

MTH 221

Fundamentals of Machine Learning

Batuhan Bardak

Lecture 3: Perceptron & Evaluation Metrics

Date: 17.10.2023

Supervised
binary classification

Announcements:

- Group Formation for the Project: **13/10/2023**
 - New Deadline: 20/10/2023 ↴
 - For those who do not decide groups by this deadline, we will automatically create the groups.
- Project Proposal Submission: **20/10/2023**
 - New Deadline: 27/10/2023 ↴
 - As for groups that did not submit their project proposals on time, their points will decrease daily by -10%.

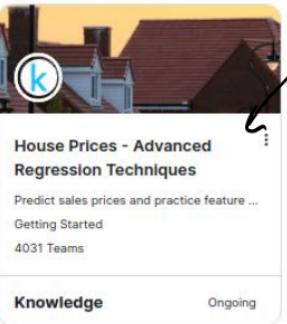
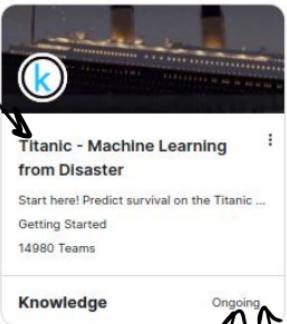
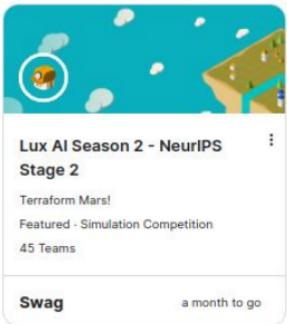
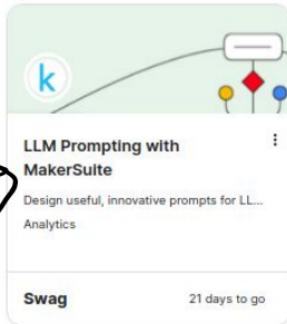
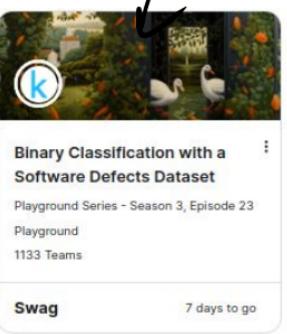
Announcements:

- Group Formation for the Project: **13/10/2023**

- New De...
○ For those
the group

- Project Propo

- New De...
○ As for gr
decrease



Recap

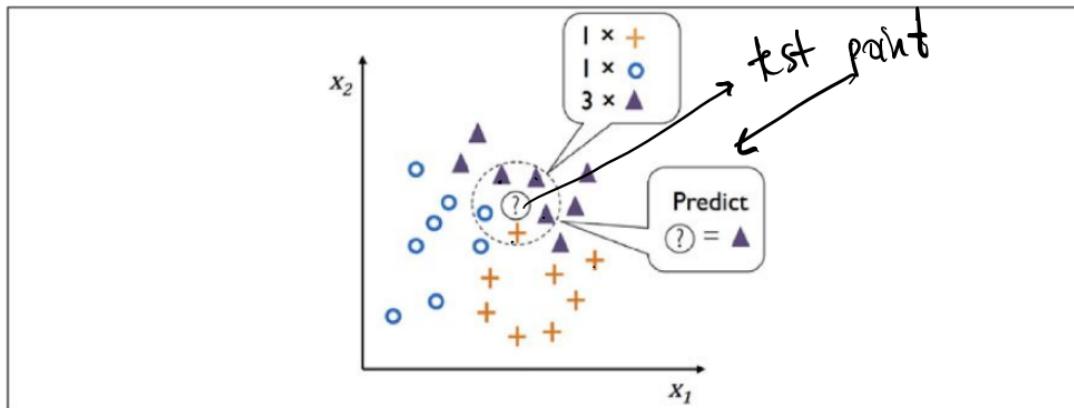
K-NN
=

- Choose the number of k and a distance metric.
- Find the k -nearest neighbors of the data record that want to classify.
- Assign the class label by majority vote.
- Check this [link](#) for other hyperparameters of kNN.
- Demo: <http://vision.stanford.edu/teaching/cs231n-demos/knn/>

=

3 Δ
1 o
1 +

} → Δ



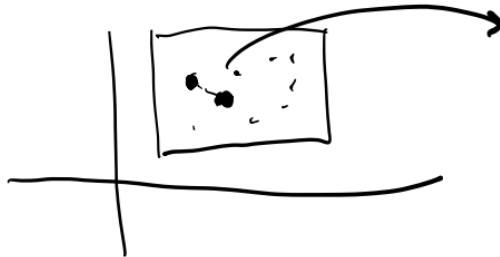
Recap

- T/F: We can use train-validation trick to determine the parameter K
- T/F: K-NN will fail when feature dimension is high
- T F: In K-NN, as K increases, the decision boundary becomes more complex.
- T/F: In high-dimensional spaces, most data points are far away from each other, making distance-based methods like K-NN less effective.

$K=1$



$K=10$



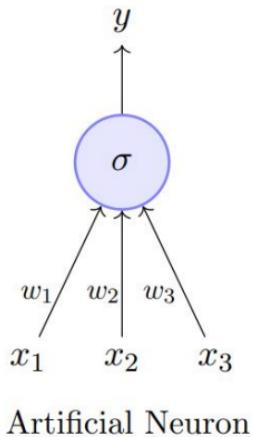
Recap

- T/F: We can use train-validation trick to determine the parameter K.
 - True //
- T/F: K-NN will fail when feature dimension is high.
 - True //
- T/F: In K-NN, as K increases, the decision boundary becomes more complex.
 - False, As K increases, the decision boundary generally becomes smoother (less complex) //
- T/F: In high-dimensional spaces, most data points are far away from each other, making distance-based methods like K-NN less effective.
 - True //

Outline for Today

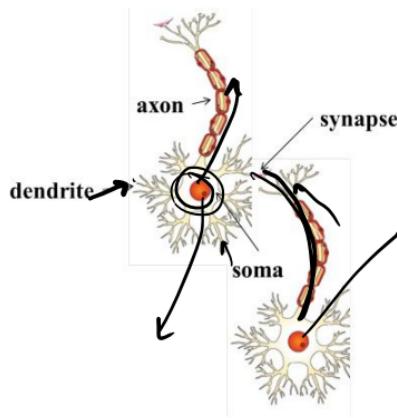
1. Perceptron Algorithm
2. Why it Works
3. Evaluation Metrics

Artificial Neuron



- The most fundamental unit of a deep neural network is called an artificial neuron
- Why is it called a neuron ? Where does the inspiration come from ?
- The inspiration comes from biology (more specifically, from the brain)
- *biological neurons = neural cells = neural processing units*
- We will first see what a biological neuron looks like ...

Artificial Neuron



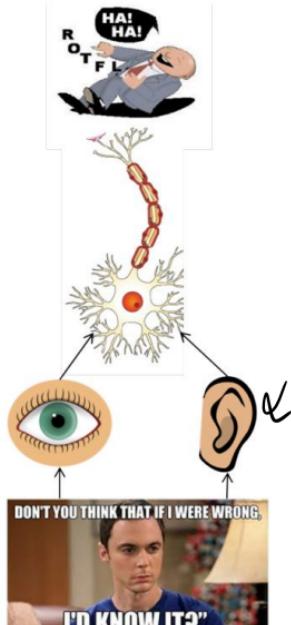
Biological Neurons*

- **dendrite:** receives signals from other neurons
- **synapse:** point of connection to other neurons
- **soma:** processes the information
- **axon:** transmits the output of this neuron

*Image adapted from

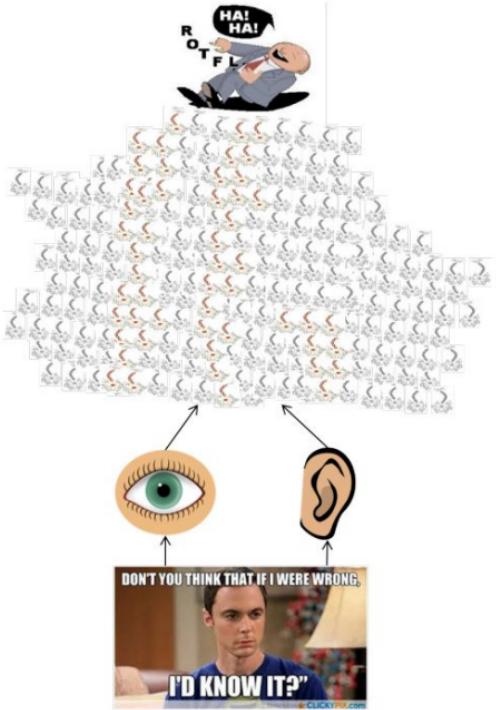
<https://cdn.vectorstock.com/i/composite/12,25/neuron-cell-vector-81225.jpg>

Artificial Neuron



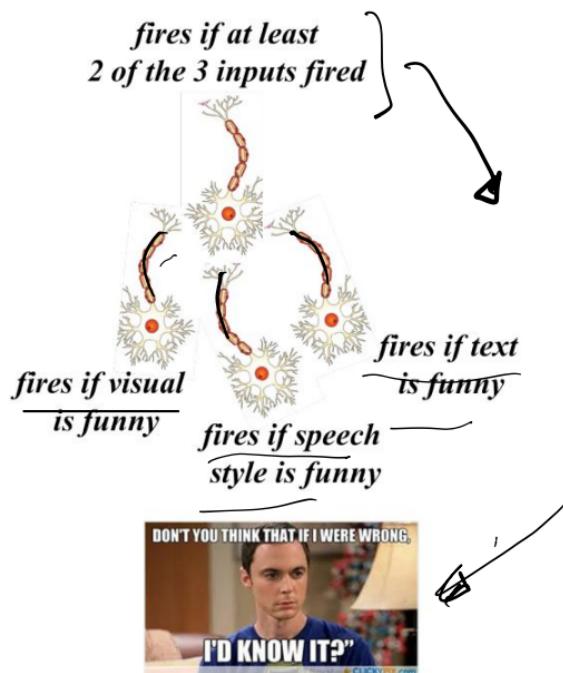
- Let us see a very cartoonish illustration of how a neuron works
- Our sense organs interact with the outside world
- They relay information to the neurons
- The neurons (may) get activated and produces a response (laughter in this case)

Artificial Neuron



- Of course, in reality, it is not just a single neuron which does all this
- There is a massively parallel interconnected network of neurons
- The sense organs relay information to the lowest layer of neurons
- Some of these neurons may fire (in red) in this information and in turn relay information to other neurons they are connected to
- These neurons may also fire (again, in red) and the process continues eventually resulting in a response (laughter in this case)
- An average human brain has around 10^{11} (100 billion) neurons!

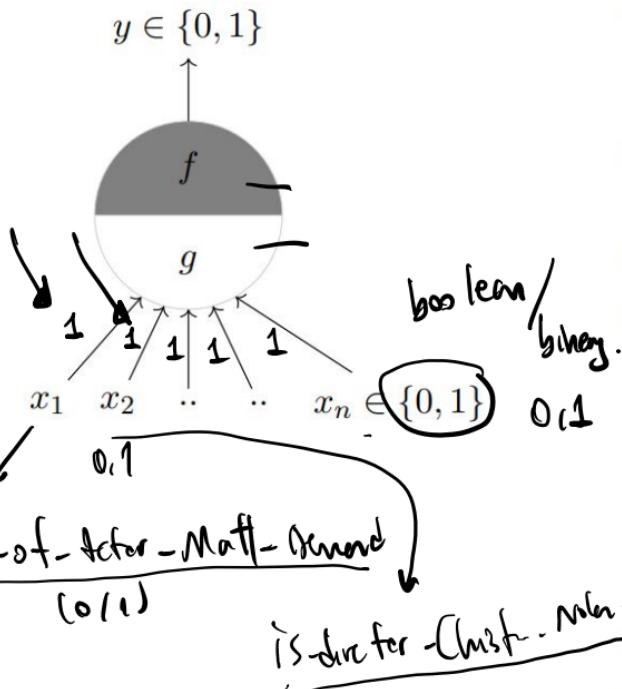
Artificial Neuron



- This massively parallel network also ensures that there is division of work
- Each neuron may perform a certain role or respond to a certain stimulus

A simplified illustration

McCulloch Pitts Neuron



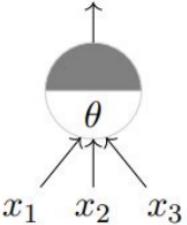
- McCulloch (neuroscientist) and Pitts (logician) proposed a highly simplified computational model of the neuron (1943)
- g aggregates the inputs and the function f takes a decision based on this aggregation
- The inputs can be excitatory or inhibitory
- $y = 0$ if any x_i is inhibitory, else

$$g(x_1, x_2, \dots, x_n) = g(\mathbf{x}) = \sum_{i=1}^n x_i$$
$$y = f(g(\mathbf{x})) = 1 \quad \text{if } \underline{g(\mathbf{x})} \geq \theta$$
$$= 0 \quad \text{if } \underline{g(\mathbf{x})} < \theta$$

- θ is called the thresholding parameter
- This is called Thresholding Logic

McCulloch Pitts Neuron

$$y \in \{0, 1\}$$



A McCulloch Pitts unit

$$y \in \{0, 1\}$$

3

AND function

$$y \in \{0, 1\}$$

1

NOR function

$$g(x) = 1 + 1 + 1 \\ = 3$$

$$3 \geq \theta \\ 3 \geq 1$$

$\{0, 1\}$

$$y \in \{0, 1\}$$

0

$$y \in \{0, 1\}$$

1

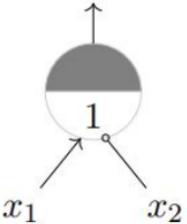
OR function

$$y \in \{0, 1\}$$

0

NOT function

$$y \in \{0, 1\}$$



x_1 AND $\neg x_2^*$

0

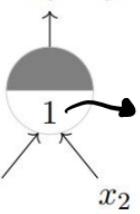
NOT function

$$x_1 + \underline{x_2} = 1$$

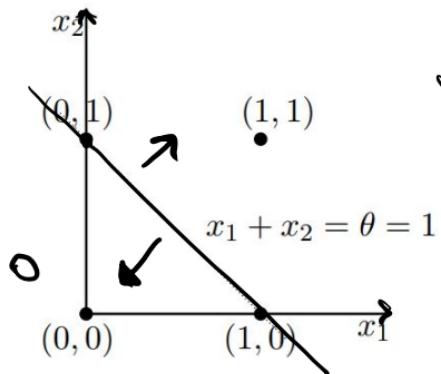
McCulloch Pitts Neuron

0	1	=	1
1	0	=	1
1	1	=	1
0	0	=	0

$$y \in \{0, 1\}$$



OR function
 $x_1 + x_2 = \sum_{i=1}^2 x_i \geq 1$



↳ Boolean

$$g(x) = x_1 + x_2 \geq 1$$

- A single MP neuron splits the input points (4 points for 2 binary inputs) into two halves
- Points lying on or above the line $\sum_{i=1}^n x_i - \theta = 0$ and points lying below this line
- In other words, all inputs which produce an output 0 will be on one side ($\sum_{i=1}^n x_i < \theta$) of the line and all inputs which produce an output 1 will lie on the other side ($\sum_{i=1}^n x_i \geq \theta$) of this line
- Let us convince ourselves about this with a few more examples (if it is not already clear from the math)

$$x_1 + x_2 - 1 = 0$$

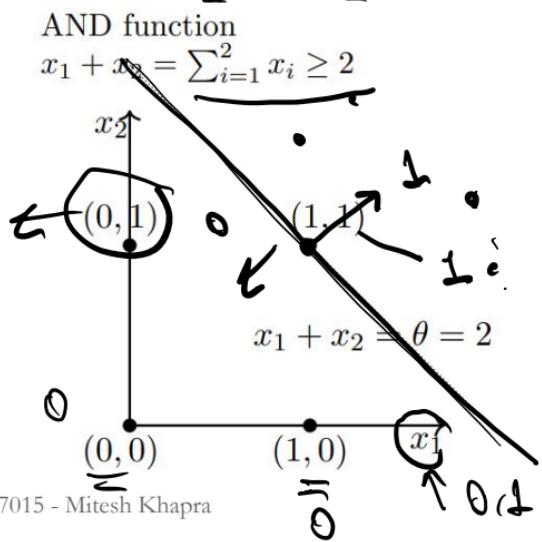
$$\Rightarrow x_1 + \underline{x_2} = 1$$

0	0	1	1
0	1	0	1
1	1	1	0

$$x_1 + x_2 \geq 2$$

$$x_1 + x_2 = 2$$

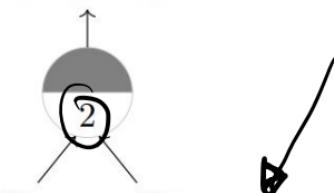
$$0 \wedge 1 = 0$$



Slide Credit: CS7015 - Mitesh Khapra

McCulloch Pitts Neuron

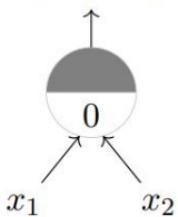
$$y \in \{0, 1\}$$



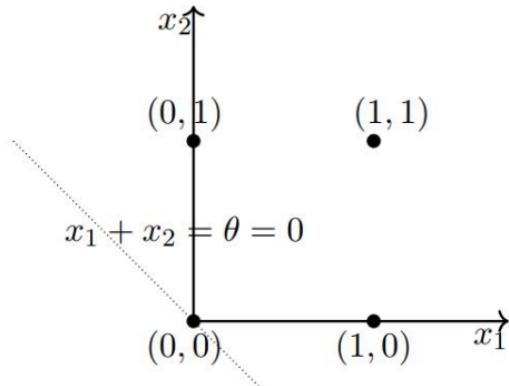
AND function

$$x_1 + x_2 = \sum_{i=1}^2 x_i \geq 2$$

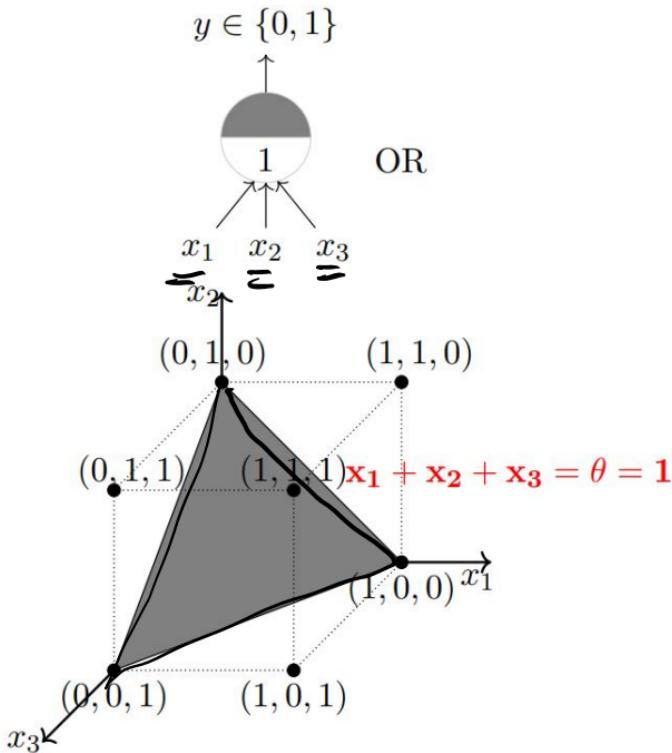
$$y \in \{0, 1\}$$



Tautology (always ON)



McCulloch Pitts Neuron



- What if we have more than 2 inputs?
- Well, instead of a line we will have a plane
- For the OR function, we want a plane such that the point $(0,0,0)$ lies on one side and the remaining 7 points lie on the other side of the plane

1
7 / 1 → $(0, 0, 0)$

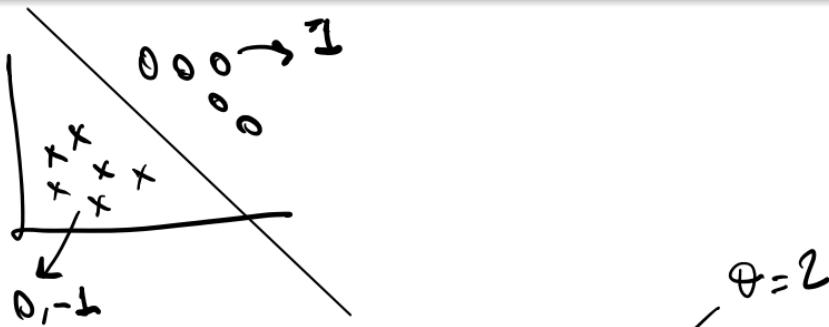
$\odot \rightarrow \circ e$

McCulloch Pitts Neuron

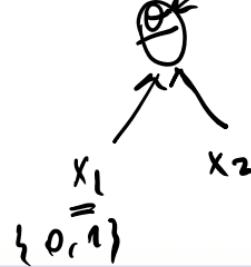
AND, OR, NOT - - -

The story so far ...

- A single McCulloch Pitts Neuron can be used to represent boolean functions which are linearly separable
- Linear separability (for boolean functions) : There exists a line (plane) such that all inputs which produce a 1 lie on one side of the line (plane) and all inputs which produce a 0 lie on other side of the line (plane)



Perceptrons

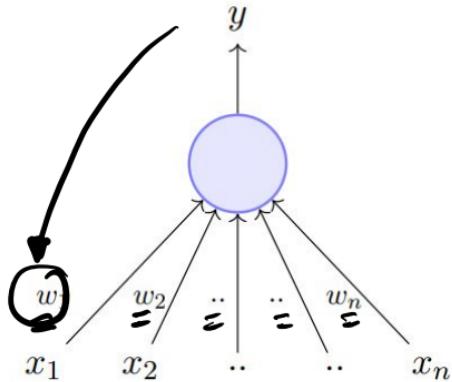


The story ahead ...

- What about non-boolean (say, real) inputs ?
- Do we always need to hand code the threshold ?
- Are all inputs equal ? What if we want to assign more weight (importance) to some inputs ?
- What about functions which are not linearly separable ?



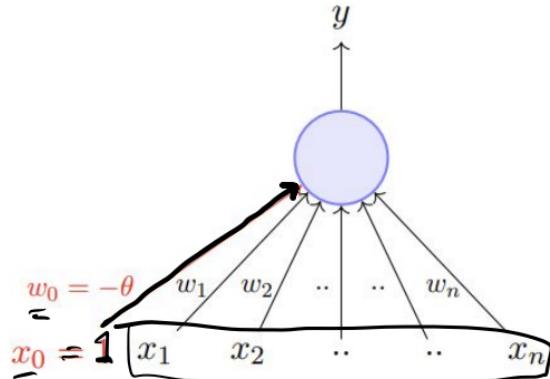
Perceptron



↳
real numbers
72

- Frank Rosenblatt, an American psychologist, proposed the **classical perceptron** model (1958)
- A more general computational model than McCulloch–Pitts neurons
- **Main differences:** Introduction of numerical weights for inputs and a mechanism for learning these weights
- Inputs are no longer limited to boolean values
- Refined and carefully analyzed by Minsky and Papert (1969) - their model is referred to as the **perceptron** model here

Perceptron



A more accepted convention,

$$y = 1 \quad \text{if} \sum_{i=0}^n w_i * x_i \geq 0$$

$$= 0 \quad \text{if} \sum_{i=0}^n w_i * x_i < 0$$

where, $x_0 = 1$ and $w_0 = -\theta$

Rewriting the above,

$$\left. \begin{array}{ll} y = 1 & \text{if} \sum_{i=1}^n w_i * x_i - \theta \geq 0 \\ = 0 & \text{if} \sum_{i=1}^n w_i * x_i - \theta < 0 \end{array} \right\}$$

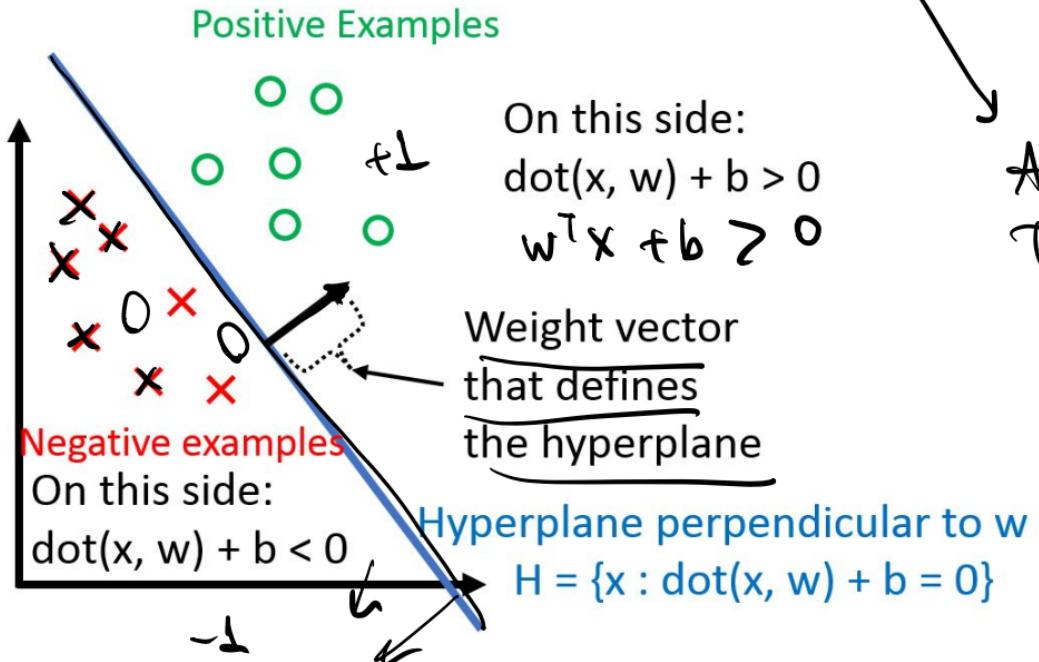
$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$

$\mathbf{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix}$

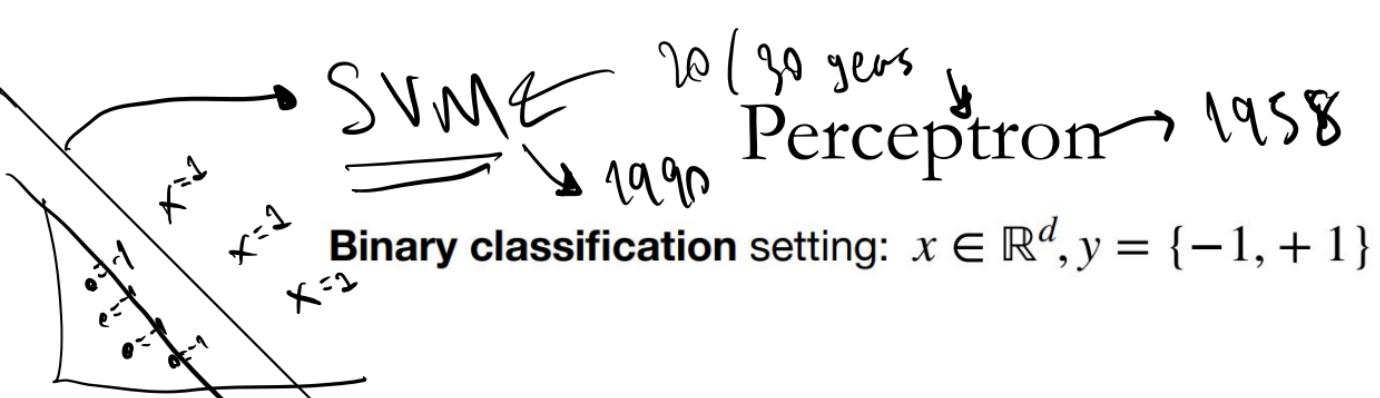
Perceptron

$$y = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \\ 1 \end{bmatrix}^T \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \\ b \end{bmatrix} = \sum_{i=1}^n w_i x_i + b$$

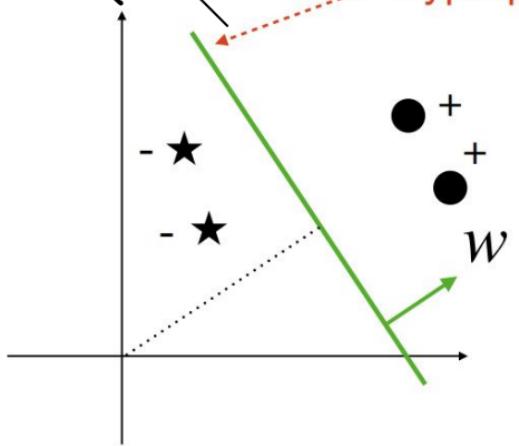
Question: Is it better or worse for the perceptron to run in higher dimensions?



Assume for 3:
The data should
be linearly separable.

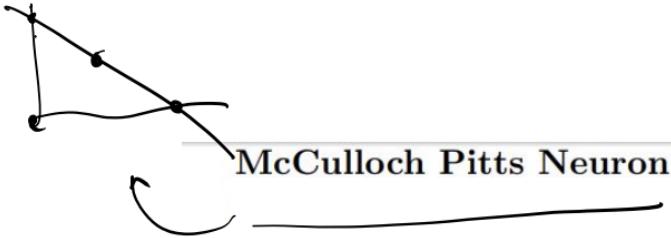


$$\text{Hyperplane } H = \{x : w^T x + b = 0\}$$



w : weight vector, wlog assume $\|w\|_2 = 1$

b : bias term; $|b|$ determines the distance of the hyperplane to origin



$$y = 1 \quad \text{if } \sum_{i=0}^n x_i \geq 0$$

$$= 0 \quad \text{if } \sum_{i=0}^n x_i < 0$$

\Downarrow
 $\{0,1\}$

Perceptron

$$y = 1 \quad \text{if } \sum_{i=0}^n w_i * x_i \geq 0$$

$$= 0 \quad \text{if } \sum_{i=0}^n w_i * x_i < 0$$

Perceptron

- From the equations it should be clear that even a perceptron separates the input space into two halves
- All inputs which produce a 1 lie on one side and all inputs which produce a 0 lie on the other side
 \Downarrow
- In other words, a single perceptron can only be used to implement linearly separable functions
- Then what is the difference? The weights (including threshold) can be learned and the inputs can be real valued
- We will first revisit some boolean functions and then see the perceptron learning algorithm (for learning weights)

$$1.1x_1 + 1.1x_2 - 1 = 0$$

Perceptron

$w_{0,1}$

$x_0 = 1$

x_1	x_2	OR	$w_0 + \sum_{i=1}^2 w_i x_i < 0$
0	0	0	$w_0 + \sum_{i=1}^2 w_i x_i < 0$
1	0	1	$w_0 + \sum_{i=1}^2 w_i x_i \geq 0$
0	1	1	$w_0 + \sum_{i=1}^2 w_i x_i \geq 0$
1	1	1	$w_0 + \sum_{i=1}^2 w_i x_i \geq 0$

1. $w_0 + w_1 \cdot 0 + w_2 \cdot 0 < 0 \implies w_0 < 0$ ✓

$$w_0 + \overbrace{w_1 \cdot 0} + \overbrace{w_2 \cdot 1} \geq 0 \implies w_2 \geq -w_0 \quad \checkmark$$

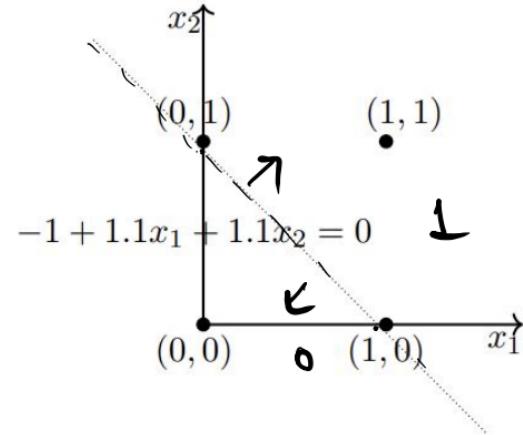
$$w_0 + w_1 \cdot 1 + w_2 \cdot 0 \geq 0 \implies w_1 \geq -w_0 \quad \checkmark$$

$$w_0 + w_1 \cdot 1 + w_2 \cdot 1 \geq 0 \implies w_1 + w_2 \geq -w_0 \quad \checkmark$$

- One possible solution to this set of inequalities is $w_0 = -1, w_1 = 1.1, w_2 = 1.1$ (and various other solutions are possible)



o



- Note that we can come up with a similar set of inequalities and find the value of θ for a McCulloch Pitts neuron also (Try it!)

Errors

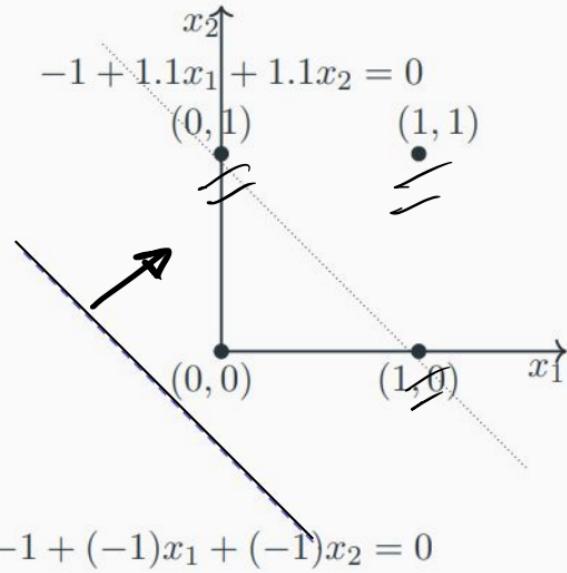
Let us fix the threshold ($-w_0 = 1$) and try different values of w_1, w_2

Say, $w_1 = -1, w_2 = -1$

What is wrong with this line? We make an error on 1 out of the 4 inputs

Lets try some more values of w_1, w_2 and note how many errors we make

w_1	w_2	errors
-1	-1	3

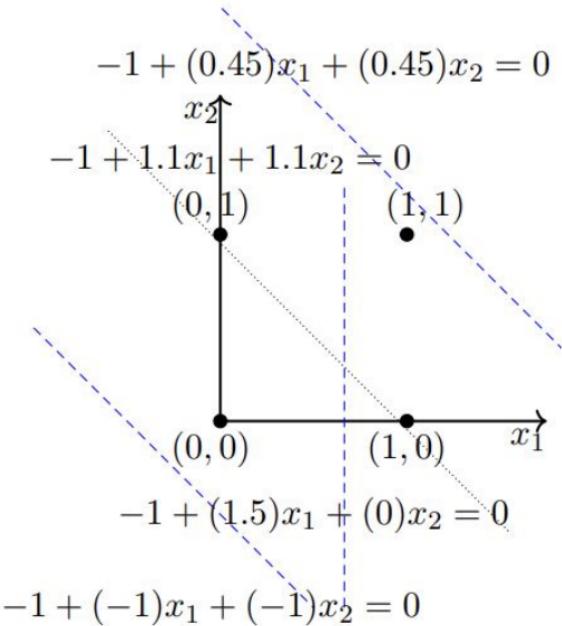


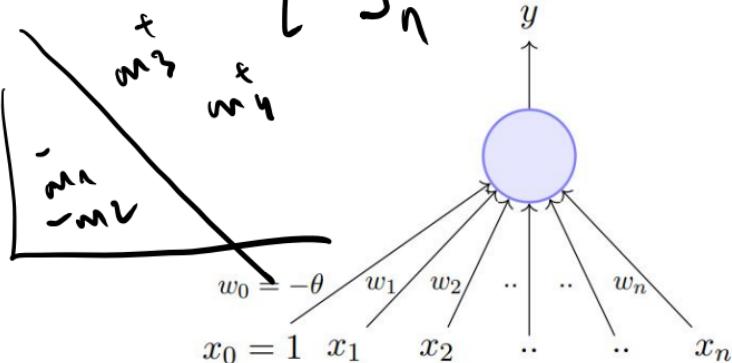
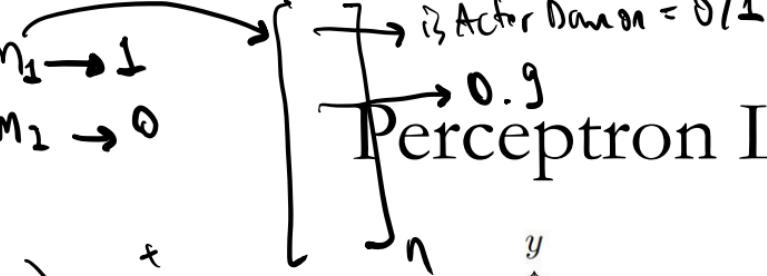
Errors

- Let us fix the threshold ($-w_0 = 1$) and try different values of w_1, w_2
- Say, $w_1 = -1, w_2 = -1$
- What is wrong with this line? We make an error on 1 out of the 4 inputs
- Lets try some more values of w_1, w_2 and note how many errors we make

w_1	w_2	errors
1	-1	3
1.5	0	1
0.45	0.45	3

- We are interested in those values of w_0, w_1, w_2 which result in 0 error
- Let us plot the error surface corresponding to different values of w_0, w_1, w_2





- $x_1 = \text{isActorDamon}$
- $x_2 = \text{isGenreThriller}$
- $x_3 = \text{isDirectorNolan}$
- $x_4 = \text{imdbRating} (\text{scaled to } 0 \text{ to } 1)$
- ...
- $x_n = \text{criticsRating} (\text{scaled to } 0 \text{ to } 1)$

- Let us reconsider our problem of deciding whether to watch a movie or not
- Suppose we are given a list of m movies and a label (class) associated with each movie indicating whether the user liked this movie or not : binary decision
- Further, suppose we represent each movie with n features (some boolean, some real valued)
- We will assume that the data is linearly separable and we want a perceptron to learn how to make this decision
- In other words, we want the perceptron to find the equation of this separating plane (or find the values of $w_0, w_1, w_2, \dots, w_m$)

Perceptron Learning Algorithm

Algorithm: Perceptron Learning Algorithm

$P \leftarrow$ inputs with label 1;

$N \leftarrow$ inputs with label 0; \downarrow

Initialize w randomly;

while !convergence **do**

 Pick random $x \in P \cup N$;

if $x \in P$ and $\sum_{i=0}^n w_i * x_i < 0$ **then**

$w = w + x$;

end

if $x \in N$ and $\sum_{i=0}^n w_i * x_i \geq 0$ **then**

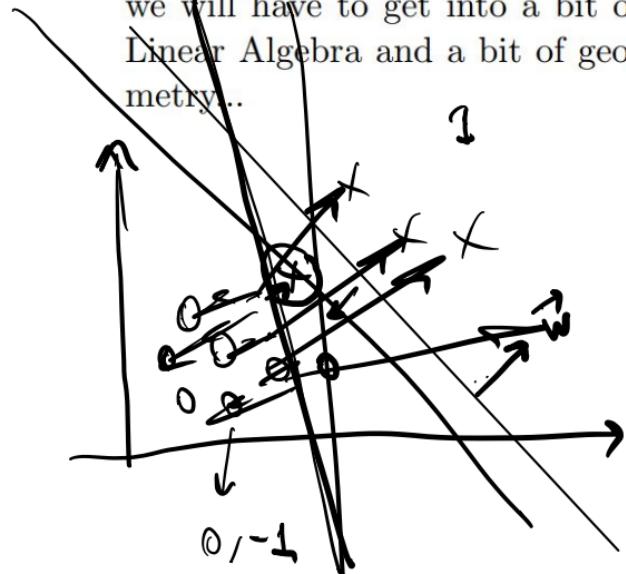
$w = w - x$;

end

end

//the algorithm converges when all the
inputs are classified correctly

- Why would this work ?
- To understand why this works we will have to get into a bit of Linear Algebra and a bit of geometry...



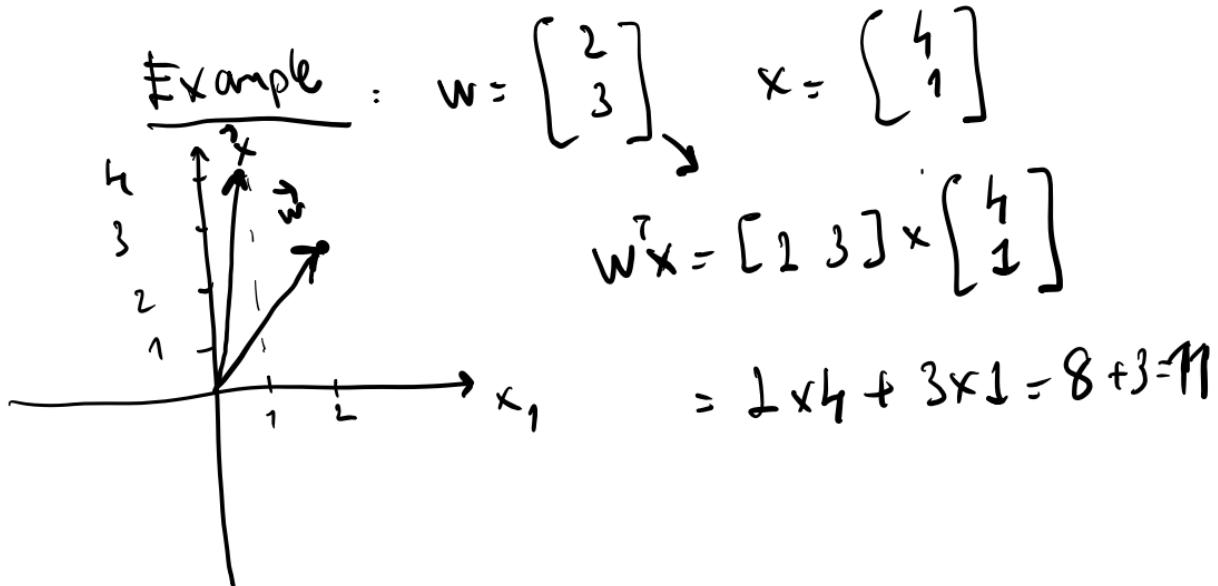
① examples $\in N$
X example $\in P$

Basic Linear Algebra

Dot Product: Also called scalar product, is an operation that takes in two vectors and produces a single number (scalar) as the output.

- This number can provide information about the angle between the two vectors or the magnitude of one vector in the direction of the other.

$w^T x$
↓
weights
vector



Basic Linear Algebra

Norm: A vector, in simple terms, is a quantity that has both direction and magnitude.

Imagine an arrow pointing in a certain direction - the length of this arrow represents the magnitude of the vector, which is also referred to as its norm.

Example

For 2d

$$v = [a, b]$$

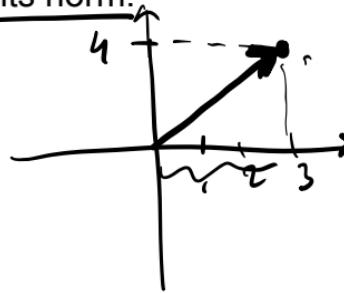
$$\|v\| = \sqrt{a^2 + b^2}$$

size / length of a vector

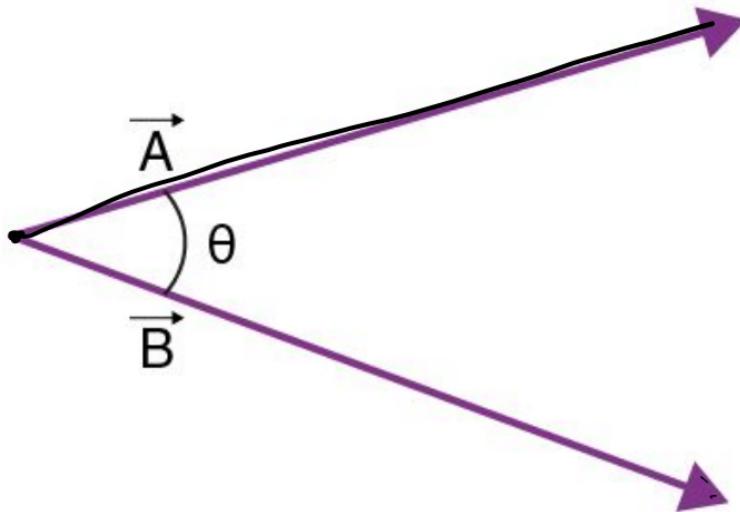
Ex/ $v = [3, 4]$

Pythagorean Theorem.

$$\begin{aligned}\|v\| &= \sqrt{3^2 + 4^2} \\ &= \sqrt{25} \\ &= 5\end{aligned}$$



Angle between two vectors



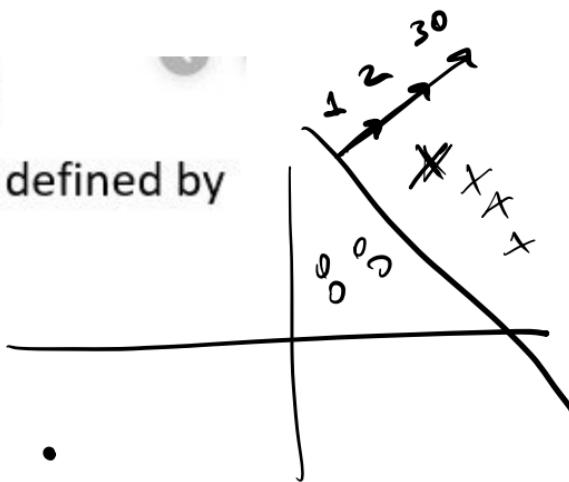
© Byjus.com

Angle between two vectors

Angle Between Two Vectors

The angle between vectors \mathbf{u} and \mathbf{v} can be defined by

$$\cos \theta = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\| \|\mathbf{v}\|}$$



The vectors are parallel if \mathbf{u} and \mathbf{v} are scalar multiples. $0^\circ \leq \theta \leq 180^\circ //$

The vectors are perpendicular if $\underline{\mathbf{u} \cdot \mathbf{v} = 0}$

Perceptron Learning Algorithm

Algorithm: Perceptron Learning Algorithm

```
P ← inputs with label 1;  
N ← inputs with label 0;  
Initialize w randomly;  
while !convergence do  
    Pick random x ∈ P ∪ N ;  
    if x ∈ P and w.x < 0 then  
        w = w + x ;  
    end  
    if x ∈ N and w.x ≥ 0 then  
        w = w - x ;  
    end  
end  
//the algorithm converges when all the  
inputs are classified correctly
```

$$\cos\alpha = \frac{\mathbf{w}^T \mathbf{x}}{\|\mathbf{w}\| \|\mathbf{x}\|}$$

- For $\mathbf{x} \in N$ if $\mathbf{w} \cdot \mathbf{x} \geq 0$ then it means that the angle (α) between this \mathbf{x} and the current \mathbf{w} is less than 90° (but we want α to be greater than 90°)

- What happens to the new angle (α_{new}) when $\mathbf{w}_{new} = \mathbf{w} - \mathbf{x}$

$$\begin{aligned}\cos(\alpha_{new}) &\propto \mathbf{w}_{new}^T \mathbf{x} \\ &\propto (\mathbf{w} - \mathbf{x})^T \mathbf{x} \\ &\propto \mathbf{w}^T \mathbf{x} - \mathbf{x}^T \mathbf{x} \\ &\propto \boxed{\cos\alpha - \mathbf{x}^T \mathbf{x}}\end{aligned}$$

$$\cos(\alpha_{new}) < \cos\alpha$$

Thus α_{new} will be greater than α and this is exactly what we want

$$\cos(0) = 1$$

$$\cos(\underline{180}) = -1$$

Perceptron Learning Algorithm

Algorithm: Perceptron Learning Algorithm

$P \leftarrow$ inputs with label 1;

Angles (in Degrees)	0°	30°	45°	60°	90°	180°	270°
Angles (in Radians)	0	$\pi/6$	$\pi/4$	$\pi/3$	$\pi/2$	π	$3\pi/2$
sin	0	$1/\sqrt{2}$	$1/\sqrt{2}$	$\sqrt{3}/2$	-1	0	-1
cos	1	$\sqrt{3}/2$	$1/\sqrt{2}$	$1/\sqrt{2}$	0	-1	0

```

    w
    |
end
if x ∈ N and w.x ≥ 0 then
    w = w - x ;
end
end
//the algorithm converges when all the
inputs are classified correctly

```

$$\cos\alpha = \frac{\mathbf{w}^T \mathbf{x}}{\|\mathbf{w}\| \|\mathbf{x}\|}$$

- For $\mathbf{x} \in N$ if $\mathbf{w} \cdot \mathbf{x} \geq 0$ then it means that the angle (α) between this \mathbf{x} and the current \mathbf{w} is less than 90° (but we want α to be greater than 90°)

What happens to the new angle (α_{new}) when $\mathbf{w}_{new} = \mathbf{w} - \mathbf{x}$

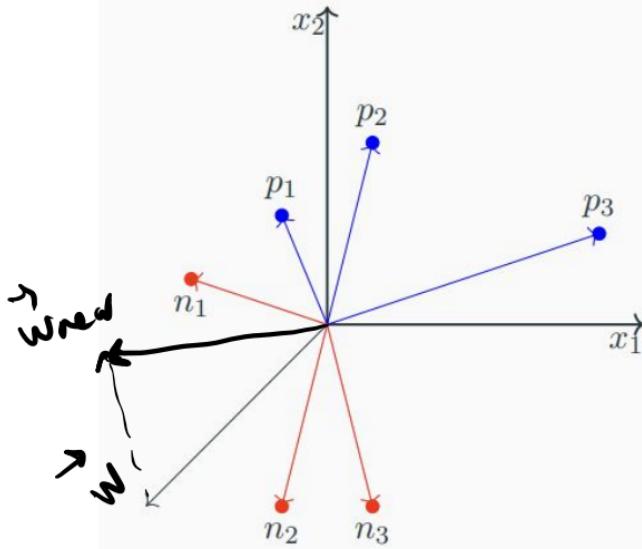
$$\begin{aligned} \cos(\alpha_{new}) &\propto \mathbf{w}_{new}^T \mathbf{x} \\ &\propto (\mathbf{w} - \mathbf{x})^T \mathbf{x} \\ &\propto \mathbf{w}^T \mathbf{x} - \mathbf{x}^T \mathbf{x} \\ &\propto \cos\alpha - \mathbf{x}^T \mathbf{x} \end{aligned}$$

$$\cos(\alpha_{new}) < \cos\alpha$$

- Thus α_{new} will be greater than α and this is exactly what we want

Example.

Perceptron Learning Algorithm



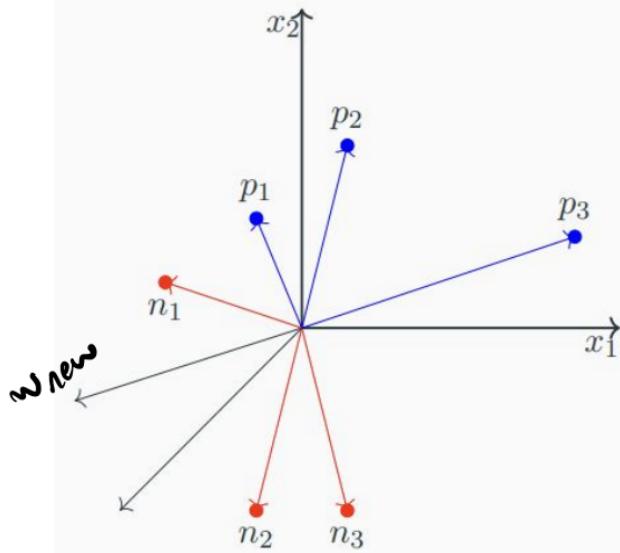
We initialized w to a random value

We observe that currently, $w \cdot x < 0$ (\because angle $> 90^\circ$) for all the positive points and $w \cdot x \geq 0$ (\because angle $< 90^\circ$) for all the negative points (the situation is exactly opposite of what we actually want it to be)

We now run the algorithm by randomly going over the points

Randomly pick a point (say, p_1), apply correction $w = w + x \because w \cdot x < 0$ (you can check the angle visually)

Perceptron Learning Algorithm



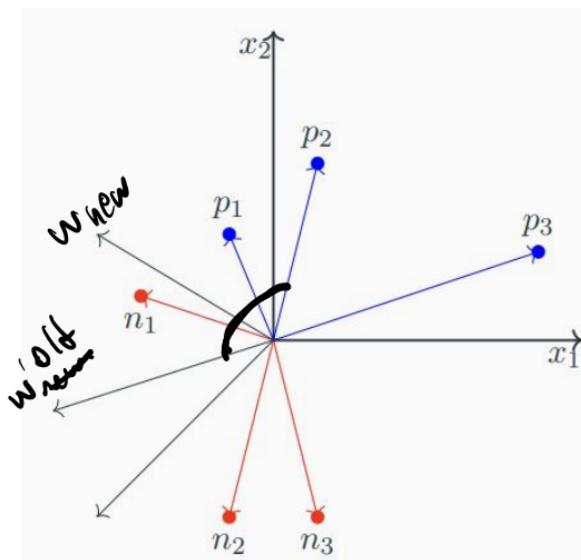
We initialized \mathbf{w} to a random value

We observe that currently, $\mathbf{w} \cdot \mathbf{x} < 0$ (\because angle $> 90^\circ$) for all the positive points and $\mathbf{w} \cdot \mathbf{x} \geq 0$ (\because angle $< 90^\circ$) for all the negative points (the situation is exactly opposite of what we actually want it to be)

We now run the algorithm by randomly going over the points

Randomly pick a point (say, p_1), apply correction
 $\mathbf{w} = \mathbf{w} + \mathbf{x} \because \mathbf{w} \cdot \mathbf{x} < 0$ (you can check the angle visually)

Perceptron Learning Algorithm



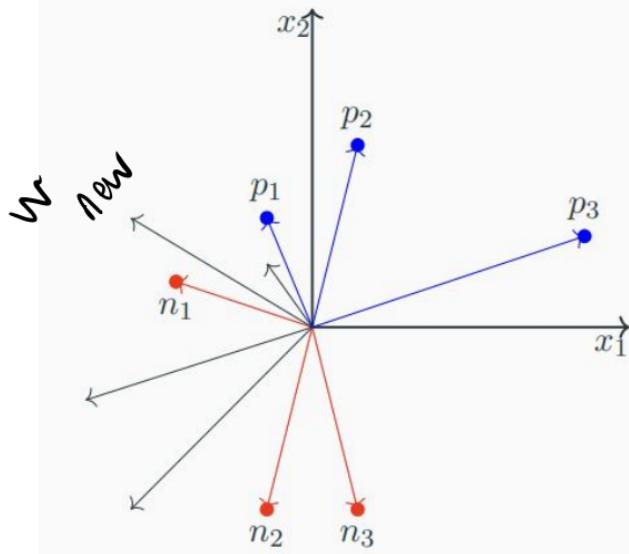
We initialized \mathbf{w} to a random value

We observe that currently, $\mathbf{w} \cdot \mathbf{x} < 0$ (\therefore angle $> 90^\circ$) for all the positive points and $\mathbf{w} \cdot \mathbf{x} \geq 0$ (\therefore angle $< 90^\circ$) for all the negative points (the situation is exactly opposite of what we actually want it to be)

We now run the algorithm by randomly going over the points

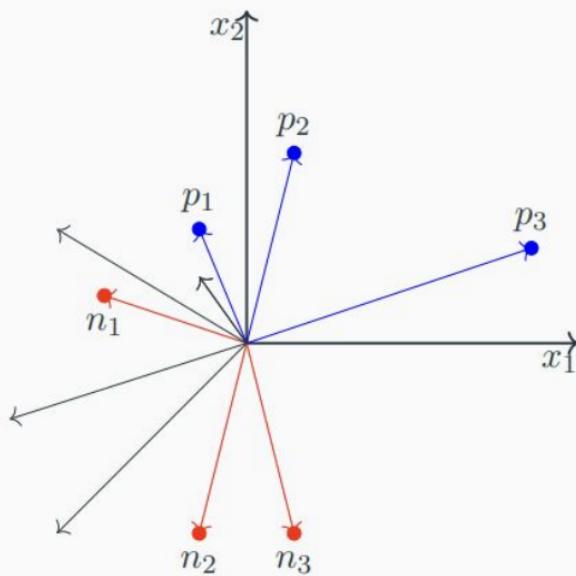
Randomly pick a point (say, p_2), apply correction
 $\mathbf{w} = \mathbf{w} + \mathbf{x} \because \mathbf{w} \cdot \mathbf{x} < 0$ (you can check the angle visually)

Perceptron Learning Algorithm



- We initialized \mathbf{w} to a random value
- We observe that currently, $\mathbf{w} \cdot \mathbf{x} < 0$ (\because angle $> 90^\circ$) for all the positive points and $\mathbf{w} \cdot \mathbf{x} \geq 0$ (\because angle $< 90^\circ$) for all the negative points (the situation is exactly opposite of what we actually want it to be)
- We now run the algorithm by randomly going over the points
 - Randomly pick a point (say, n_1) apply correction
 $\mathbf{w} = \mathbf{w} - \mathbf{x} \because \mathbf{w} \cdot \mathbf{x} \geq 0$ (you can check the angle visually)

Perceptron Learning Algorithm



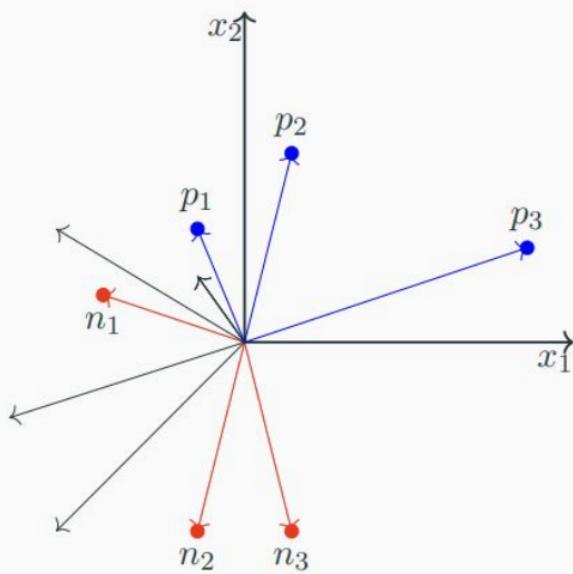
We initialized \mathbf{w} to a random value

We observe that currently, $\mathbf{w} \cdot \mathbf{x} < 0$ (\because angle $> 90^\circ$) for all the positive points and $\mathbf{w} \cdot \mathbf{x} \geq 0$ (\because angle $< 90^\circ$) for all the negative points (the situation is exactly opposite of what we actually want it to be)

We now run the algorithm by randomly going over the points

Randomly pick a point (say, n_3), no correction needed $\because \mathbf{w} \cdot \mathbf{x} < 0$ (you can check the angle visually)

Perceptron Learning Algorithm



We initialized w to a random value

We observe that currently, $w \cdot x < 0$ (\because angle $> 90^\circ$) for all the positive points and $w \cdot x \geq 0$ (\because angle $< 90^\circ$) for all the negative points (the situation is exactly opposite of what we actually want it to be)

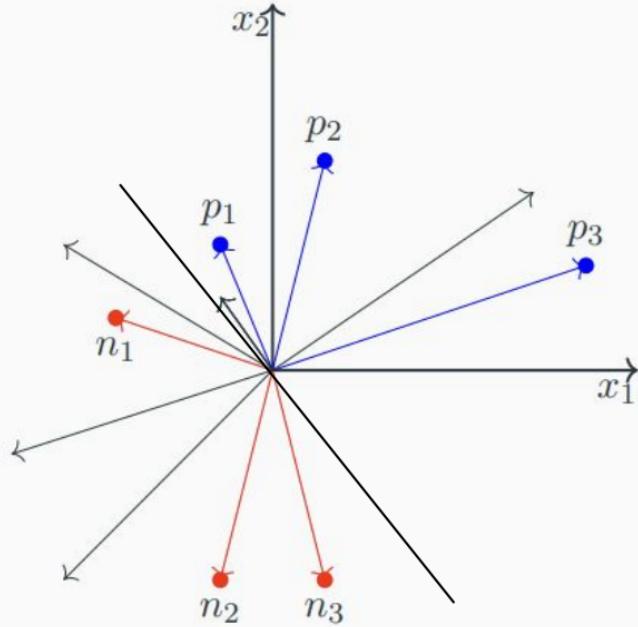
We now run the algorithm by randomly going over the points

Randomly pick a point (say, n_2), no correction needed $\because w \cdot x < 0$ (you can check the angle visually)

$$x \in P \text{ and } w^T x \leq 0$$

Perceptron Learning Algorithm

$$\begin{aligned} \mathbf{w} &\leq \mathbf{w}' \\ \text{and } \mathbf{w}'^\top \mathbf{x} &\geq 0 \\ \mathbf{w} &= \mathbf{w}' + \mathbf{x} \end{aligned}$$



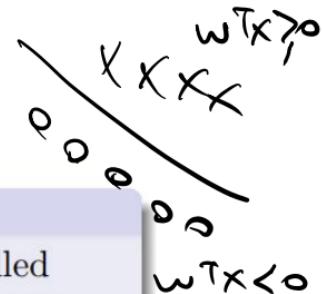
We initialized \mathbf{w} to a random value

We observe that currently, $\mathbf{w} \cdot \mathbf{x} < 0$ (\because angle $> 90^\circ$) for all the positive points and $\mathbf{w} \cdot \mathbf{x} \geq 0$ (\because angle $< 90^\circ$) for all the negative points (the situation is exactly opposite of what we actually want it to be)

We now run the algorithm by randomly going over the points

Randomly pick a point (say, p_3), apply correction
 $\mathbf{w} = \mathbf{w} + \mathbf{x} \because \mathbf{w} \cdot \mathbf{x} < 0$ (you can check the angle visually)

Proof of Convergence



Theorem

Definition: Two sets P and N of points in an n -dimensional space are called absolutely linearly separable if $n + 1$ real numbers w_0, w_1, \dots, w_n exist such that every point $\underline{(x_1, x_2, \dots, x_n)} \in P$ satisfies $\sum_{i=1}^n w_i * \underline{x_i} > w_0$ and every point $\underline{(x_1, x_2, \dots, x_n)} \in N$ satisfies $\sum_{i=1}^n w_i * \underline{x_i} < w_0$.

Proposition: If the sets P and N are finite and linearly separable, the perceptron learning algorithm updates the weight vector \mathbf{w}_t a finite number of times. In other words: if the vectors in P and N are tested cyclically one after the other, a weight vector \mathbf{w}_t is found after a finite number of steps t which can separate the two sets.

\equiv

Proof: On the next slide

$$w_0 \xrightarrow{\quad} + \Rightarrow w_t$$

Proof of Convergence

Setup:

- If $x \in N$ then $-x \in P$ ($\because w^T x < 0 \implies w^T(-x) \geq 0$)
- We can thus consider a single set $P' = P \cup N^-$ and for every element $p \in P'$ ensure that $w^T p \geq 0 //$
- Further we will normalize all the p 's so that $\|p\| = 1$ (notice that this does not affect the solution \because if $w^T \frac{p}{\|p\|} \geq 0$ then $w^T p \geq 0$)
- Let \underline{w} be the normalized solution vector (we know one exists as the data is linearly separable)

Algorithm: Perceptron Learning Algorithm

$P \leftarrow$ inputs with label 1; ✓

$N \leftarrow$ inputs with label 0; ✓

N^- contains negations of all points in N ;

$P' = P \cup N^-$; ✓

Initialize \mathbf{w} randomly;

while !convergence **do**

 Pick random $\mathbf{p} \in P'$;

$\mathbf{p} \leftarrow \frac{\mathbf{p}}{\|\mathbf{p}\|}$ (so now, $\|\mathbf{p}\| = 1$) ; ✓

if $\mathbf{w} \cdot \mathbf{p} < 0$ **then**

$\mathbf{w} = \mathbf{w} + \mathbf{p}$;

end

end

//the algorithm converges when all the inputs are classified correctly

//notice that we do not need the other **if** condition because by construction we want all points in P' to lie in the positive half space $\mathbf{w} \cdot \mathbf{p} \geq 0$

P
Inputs

$\overline{P \cup N^-}$

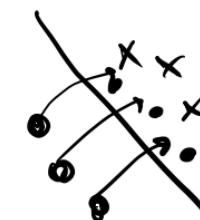
*

w

w

P
 N
 $P' \rightarrow P \cup N^-$

P''
 $w^T p \geq 0$



$\|p\| = 1$

<

Proof of Convergence

Observations:

- w^* is some optimal solution which exists but we don't know what it is
- We do not make a correction at every time-step
- We make a correction only if $w^T \cdot p_i \leq 0$ at that time step
- So at time-step t we would have made only \bar{k} ($\leq t$) corrections
- Every time we make a correction a quantity δ gets added to the numerator
- So by time-step t , a quantity $k\delta$ gets added to the numerator

Proof:

- Now suppose at time step t we inspected the point p_i and found that $w^T \cdot p_i < 0$
- We make a correction $w_{t+1} = w_t + p_i$
- Let β be the angle between w^* and w_{t+1}

$$\cos \beta = \frac{w^* \cdot w_{t+1}}{\|w_{t+1}\| \|w^*\|}$$

$$\text{Numerator} = w^* \cdot w_{t+1} = w^* \cdot (w_t + p_i)$$

$$= w^* \cdot \overline{w_t} + w^* \cdot \overline{p_i}$$

$$\geq w^* \cdot \overline{w_t} + \underline{\delta} \quad (\delta = \min\{w^* \cdot p_i | \forall i\})$$

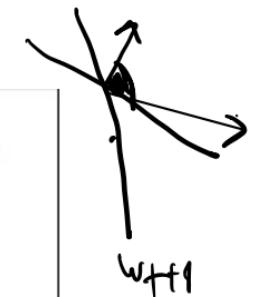
$$\geq w^* \cdot (w_{t-1} + p_j) + \delta$$

$$\geq w^* \cdot \overline{w_{t-1}} + \underline{w^* \cdot p_j} + \delta$$

$$\geq w^* \cdot \overline{w_{t-1}} + 2\underline{\delta}$$

Numerator

$$\geq w^* \cdot w_0 + (k)\delta \quad (\text{By induction})$$



$$w_{t+1} = w_t + p_i$$

Proof of Convergence

Proof (continued:)

So far we have, $w^T \cdot p_i \leq 0$ (and hence we made the correction)

$$\cos\beta = \frac{w^* \cdot w_{t+1}}{\|w_{t+1}\|} \quad (\text{by definition})$$

Numerator $\geq w^* \cdot w_0 + k\delta$ (proved by induction)

$$\begin{aligned} \text{Denominator}^2 &= \|w_{t+1}\|^2 \\ &= (w_t + p_i) \cdot (w_t + p_i) \\ &= \|w_t\|^2 + 2w_t \cdot p_i + \|p_i\|^2 \\ &\leq \|w_t\|^2 + \|p_i\|^2 \quad (\because w_t \cdot p_i \leq 0) \\ &\leq \|w_t\|^2 + 1 \quad (\because \|p_i\|^2 = 1) \\ &\leq (\|w_{t-1}\|^2 + 1) + 1 \\ &\leq \|w_{t-1}\|^2 + 2 \\ &\leq \|w_0\|^2 + (k) \quad (\text{By same observation that we made about } \delta) \end{aligned}$$

$$w_{t+1} = w_t + p_i$$

Proof of Convergence

Proof (continued:)

So far we have, $w^T \cdot p_i \leq 0$ (and hence we made the correction)

$$\cos\beta = \frac{w^* \cdot w_{t+1}}{\|w_{t+1}\|} \quad (\text{by definition})$$

Numerator $\geq w^* \cdot w_0 + k\delta$ (proved by induction)

Denominator² $\leq \|w_0\|^2 + k$ (By same observation that we made about δ)

$$\cos\beta \geq \frac{w^* \cdot w_0 + k\delta}{\sqrt{\|w_0\|^2 + k}}$$

- $\cos\beta$ thus grows proportional to \sqrt{k}
- As k (number of corrections) increases $\cos\beta$ can become arbitrarily large
- But since $\cos\beta \leq 1$, k must be bounded by a maximum number
- Thus, there can only be a finite number of corrections (k) to w and the algorithm will converge!





Proof of Convergence

Coming back to our questions ...

- What about non-boolean (say, real) inputs? **Real valued inputs are allowed in perceptron**
- Do we always need to hand code the threshold? **No, we can learn the threshold**
- Are all inputs equal? What if we want to assign more weight (importance) to some inputs? **A perceptron allows weights to be assigned to inputs**
- What about functions which are not linearly separable ? **Not possible with a single perceptron but we will see how to handle this ..**

Evaluation Metrics for Binary Classification

Evaluation metrics for binary classification

Knn
 perception
 supervised
 →
 test set
 100 → movies.
 So → +1
 So → -1
 Recall = $\frac{TP}{TP+FN}$

		Confusion matrix	
		0	1
0	actual negative	True Negative	False Positive
	actual positive	False Negative	True Positive
		predicted negative	predicted positive

$x_1 \rightarrow y_1$
 movie
 actual class
 ground truth
 $\hat{y} \rightarrow \text{prediction.}$
 0, 1
 1, 1

$$\text{Accuracy} = \frac{\frac{TP + TN}{TP + TN + FP + FN}}{\Sigma \# \text{ of pred.}}$$

≠ correct pred.

imbalanced

Problems with Accuracy

test cases
1 disease
gg → disease

Data with 90% negatives:

```
from sklearn.metrics import accuracy_score
for y_pred in [y_pred_1, y_pred_2, y_pred_3]:
    print(accuracy_score(y_true, y_pred))
```

0.9
0.9
0.9

90000.00 M -
1000.00 M +
9.000.00 0

		y_pred_1	
		N	P
True label	N	90	0
	P	10	0

Predicted label

		y_pred_2	
		N	P
True label	N	80	10
	P	0	10

Predicted label

		y_pred_3	
		N	P
True label	N	85	5
	P	5	5

Predicted label

~.99
every day
↓ M:
Y.L

Evaluation Metrics for Classification

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Positive Predicted Value (PPV)

~~100 disease~~ → Positive

~~100 test result~~

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

95 95
100

Sensitivity, coverage, true positive rate.

FP

$$F_1 = 2 \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Harmonic mean of precision and recall

Changing Thresholds

```
y_pred = rf.predict(X_test)  
print(classification_report(y_test, y_pred))
```

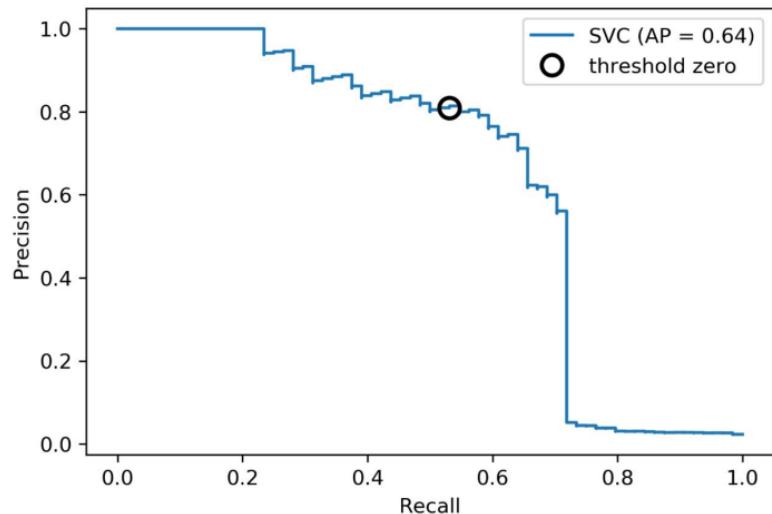
	precision	recall	f1-score	support
False	0.99	1.00	0.99	2732
True	0.90	0.56	0.69	64
accuracy			0.99	2796
macro avg	0.94	0.78	0.84	2796
weighted avg	0.99	0.99	0.99	2796

```
y_pred = rf.predict_proba(X_test)[:, 1] > .30  
  
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
False	0.99	0.99	0.99	2732
True	0.71	0.64	0.67	64
accuracy			0.99	2796
macro avg	0.85	0.82	0.83	2796
weighted avg	0.99	0.99	0.99	2796

Precision-Recall Curve

```
svc = make_pipeline(StandardScaler(), SVC(C=100, gamma=0.1))
svc.fit(X_train, y_train)
plot_precision_recall_curve(svc, X_test, y_test, name='SVC')
```

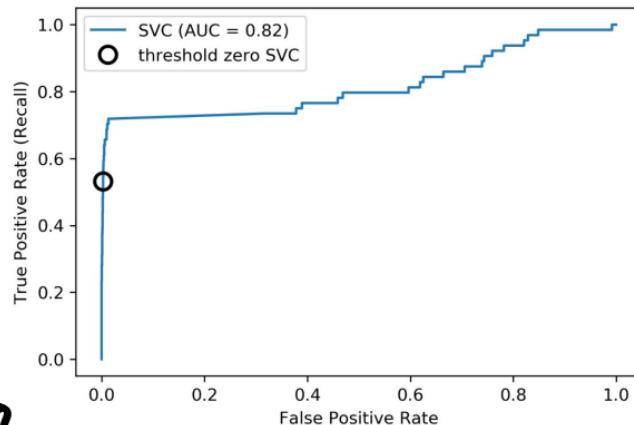


16

ROC Curve

(Receiver Operating Characteristic)

```
plot_roc_curve(svc, X_test, y_test, name='SVC')
```



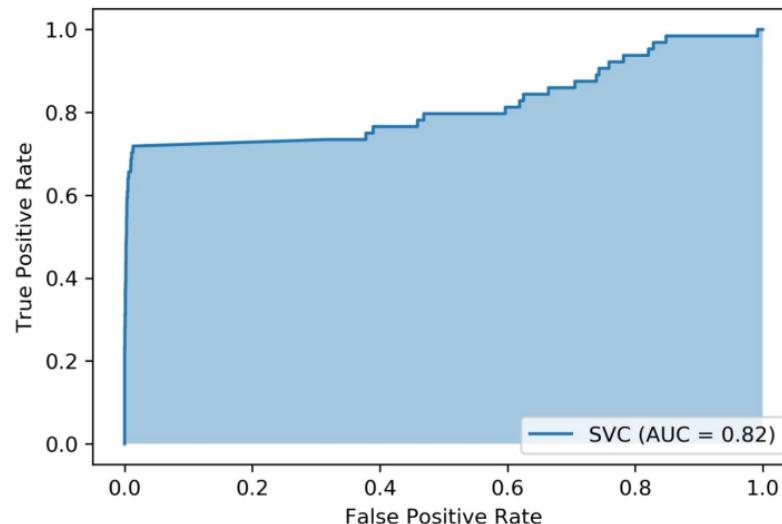
- True positive rate (recall)
- False Positive Rate (FPR)
 - Negative instances that are incorrectly classified as positive.
 - $1 - \text{True negative rate}$ (specificity)

Area Under ROC Curve (AUC)

$k_{nn} \rightarrow 0, 1$

perception $\rightarrow \dots$

logistic reg $\rightarrow 0.8$



- Always .5 for random/constant prediction

Summary of metrics for binary classification

- Threshold-based
 - (balanced) accuracy
 - precision , recall, f1
- Ranking
 - Average precision
 - ROC AUC

Picking metrics

- Accuracy rarely what you want
- Problems are rarely balanced
- Find the right criterion for the task
- OR pick a substitute, but at least think about it
- Emphasis on recall or precision?
- Which classes are the important ones?

Code Example

- 6-Evaluation Metrics.ipynb