MTH 221

Fundamentals of Machine Learning

Batuhan Bardak

Lecture 3: Perceptron & Evaluation Metrics

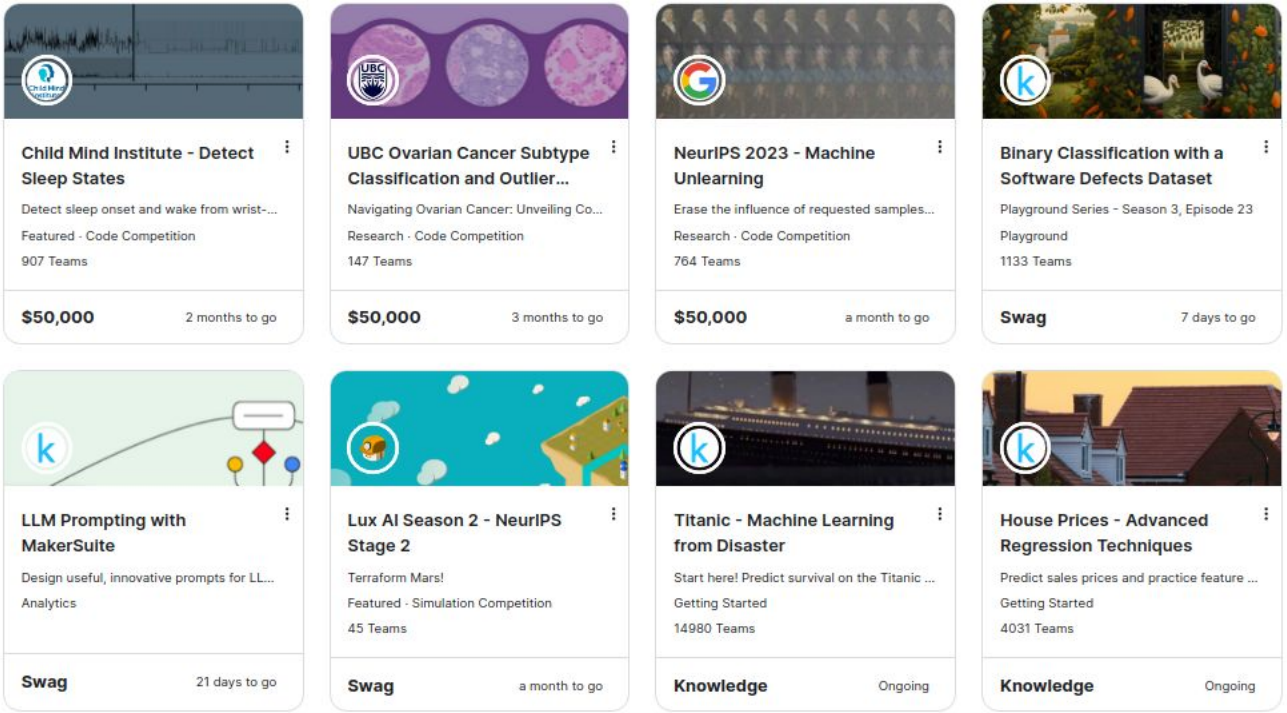
Date: 17.10.2023

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 Submssion: 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%.

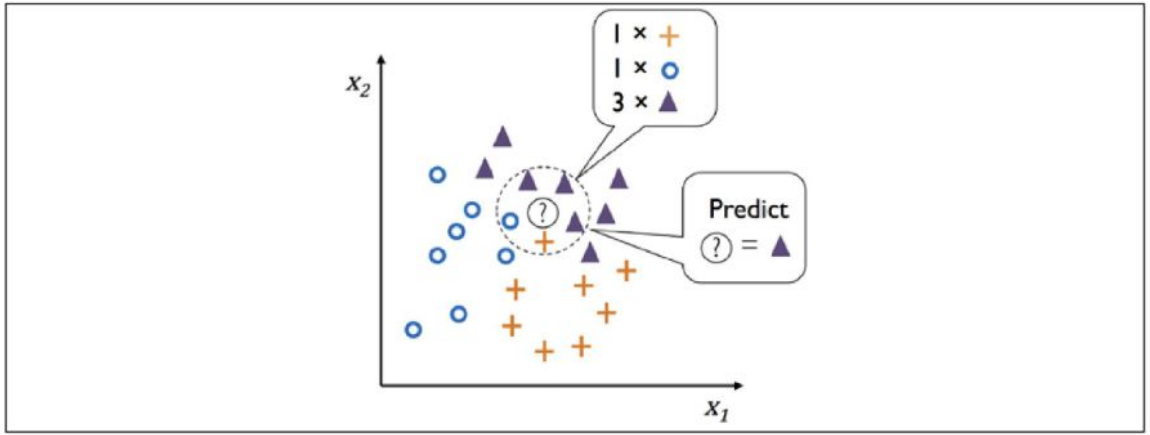
Announcements:

- Group Formation for the Project: 13/10/2023
  - New Dea
  - For thos
    the grou
- Project Propo
  - New Dea
  - As for gr
    decrease



Recap

- 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/



Slide Credit: Sebastian Raschka, Python Machine Learning

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.

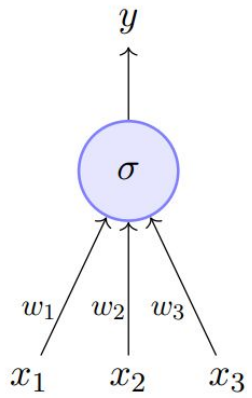
Recap

- 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

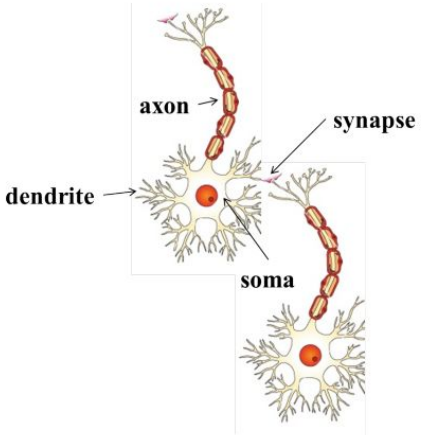


Artificial Neuron

- 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 ...

Slide Credit: CS7015 - Mitesh Khapra

Artificial Neuron



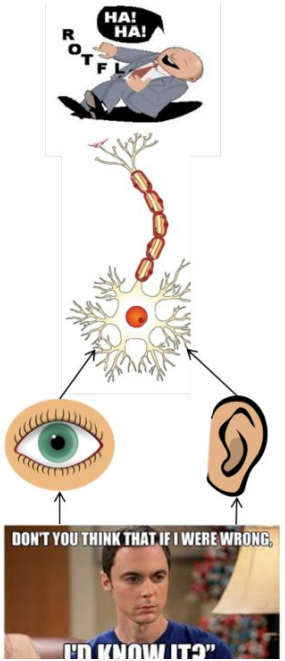
Biological Neurons*

- **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`

Slide Credit: CS7015 - Mitesh Khapra

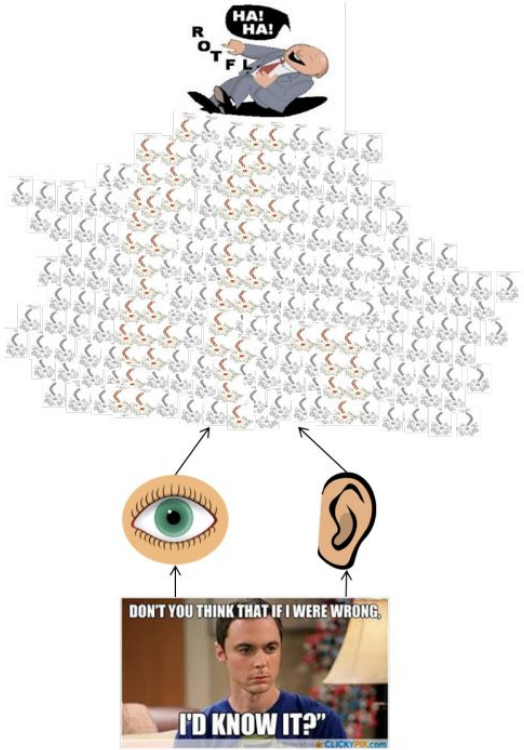
Artificial Neuron



- 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)

Slide Credit: CS7015 - Mitesh Khapra

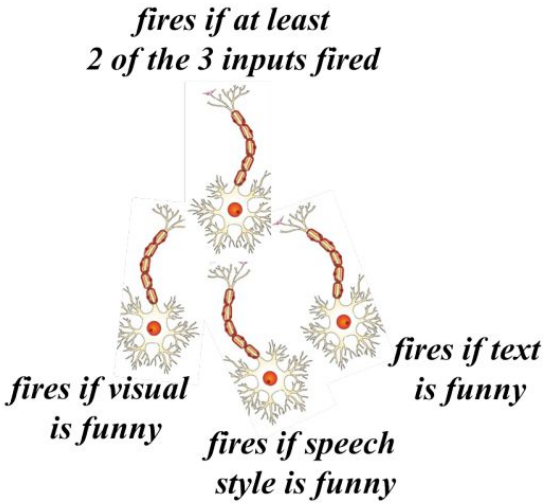
Artificial Neuron



- 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 response to 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!

Slide Credit: CS7015 - Mitesh Khapra

Artificial Neuron

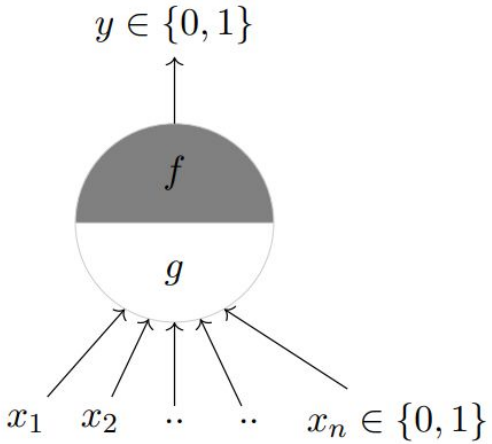


A simplified illustration

- 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

Slide Credit: CS7015 - Mitesh Khapra

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

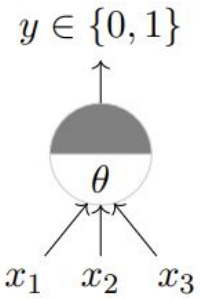
$$g(x_1, x_2, ..., x_n) = g(\mathbf{x}) = \sum_{i=1}^{n} x_i$$

$$y = f(g(\mathbf{x})) = 1 \quad if \quad g(\mathbf{x}) \geq \theta$$
$$= 0 \quad if \quad g(\mathbf{x}) < \theta$$

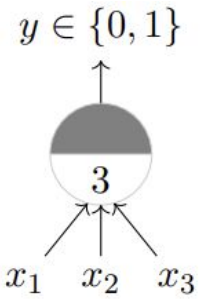
- θ is called the thresholding parameter
- This is called Thresholding Logic

Slide Credit: CS7015 - Mitesh Khapra

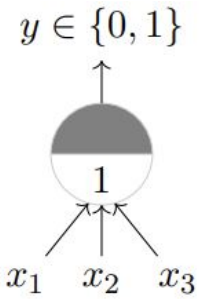
McCulloch Pitts Neuron



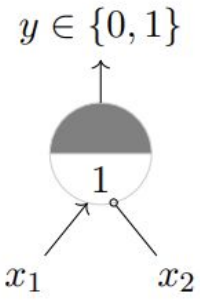
A McCulloch Pitts unit



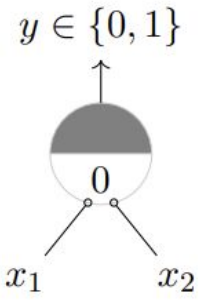
AND function



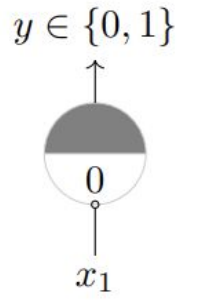
OR function



x_1 AND $!x_2$*



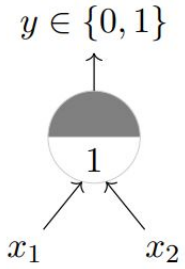
NOR function



NOT function

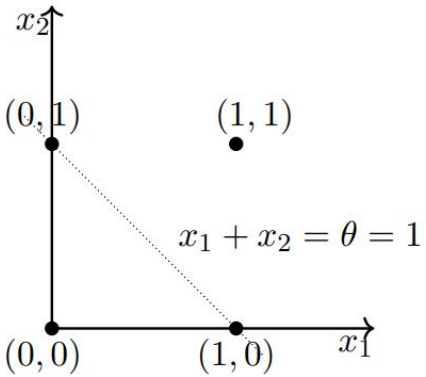
Slide Credit: CS7015 - Mitesh Khapra

McCulloch Pitts Neuron



OR function

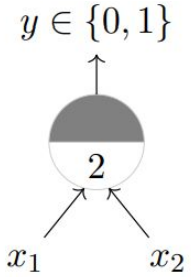
$x_1 + x_2 = \sum_{i=1}^{2} x_i \geq 1$

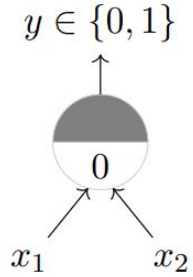


- A single MP neuron splits the input points (4 points for 2 binary inputs) into two halves
- Points lying on or above the line $\sum_{i=1}^{n} x_i - \theta = 0$ and points lying below this line
- In other words, all inputs which produce an output 0 will be on one side ($\sum_{i=1}^{n} x_i < \theta$) 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)

Slide Credit: CS7015 - Mitesh Khapra

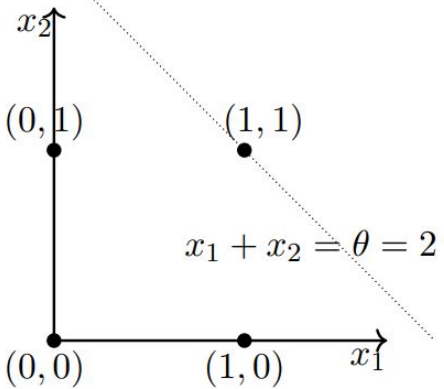
McCulloch Pitts Neuron



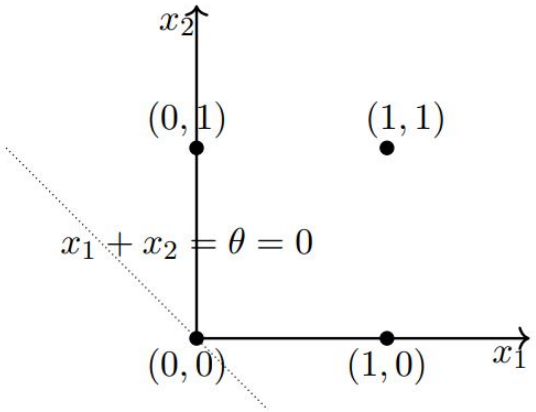


AND function

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

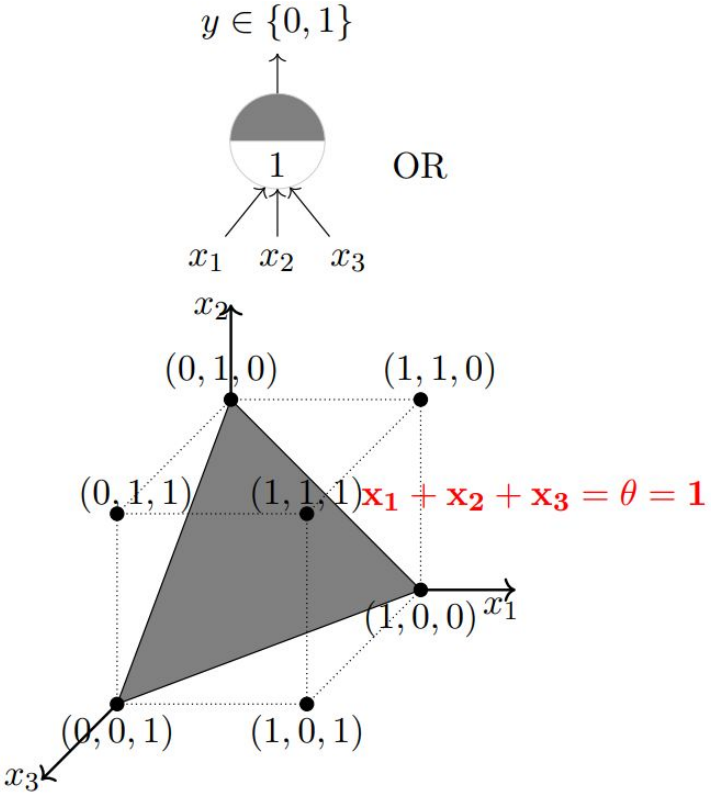


Tautology (always ON)



Slide Credit: CS7015 - Mitesh Khapra

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

Slide Credit: CS7015 - Mitesh Khapra

McCulloch Pitts Neuron

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)

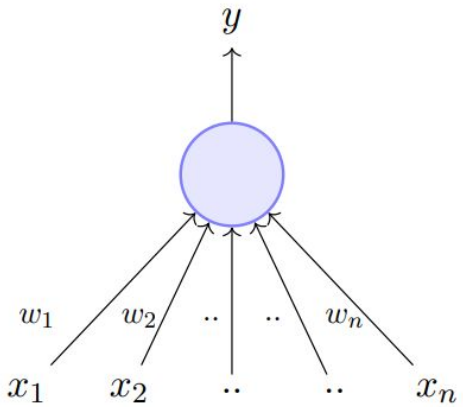
Perceptrons

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 ?

Slide Credit: CS7015 - Mitesh Khapra

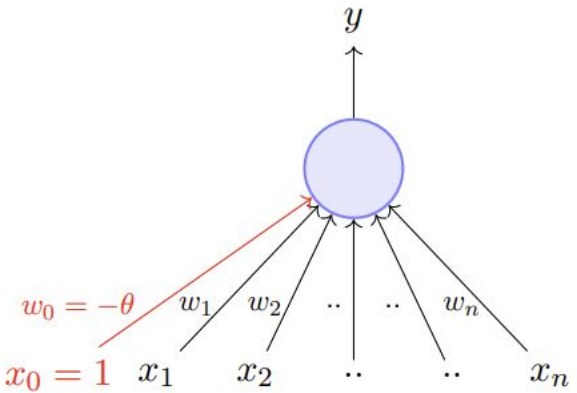
Perceptron



- 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

Slide Credit: CS7015 - Mitesh Khapra

Perceptron



A more accepted convention,

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

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

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

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

$$= 0 \quad if \sum_{i=1}^{n} w_i * x_i < \theta$$

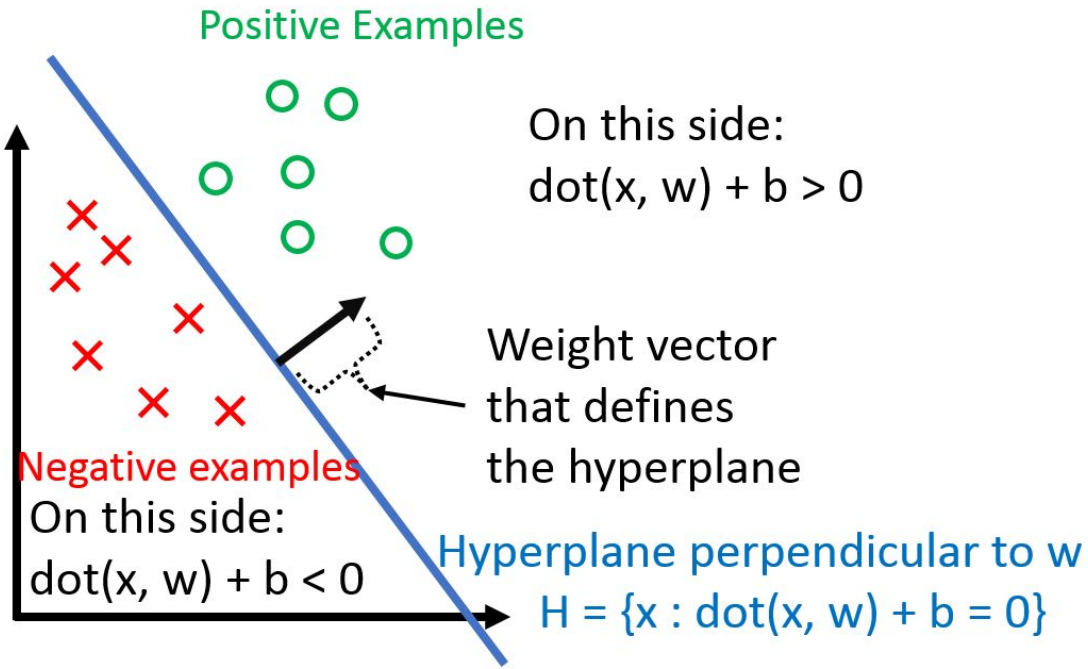
Rewriting the above,

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

$$= 0 \quad if \sum_{i=1}^{n} w_i * x_i - \theta < 0$$

Slide Credit: CS7015 - Mitesh Khapra

Perceptron

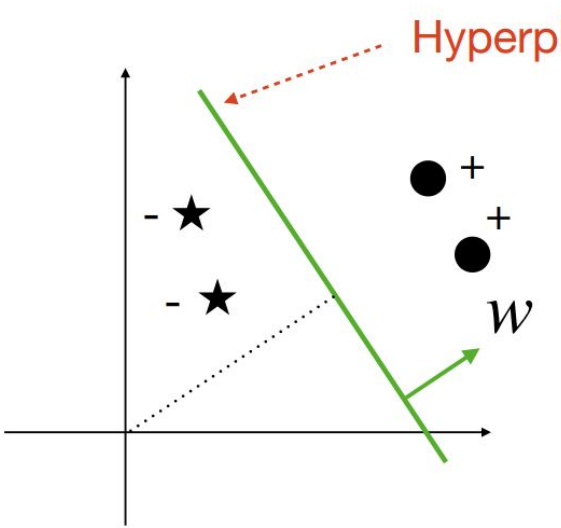


Question: Is it better or worse for the perceptron to run in higher dimensions?

Slide Credit: Cornell CS4780

Perceptron

Binary classification setting: $x \in \mathbb{R}^d, y = \{-1, +1\}$



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

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

Slide Credit: Cornell CS4780

Perceptron

McCulloch Pitts Neuron

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

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

Perceptron

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

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

- 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
- 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)

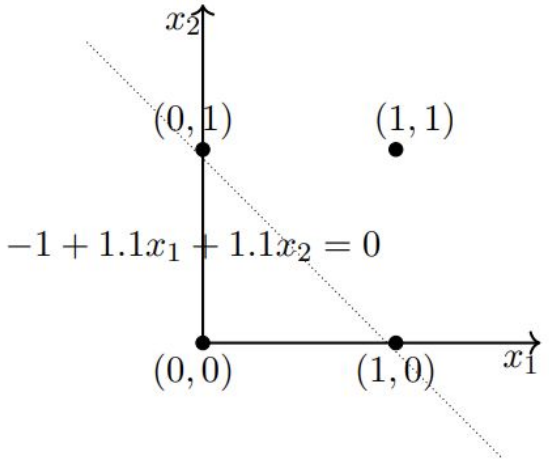
Slide Credit: CS7015 - Mitesh Khapra

Perceptron

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

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

- 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)



- 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!)

Slide Credit: CS7015 - Mitesh Khapra

Errors

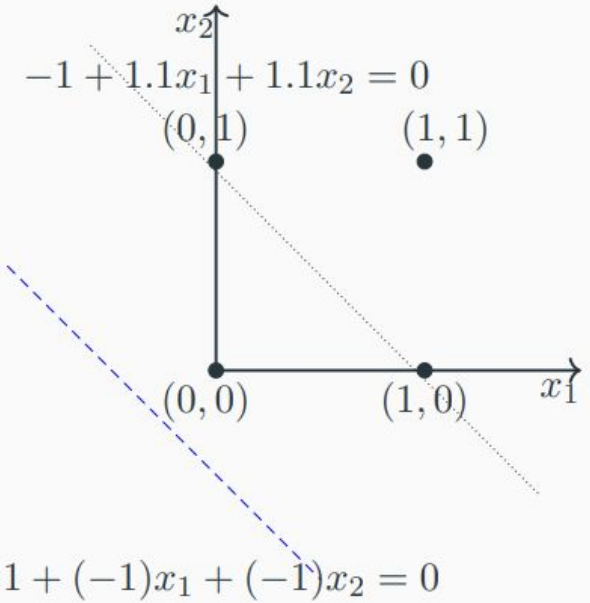
Let us fix the threshold ($-w_0 = 1$) and try different values of w_1, w_2

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



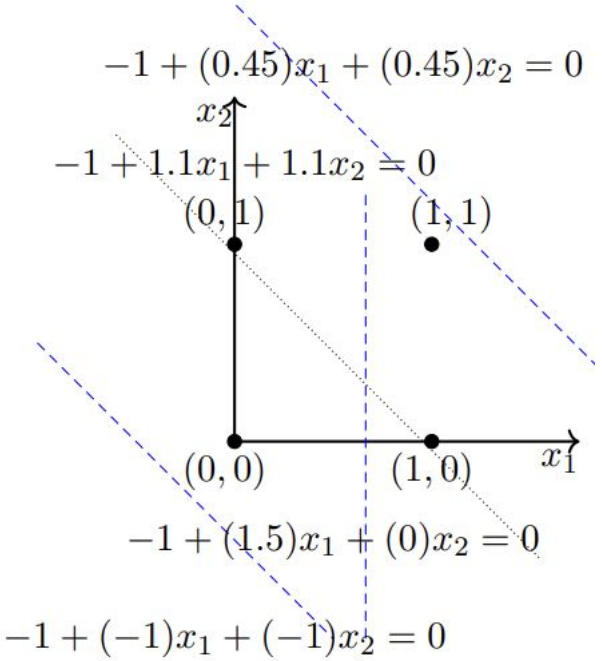
Slide Credit: CS7015 - Mitesh Khapra

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

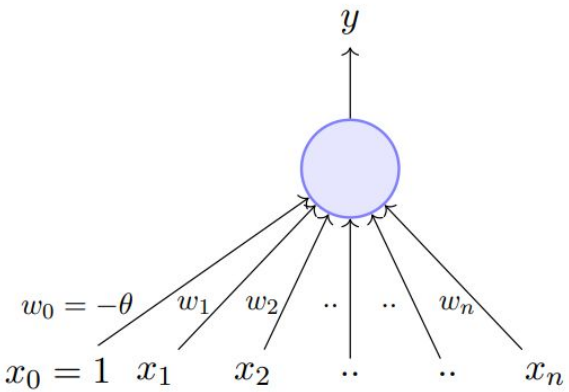
w_1	w_2	errors
-1	-1	3
1.5	0	1
0.45	0.45	3

- 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



Slide Credit: CS7015 - Mitesh Khapra

Perceptron Learning Algorithm



$x_1 = isActorDamon$

$x_2 = isGenreThriller$

$x_3 = isDirectorNolan$

$x_4 = imdbRating (scaled\ to\ 0\ to\ 1)$

... ...

$x_n = criticsRating (scaled\ to\ 0\ to\ 1)$

- 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 $w_0, w_1, w_2, .., w_m$)

Slide Credit: CS7015 - Mitesh Khapra

Perceptron Learning Algorithm

Algorithm: Perceptron Learning Algorithm

$P \leftarrow$ *inputs with label* 1;
$N \leftarrow$ *inputs with label* 0;
Initialize **w** randomly;
while !*convergence* **do**
  Pick random $\mathbf{x} \in P \cup N$;
  if $\mathbf{x} \in P$ *and* $\sum_{i=0}^{n} w_i * x_i < 0$ **then**
    $\mathbf{w} = \mathbf{w} + \mathbf{x}$;
  end
  if $\mathbf{x} \in N$ *and* $\sum_{i=0}^{n} w_i * x_i \geq 0$ **then**
    $\mathbf{w} = \mathbf{w} - \mathbf{x}$;
  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...

Slide Credit: CS7015 - Mitesh Khapra

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.

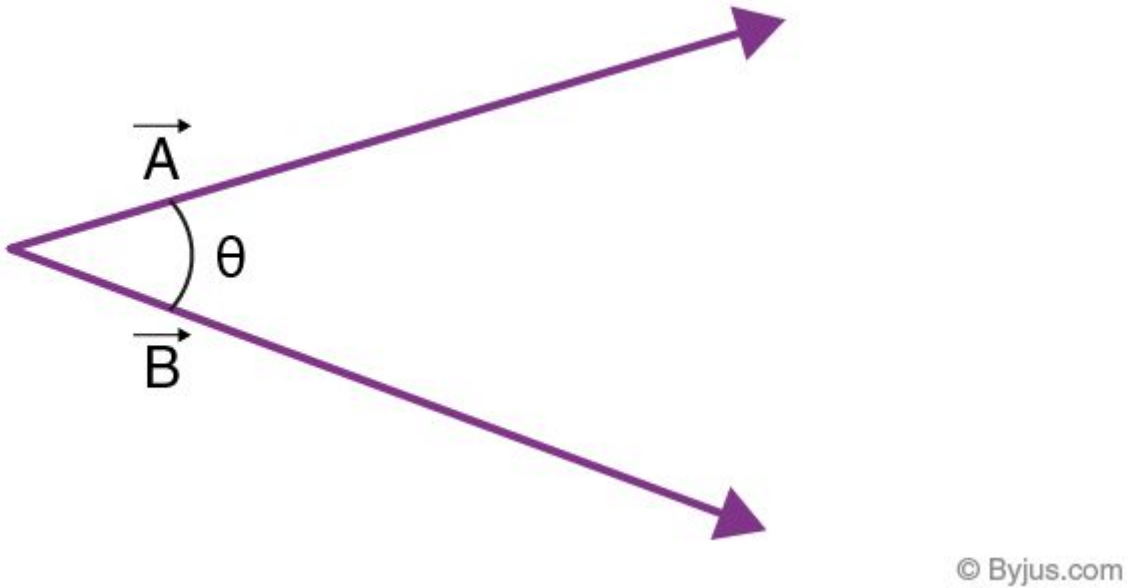
- This number can provide information about the angle between the two vectors or the magnitude of one vector in the direction of the other.

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.

Angle between two vectors



Slide Credit: CS7015 - Mitesh Khapra

Angle between two vectors

Angle Between Two Vectors

The angle between vectors ***u*** and ***v*** can be defined by

$$\cos\theta = \frac{u \bullet v}{\|u\|\|v\|}$$

The vectors are parallel if ***u*** and ***v*** are scalar multiples.

The vectors are perpendicular if $u \bullet v = 0$

Slide Credit: CS7015 - Mitesh Khapra

Perceptron Learning Algorithm

Algorithm: Perceptron Learning Algorithm

$P \leftarrow$ *inputs with label* 1;
$N \leftarrow$ *inputs with label* 0;
Initialize $\mathbf{w}$ randomly;
while !*convergence* **do**
  Pick random $\mathbf{x} \in P \cup N$;
  if $\mathbf{x} \in P$ *and* $\mathbf{w}.\mathbf{x} < 0$ **then**
    $\mathbf{w} = \mathbf{w} + \mathbf{x}$;
  end
  if $\mathbf{x} \in N$ *and* $\mathbf{w}.\mathbf{x} \geq 0$ **then**
    $\mathbf{w} = \mathbf{w} - \mathbf{x}$;
  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 $\mathbf{w}.\mathbf{x} \geq 0$ 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

Slide Credit: CS7015 - Mitesh Khapra

Perceptron Learning Algorithm

Algorithm: Perceptron Learning Algorithm

$P \leftarrow inputs \quad with \quad label \quad 1;$

Angles (in Degrees)	0°	30°	45°	60°	90°	180°	270°
Angles (in Radians)	0	π/6	π/4	π/3	π/2	π	3π/2
sin	0	1/2	1/√2	√3/2	-1	0	-1
cos	1	√3/2	1/√2	1/2	0	-1	0

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

end

if $\mathbf{x} \in N \quad and \quad \mathbf{w}.\mathbf{x} \geq 0$ **then**

$\mathbf{w} = \mathbf{w} - \mathbf{x}$;

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 $\mathbf{w}.\mathbf{x} \geq 0$ 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 $\mathbf{w_{new}} = \mathbf{w} - \mathbf{x}$

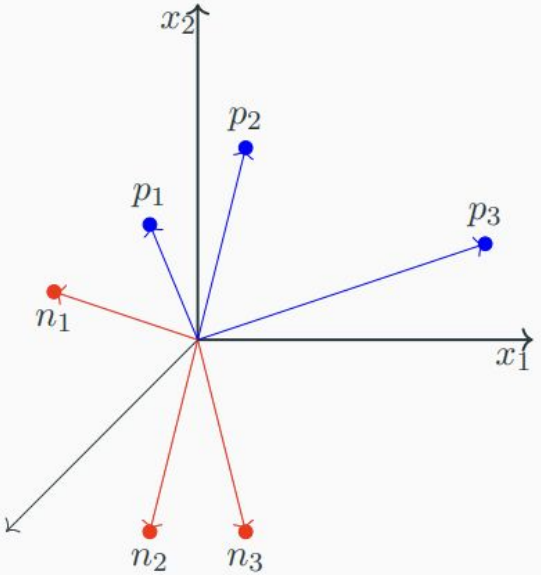
$$\begin{aligned} cos(\alpha_{new}) &\propto \mathbf{w_{new}}^T\mathbf{x} \\ &\propto (\mathbf{w} - \mathbf{x})^T\mathbf{x} \\ &\propto \mathbf{w}^T\mathbf{x} - \mathbf{x}^T\mathbf{x} \\ &\propto cos\alpha - \mathbf{x}^T\mathbf{x} \end{aligned}$$

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

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

Slide Credit: CS7015 - Mitesh Khapra

Perceptron Learning Algorithm



We initialized $\mathbf{w}$ to a random value

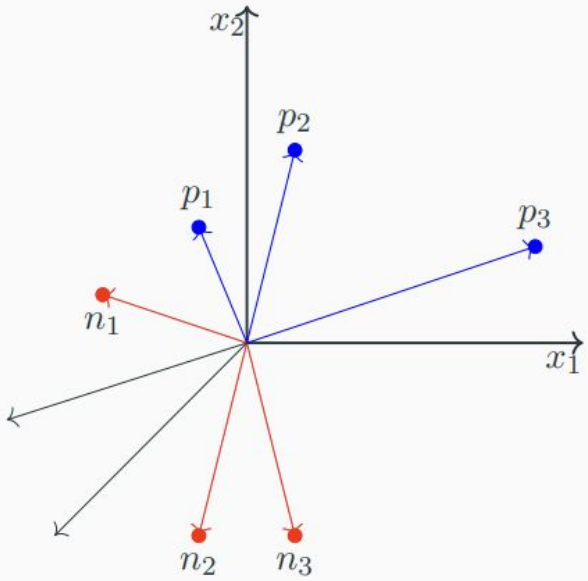
We observe that currently, $\mathbf{w} \cdot \mathbf{x} < 0$ ($\because$ angle $>$ 90°) for all the positive points and $\mathbf{w} \cdot \mathbf{x} \geq 0$ ($\because$ angle $<$ 90°) for all the negative points (the situation is exactly oppsite of what we actually want it to be)

We now run the algorithm by randomly going over the points

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

Slide Credit: CS7015 - Mitesh Khapra

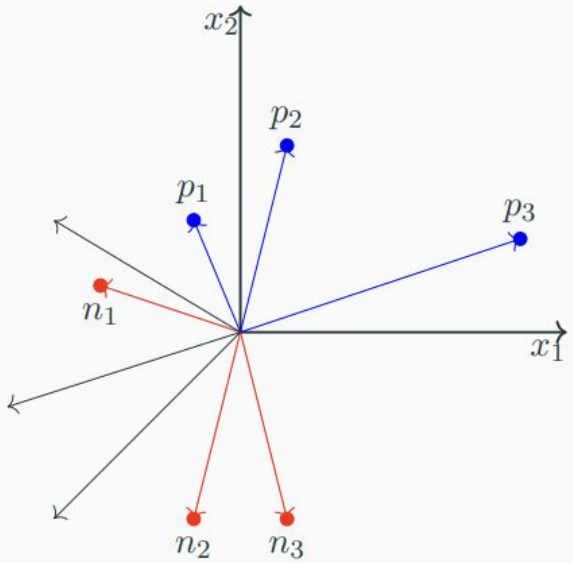
Perceptron Learning Algorithm



- We initialized $\mathbf{w}$ to a random value
- We observe that currently, $\mathbf{w} \cdot \mathbf{x} < 0$ ($\because$ angle $>$ 90°) for all the positive points and $\mathbf{w} \cdot \mathbf{x} \geq 0$ ($\because$ angle $<$ 90°) for all the negative points (the situation is exactly oppsite 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 $\mathbf{w} = \mathbf{w} + \mathbf{x}$ $\because \mathbf{w} \cdot \mathbf{x} < \mathbf{0}$ (you can check the angle visually)

Slide Credit: CS7015 - Mitesh Khapra

Perceptron Learning Algorithm



We initialized $\mathbf{w}$ to a random value

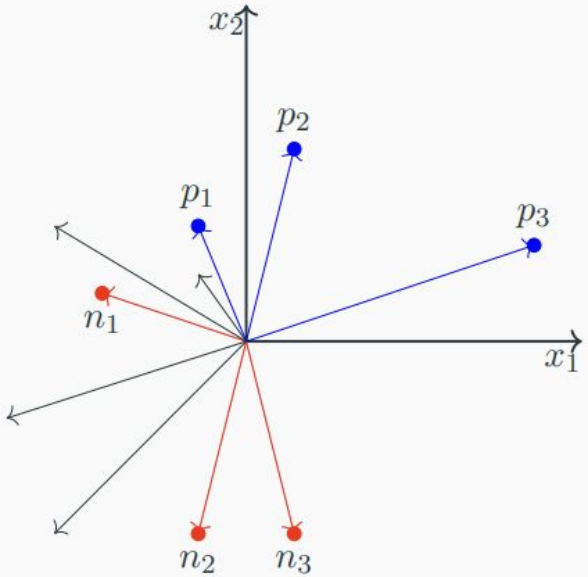
We observe that currently, $\mathbf{w} \cdot \mathbf{x} < 0$ ($\because$ angle $>$ 90°) for all the positive points and $\mathbf{w} \cdot \mathbf{x} \geq 0$ ($\because$ angle $<$ 90°) for all the negative points (the situation is exactly oppsite of what we actually want it to be)

We now run the algorithm by randomly going over the points

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

Slide Credit: CS7015 - Mitesh Khapra

Perceptron Learning Algorithm



We initialized $\mathbf{w}$ to a random value

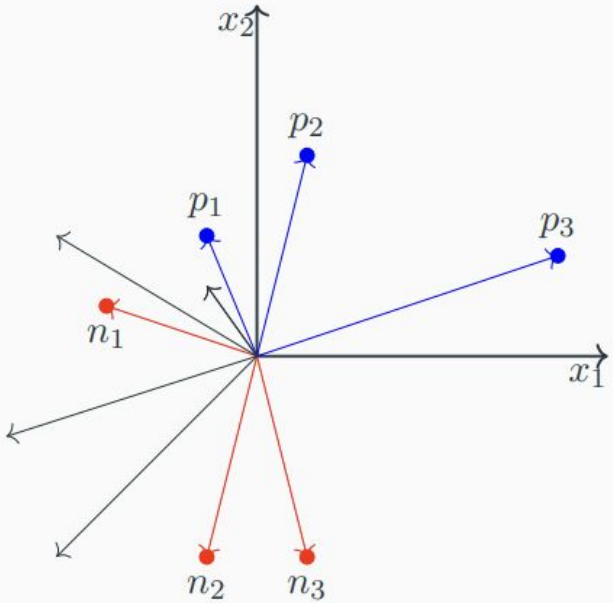
We observe that currently, $\mathbf{w} \cdot \mathbf{x} < 0$ ($\because$ angle $>$ 90°) for all the positive points and $\mathbf{w} \cdot \mathbf{x} \geq 0$ ($\because$ angle $<$ 90°) for all the negative points (the situation is exactly oppsite of what we actually want it to be)

We now run the algorithm by randomly going over the points

Randomly pick a point (say, n_1), apply correction $\mathbf{w} = \mathbf{w} - \mathbf{x}$ $\because \mathbf{w} \cdot \mathbf{x} \geq \mathbf{0}$ (you can check the angle visually)

Slide Credit: CS7015 - Mitesh Khapra

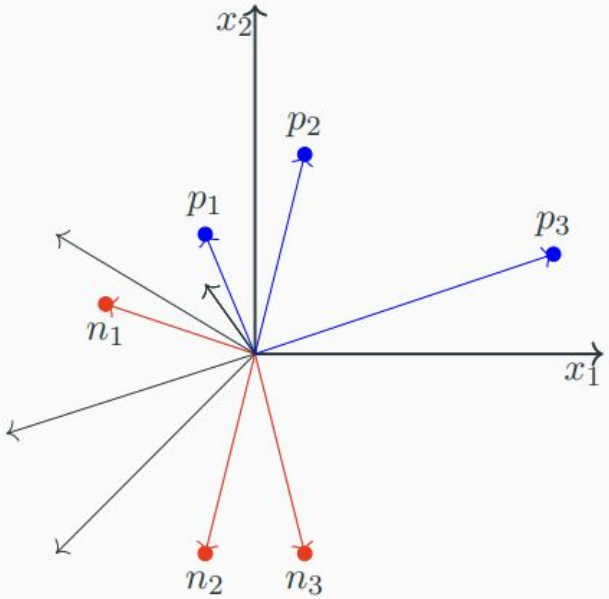
Perceptron Learning Algorithm



- We initialized $\mathbf{w}$ to a random value
- We observe that currently, $\mathbf{w} \cdot \mathbf{x} < 0$ ($\because$ angle $>$ 90°) for all the positive points and $\mathbf{w} \cdot \mathbf{x} \geq 0$ ($\because$ angle $<$ 90°) for all the negative points (the situation is exactly oppsite 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 $\because$ $\mathbf{w} \cdot \mathbf{x} < \mathbf{0}$ (you can check the angle visually)

Slide Credit: CS7015 - Mitesh Khapra

Perceptron Learning Algorithm



We initialized $\mathbf{w}$ to a random value

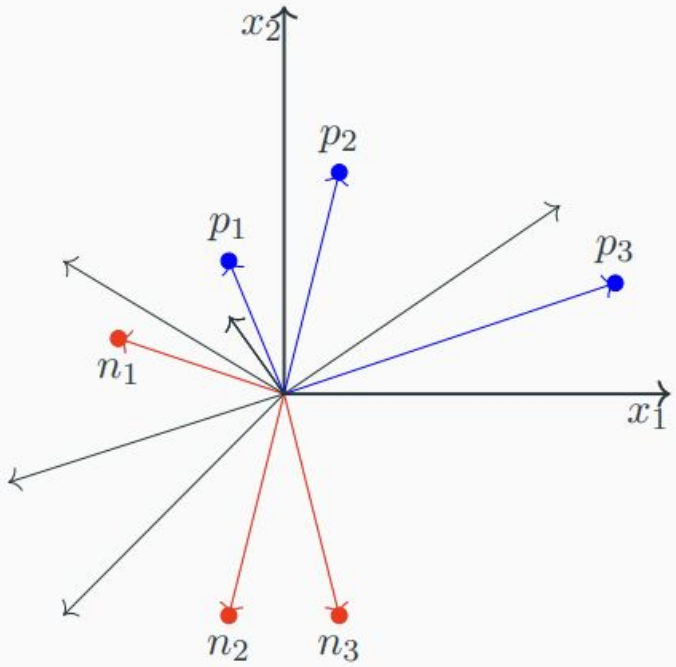
We observe that currently, $\mathbf{w} \cdot \mathbf{x} < 0$ ($\because$ angle $>$ 90°) for all the positive points and $\mathbf{w} \cdot \mathbf{x} \geq 0$ ($\because$ angle $<$ 90°) for all the negative points (the situation is exactly oppsite of what we actually want it to be)

We now run the algorithm by randomly going over the points

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

Slide Credit: CS7015 - Mitesh Khapra

Perceptron Learning Algorithm



- We initialized $\mathbf{w}$ to a random value
- We observe that currently, $\mathbf{w} \cdot \mathbf{x} < 0$ ($\because$ angle $>$ 90°) for all the positive points and $\mathbf{w} \cdot \mathbf{x} \geq 0$ ($\because$ angle $<$ 90°) for all the negative points (the situation is exactly oppsite 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 $\mathbf{w} = \mathbf{w} + \mathbf{x}$ $\because \mathbf{w} \cdot \mathbf{x} < \mathbf{0}$ (you can check the angle visually)

Slide Credit: CS7015 - Mitesh Khapra

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 $w_0, w_1, ..., w_n$ exist such that every point $(x_1, x_2, ..., x_n) \in P$ satisfies $\sum_{i=1}^{n} w_i * x_i > w_0$ and every point $(x_1, x_2, ..., x_n) \in N$ satisfies $\sum_{i=1}^{n} w_i * x_i < w_0$.

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.

Proof: On the next slide

Slide Credit: CS7015 - Mitesh Khapra

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 $\because if \quad w^T \frac{p}{||p||} \geq 0$ then $w^T p \geq 0$)
- Let 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;
$P' \leftarrow P \cup N^-$;
Initialize **w** randomly;
while !*convergence* **do**
    Pick random $\mathbf{p} \in P'$;
    $\mathbf{p} \leftarrow \frac{p}{||p||}$ (*so now*, $||p|| = 1$) ;
    if $\mathbf{w}.\mathbf{p} < 0$ **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 $\mathbf{w}.\mathbf{p} \geq 0$

Slide Credit: CS7015 - Mitesh Khapra

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 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 numer-

Proof:

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

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

$$\begin{aligned} Numerator &= w^* \cdot w_{t+1} = w^* \cdot (w_t + p_i) \\ &= w^* \cdot w_t + w^* \cdot p_i \\ &\geq w^* \cdot w_t + \delta \quad (\delta = min\{w^* \cdot p_i | \forall i\} \\ &\geq w^* \cdot (w_{t-1} + p_j) + \delta \\ &\geq w^* \cdot w_{t-1} + w^* \cdot p_j + \delta \\ &\geq w^* \cdot w_{t-1} + 2\delta \\ &\geq w^* \cdot w_0 + (k)\delta \quad (By\ induction) \end{aligned}$$

Slide Credit: CS7015 - Mitesh Khapra

Proof of Convergence

Proof (continued:)

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

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

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

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

Slide Credit: CS7015 - Mitesh Khapra

Proof of Convergence

Proof (continued:)

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

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

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

$$Denominator^2 \leq ||w_0||^2 + k \quad (By\ same\ observation\ that\ we\ made\ about\ \delta)$$

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

- $cos\beta$ 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!

Slide Credit: CS7015 - Mitesh Khapra

Proof of Convergence

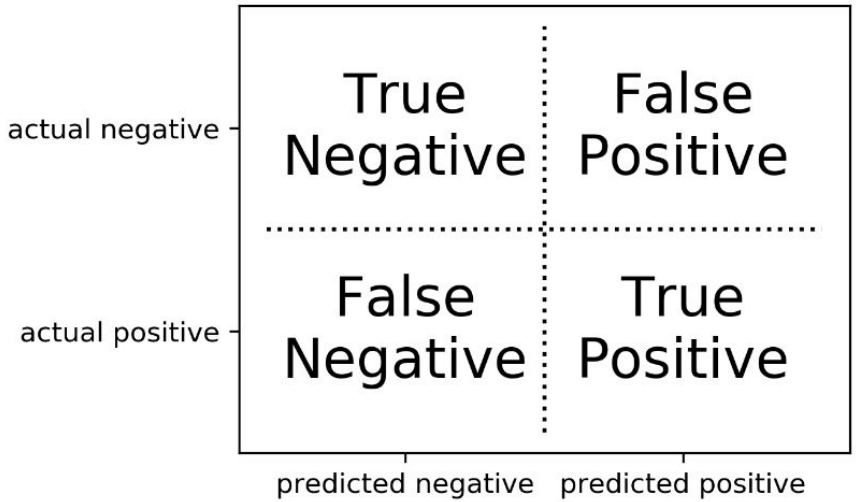
Coming back to our questions ...

- 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 ..

Slide Credit: CS7015 - Mitesh Khapra

Evaluation Metrics for Binary Classification

Evaluation metrics for binary classification



$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

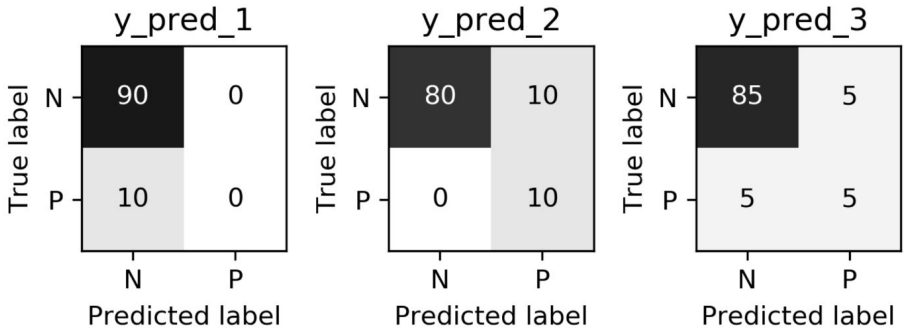
Slide Credit: Amuller, COMS4995

Problems with Accuracy

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
```



Slide Credit: Amuller, COMS4995

Evaluation Metrics for Classification

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

Positive Predicted Value (PPV)

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

Sensitivity, coverage, true positive rate.

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

Harmonic mean of precision and recall

Slide Credit: Amuller, COMS4995

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
```

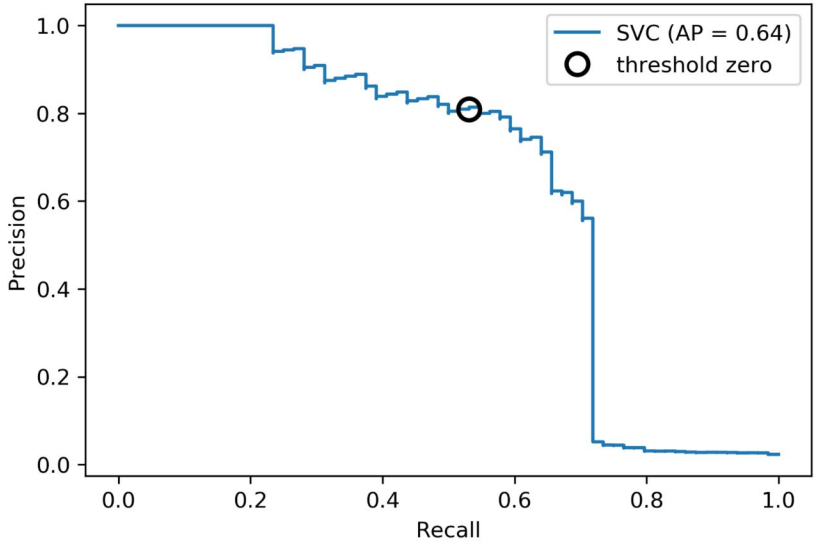
```
y_pred = rf.predict_proba(X_test)[:, 1] > .30


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
```

Slide Credit: Amuller, COMS4995

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')
```

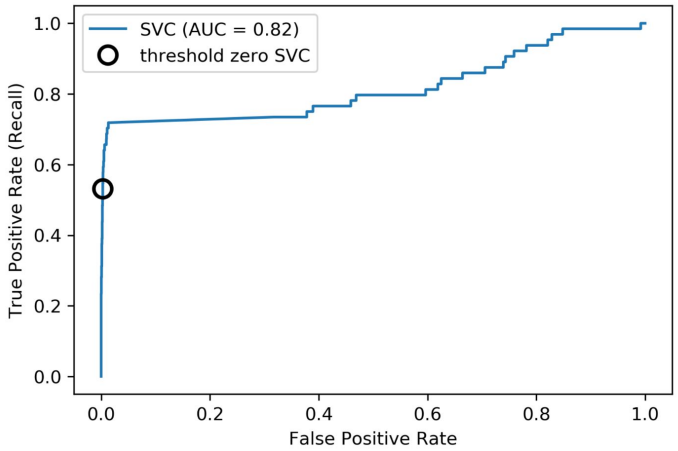


Slide Credit: Amuller, COMS4995

ROC Curve

(Receiver Operating Characteristic)

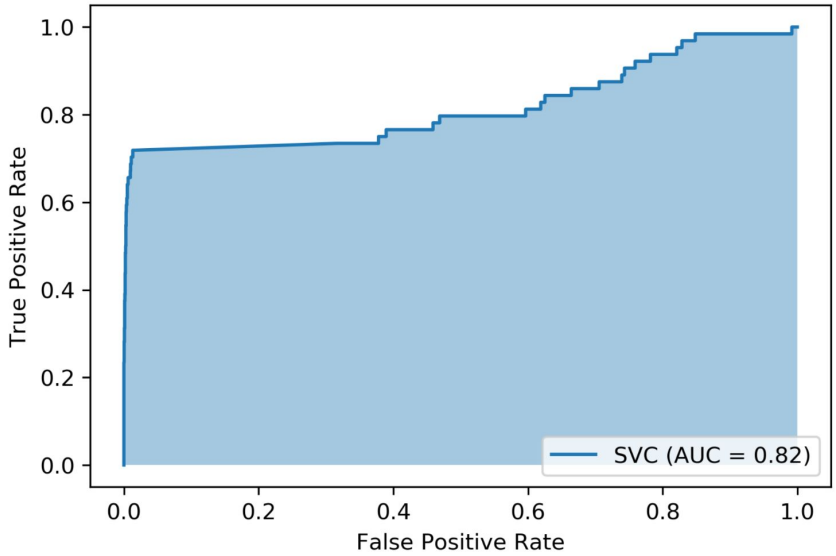
```
plot_roc_curve(svc, X_test, y_test, name='SVC')
```



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

Slide Credit: Amuller, COMS4995

Area Under ROC Curve (AUC)



- Always .5 for random/constant prediction

Slide Credit: Amuller, COMS4995

Summary of metrics for binary classification

- Threshold-based
  - (balanced) accuracy
  - precision , recall, f1

- Ranking
  - Average precision
  - ROC AUC

Slide Credit: Amuller, COMS4995

Picking metrics

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

Slide Credit: Amuller, COMS4995

Code Example

- 6-Evaluation Metrics.ipynb