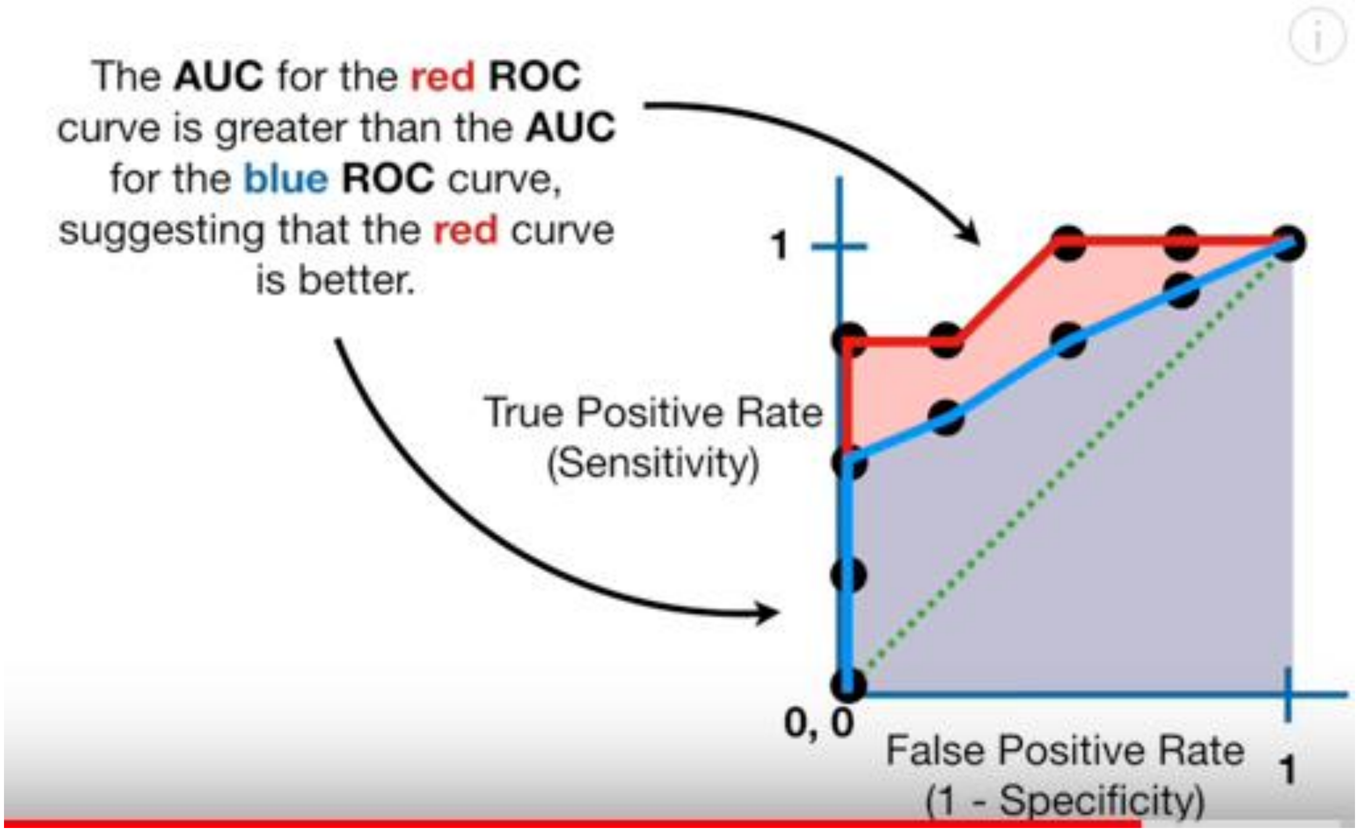
The **ROC** graph summarizes all of the confusion matrices that each threshold produced.



The **AUC** (Area Under the Curve) is **0.9**

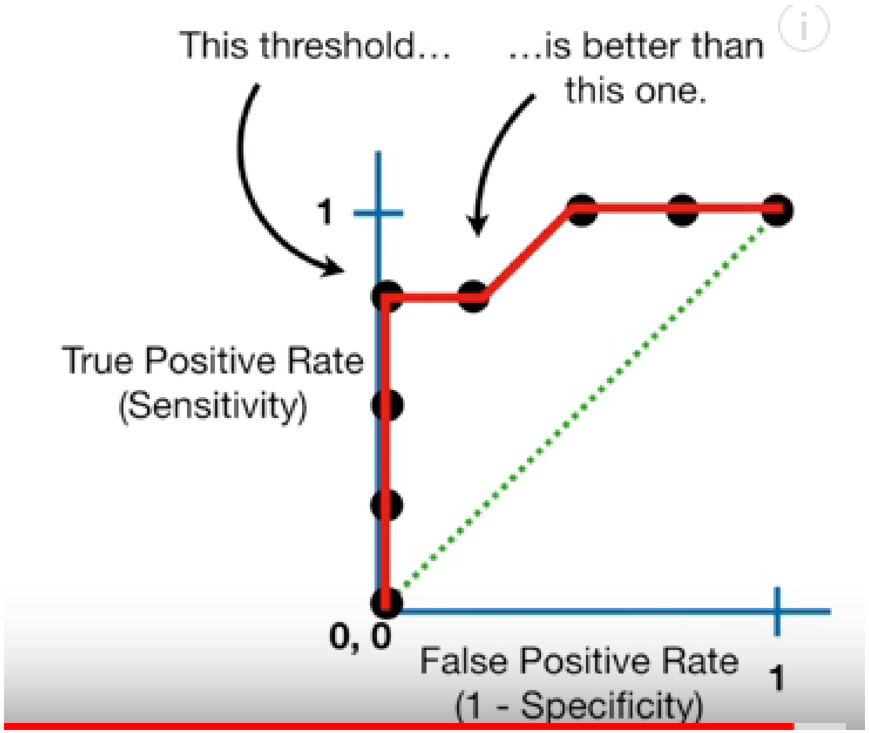
The **AUC** makes it easy to compare one **ROC** curve to another.



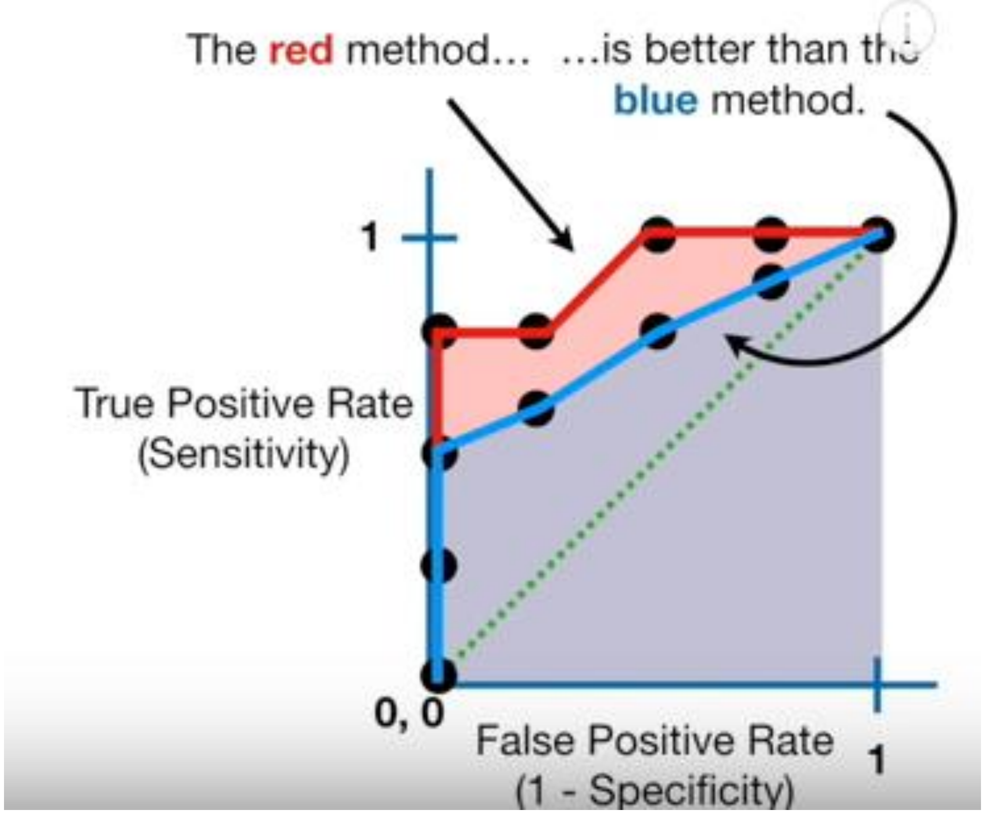


The **AUC** for the **red ROC** curve is greater than the **AUC** for the **blue ROC** curve, suggesting that the **red** curve is better.

- Other matrices can be used to draw ROC
- FPR can be replaced by Precision
- ROC curves make it easy to identify the best threshold for decision making



- And AUC can help you to choose which classification method is better



Multiclass Classification

- Classification task with more than two classifiers.
- OVA
- OVO

Multi-class classification

- One versus All(one-versus-rest)scheme
  - To train k binary classifiers, the first of which separates class C1 from C2, . . . ,Cn, the second of which separates C2 from all other classes, and so on.
  - To train k binary classifiers, the first of which separates class C1 from C2, . . . ,Cn, the second of which separates *C2 from all other classes, and so on.*

Multi-class classification

- one-versus-one scheme
  - we train $k(k-1)/2$ binary classifiers, one for each pair of different classes.
  - If a binary classifier treats the classes asymmetrically, as happens with certain models, it makes more sense to train two classifiers for each pair, leading to a total of $k(k-1)$ classifiers.

Output code matrix

- A convenient way to describe all these schemes to decompose a k-class task into l binary classification tasks is by means of output code matrix.
- This is a k-by-l matrix whose entries are +1, 0 or −1.
- Each column of these matrices describes a binary classification task, using the class corresponding to the row with the +1 entry as positive class and the class with the −1 entry as the negative class.

Output code matrix

- Output code matrix for one versus rest scheme

$$\begin{array}{cc} \begin{matrix} \rightarrow c_1 \\ c_2 \\ c_3 \end{matrix} & \begin{pmatrix} +1 & -1 & -1 \\ -1 & +1 & -1 \\ -1 & -1 & +1 \end{pmatrix} \end{array} \qquad\qquad \begin{pmatrix} +1 & 0 \\ -1 & +1 \\ -1 & -1 \end{pmatrix}$$

- unordered ordered
- Predict $\underline{\underline{-1+1-1}}$ for unordered scheme

Output code matrix

- Output code matrix for one versus one scheme

$$\begin{array}{r} C_1 \rightarrow \\[1.2em] \\[1.2em] \end{array}\left(\begin{array}{ccc} +1 & \underline{+1} & \underline{0} \\ -1 & 0 & +1 \\ 0 & -1 & -1 \end{array}\right)\left.\vphantom{\begin{array}{c}1\\1\\1\end{array}}\right\} \qquad \left(\begin{array}{cccccc} +1 & -1 & +1 & -1 & 0 & 0 \\ -1 & +1 & 0 & 0 & +1 & -1 \\ 0 & 0 & -1 & +1 & -1 & +1 \end{array}\right)$$

- Symmetric asymmetric
- Predict $\underline{\overset{w}{\boxed{0+1\ 0}}}$ for symmetric scheme
- the nearest code word is the first row in the matrix, and so we should predict C_1.

- By using the formula the distance between a word W and a code word c is calculated

$$\checkmark \qquad d(\underline{w}, \underline{c}) = \underline{\sum_i \underline{(1 - w_i c_i)}/2}$$

- Minimum distance is the class we predicted

$$\begin{array}{cccc} & c_1 & c_2 & c_3 \\ \rightarrow & +1 & +1 & 0 \\ & -1 & 0 & +1 \\ & c_1 & c_2 & c_3 \\ & 0 & -1 & -1 \\ & c_1 & c_2 & c_3 \end{array}$$

$$d(w,c) = \sum_{i=1}^{3} (1 - w_i c_i)/2$$ ✓

- Predict $\underline{\overset{w_1}{0}\ \overset{w_2}{+1}\ \overset{w_3}{0}}$

$$\downarrow$$

$$C_1$$

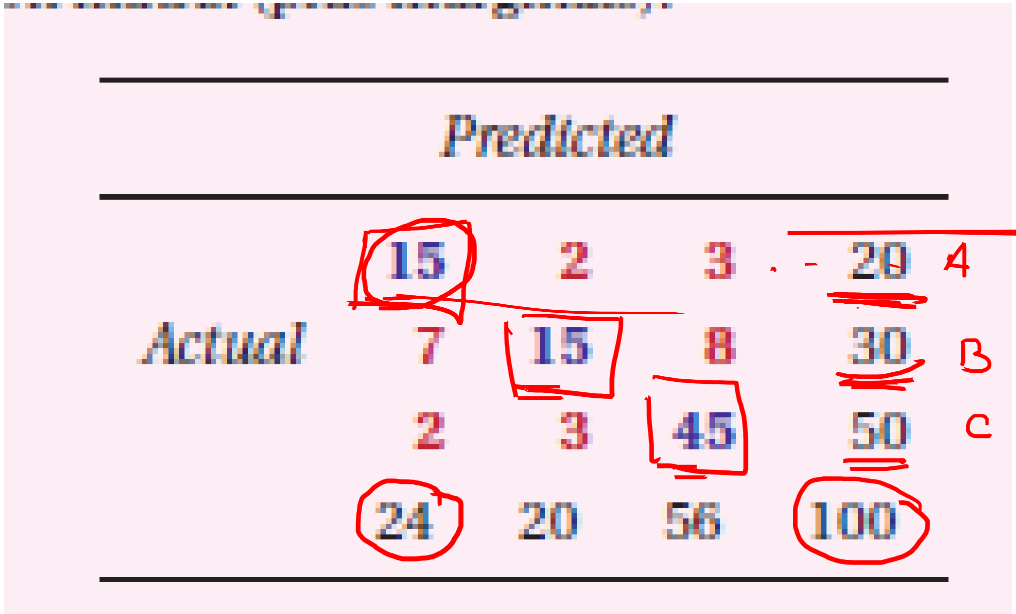
Hamming distance.

$$d(w, \underline{\underline{c_1}}) = \frac{(1-0)+(1-1)+(1-0)}{2} = \boxed{1}$$ ✓

$$d(w\cancel{1}\,\underline{\underline{c_2}}) = 1.5$$

$$d(w, \underline{\underline{c_3}}) = 1.5$$

3 class ·



	Predicted				
	15	2	3	20	A
Actual	7	15	8	30	B
	2	3	45	50	C
	24	20	56	100	

$$\frac{15+15+45}{100} = \frac{75}{100}$$

0.75

per class precision A

$$\frac{15}{24} = 0.63$$

Recall A

$$\frac{15}{20} = 0.75$$

precision B $= \frac{15}{20} = 0.75$

recall B

pre C $\frac{45}{50} = 0.9$

rec C

Weighted Avg. prec. = 0.63 * 0.2 +
0.75 * 0.3 +
0.9 * 0.5
= 0.75

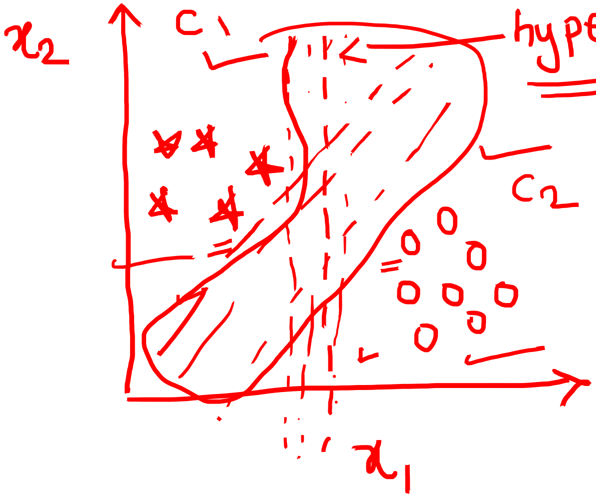
Accuracy as a weighted average

	Predicted ⊕	Predicted ⊖		
Actual ⊕	60	15	75	50
Actual ⊖	10	15	25	50
	70	30	100	

From this table, we see that the true positive rate is $tpr = 60/75 = 0.80$ and the true negative rate is $tnr = 15/25 = 0.60$. The overall accuracy is $acc = (60 + 15)/100 = 0.75$, which is no longer the average of true positive and negative rates. However, taking into account the proportion of positives $pos = 0.75$ and the proportion of negatives $neg = 1 - pos = 0.25$, we see that

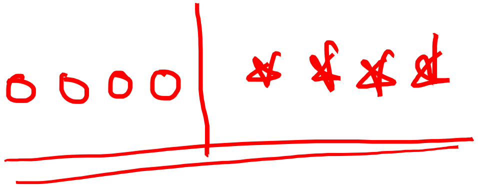
$$acc = pos \cdot tpr + neg \cdot tnr$$

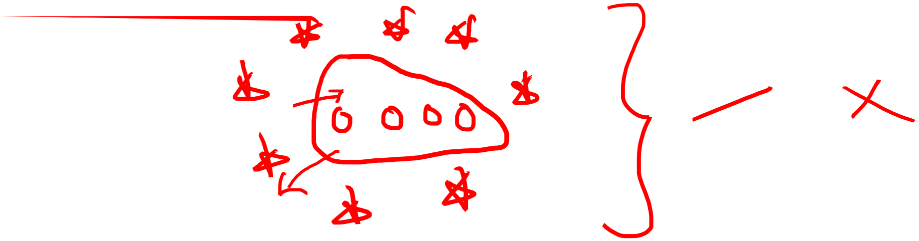
features



hyperplane

1) Linear Seperable





Perceptron

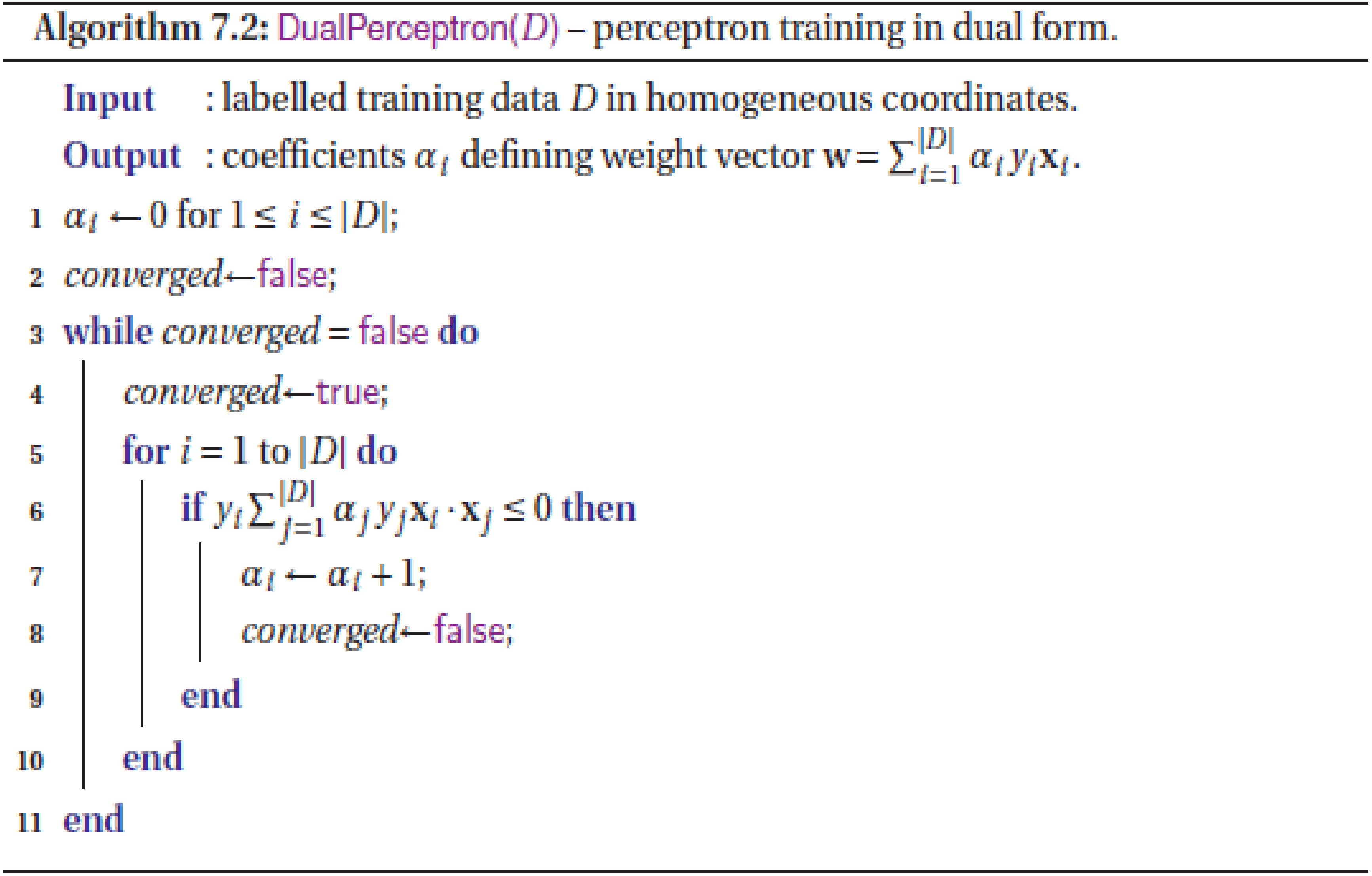
- Linearly Separable
- Perceptron : Linear Classifier that will achieve perfect separation on Linearly Separable data
- Simple NN
- Learning rate(η)

```
Algorithm Perceptron(D, η)
    i/p: Training Data and learning rate
    o/p : weight vector w
 w<- 0
converged<-false
while converged= false do
    Converged <- true
    for(i=1to |D|)
        If($y_i$ w .$x_i \le 0$)
            W<- w+ η$y_i$ $x_i$
            Converged <- false
        end
    end
end
```

- Every time an example is misclassified we add y_i x_i to weight vector.
- Weight vector can be expressed as

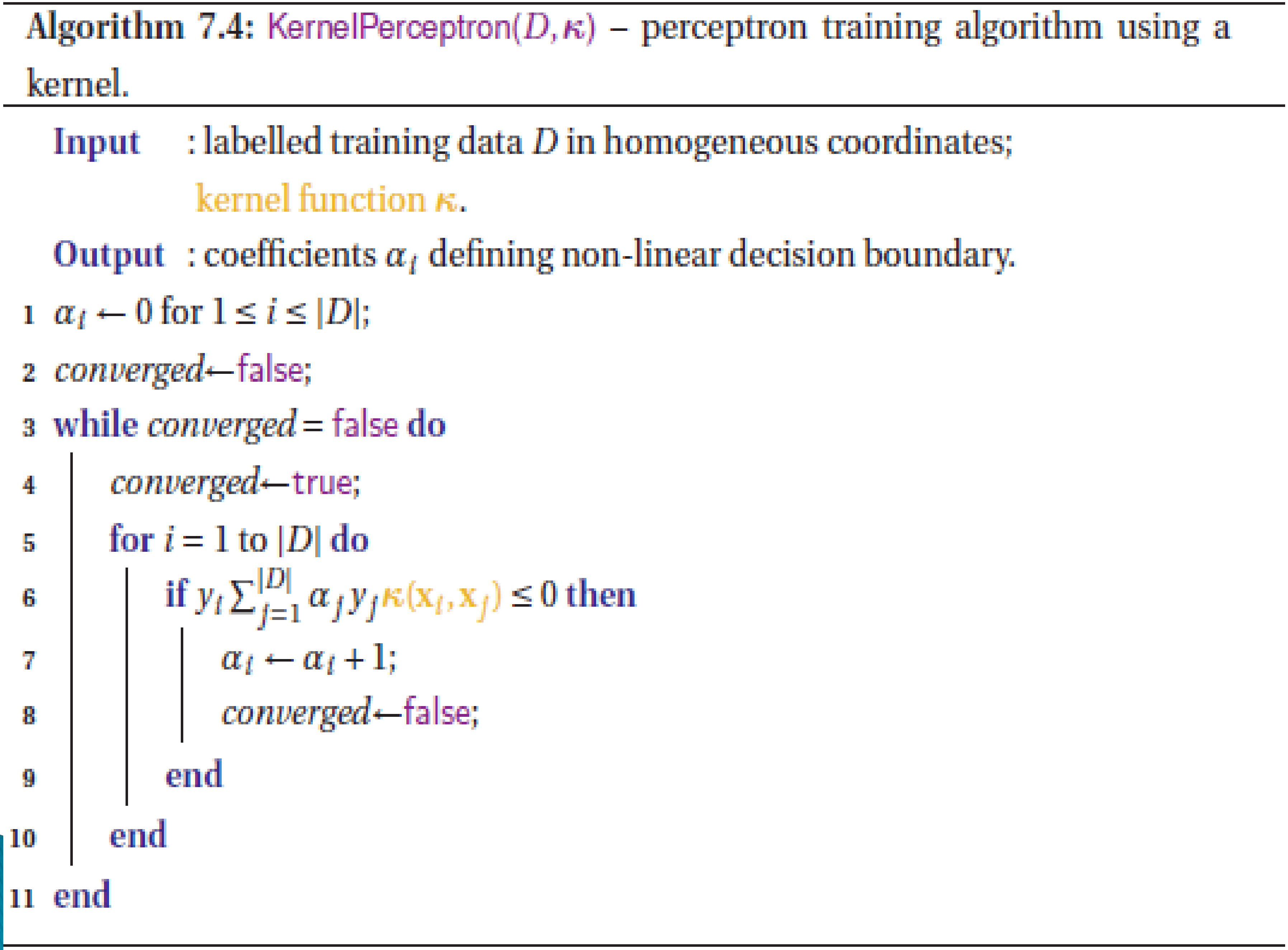
$$w = \sum_{i=1}^{n} \alpha_i y_i x_i$$

Dual Perceptron



Algorithm 7.2: DualPerceptron(D) – perceptron training in dual form.

```
Input  : labelled training data $D$ in homogeneous coordinates.
Output : coefficients $\alpha_i$ defining weight vector $\mathbf{w} = \sum_{i=1}^{|D|} \alpha_i y_i \mathbf{x}_i$.
$\alpha_i \leftarrow 0$ for $1 \leq i \leq |D|$;
$converged \leftarrow$ false;
while $converged$ = false do
    $converged \leftarrow$ true;
    for $i = 1$ to $|D|$ do
        if $y_i \sum_{j=1}^{|D|} \alpha_j y_j \mathbf{x}_i \cdot \mathbf{x}_j \leq 0$ then
            $\alpha_i \leftarrow \alpha_i + 1$;
            $converged \leftarrow$ false;
        end
    end
end
```



Algorithm 7.4: KernelPerceptron(D,κ) – perceptron training algorithm using a kernel.

```
Input  : labelled training data $D$ in homogeneous coordinates;
         kernel function $\kappa$.
Output : coefficients $\alpha_i$ defining non-linear decision boundary.
$\alpha_i \leftarrow 0$ for $1 \le i \le |D|$;
converged←false;
while converged = false do
    converged←true;
    for $i = 1$ to $|D|$ do
        if $y_i \sum_{j=1}^{|D|} \alpha_j y_j \kappa(\mathbf{x}_i, \mathbf{x}_j) \le 0$ then
            $\alpha_i \leftarrow \alpha_i + 1$;
            converged←false;
        end
    end
end
```

Basic operation on vectors in $\mathbb{R}^n$

1. Multiplication by a scalar

Consider a vector $\vec{a} = (a_1, a_2, \ldots, a_n)$ and a scalar c

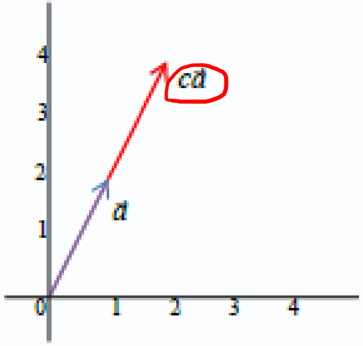
Define: $c\vec{a} = (ca_1, ca_2, \ldots, ca_n)$

When you multiply a vector by a scalar, you "stretch" it in the same or opposite direction depending on whether the scalar is positive or negative.

$\vec{a} = (1,2)$

$c = 2$

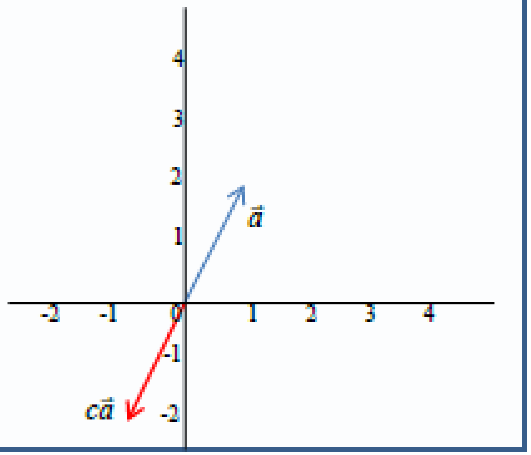
$\vec{c}a = (2,4)$



$\vec{a} = (1,2)$

$c = -1$

$\vec{c}a = (-1,-2)$



Basic operation on vectors in $\mathbb{R}^n$

2. Addition

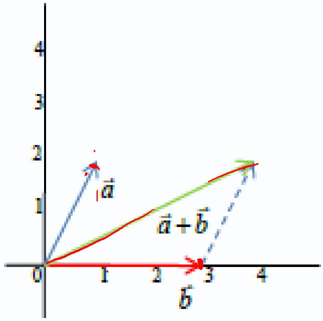
Consider vectors $\vec{a} = (a_1, a_2, \ldots, a_n)$ and $\vec{b} = (b_1, b_2, \ldots, b_n)$

Define: $\vec{a} + \vec{b} = (a_1 + b_1, a_2 + b_2, \ldots, a_n + b_n)$

$\vec{a} = (1,2)$

$\vec{b} = (3,0)$

$\vec{a} + \vec{b} = (4,2)$



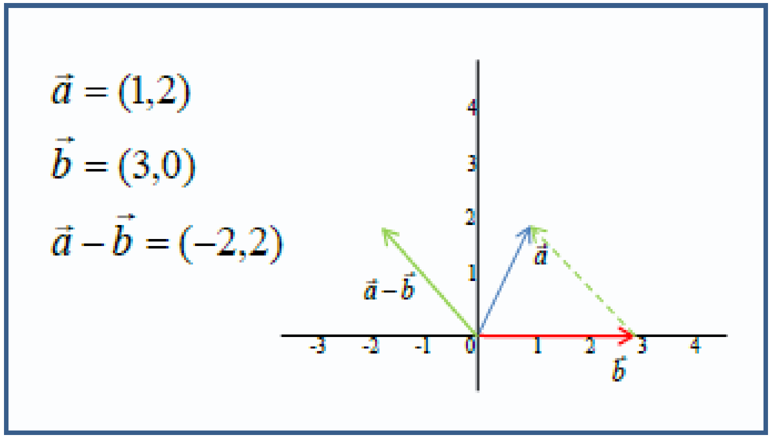
Recall addition of forces in classical mechanics.

Basic operation on vectors in $\mathbb{R}^n$

3. Subtraction

Consider vectors $\vec{a}=(a_1,a_2,\ldots,a_n)$ and $\vec{b}=(b_1,b_2,\ldots,b_n)$

Define: $\vec{a}-\vec{b}=(a_1-b_1,a_2-b_2,\ldots,a_n-b_n)$



$\vec{a}=(1,2)$

$\vec{b}=(3,0)$

$\vec{a}-\vec{b}=(-2,2)$

What vector do we need to add to $\vec{b}$ to get $\vec{a}$? I.e., similar to subtraction of real numbers.

Basic operation on vectors in $\mathbb{R}^n$

4. Euclidian length or L2-norm

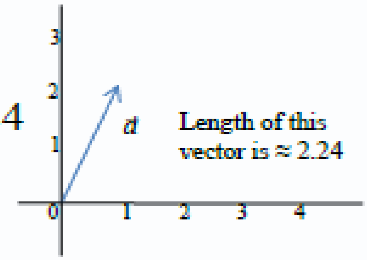
Consider a vector $\vec{a} = (\underline{a_1}, \underline{a_2}, \ldots, \underline{a_n})$

Define the L2-norm: $\underline{\|\vec{a}\|_2} = \sqrt{a_1^2 + a_2^2 + \ldots + a_n^2}$

We often denote the L2-norm without subscript, i.e. $\|\vec{a}\|$

$\vec{a} = (1,2)$

$\|\vec{a}\|_2 = \sqrt{5} \approx 2.24$



Length of this vector is ≈ 2.24

L2-norm is a typical way to measure length of a vector; other methods to measure length also exist.

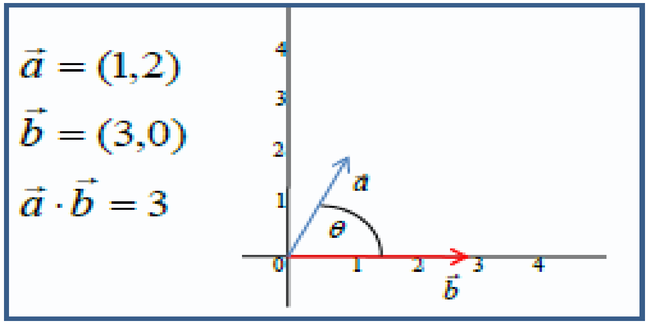
Basic operation on vectors in $\mathbb{R}^n$

5. Dot product

Consider vectors $\vec{a} = (a_1, a_2, \ldots, a_n)$ and $\vec{b} = (b_1, b_2, \ldots, b_n)$

Define dot product: $\underline{\vec{a} \cdot \vec{b}} = a_1 b_1 + a_2 b_2 + \ldots + a_n b_n = \sum_{i=1}^{n} a_i b_i$ ✓

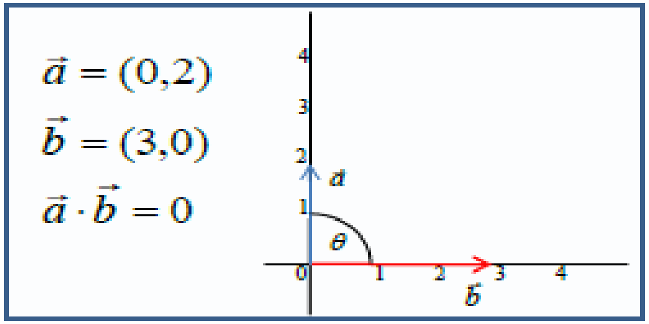
The law of cosines says that $\vec{a} \cdot \vec{b} = \|\vec{a}\|_2 \|\vec{b}\|_2 \underline{\cos\theta}$ where θ is the angle between $\vec{a}$ and $\vec{b}$. Therefore, when the vectors are perpendicular $\underline{\vec{a} \cdot \vec{b} = 0.}$



$\vec{a} = (1,2)$

$\vec{b} = (3,0)$

$\vec{a} \cdot \vec{b} = 3$



$\vec{a} = (0,2)$

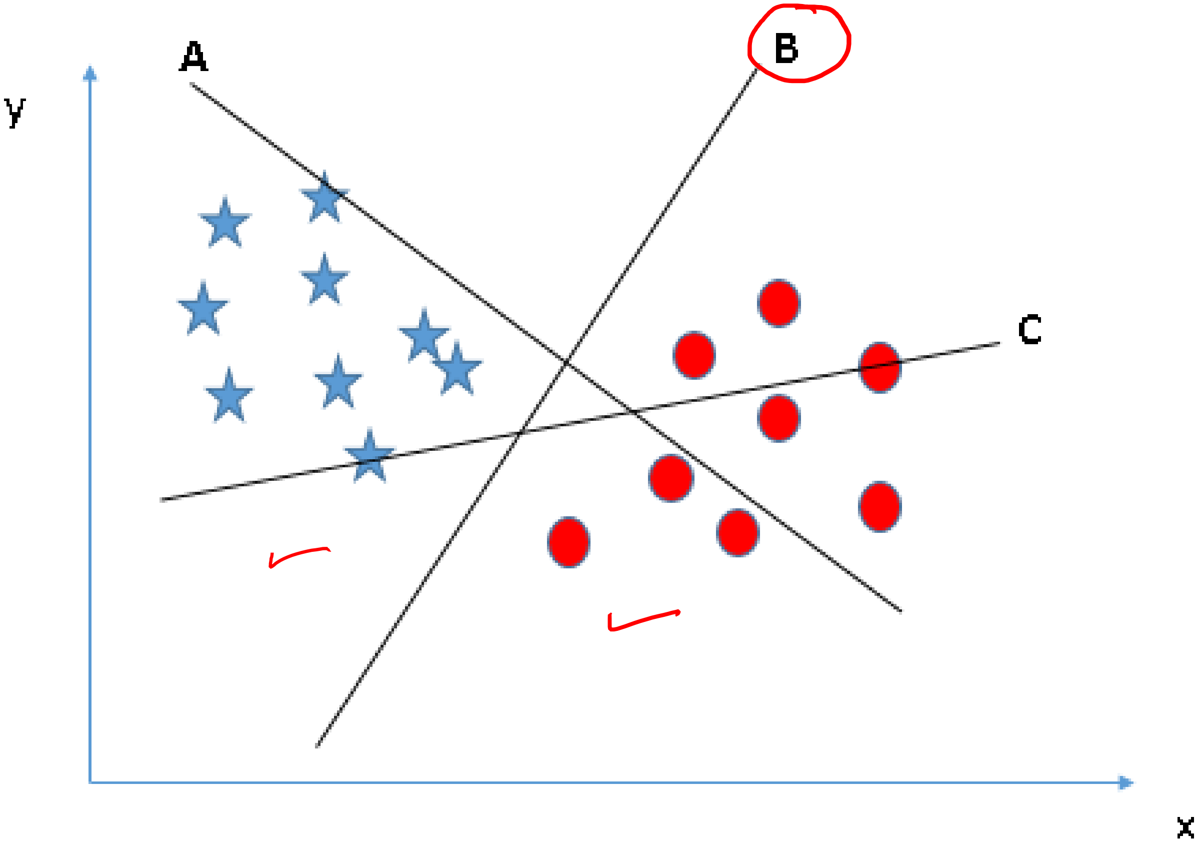
$\vec{b} = (3,0)$

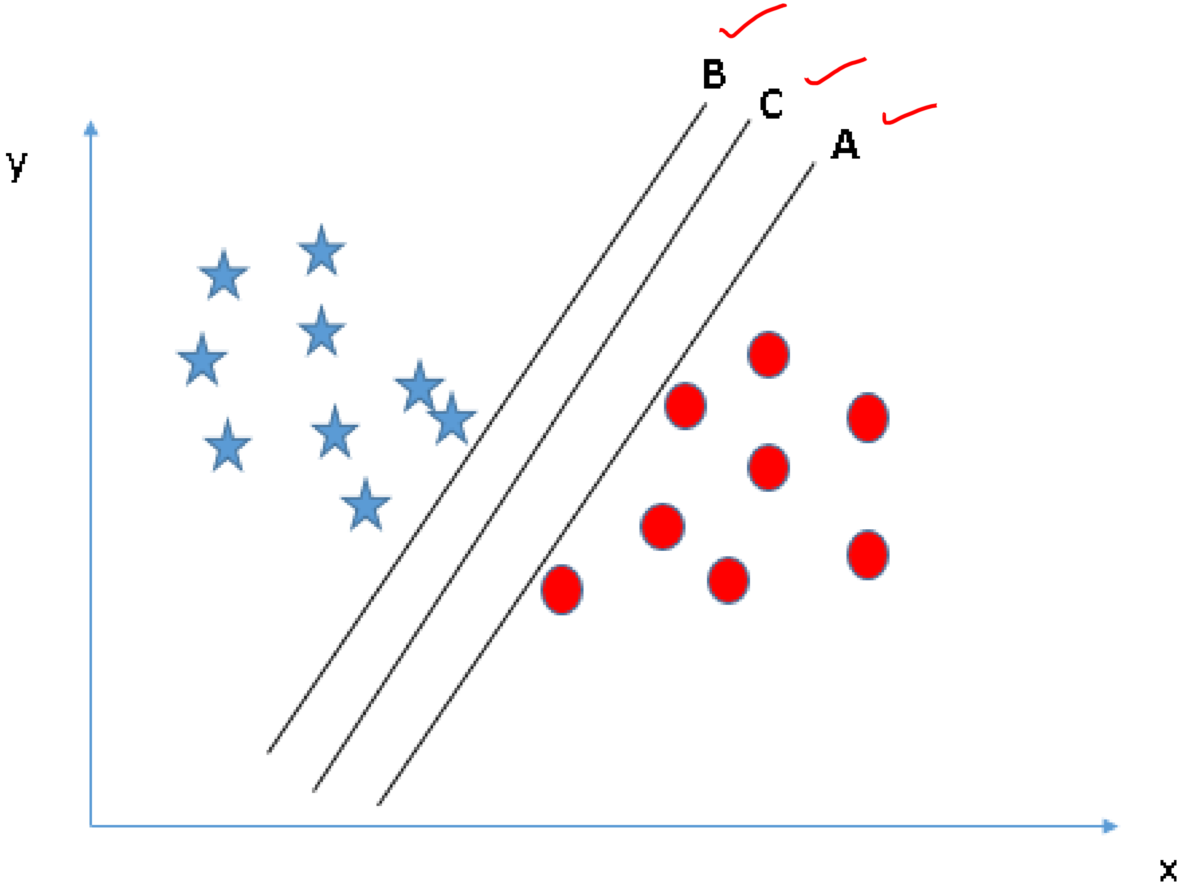
$\vec{a} \cdot \vec{b} = 0$

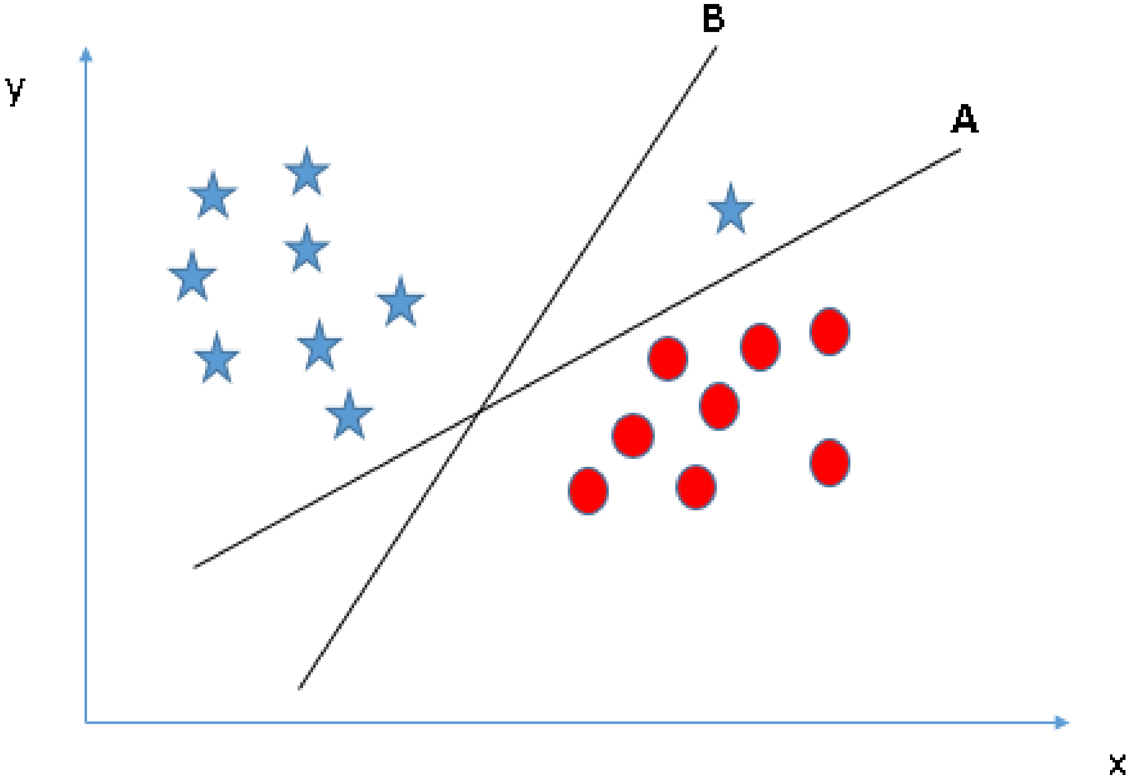
5. Dot product (continued)

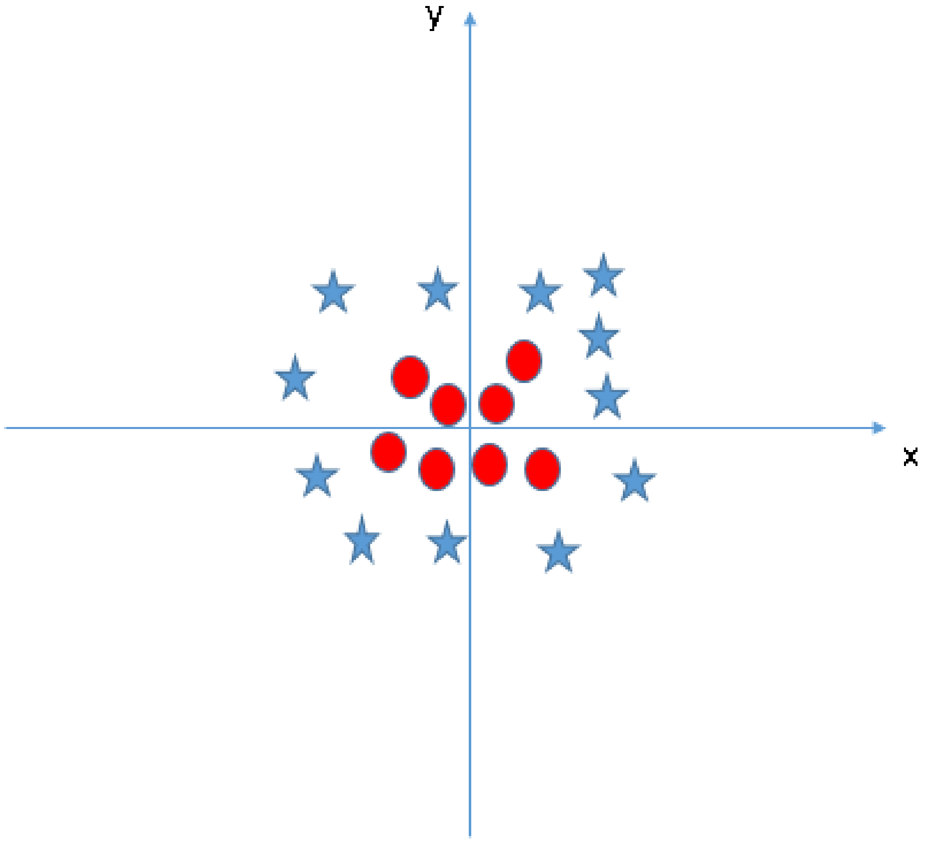
$$\vec{a}\cdot\vec{b} = a_1 b_1 + a_2 b_2 + \ldots + a_n b_n = \sum_{i=1}^{n} a_i b_i$$

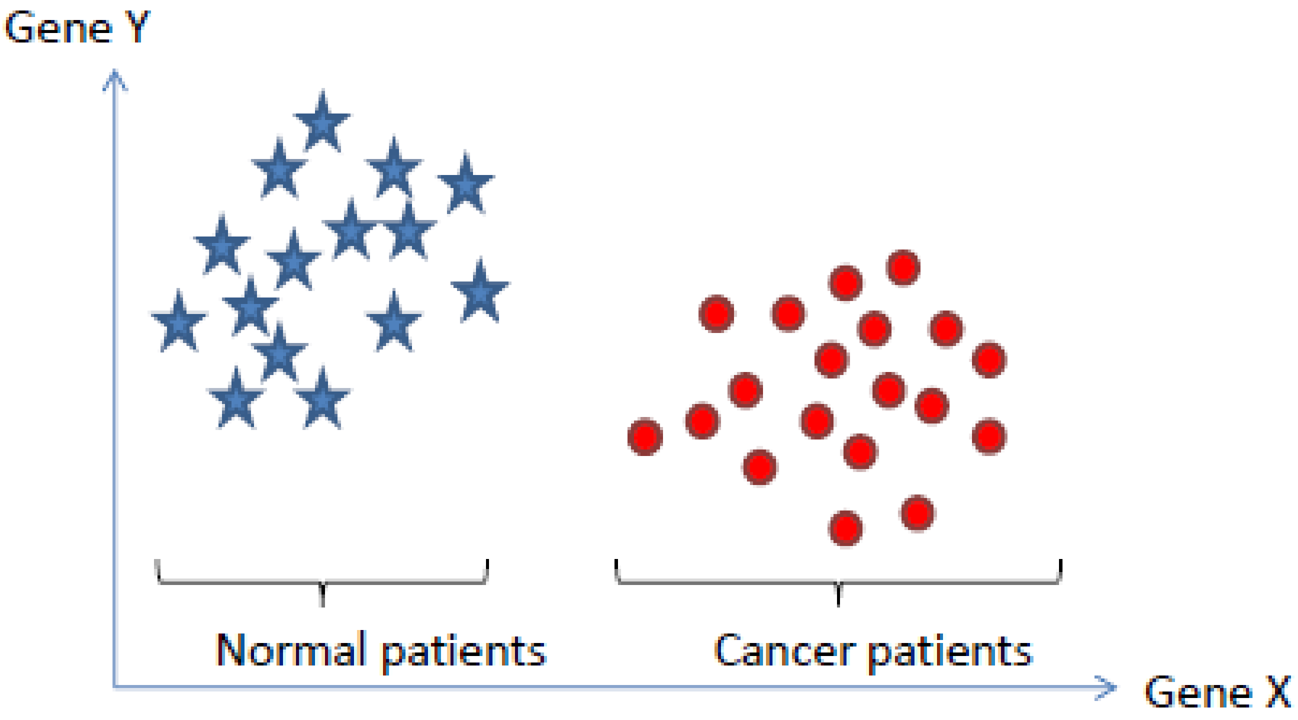
- Property: $\vec{a}\cdot\vec{a} = a_1 a_1 + a_2 a_2 + \ldots + a_n a_n = \|\vec{a}\|_2^2$
- In the classical regression equation $y = \vec{w}\cdot\vec{x} + b$ the response variable *y* is just a dot product of the vector representing patient characteristics ($\vec{x}$) and the regression weights vector ($\vec{w}$) which is common across all patients plus an offset *b*.

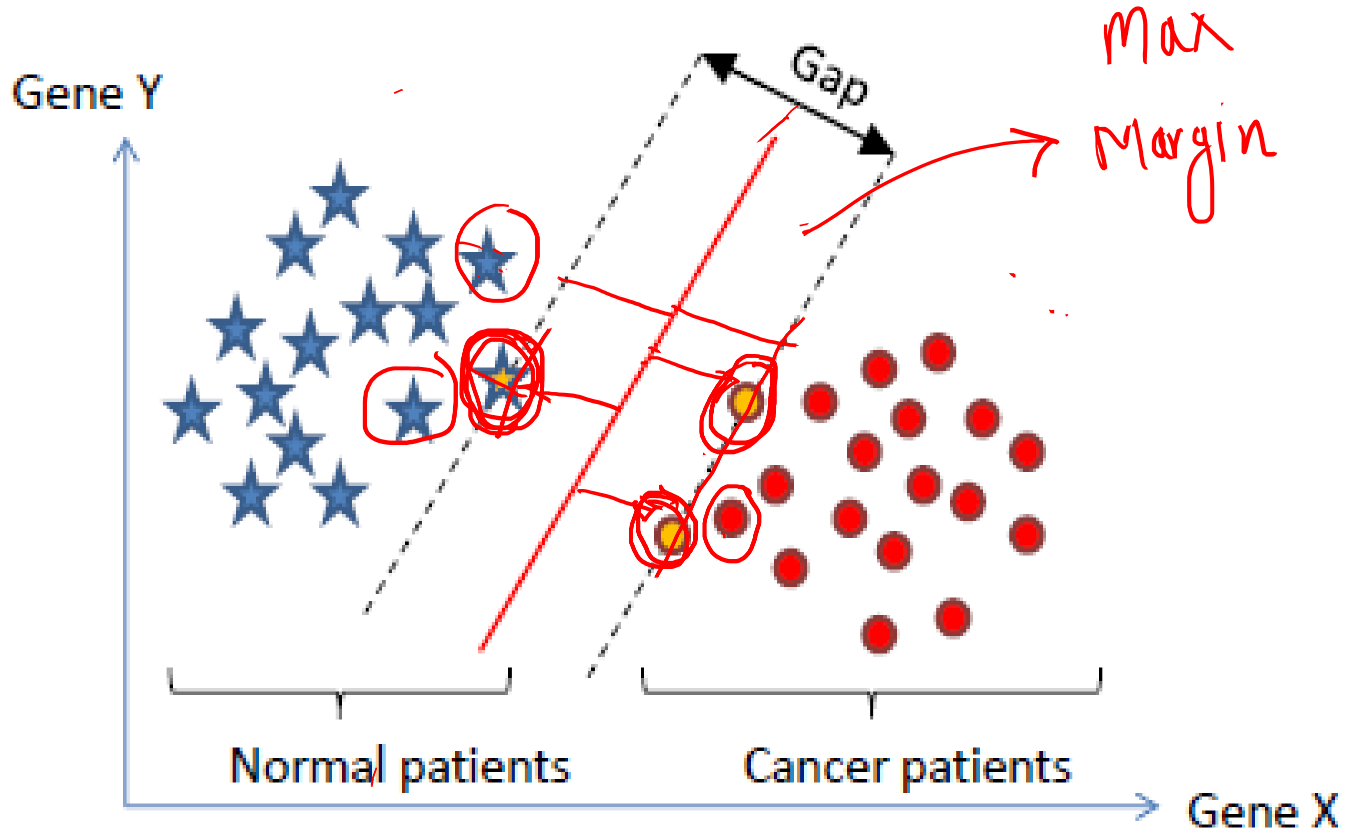

A
B
C
y
x


y
x
B ✓
C ✓
A ✓


B
A
y
x


y
x

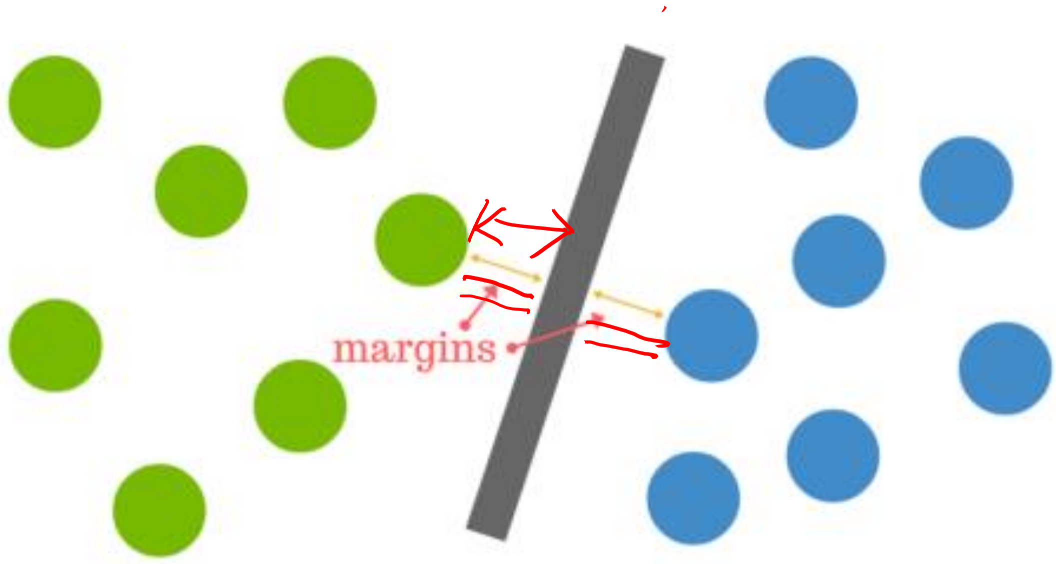

Gene Y
Normal patients
Cancer patients
Gene X


Gene Y
Gap
Max
Margin
Normal patients
Cancer patients
Gene X

Support Vectors

- The data points nearest to the hyperplane
- the points of a data set that
- if removed, would alter the position of the dividing hyperplane.
- They can be considered the critical elements of a data set.

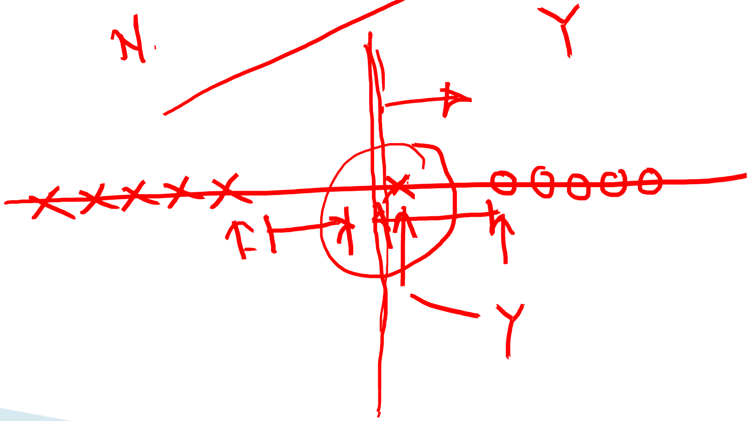
SVM

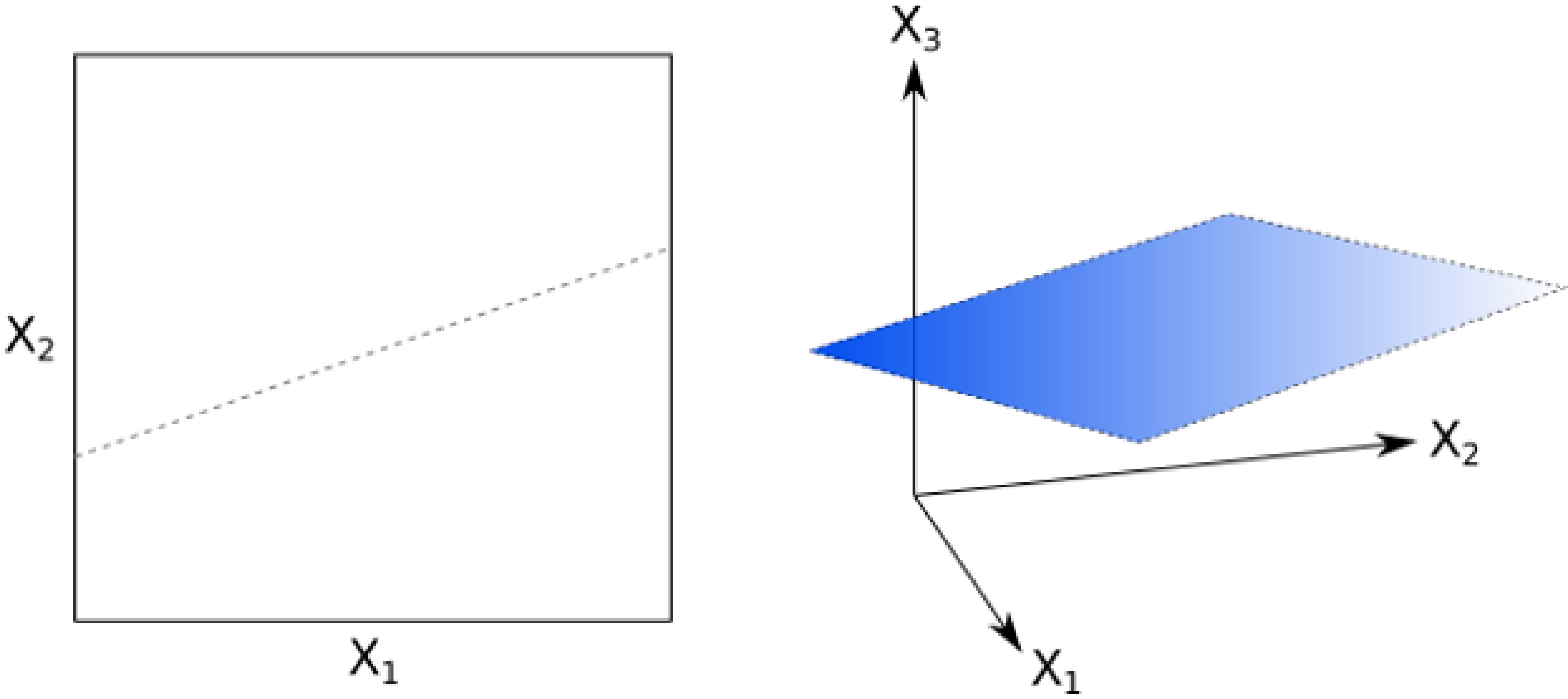


Support Vector Machines

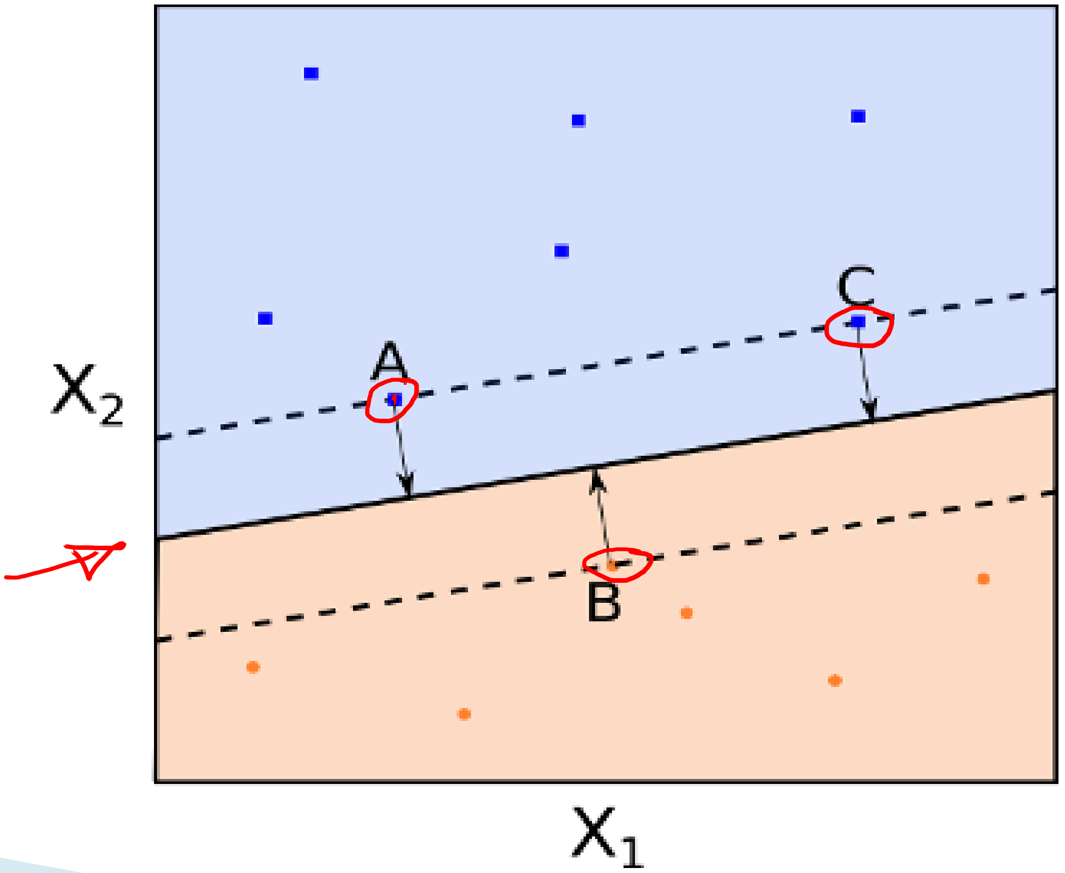
- supervised classification techniques
- **supervised binary classification**
- The goal of the SVM is to train a model that assigns new unseen objects into a particular category.
- The SVM is an effective tool in high-dimensional spaces → sensitive → outliers

  Noise
- **Memory Efficiency**




X_2
X_1
X_3
X_2
X_1

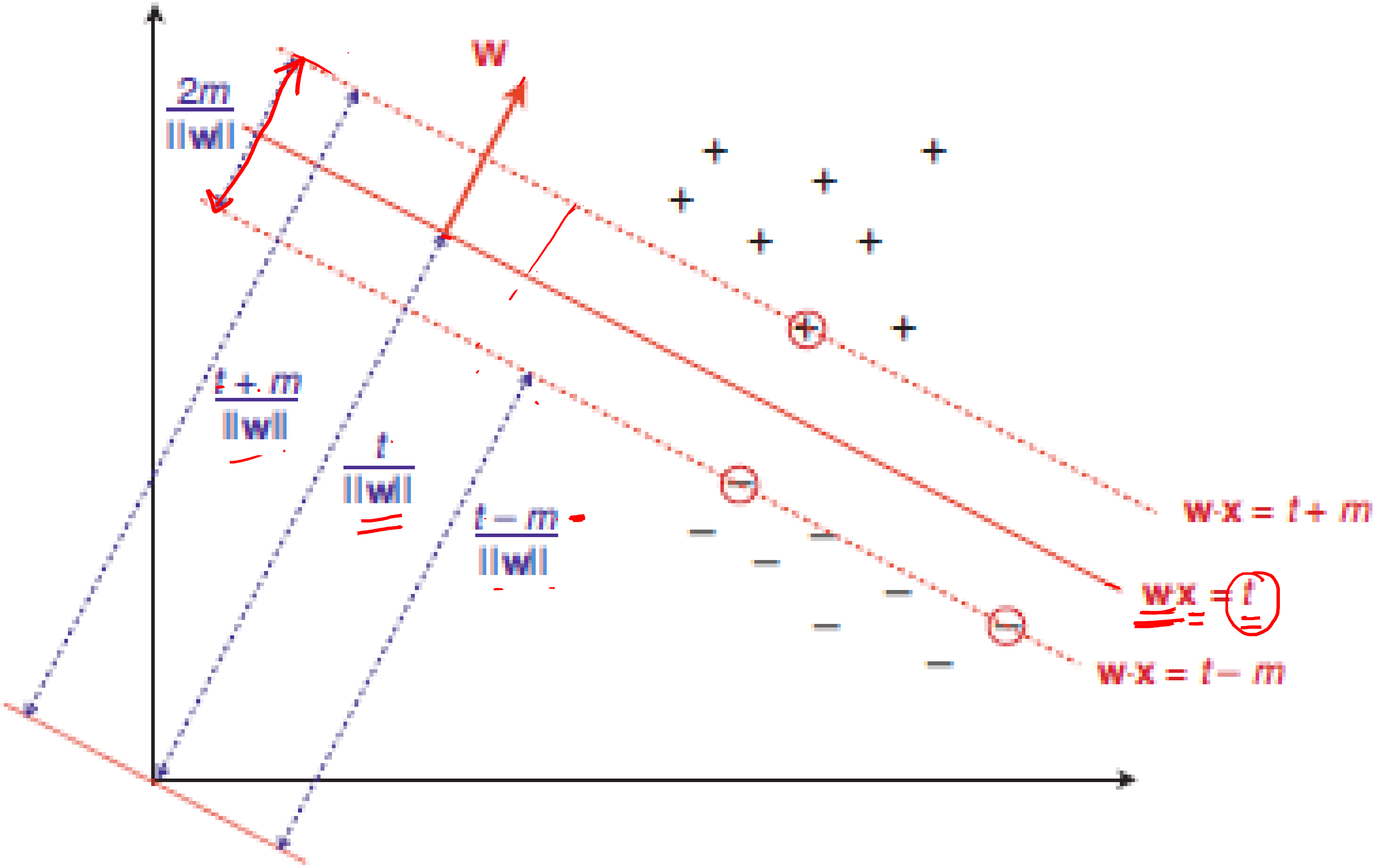
support vectors.



SVM

- The training examples nearest to decision boundary are called support vectors.
- Margin is the distance between the decision boundary and the nearest training instance measured along w

$y = mx + c$



$\frac{2m}{||\mathbf{w}||}$
w
$\frac{t+m}{||\mathbf{w}||}$
$\frac{t}{||\mathbf{w}||}$
$\frac{t-m}{||\mathbf{w}||}$
$\mathbf{w}\cdot\mathbf{x} = t + m$
$\mathbf{wx} = t$
$\mathbf{w}\cdot\mathbf{x} = t - m$

- Margin = 2m/||W||
- Maximizing the margin leads to minimizing ||W|| i.e. None of the points fall inside the margin

$$\mathbf{w}^*, t^* = \arg\min_{\mathbf{w},t} \frac{1}{2}||\mathbf{w}||^2 \qquad \text{subject to } y_i(\mathbf{w}\cdot\mathbf{x}_i - t) \geq 1, 1 \leq i \leq n$$

$$\begin{aligned}\Lambda(\mathbf{w},t,\alpha_1,\ldots,\alpha_n) &= \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n}\alpha_i(y_i(\mathbf{w}\cdot\mathbf{x}_i - t) - 1)\\ &= \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n}\alpha_i y_i(\mathbf{w}\cdot\mathbf{x}_i) + \sum_{i=1}^{n}\alpha_i y_i t + \sum_{i=1}^{n}\alpha_i\\ &= \frac{1}{2}\mathbf{w}\cdot\mathbf{w} - \mathbf{w}\cdot\left(\sum_{i=1}^{n}\alpha_i y_i \mathbf{x}_i\right) + t\left(\sum_{i=1}^{n}\alpha_i y_i\right) + \sum_{i=1}^{n}\alpha_i\end{aligned}$$

By taking the partial derivative of the Lagrange function with respect to *t and setting* it to 0
for the optimal threshold *t we have*

$$\sum_{i=1}^{n}\alpha_i y_i = 0$$

Similarly taking the partial derivative of the Lagrange function with respect to **w** Lagrange multipliers define the weight vector as a linear combination of the training examples:

$$\frac{\partial}{\partial\mathbf{w}}\Lambda(\mathbf{w},t,\alpha_1,\ldots,\alpha_n) = \frac{\partial}{\partial\mathbf{w}}\frac{1}{2}\mathbf{w}\cdot\mathbf{w} - \frac{\partial}{\partial\mathbf{w}}\mathbf{w}\cdot\left(\sum_{i=1}^{n}\alpha_i y_i\mathbf{x}_i\right) = \mathbf{w} - \sum_{i=1}^{n}\alpha_i y_i\mathbf{x}_i$$

- Putting the values in Lagrangian we get

$$\mathbf{w} = \sum_{i=1}^{l} a_i y_i \mathbf{x}_i \qquad \sum_{i=1}^{n} \alpha_i y_i = 0$$

$$\begin{aligned}
\Lambda(\alpha_1, \ldots, \alpha_n) &= -\frac{1}{2}\left(\sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i\right) \cdot \left(\sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i\right) + \sum_{i=1}^{n} \alpha_i \\
&= -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j + \sum_{i=1}^{n} \alpha_i
\end{aligned}$$

- Maximize

$$\alpha_1^*, \ldots, \alpha_n^* = \underset{\alpha_1, \ldots, \alpha_n}{\arg\max} -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j + \sum_{i=1}^{n} \alpha_i$$

$$\text{subject to } \alpha_i \geq 0, 1 \leq i \leq n \text{ and } \sum_{i=1}^{n} \alpha_i y_i = 0$$

- Dual form of the optimization problem for SVM illustrates
  - Searching for the maximum margin decision boundary is equivalent to searching for support vectors
  - Optimization problem is defined by pair wise dot products between training examples

Soft Margin SVM

- If data is not linearly separable , slack variable is introduced (one for each example)
- Which allows some of the instances to be inside margin or even at wrong side.

$$\mathbf{w}^*, t^*, \xi_i^* = \mathop{\arg\min}_{\mathbf{w}, t, \xi_i} \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n} \xi_i$$

$$\text{subject to } y_i(\mathbf{w}\cdot\mathbf{x}_i - t) \geq 1 - \xi_i \text{ and } \xi_i \geq 0, 1 \leq i \leq n$$

- *C is a user-defined parameter trading off margin maximization against slack variable*
- minimization: a high value of *C means that margin errors incur a high penalty,*
- a low value permits more margin errors in order

  to achieve a large margin.

  If we allow more margin errors we need fewer support
- vectors, hence *C controls to some extent the ‘complexity’ of the SVM and hence is often*
- referred to as the ***complexity parameter.***

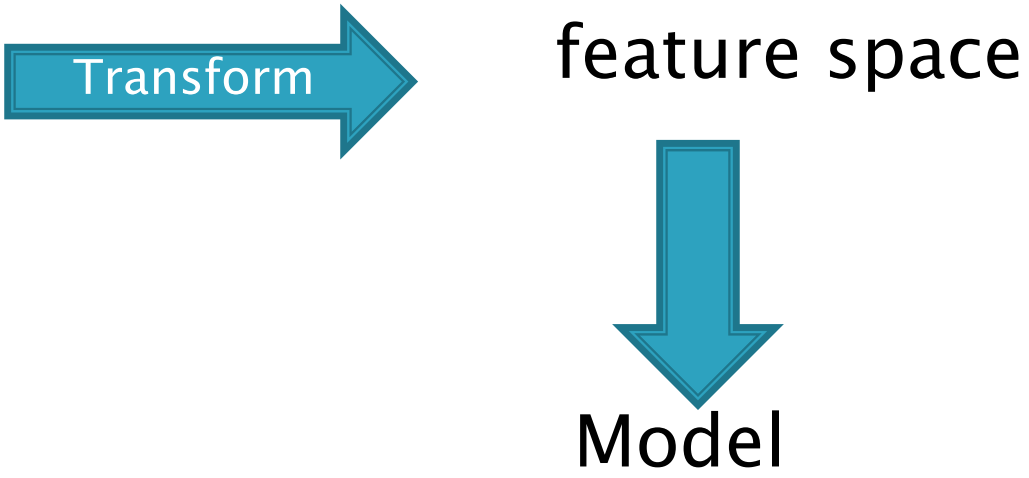
$$\begin{aligned}
\Lambda(\mathbf{w},t,\xi_i,\alpha_i,\beta_i) &= \frac{1}{2}||\mathbf{w}||^2+C\sum_{i=1}^{n}\xi_i-\sum_{i=1}^{n}\alpha_i(y_i(\mathbf{w}\cdot\mathbf{x}_i-t)-(1-\xi_i))-\sum_{i=1}^{n}\beta_i\xi_i \\
&= \frac{1}{2}\mathbf{w}\cdot\mathbf{w}-\mathbf{w}\cdot\left(\sum_{i=1}^{n}\alpha_i y_i\mathbf{x}_i\right)+t\left(\sum_{i=1}^{n}\alpha_i y_i\right)+\sum_{i=1}^{n}\alpha_i+\sum_{i=1}^{n}(C-\alpha_i-\beta_i)\xi_i \\
&= \Lambda(\mathbf{w},t,\alpha_i)+\sum_{i=1}^{n}(C-\alpha_i-\beta_i)\xi_i
\end{aligned}$$

$$\alpha_1^*,\ldots,\alpha_n^*=\underset{\alpha_1,\ldots,\alpha_n}{\operatorname{argmax}}-\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j\mathbf{x}_i\cdot\mathbf{x}_j+\sum_{i=1}^{n}\alpha_i$$

$$\text{subject to } 0\leq\alpha_i\leq C \text{ and } \sum_{i=1}^{n}\alpha_i y_i=0$$

Kernel methods for non-Linearity

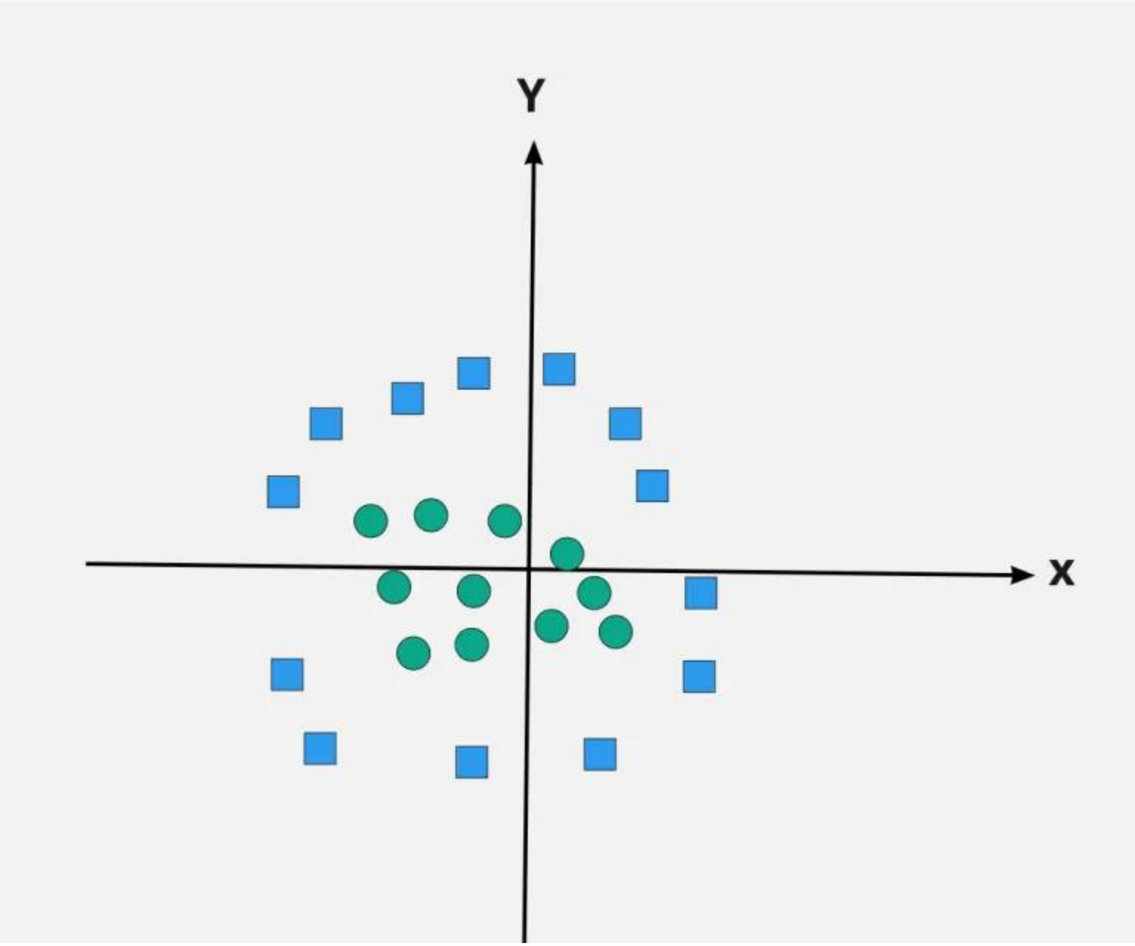
- Transform the data non- linearly to feature space in which linear classification can be applied.
- Training data



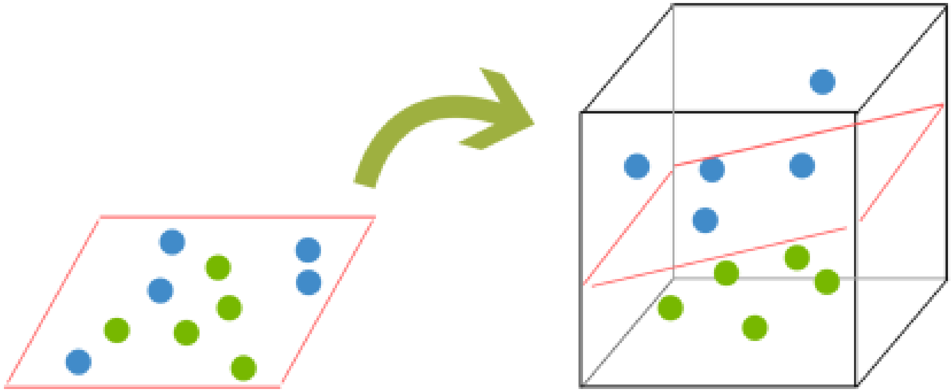
Transform

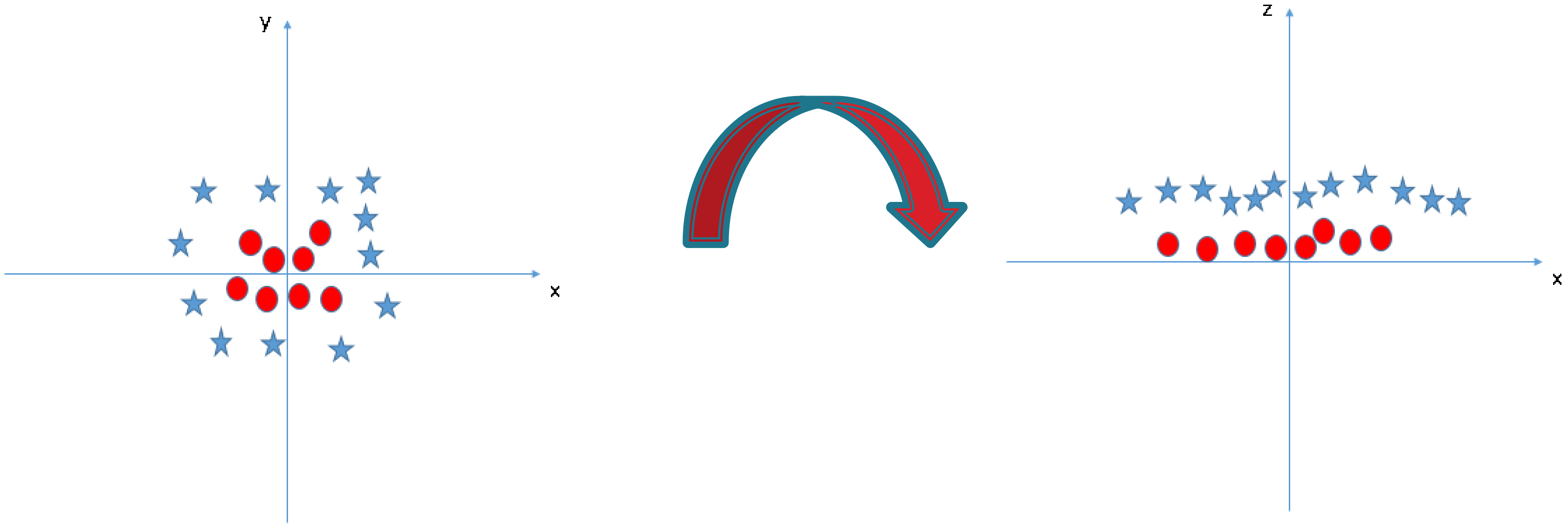
feature space

Model


Y
X

- move away from a 2d view of the data to a 3d view
- i.e.the mapping of data into a higher dimension. This is known as kernelling.
- These are functions which takes low dimensional input space and transform it to a higher dimensional space i.e. it converts not separable problem to separable problem, these functions are called kernels.
- It is mostly useful in non-linear separation problem.




y
x
z
x

- A function that calculates the dot product in feature space directly from the vectors in the original space is called a *kernel*

κ(x1,x2) = (x1 · x2)²

Dot product :

$$x_i \bullet x_j = x_i x_j + y_i y_j$$

Corresponding features in quadratic space

$$(x_i^2, y_i^2) \bullet (x_j^2, y_j^2) = x_i^2 x_j^2 + y_i^2 y_j^2$$

- The instance (x,y) can be transformed into 3D feature space as (x , y, √2xy)

$$\phi(\mathbf{x}_i) = \left(x_i^2, y_i^2, \sqrt{2}x_i y_i\right) \qquad \phi(\mathbf{x}_j) = \left(x_j^2, y_j^2, \sqrt{2}x_j y_j\right)$$

$$\phi(\mathbf{x}_i)\cdot\phi(\mathbf{x}_j) = x_i^2 x_j^2 + y_i^2 y_j^2 + 2x_i x_j y_i y_j = (\mathbf{x}_i \cdot \mathbf{x}_j)^2$$

Kernel Method – Polynomial

- a polynomial kernel of any degree *p as*

  $\kappa(xi, xj) = (xi \cdot xj)^p$.

- *This transforms* a *d-dimensional input space into a high-dimensional feature space, such that each* new feature is a product of *p terms*

Kernel Method–Polynomial

- Polynomials of degree d

- Polynomials of degree up to d

$$\boxed{K(\mathbf{u}, \mathbf{v}) = (\mathbf{u} \cdot \mathbf{v})^d}$$

all lower-order terms are also included

$$\boxed{K(\mathbf{u}, \mathbf{v}) = (\mathbf{u} \cdot \mathbf{v} + 1)^d}$$

$$\phi(\mathbf{x}) = \left(x^2, y^2, \sqrt{2}xy, \sqrt{2}x, \sqrt{2}y, 1\right)$$

Kernel Method– Gaussian

- Gaussian Method

$$\kappa(\mathbf{x}_\ell, \mathbf{x}_j) = \exp\left(\frac{-||\mathbf{x}_\ell - \mathbf{x}_j||^2}{2\sigma^2}\right)$$

Where σ is known as bandwidth

- Kernel methods with support vector machines

$$\alpha_1^*,\ldots,\alpha_n^* = \underset{\alpha_1,\ldots,\alpha_n}{\operatorname{argmax}} -\frac{1}{2}\sum_{\ell=1}^{n}\sum_{j=1}^{n}\alpha_\ell\alpha_j y_\ell y_j \kappa(\mathbf{x}_\ell,\mathbf{x}_j) + \sum_{\ell=1}^{n}\alpha_\ell$$

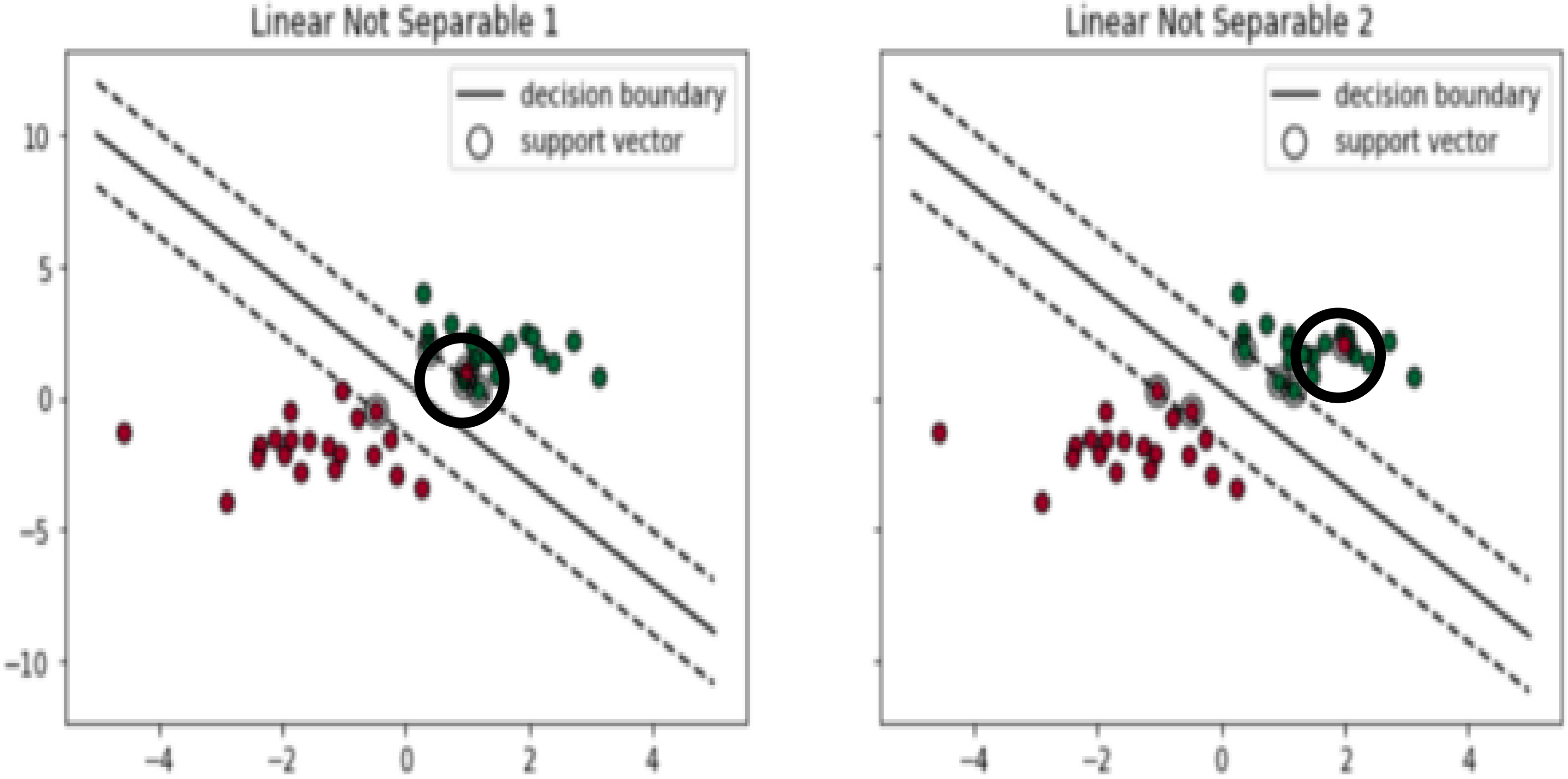
$$\text{subject to } 0 \leq \alpha_\ell \leq C \text{ and } \sum_{\ell=1}^{n}\alpha_\ell y_\ell = 0$$

Hard Margin SVM vs. Soft margin SVM

1. Hard margin given by Boser et al. 1992 in COLT and soft margin given by Vapnik et al. 1995.
2. Soft margin is extended version of hard margin SVM.
3. Hard margin SVM can work only when data is completely linearly separable without any errors (noise or outliers). In case of errors either the margin is smaller or hard margin SVM fails.
4. On the other hand soft margin SVM was proposed by Vapnik to solve this problem by introducing slack variables.
5. As for as their usage is concerned since Soft margin is extended version of hard margin SVM so we use Soft margin SVM.

Soft Margin

- try to find a line to separate, but tolerate one or few misclassified instances
- Two types of misclassifications are tolerated by SVM under soft margin:
- The instance is on the wrong side of the decision boundary but on the correct side/ on the margin
- The instance is on the wrong side of the decision boundary and on the wrong side of the margin


Linear Not Separable 1
decision boundary
support vector
10
5
0
−5
−10
−4
−2
0
2
4
Linear Not Separable 2
decision boundary
support vector
−4
−2
0
2
4

- **Pros**
  - Accuracy
  - Works well on smaller cleaner datasets
  - It can be more efficient because it uses a subset of training points
- **Cons**
  - Isn’t suited to larger datasets as the training time with SVMs can be high
  - Less effective on noisier datasets with overlapping classes