Perceptron

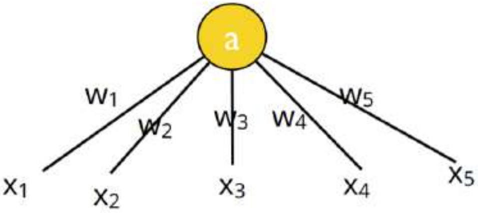
UNIVERSITY OF LIVERPOOL

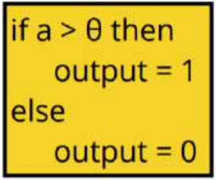
Bio-inspired model

- Perceptron is a bio-inspired algorithm that tries to mimic a single neuron.
- We simply multiply each input (feature) by a weight and check whether this weighted sum (activation) is greater than a threshold.
- If so, then we “fire” the neuron (i.e., a decision is made based on the activation).

A Single neuron

activation (score) = $a = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5$





If the activation is greater than a predefined threshold, then the neuron fires.

Bias

- Often we need to adjust a fixed *shift* from zero, if the “interesting” region happens to be far from the origin.
- We adjust the previous model by including a bias term b as follows:

$$a = b + \sum_{i=1}^{D} w_d x_d$$

Notational trick

- By introducing a feature that is always ON (i.e., $X_0 = 1$ for all instances), we can squeeze the bias term b into the weight vector by setting $w_0 = b$

$$a = \sum_{i=1}^{D} w_d x_d = \mathbf{w}^\top \mathbf{x}$$

This is more “elegant” as we can write the activation as the inner-product between weight vector and feature vector. However, we should keep in mind that bias term still appears in the model

Perceptron

- Consider only one training instance at a time
  - online learning
  - k-NN considers ALL instances (batch learning)
- Learn only if we make a mistake when we classify using the current weight vector. Otherwise, we do not make adjustments to the weight vector
  - Error-driven learning

Perceptron

Algorithm 5 PerceptronTrain($\mathbf{D}$, *MaxIter*)

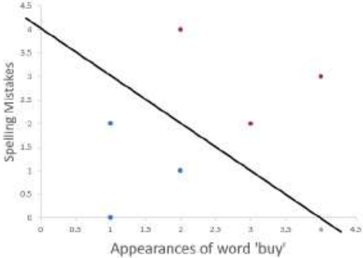
```
w_d ← 0, for all d = 1 … D                              // initialize weights
b ← 0                                                    // initialize bias
for iter = 1 … MaxIter do
    for all (x,y) ∈ D do
        a ← Σ_{d=1}^{D} w_d x_d + b                      // compute activation for this example
        if ya ≤ 0 then
            w_d ← w_d + y x_d, for all d = 1 … D         // update weights
            b ← b + y                                    // update bias
        end if
    end for
end for
return w_0, w_1, …, w_D, b
```

Algorithm 6 PerceptronTest($w_0, w_1, \ldots, w_D, b, \hat{x}$)

```
a ← Σ_{d=1}^{D} w_d x̂_d + b                              // compute activation for the test example
return SIGN(a)
```

Example

Spelling Mistakes	Appearances of word 'buy'	Spam or Ham
x_1	x_2	y
3	2	-1
2	1	1
4	3	-1
1	0	1
1	2	1
2	4	-1



Update Weights

	x_1	x_2	y	w_1	w_2	b	a	ya
				0	0	0		
1	3	2	-1	-3	-2	-1	0	0

1. initialise weight vector w_d $(w_1 = w_2 = 0)$ and bias $(b = 0)$
2. compute activation :

$$a = \sum_{i=0}^{D} w_d x_d + b \implies w_1.x_1 + w_2.x_2 + b \implies a = 0, ya = 0$$

3. since $ya \leq 0$, update weight vector and bias

$$w_d \leftarrow w_d + yx_d \text{ , thus } w_1 = w_1 - y.x_1 = -3$$

4. similarly, we get $w_2 = -2$ and $b = b + y = -1$

Update Weights

	x_1	x_2	y	w_1	w_2	b	a	ya
				0	0	0		
1	3	2	-1	-3	-2	-1	0	0
2	2	1	1	-1	-1	0	-9	-9
3	4	3	-1				-7	7

Second training example:

1. we use updated weights from first training example
2. we get $a = -9$ and $ya = -9$ (because $y = 1$)
3. since $ya \leq 0$, update parameters

Third training example:

1. we use updated weights from the second training example
2. we get $a = -7$ and $ya = 7$ (because $y = -1$)
3. since $ya > 0$, we don't update and go to next instance.

Detected Errors

	x_1	x_2	y	w_1	w_2	b	a	ya
				0	0	0		
1	3	2	-1	-3	-2	-1	0	0
2	2	1	1	-1	-1	0	-9	-9
3	4	3	-1				-7	7

- In Line 6 of Perceptron Train code, we have:
  - $ya \leq 0$
  - if the current instance is positive ($y = +1$), we need a positive activation ($a > 0$) to have a correct prediction
  - if the current instance is negative ($y = -1$), we need a negative activation ($a < 0$) to have a correct prediction
  - in both cases $ya > 0$
  - therefore, if $ya \leq 0$, we have a misclassification

Update Rule - Intuitive Explanation

	x_1	x_2	y	w_1	w_2	b	a	ya
				0	0	0		
1	3	2	-1	-3	-2	-1	0	0
2	2	1	1	-1	-1	0	-9	-9
3	4	3	-1				-7	7

- Perceptron update rule is
  - $\mathbf{w} = \mathbf{w} + y\mathbf{x}$
- If we incorrectly classify a positive instance as negative
  - We should have a higher (more positive) activation to avoid this
  - We should increase $\mathbf{w}^T\mathbf{x}$
  - Therefore, we should ADD the current instance to the weight vector

Update Rule - Intuitive Explanation

	x_1	x_2	y	w_1	w_2	b	a	ya
				0	0	0		
1	3	2	-1	-3	-2	-1	0	0
2	2	1	1	-1	-1	0	-9	-9
3	4	3	-1				-7	7

- If we incorrectly classify a negative instance as positive
  - We should have a lower (more negative) activation to avoid this
  - We should decrease $\mathbf{w}^\top \mathbf{x}$
  - Therefore, we should DEDUCT the current instance from the weight vector

Update rule - Math Explanation

- Let's look at a misclassified positive example ($y_n = +1$)
  - Perceptron (wrongly) thinks $\mathbf{w}_{old}^\top \mathbf{x}_n + b_{old} < 0$
- Updates would be:
  - $\mathbf{w}_{new} = \mathbf{w}_{old} + y_n \mathbf{x}_n = \mathbf{w}_{old} + \mathbf{x}_n$ (since $y_n = +1$)
  - $b_{new} = b_{old} + y_n = b_{old} + 1$

$$\begin{aligned}\mathbf{w}_{new}^\top \mathbf{x}_n + b_{new} &= (\mathbf{w}_{old} + \mathbf{x}_n)^\top \mathbf{x}_n + b_{old} + 1 \\ &= (\mathbf{w}_{old}^\top \mathbf{x}_n + b_{old}) + \mathbf{x}_n^\top \mathbf{x}_n + 1\end{aligned}$$

- Thus $\mathbf{w}_{new}^\top \mathbf{x}_n + b_{new}$ is less negative than $\mathbf{w}_{old}^\top \mathbf{x}_n + b_{old}$
  - we are making ourselves more correct on this example.

Update rule - Math Explanation

- Let's look at a misclassified negative example ($y_n = -1$)
  - Perceptron (wrongly) thinks $\mathbf{w}_{old}^\top \mathbf{x}_n + b_{old} > 0$
- Updates would be:
  - $\mathbf{w}_{new} = \mathbf{w}_{old} + y_n \mathbf{x}_n = \mathbf{w}_{old} - \mathbf{x}_n$ (since $y_n = -1$)
  - $b_{new} = b_{old} + y_n = b_{old} - 1$

$$\begin{aligned}\mathbf{w}_{new}^\top \mathbf{x}_n + b_{new} &= (\mathbf{w}_{old} - \mathbf{x}_n)^\top \mathbf{x}_n + b_{old} - 1 \\ &= (\mathbf{w}_{old}^\top \mathbf{x}_n + b_{old}) - \mathbf{x}_n^\top \mathbf{x}_n - 1\end{aligned}$$

- Thus $\mathbf{w}_{new}^\top \mathbf{x}_n + b_{new}$ is less positive than $\mathbf{w}_{old}^\top \mathbf{x}_n + b_{old}$
  - we are making ourselves more correct on this example.

Things to Remember

- There is no guarantee that we will correctly classify a misclassified instance in the next round.
- We have simply increased/decreased the activation but this adjustment might not be sufficient. We might need to do more aggressive adjustments
- There are algorithms that enforce such requirements explicitly such as the Passive Aggressive Classifier (not discussed here)

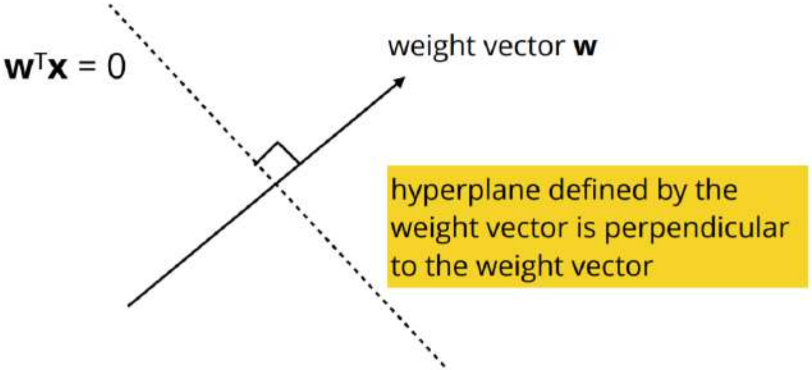
Ordering of Instances

- Ordering training instances randomly within each iteration produces good results in practice.
- Showing only all the positives first and all the negatives next is a bad idea.

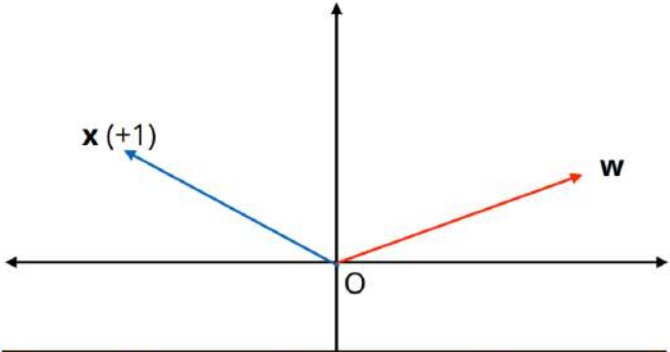
Hyperplane

- The decision in perceptron is made depending on $\mathbf{w}^T\mathbf{x} > 0$ or $\mathbf{w}^T\mathbf{x} <= 0$
- Therefore, $\mathbf{w}^T\mathbf{x} = 0$ is the critical region (decision boundary)
- $\mathbf{w}^T\mathbf{x} = 0$ defines a hyperplane
- Example:
  - In 2D space we have $w_1x_1 + w_2x_2 = 0$ (ignoring the bias term), which is a straight line through the origin.
  - In N dimensional space this is an (N-1) dimensional hyperplane

Geometric Interpretation of Hyperplane

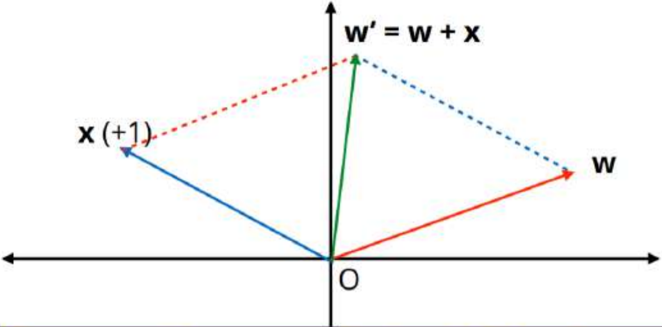


Geometric Interpretation



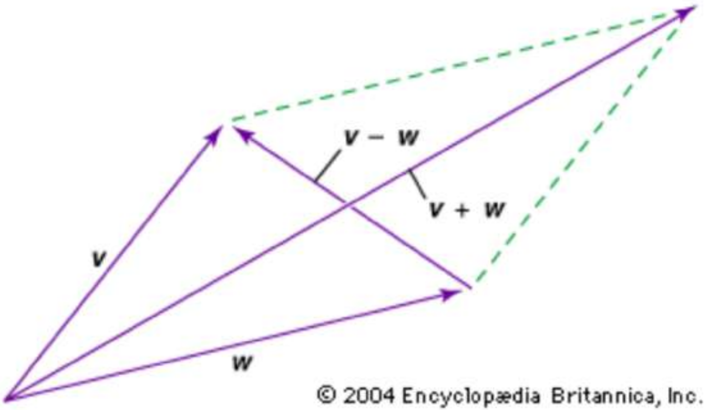
The angle between the current weight vector **w** and the positive instance **x** is greater than 90º. Therefore, $\mathbf{w}^T\mathbf{x} < 0$, and this instance is going to get misclassified as negative.

Geometric Interpretation



The new weight vector **w′** is the addition of **w** + **x** according to the perceptron update rule. It lies in between **x** and **w**. Notice that the angle between **w′** and **x** is less than 90º. Therefore, **x** will be classified as positive by **w′**.

Vector algebra revision



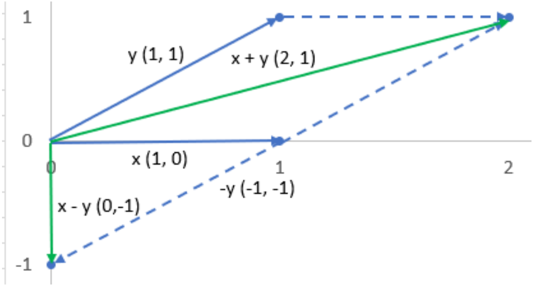
Quiz 1

- Let $\mathbf{x} = (1, 0)^\top$ and $\mathbf{y} = (1, 1)^\top$. compute $\mathbf{x} + \mathbf{y}$ and $\mathbf{x} - \mathbf{y}$ using the parallelogram approach described in the previous slide.

Quiz 1

- Let $\mathbf{x} = (1, 0)^\top$ and $\mathbf{y} = (1, 1)^\top$. compute $\mathbf{x} + \mathbf{y}$ and $\mathbf{x} - \mathbf{y}$ using the parallelogram approach described in the previous slide.

  $\mathbf{x} + \mathbf{y} = (2, 1); \mathbf{x} - \mathbf{y} = (0, -1)$

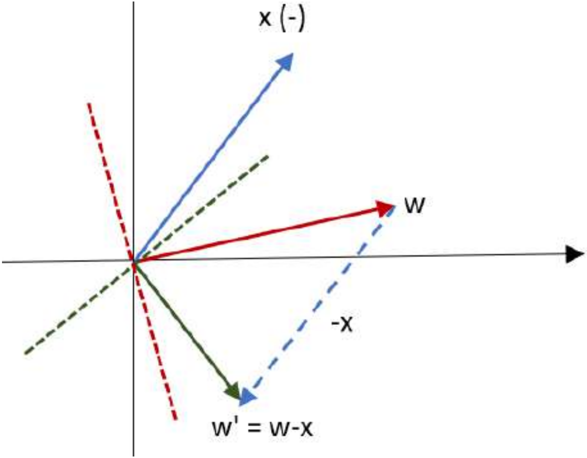


Quiz 2

- Provide a geometric interpretation for the update rule in Perceptron when a negative instance is mistaken to be positive.

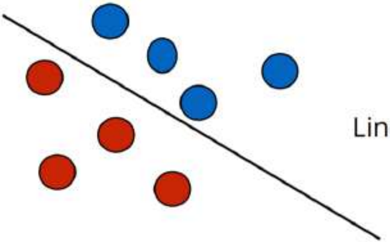
Quiz 2

- Provide a geometric interpretation for the update rule in Perceptron when a negative instance is mistaken to be positive.



Linear separability

- If a given set of positive and negative training instances can be separated into those two groups using a straight line (hyperplane), then we say that the dataset is *linearly separable*.



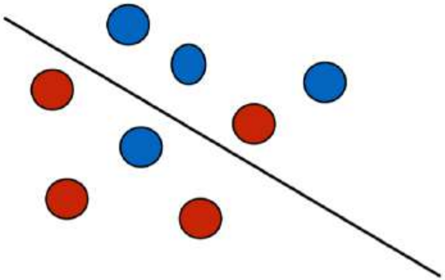
Linearly separable

Remarks

- When a dataset is linearly separable, there can exist more than one hyperplanes that separates the dataset into positive/negative groups.
- In other words, the hyperplane that linearly separates a linearly separable dataset might not be unique.
- However, (by definition) if a dataset is nonlinearly separable, then there exist NO hyperplane that separates the dataset into positive/negative groups.

A non-linearly separable case

No matter how we draw straight lines, we cannot separate the red instances from the blue instances



Further Remarks

- When a dataset is linearly separable it can be proved that the Perceptron will always find a separating hyperplane!
- The final weight vector returned by the Perceptron is more influenced by the final training instances it sees.
    - Take the average over all weight vectors during the training (averaged Perceptron algorithm)