The Perceptron

Eitan Kosman
The Technion
Department of Computer Science

The perceptron is an algorithm designed to solve the problem of learning binary classifiers. Bach in time, the Perceptron was considered a very powerfull tool since it could automatically learn a predictor function for any two linearly separable sets.

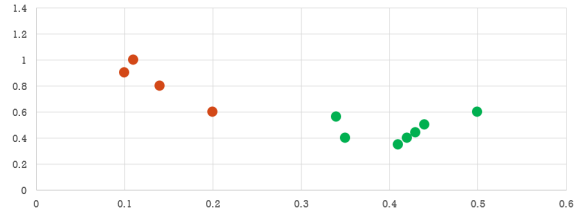
The Problem

Before showing the Perceptron algorithm, I would like to shortly introduce the problem we try to solve. Given a set of n points in $\mathbb{R}^d$ with binary labels:

$$X = \{x_i | i \in [n], x \in \mathbb{R}^d\}$$

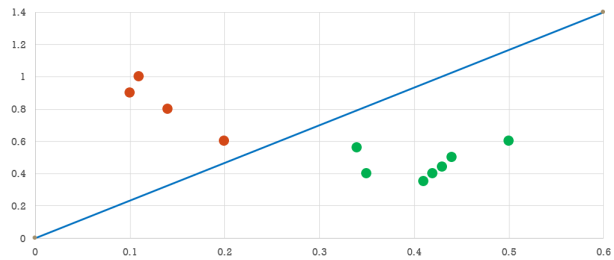
$$Y = \{y_i | i \in [n], y_i \in \{-1, 1\}\}$$

we want to find a transformation $f : X \rightarrow Y$ so that $x_i \mapsto_f y_i$. In the following picture you can see an example for this problem where X is the set of the drawn points in $\mathbb{R}^2$ and the two colors represent two different classes.



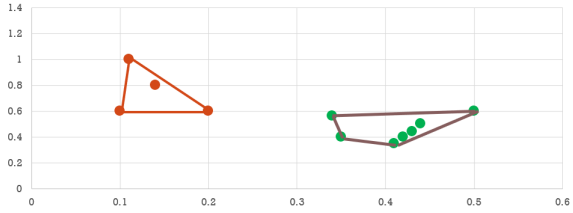
Definition (1) - Linear Separability

Let $X_0, X_1 \subseteq \mathbb{R}^d$ be two sets of points. X_0, X_1 are linearly separable if there exist $n + 1$ real numbers $w_1, w_2, ..., w_n, k$ such that:$\forall x \in X_0 : \sum_{i=1}^{n} w_i x_i > k$, $\forall x \in X_1 : \sum_{i=1}^{n} w_i x_i < k$. These terms could also be written as the inner product $\langle w, x \rangle$ where $w = (w_1, w_2, ..., w_n)$ and $x = (x_1, x_2, ..., x_n)$. This way, we can treat it as a hyper-plane: $\langle w, x \rangle - k = 0$ that separates the vector space intro two regions such that all points belong to X_0 are in one region and all points belong to X_1 are in the other region. In $\mathbb{R}^2$, this hyper-place would be a straight line that for our example would look like the blue line in the following picture:



Definition (2) - Linear Separability

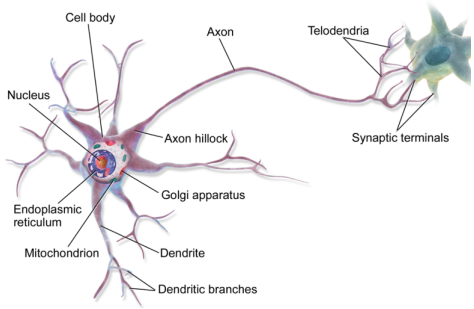
Let $X_0, X_1 \subseteq \mathbb{R}^d$ be two sets of points. X_0, X_1 are linearly separable precisely when their repective convex hulls are disjoint (do not overlap)



The Bioligical Neuron

Structure

Like any other body cell, the neuron has a cell body which contains a nucleus where the DNA is stored. From our perspective, the interesting parts are: Dendrites – make connections with tens of thousand of other cells; other neurons. The behave as "inputs". Axon – transmits information to different neurons, muscles, and other body cells based on the signals the cell receives. It's signals are received by other cells' dendrites.



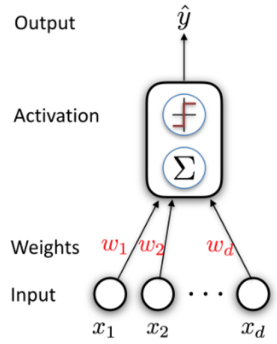
Mathematical model of the neuron cel

Mathematicians try to model various phenomenas that happen in our world. For the neuron, the model suggested for the neuron is that given an input vector x:

- x will be the inputs of the neuron (dendrites)

- Define a weight, w_i, for each input x_i, and sum all the multiplications.
- Output the result at $\hat{y}$ (Axon)

This model scheme is shown more clearly in the following picture, However, there's still a problem - How do we find the weights?



The Solution

Prehistory

In 1943, W.S. McCulloch & W. Pitts published their article: "A logical calculus of the ideas immanent in nervous activity". This paper pointed out that simple artificial "neurons" could be made to perform basic logical operations such as AND, OR and NOT. The citation from the abstract shows that notes that clearly: Because of the "all-or-none" character of nervous activity, neural events and the relations among them can be treated by means of propositional logic". They also attempted to demonstrate that a Turing machine program could be implemented in a finite network of formal neurons, the base logic unit of the brain. Another achievement is showing that any complex and dynamic neural network with delays (that occur when transferring signals with chemicals thru the synapse, or because of poor conductivity of the axon) has an equivalent network of logical units which computes the same function.

In this context, we want to represent the logical units by the mathematical model proposed for the neuron. A simple solution for it is:

$$OR(x,y) = \begin{cases} 0 & x+y-1<0 \\ 1 & \text{otherwise} \end{cases}$$

$$AND(x,y) = \begin{cases} 0 & x+y-1.5<0 \\ 1 & \text{otherwise} \end{cases}$$

$$NOT(x) = -x$$

The weights for the model will be the coefficients of the variables. In the OR example, we can define $w = (1,1), k = 1$ and therefore k means "at least one input is on". for the AND example we can define $w = (1,1), k = 1.5$ and thefre "at least two inputs are on".

Eliminating k

The goal is to find a hyper-place separating two known classes. Consider definition (1) for linear separability: $\forall x \in X_0 : \langle w,x\rangle > k, \forall x \in X_1 : \langle w,x\rangle < k$. By subtracting k for both sides we get: $\forall x \in X_0 : \langle w,x\rangle - k > 0, \forall x \in X_1 : \langle w,x\rangle - k < 0$.

Important feature of those expressions is that we can eliminate k by augmenting representation with one dimension: $x' = (x,1), w' = (w,-x)$ and therefore we get: $\langle w',x'\rangle = (w,-k)\begin{pmatrix} x \\ 1 \end{pmatrix} = wx - k$. In the presentation I added a picture a baby, being happy for this solution.

The Perceptron algorithm

$P \leftarrow$ inputs with label +1 (Positive)
$N \leftarrow$inputs with label -1 (Negative)
$w \leftarrow \bar{0}$
while a stopping criterion isn't met:
  Iterate over all $(x,y) \in X \times Y$:
    $\hat{y} = sign(\langle w,x\rangle)$
    if $\hat{y} \neq y$:
      $w \leftarrow w + yx$

Building some intuition on the weights update rule

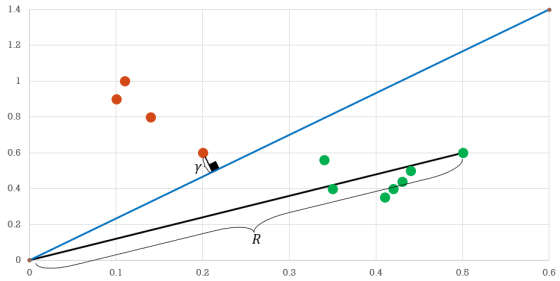
In the presentation I added an example that demonstrates how the weights update rule affects the position of the decision line. The detailed example is in the presentation and doesn't appear here. You can check the additional presentation

Perceptron's mistake bound theorm

Let $(x_1,y_1),...,(x_n,y_n)$, where $x_i \in \mathbb{R}^d$ and $y_i \in \{-1,1\}$ be a sequence of labeled examples and assume it is linearly separable, and denote: $R = max_i|x_i|$.

The set of points is linearly separable, thus there exist $w^*, \gamma > 0$ such that $|w^*| = 1, \forall i : y_i {w^*}^T x_i >= \gamma$. The number of mistakes made by the Perceptron algorithm on this sequence of examples is $O((\frac{R}{\gamma})^2)$.

Before proving the theorm, lets understand what does R and γ mean. $R = max_i|x_i|$ could be interpreted as the maximum distance of a point from the origin (or the minimum radius of the sphere that contains all the points). γ could be any positive real number between zero and the distance from the decision line to the closest point.



Proof

Let $w_0 = \bar{0}$ (initial weights vector) and denote w_k the weights vector after the $k'th$ mistake.

Lemma 1: $w_{t+1} \cdot w^* \geq w_t \cdot w^* + \gamma$

The $t's$ update occured when the perceptron made a mistake on sample (x_i, y_i).

If $y_i = 1 : w_{t+1} \cdot w^* = (w_t + x_i) \cdot w^* = w_t \cdot w^* + \overset{\geq \gamma}{x_i \cdot w^*} \geq w_t \cdot w^* + \gamma$

If $y_i = -1 : w_{t+1} \cdot w^* = (w_t - x_i) \cdot w^* = w_t \cdot w^* - \overset{\leq -\gamma}{x_i \cdot w^*} \geq w_t \cdot w^* + \gamma$

Lemma 2: $|w_{t+1}|^2 \leq |w_t|^2 + R^2$

The $t's$ update occured when the perceptron made a mistake on sample (x_i, y_i).

<0, since a mistake occured

If $y_i = 1 : |w_{t+1}|^2 = |w_t + x_i|^2 = |w_t|^2 + 2\overset{<0}{w_t \cdot x_i} + \overset{\leq R^2}{|x_i|^2} \leq |w_t|^2 + R^2$

If $y_i = -1 : |w_{t+1}|^2 = |w_t - x_i|^2 = |w_t|^2 - 2\overset{>0}{w_t \cdot x_i} + \overset{\leq R^2}{|x_i|^2} \leq |w_t|^2 + R^2$

Lets head to proving the theorm. I want to prove by induction that $w_t \cdot w^* \geq \gamma$. From Lemma 1 we know that:

$$w_0 = \bar{0}$$

$$w_1 \cdot w^* \geq w_0 \cdot w^* + \gamma = \gamma$$

$$w_2 \cdot w^* \geq w_1 \cdot w^* + \gamma \geq \gamma + \gamma = 2\gamma$$

Closure: Assume $w_{t-1} \cdot w^* \geq (t-1) \cdot \gamma$,thus $w_t \cdot w^* \geq w_t + \gamma \geq (t-1) \cdot \gamma + \gamma = t \cdot \gamma$ (*)

Also, I want to prove by induction that $|w_t|^2 \leq tR^2$. From Lemma 2 we know that:

$$|w_0| = \bar{0}$$

$$|w_1|^2 \leq |w_0|^2 + R^2 = R^2$$

$$|w_2|^2 \leq |w_1|^2 + R^2 \leq R^2 + R^2 = 2R^2$$

Closure: Assume $|w_{t-1}|^2 \leq (t-1)R^2$,thus $|w_t|^2 \leq |w_{t-1}|^2 + R^2 \leq (t-1)R^2 + R^2 = tR^2$ (**)

Recap: From (*) and (**) we know that:

$$w_t \cdot w^* \geq t \cdot \gamma$$

$$|w_{t-1}|^2 \leq tR^2$$

Therefore, after T mistakes:

$$\gamma \cdot T \leq w_{T+1} \cdot w^* \overset{scalar}{=} |w_{T+1} \cdot w^*| \underset{Cauchy-Schwartz}{\leq} |w_{T+1}| \cdot \underset{=1|}{|w^*|} = |w_{T+1}|$$

$$\Downarrow$$

$$\gamma^2 T^2 \leq |w_{T+1}|^2 \leq TR^2$$

$$\Downarrow$$

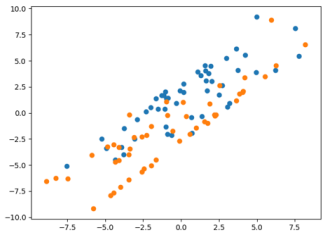
$$T \leq \frac{R^2}{\gamma^2} = O((\frac{R}{\gamma})^2)$$

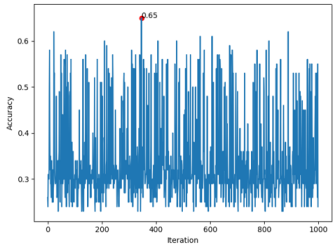
■

Stopping critertions

The mistake bound holds only if the dataset is linearly separable. If it's not the case, one could define other critertions:

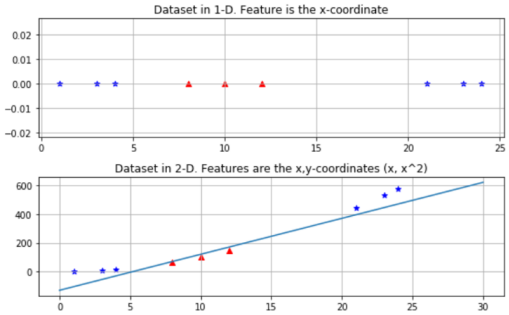
- Approach 1: Consider the perceptron as an any-time algorithm. When the user is out of time or resources, return the current weights.
- Approach 2: After each update, calculate the accuracy, and remember the weights corresponding to the highest accuracy:





Kernel Perceptron

While there exist datasets that aren't linearly separable in a given form (set of features), we can find a transformations to another space where the points are linearly separable:



Our next goal is to find transformations to spaces where our datasets are linearly separable. But firstly, we have to find a method to apply these transformations effectively.

Let $\mathbb{R}^n$ be the original features' space and $\mathbb{R}^m$ be the new features' space. Assume we have a transformation $\varphi : \mathbb{R}^n \rightarrow \mathbb{R}^m$. In a low dimensions space, i.e. if m is small, we cannot deal with more complex datasets. Otherwise, In a high

dimensions space, the computations become very slow. I'll introduce you **"The Kernel Trick"** - which helps us make these computations fast!

Firstly, lets have a look at the learning rule:

if $\hat{y} \neq y_{i:}$:

$$w \leftarrow w + yx$$

Thus, we can infer that the weights vector w learned by the Perceptron's algorithm is a linear combination of all the data points: $w = \sum_i \alpha_i y_i x_i$. From the algorithm, we can infer that $\alpha_i$is a "mistake-counter", where α_i is means "how many times the perceptron made a mistake on sample x_i. We can rewrite the prediction rule of a new observation x' as: $\hat{y} = sign[(\sum_i \alpha_i y_i x_i)^T x'] = sign[\sum_i \alpha_i y_i x_i^T x']$. In the new features' space, the prediction rule would become: $\hat{y} = sign[\sum_i \alpha_i y_i \varphi^T(x_i)\varphi(x')]$

This leads us to the dual problem, that is finding the α_i's. The new learning algorithm would loop thru the samples and make predictions, but now it will update the mistake counters vector α, rather than updating a weights vector m.

Observation: We can define a threshold s, and during learning we'll zero (and consider dropping samples from the dataset since those samples are probably outliers) any mistake-counter that reaches a value above it, i.e.:

if $\alpha_i \geq s$:

$$\alpha_i \leftarrow 0$$

drop x_i from the dataset

Now back to our mission, it makes it possible to get the same results as if you added many features, without adding them in practice. Usually, we define a kernel K such that $K(x, y) = \varphi^T(x)\varphi(y)$ and find a direct solution that doesn't involve any transformation and calculations in a higher dimention space. Example:

$$\varphi(u) = \varphi(\begin{bmatrix} u_1 \\ u_2 \end{bmatrix}) = \begin{bmatrix} 1 \\ \sqrt{2}u_1 \\ \sqrt{2}u_2 \\ \sqrt{2}u_1u_2 \\ u_1^2 \\ u_2^2 \end{bmatrix}$$

$$\Downarrow$$

$$\varphi^T(u)\varphi(v) = 1+2u_1v_1+2u_2v_2+2u_1u_2v_1v_2+u_1^2v_1^2+u_2^2v_2^2 = (1+u^Tv)^2$$

Note that computing the dot product u^Tv in the original features' space usually less expensive than computing the dot product in the transformed space. However, it appears that there exist transformation that "The Kernel Trick" could not be applied. Actually, the trick holds only if the the matrix K that corresponds to the inner product $\langle u, v\rangle_k = \varphi^T(u)\varphi(v)$ is positive semi-definite. This is a private case of Mercer's Theorm.

In practive, engineers and scientists use some sidely used kernels, for example:

1. $K(x, y) = (x^Ty + 1)^p$ - Polunomial kernel of degree p
2. $K(x, y) = e^{-\frac{1}{2\sigma^2}|x-y|^2}$ - Gaussian kernel
3. $K(x, y) = e^{-\gamma|x-y|^2}$ - RBF kernel (Radial basis function)
4. $K(x, y) = tanh(\eta x^Ty + \theta)$ - Sigmoid kernel

The new algorithm

After considering all the outcomes of the last formulations, we can define the new learnig algorithm and prediction rule as:

Learning:

Initialize a mistake-counter vector $\alpha \leftarrow \bar{0}$

while a stopping criterion isn't met:

Iterate over all $(x_j, y_j) \in X \times Y$:

$\hat{y} = sign(\sum_i \alpha_i y_i K(x_i, x_j))$

if $\hat{y} \neq y$:

$\alpha_j \leftarrow \alpha_j + 1$

Prediction:

$$\hat{y} = sign[\sum_i \alpha_i y_i K(x_i, x')]$$