Perceptron: for binary classification.

- Algorithm for learning half spaces.

• Script $\mathcal{C}$ = {half spaces}

definition of half space: $f(x) = \text{sign}\left(\sum_{i=1}^{n} \underset{\substack{\uparrow \\ \text{dot product}}}{w_i \cdot x_i} - \overset{\text{threshold.}}{\theta}\right)$

vector w = unknown.

$f(x) \in \{-1, +1\}$

$x_i \in \mathbb{R}^n$.

A "complicated" algorithm for learning halfspace:

$(x^{(1)}, \underset{\substack{\uparrow \\ \text{label}}}{+1}) \longrightarrow w_1 x_1^{(1)} + \dots + w_n x_n^{(1)} \geq \theta$

$(x^{(2)}, -1) \longrightarrow w_1 x_1^{(2)} + \dots + w_n x_n^{(2)} < \theta$.

$\vdots \qquad \vdots$

m training examples $\longrightarrow$ m inequalities.

we can use linear programming to find weight vector W consistent w/ training set.

• LP is very expensive. use Perceptron.

Perceptron:

1) Initially, $w^0 = (0, \dots, 0)$ or unit $w^0 = (\frac{1}{\sqrt{n}}, \dots, \frac{1}{\sqrt{n}})$.
2) Learner has w as its state.
3) Teacher presents challenge:

$$\xleftarrow{\quad x \in \mathbb{R}^n \quad} \text{Teacher}$$

4) Learner responds with $\text{sign}(w \cdot x)$, which is current state.

$$\text{Learner} \xrightarrow{\quad \text{sign}(w \cdot x) \quad}$$

5) If mistake is made:

Case 1: x was truly a negative example:

$$w_{new} = w_{old} - x.$$

Case 2: x was truly a positive example:

$$w_{new} = w_{old} + x.$$

Equivalent way to update rule (applies only when mistake).

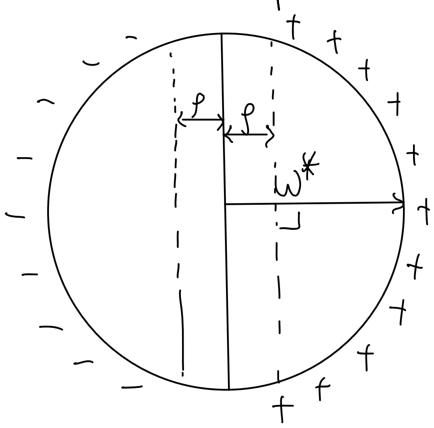
$$w_{new} = w_{old} + y \cdot \vec{x}$$

label $\in \{-1, +1\}$

Assumptions:

- Assume $\exists\, w^*$, which is true unknown weight vector, which has norm 1, $\|w^*\| = 1$.
- Assume x has norm 1, $\|x\| = 1$.
- $\theta = 0$.

Main Assumption: There exists a margin ρ where all points are at least distance ρ from w^*.



All +/− points have distance $\geq \rho$ from half space.

$\|x\| = \|w^*\| = 1$ (assumption).

Equivalently, $\underbrace{|\langle x, w^* \rangle| \geq \rho}_{\text{"margin Assumption"}}$

ρ = "cushion" of the classifier.

Main Theorem: "Perceptron convergence theorem":

The mistake bound of the perceptron algorithm is $O\left(\frac{1}{\rho^2}\right)$ mistakes.

proof:

Recall update step: $w_{new} = w_{old} + y \cdot x$

Let's say: w is current state of learner,

w^* is true normal to half space.

Claim 1: on every mistake, $w \cdot w^*$ increases by at least ρ.

claim 2: $\|w\|^2$ increases after every mistake by at most 1.

Question: How to obtain $O\left(\frac{1}{\rho^2}\right)$ mistake bound given claims 1 and 2?

Let t = # mistakes we've made at some point during execution.

$$\underbrace{t \cdot \rho \leq w \cdot w^*}_{\text{claim 1.}} \underset{\substack{\uparrow \\ \text{Cauchy-Schwartz} \\ \text{inequality}}}{\leq} \overset{\leq \sqrt{t} \text{ since } \underbrace{\|w\|^2 \leq t}_{\text{Claim 2.}}}{\|w\|} \underbrace{\|w^*\|}_{\to 1}$$

$$\Rightarrow t\rho \leq \sqrt{t}$$

$$\Rightarrow \boxed{t \leq \frac{1}{\rho^2}}$$

Proving the claims:

Claim 1: $w \cdot w^*$ increases on every mistake by $\geq \rho$.

$w_{new} = w_{old} + y\vec{x}$ (only when there is a mistake).

$$\Rightarrow w_{new} \cdot w^* = \left(w_{old} + y \cdot \vec{x}\right) \cdot \vec{w}^*$$

$$= \Big(w_{old} \cdot w^* + \underbrace{\underset{\substack{\uparrow \\ -1 \text{ since updates} \\ \text{only when} \\ \text{mistake.}}}{y} \cdot \underbrace{\vec{x} \cdot \vec{w}^*}_{\substack{\ominus \text{ since mistake is} \\ \text{made, and we know} \\ \text{the absolute value} \geq \rho \\ (\leq -\rho) \text{ per the} \\ \text{margin assumption}}}}_{\geq \rho}\Big)$$

proving claim 2:

$\|w\|^2$ increases by at most 1 on every mistake.

$$\|\omega_{new}\|^2 = \|w_{old} + y \cdot x\|^2 = \|w_{old}\|^2 + \underbrace{\underbrace{2 \cdot y \langle \vec{x}, w_{old} \rangle}_{\text{negative, bc } \langle \vec{x} \cdot w_{old} \rangle \text{ has different sign from } y \text{ since mistake is made, therefore their product = negative.}} + \underbrace{\|x\|^2}_{1}}_{\leq 1}$$

Look back at some assumptions:

- $\theta = 0$: Add a new feature, call it x_{n+1}

  $(x_1, \dots x_n) \longrightarrow (x_1, \dots, x_{n+1})$

  always set to 1.

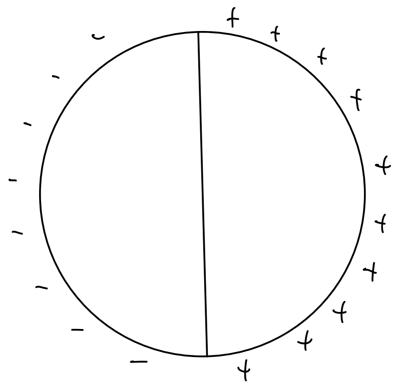
- $\|x\| = 1$. If $\|x\| = R \Rightarrow$ M.B. $O\left(\frac{R^2}{\rho^2}\right)$

Perceptron Learning: Polynomial Threshold Functions (PTFs)

Definition: $f = \text{sign}(p(x))$

p is a multivariate polynomial of degree d.

How can we use perceptron to learn this function class?



Higher degree polynomial

feature map:

$$\underbrace{(x_1, \ldots, x_n)}_{\text{degree } 2} \longrightarrow x_1^2, x_1 x_2, \ldots, x_n^2 = n^2 \text{ new variables.}$$

Call them $Y_1, \ldots, Y_{N=n^2}$, $f = \text{sign}\left(\sum_{i=1}^{N} w_i Y_i\right)$

$$\boxed{\text{Learning PTFs of degree } d \text{ is equivalent to learning halfspaces in } n^d \text{ dimensions.}}$$

↳ can run perceptron to learn this half-space in higher dimension.

- Runtime: just computing the feature map takes time n^d.
- What is the margin in this n^d dimension space?
  May be costly.

We can save on the running time, using something called the kernel trick!

Perceptron Learning: Kernel Functions:

$X \leftarrow$ input

☆ Denote $\varphi(x)$ to be the image of x in the feature space.

$$X \in R^n \xrightarrow[\text{function}]{\text{kernel}} \varphi(x) \in R^{n^d}$$

$K(X^1, X^2)$ outputs $\{\varphi(X^1), \varphi(X^2)\}$

kernel function

and let's assume $K(X^1, X^2)$ is easy to compute.

kernel function.

Kernel Perception (want to work in R^{n^d})

$W = 0^{n^d} = \{0$ vector of length $n^d\}$

Let's Assume we make a mistake on first try (point X^1)

$$W_{new} = \underbrace{W_{old}}_{\text{0 vector bc first mistake.}} + y\,\varphi(x)$$

in n^d space

$x^{(2)}$ is the new point.

We need to evaluate $W_{new} \odot \varphi(x^{(2)})$

$$= \langle y \cdot \varphi(x^1), \varphi(x^2) \rangle \qquad (1)$$

$$= \underset{\text{Scalar}}{y} \cdot \underbrace{K(x^{(1)}, x^{(2)})}_{\text{easy to compute.}}.$$

Note: $K(X^1, X^2) = \langle \varphi(X^1), \varphi(X^2) \rangle$

therefore (1) is $y \cdot K(X^1, X^2)$

$$W_{t+1} = \sum_{i=1}^{t} y^i \cdot \varphi(x^i) \quad \in \mathbb{R}^{n^d}$$

If need to compute $\langle W_{t+1}, \varphi(x^{t+1}) \rangle$:

$$\sum_{i=1}^{t} y^i \cdot \langle \underbrace{\varphi(x^i), \varphi(x^{t+1})}_{\underbrace{K(x^i, x^{t+1})}_{\text{efficiency computable.}}} \rangle$$

So, we're able to simulate the perceptron algorithm in a much higher dimensional space with just low dimensional vectors and this kernel function!

Example of Kernel Function:

Consider degree-2 polynomial threshold function:

$$\varphi(x_1, \dots, x_n) = (x_1 x_1, x_1 x_2, \dots x_{n-1} x_n, x_{n^2})$$

$$\underset{(x, z \in \mathbb{R}^n)}{K(x, z)} = \underbrace{\langle \varphi(x), \varphi(z) \rangle}_{\text{inner product}} =$$

$$= (x_1^2 z_1^2 + x_1 x_2 z_1 z_2 + \dots + x_n^2 z_n^2)$$

$$= \sum_{i,j} x_i x_j z_i z_j = \left(\sum_{i=1}^{n} x_i z_i\right) \cdot \left(\sum_{j=1}^{n} x_j z_j\right)$$

$$= \boxed{\underbrace{(x \cdot z)}_{\text{inner product}}^2 = K(x, z)}$$

Example of Kernel methods.

Other Kernels

$$K(x,z) = (x \cdot z + c)^2$$

$$\varphi(x) = (x_1^2, \dots, x_n^2, \sqrt{2c}\, x_1, \dots, \sqrt{2c}\, x_n, c)$$

Gaussian Kernels = $K(x,z) = \underbrace{e^{-\|x-z\|_2^2}}_{\text{radial basis kernel.}}$