Chapter 25

The Perceptron Algorithm

By Sariel Har-Peled, November 1, 2015[①] Version: 0.2

25.1. The perceptron algorithm

Assume, that we are given examples, say a database of cars, and you would like to determine which cars are sport cars, and which are regular cars. Each car record, can be interpreted as a point in high dimensions. For example, a sport car with 4 doors, manufactured in 1997, by Quaky (with manufacturer ID 6) will be represented by the point $(4, 1997, 6)$, marked as a sport car. A tractor made by General Mess (manufacturer ID 3) in 1998, would be stored as $(0, 1997, 3)$ and would be labeled as not a sport car.

Naturally, in a real database there might be hundreds of attributes in each record, for engine size, weight, price, maximum speed, cruising speed, etc, etc, etc.

We would like to automate this ***classification*** process, so that tagging the records whether they correspond to race cars be done automatically without a specialist being involved. We would like to have a learning algorithm, such that given several classified examples, develop its own conjecture about what is the rule of the classification, and we can use it for classifying new data.

That is, there are two stages for ***learning***: ***training*** and ***classifying***. More formally, we are trying to learn a function

$$f : \mathbb{R}^d \to \{-1, 1\}.$$

The challenge is, of course, that f might have infinite complexity – informally, think about a label assigned to items where the label is completely random – there is nothing to learn except knowing the label for all possible items.

This situation is extremely rare is the real world, and we would limit ourselves to a set of functions that can be easily described. For example, consider a set of red and blue points that are linearly separable, as demonstrated in Figure 25.1. That is, we are trying to learn a line ℓ that separates the red points from the blue points.

The natural question is now, given the red and blue points, how to compute the line ℓ? Well, a line or more generally a plane (or even a hyperplane) is the zero set of a linear function, that has the form

$$\forall x \in \mathbb{R}^d \qquad f(x) = \langle a, x \rangle + b,$$

where $a = (a_1, \ldots, a_d), b = (b_1, \ldots, b_d) \in \mathbb{R}^2$, and $\langle a, x \rangle = \sum_i a_i x_i$ is the ***dot product*** of a and x. The classification itself, would be done by computing the sign of $f(x)$; that is $sign(f(x))$. Specifically, if $sign(f(x))$ is negative, it outside the class, if it is positive it is inside.

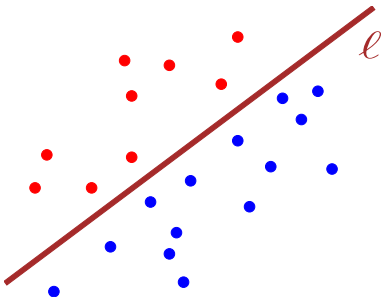


Figure 25.1: Linear separable red and blue point sets.

A set of training examples is a set of pairs

$$S = \Big\{(x_1, y_1), \ldots, (x_n, y_n)\Big\},$$

where $x_i \in \mathbb{R}^d$ and $y_i \in$ {-1,1}, for $i = 1, \ldots, n$.

A ***linear classifier*** h is a pair (w, b) where $w \in \mathbb{R}^d$ and $b \in \mathbb{R}$. The classification of $x \in \mathbb{R}^d$ is $sign(\langle w, x\rangle + b)$. For a ***labeled*** example (x, y), h classifies (x, y) correctly if $\text{sign}(\langle w, x\rangle + b) = y$.

Assume that the underlying label we are trying to learn has a linear classifier (this is a problematic assumption – more on this later), and you are given "enough" examples (i.e., n). How to compute the linear classifier for these examples?

One natural option is to use linear programming. Indeed, we are looking for $(\mathbf{w}, b)$, such that for an $(\mathbf{x}_i, y_i)$ we have $\text{sign}(\langle \mathbf{w}, \mathbf{x}_i\rangle + b) = y_i$, which is

$$\begin{aligned} & \langle \mathbf{w}, \mathbf{x}_i\rangle + b \geq 0 && \text{if } y_i = 1, \\ \text{and} \quad & \langle \mathbf{w}, \mathbf{x}_i\rangle + b \leq 0 && \text{if } y_i = -1. \end{aligned}$$

Or equivalently, let $\mathbf{x}_i = \left(x_i^1, \ldots, x_i^d\right) \in \mathbb{R}^d$, for $i = 1, \ldots, m$, and let $\mathbf{w} = \left(w^1, \ldots, w^d\right)$, then we get the linear constraint

$$\begin{aligned} & \sum_{k=1}^{d} w^k x_i^k + b \geq 0 && \text{if } y_i = 1, \\ \text{and} \quad & \sum_{k=1}^{d} w^k x_i^k + b \leq 0 && \text{if } y_i = -1. \end{aligned}$$

Thus, we get a set of linear constraints, one for each training example, and we need to solve the resulting linear program.

The main stumbling block is that linear programming is very sensitive to noise. Namely, if we have points that are misclassified, we would not find a solution, because no solution satisfying all of the constraints exists. Instead, we are going to use an iterative algorithm that converges to the optimal solution if it exists, see Figure 25.2.

Why does the **perceptron** algorithm converges to the right solution? Well, assume that we made a mistake on a sample $(\mathbf{x}, y)$ and $y = 1$. Then, $\langle \mathbf{w}_k, \mathbf{x}\rangle < 0$, and

$$\langle \mathbf{w}_{k+1}, \mathbf{x}\rangle = \langle \mathbf{w}_k + y * \mathbf{x}, x\rangle = \langle \mathbf{w}_k, \mathbf{x}\rangle + y \langle \mathbf{x}, \mathbf{x}\rangle = \langle \mathbf{w}_k, \mathbf{x}\rangle + y \|\mathbf{x}\| > \langle \mathbf{w}_k, \mathbf{x}\rangle .$$

Namely, we are "walking" in the right direction, in the sense that the new value assigned to $\mathbf{x}$ by $\mathbf{w}_{k+1}$ is larger ("more positive") than the old value assigned to $\mathbf{x}$ by $\mathbf{w}_k$.

Algorithm **perceptron**(S: a set of l examples)
  $\mathbf{w}_0 \leftarrow 0, k \leftarrow 0$
  $R = \max_{(\mathbf{x},y)\in S} \left\|\mathbf{x}\right\|$.
  repeat
    for $(\mathbf{x}, y) \in S$ **do**
      if $\text{sign}(\langle \mathbf{w}_k, \mathbf{x}\rangle) \neq y$ **then**
        $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + y * \mathbf{x}$
        $k \leftarrow k + 1$
  until no mistakes are made in the classification
  return $\mathbf{w}_k$ and k

Figure 25.2: The **perceptron** algorithm.

Theorem 25.1.1. *Let S be a training set of examples, and let $R = \max_{(x,y)\in S} \left\|x\right\|$. Suppose that there exists a vector w_{opt} such that $\left\|w_{opt}\right\| = 1$, and a number $\gamma > 0$, such that*

$$y \left\langle w_{opt},\ x \right\rangle \geq \gamma \qquad \forall (x, y) \in S.$$

Then, the number of mistakes made by the online **perceptron** *algorithm on S is at most*

$$\left(\frac{R}{\gamma}\right)^2.$$

Proof: Intuitively, the **perceptron** algorithm weight vector converges to w_{opt}, To see that, let us define the distance between w_{opt} and the weight vector in the kth update:

$$\alpha_k = \left\| w_k - \frac{R^2}{\gamma} w_{opt} \right\|^2.$$

We next quantify the change between α_k and α_{k+1} (the example being misclassified is (x, y)):

$$\begin{aligned}
\alpha_{k+1} &= \left\| w_{k+1} - \frac{R^2}{\gamma} w_{opt} \right\|^2 \\
&= \left\| w_k + y\mathbf{x} - \frac{R^2}{\gamma} w_{opt} \right\|^2 \\
&= \left\| \left(w_k - \frac{R^2}{\gamma} w_{opt} \right) + y\mathbf{x} \right\|^2 \\
&= \left\langle \left(w_k - \frac{R^2}{\gamma} w_{opt} \right) + y\mathbf{x},\ \left(w_k - \frac{R^2}{\gamma} w_{opt} \right) + y\mathbf{x} \right\rangle.
\end{aligned}$$

Expanding this we get:

$$\begin{aligned}
\alpha_{k+1} &= \left\langle \left(w_k - \frac{R^2}{\gamma} w_{opt} \right), \left(w_k - \frac{R^2}{\gamma} w_{opt} \right) \right\rangle + 2y \left\langle \left(w_k - \frac{R^2}{\gamma} w_{opt} \right), \mathbf{x} \right\rangle + \langle \mathbf{x}, \mathbf{x} \rangle \\
&= \alpha_k + 2y \left\langle \left(w_k - \frac{R^2}{\gamma} w_{opt} \right), x \right\rangle + \left\| x \right\|^2.
\end{aligned}$$

As $(\mathbf{x}, y)$ is misclassified, it must be that $\text{sign}(\langle w_k, \mathbf{x}\rangle) \neq y$, which implies that $\text{sign}(y\langle \mathbf{w}_k, \mathbf{x}\rangle) = -1$; that is $y\langle \mathbf{w}_k, \mathbf{x}\rangle < 0$. Now, since $\left\|x\right\| \leq R$, we have

$$\begin{aligned}\alpha_{k+1} &\leq \alpha_k + R^2 + 2y\langle w_k, \mathbf{x}\rangle - 2y\left\langle \frac{R^2}{\gamma} w_{opt}, \mathbf{x}\right\rangle \\ &\leq \alpha_k + R^2 + \qquad\qquad -2\frac{R^2}{\gamma} y\left\langle w_{opt}, x\right\rangle.\end{aligned}$$

Next, since $y\left\langle w_{opt}\ ,\ x\right\rangle \geq \gamma$ for $\forall (x, y) \in S$, we have that

$$\begin{aligned}\alpha_{k+1} &\leq \alpha_k + R^2 - 2\frac{R^2}{\gamma}\gamma \\ &\leq \alpha_k + R^2 - 2R^2 \\ &\leq \alpha_k - R^2.\end{aligned}$$

We have: $\alpha_{k+1} \leq \alpha_k - R^2$, and

$$\alpha_0 = \left\|0 - \frac{R^2}{\gamma} w_{opt}\right\|^2 = \frac{R^4}{\gamma^2}\left\|w_{opt}\right\|^2 = \frac{R^4}{\gamma^2}\ .$$

Finally, observe that $\alpha_i \geq 0$ for all i. Thus, what is the maximum number of classification errors the algorithm can make?

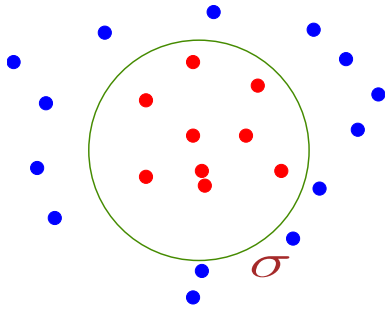
$$\left(\frac{R^2}{\gamma^2}\right).$$

■

It is important to observe that any linear program can be written as the problem of separating red points from blue points. As such, the **perceptron** algorithm can be used to solve linear programs.

25.2. Learning A Circle

Given a set of red points, and blue points in the plane, we want to learn a circle that contains all the red points, and does not contain the blue points.



How to compute the circle σ ?

It turns out we need a simple but very clever trick. For every point $(x, y) \in P$ map it to the point $\left(x,\ y,\ x^2 + y^2\right)$. Let $z(P) = \left\{\left(x,\ y,\ x^2 + y^2\right)\ \middle|\ (x, y) \in P\right\}$ be the resulting point set.

Theorem 25.2.1. *Two sets of points R and B are separable by a circle in two dimensions, if and only if $z(R)$ and $z(B)$ are separable by a plane in three dimensions.*

Proof: Let $\sigma \equiv (x-a)^2 + (y-b)^2 = r^2$ be the circle containing all the points of R and having all the points of B outside. Clearly, $(x-a)^2 + (y-b)^2 \leq r^2$ for all the points of R. Equivalently

$$-2ax - 2by + \left(x^2 + y^2\right) \leq r^2 - a^2 - b^2.$$

Setting $z = x^2 + y^2$ we get that

$$h \equiv -2ax - 2by + z - r^2 + a^2 + b^2 \leq 0.$$

Namely, $p \in \sigma$ if and only if $h(z(p)) \leq 0$. We just proved that if the point set is separable by a circle, then the lifted point set $z(R)$ and $z(B)$ are separable by a plane.

As for the other direction, assume that $z(R)$ and $z(B)$ are separable in $3d$ and let

$$h \equiv ax + by + cz + d = 0$$

be the separating plane, such that all the point of $z(R)$ evaluate to a negative number by h. Namely, for $(x, y, x^2 + y^2) \in z(R)$ we have $ax + by + c(x^2 + y^2) + d \leq 0$

and similarly, for $(x, y, x^2 + y^2) \in B$ we have $ax + by + c(x^2 + y^2) + d \geq 0$.

Let $U(h) = \left\{(x, y) \;\middle|\; h\Big((x, y, x^2 + y^2)\Big) \leq 0\right\}$. Clearly, if $U(h)$ is a circle, then this implies that $R \subset U(h)$ and $B \cap U(h) = \emptyset$, as required.

So, $U(h)$ are all the points in the plane, such that

$$ax + by + c\left(x^2 + y^2\right) \leq -d.$$

Equivalently

$$\left(x^2 + \frac{a}{c}x\right) + \left(y^2 + \frac{b}{c}y\right) \leq -\frac{d}{c}$$

$$\left(x + \frac{a}{2c}\right)^2 + \left(y + \frac{b}{2c}\right)^2 \leq \frac{a^2 + b^2}{4c^2} - \frac{d}{c}$$

but this defines the interior of a circle in the plane, as claimed. ∎

This example show that linear separability is a powerful technique that can be used to learn complicated concepts that are considerably more complicated than just hyperplane separation. This lifting technique showed above is the ***kernel technique*** or ***linearization***.

25.3. A Little Bit On VC Dimension

As we mentioned, inherent to the learning algorithms, is the concept of how complex is the function we are trying to learn. VC-dimension is one of the most natural ways of capturing this notion. (VC = Vapnik, Chervonenkis,1971).

A matter of expressivity. What is harder to learn:

1. A rectangle in the plane.
2. A halfplane.
3. A convex polygon with k sides.

Let $X = \{p_1, p_2, \ldots, p_m\}$ be a set of m points in the plane, and let R be the set of all halfplanes.
A half-plane r defines a binary vector

$$r(X) = (b_1, \ldots, b_m)$$

where $b_i = 1$ if and only if p_i is inside r.
Let

$$U(X, R) = \{r(X) \mid r \in R\}.$$

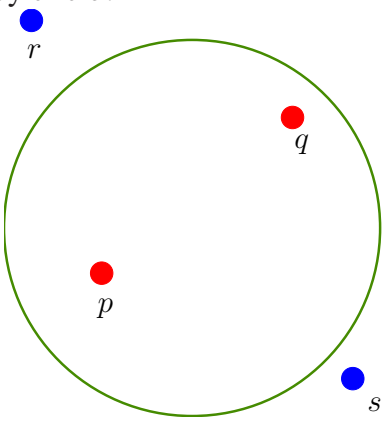
A set X of m elements is *shattered* by R if

$$|U(X, R)| = 2^m.$$

What does this mean?
The VC-dimension of a set of ranges R is the size of the largest set that it can shatter.

25.3.1. Examples

What is the VC dimensions of circles in the plane?
Namely, X is set of n points in the plane, and R is a set of all circles.
$X = \{p, q, r, s\}$
What subsets of X can we generate by circle?



$\{\}, \{r\}, \{p\}, \{q\}, \{s\}, \{p, s\}, \{p, q\}, \{p, r\}, \{r, q\}\{q, s\}$ and $\{r, p, q\}$, $\{p, r, s\}\{p, s, q\}, \{s, q, r\}$ and $\{r, p, q, s\}$
We got only 15 sets. There is one set which is not there. Which one?
The VC dimension of circles in the plane is 3.

Lemma 25.3.1 (Sauer Lemma). *If R has VC dimension d then $|U(X, R)| = O\left(m^d\right)$, where m is the size of X.*