Last time:
- Winnow 2 for $F(\delta)$ = $\delta$-sep. monotone LTFs
- Perceptron alg, PCT: $1/\delta^2$ m.b. for unit $x$, origin-sep. $v \cdot x \geq 0$ target fn, with "margin" $\delta$.

Today:
- example of applying PCT
- Dual Perceptron, "Kernelization"
- (maybe) start generic bounds for online learning

Questions?

---

Ex of applying PCT:

King with $n$ advisors ($n$ odd)
King feeds subset $S \subseteq [n]$ give bad advice

Binary dec: gets all $n$ opinions, negates votes in $S$, goes w/ maj vote.

You: political analyst. Observe repeatedly.
How many mistakes? (predicting King's decision given advisors' opinions)

A: Use Perceptron.

$MAJ(x_1, \ldots, x_n) : \{0,1\}^n \to \{0,1\}$

$= \mathbb{1}[x_1 + \ldots + x_n \geq \frac{n}{2}]$

Target: $MAJ(x_1, \bar{x}_2, \bar{x}_3, x_4, x_5, \bar{x}_6, x_7, x_8)$

$S = \{2, 3, 6\}$.

---

$\pm 1$ view on input:

$\{\pm 1\}^n \to \{\pm 1\}$

$MAJ(x) = \text{sign}(x_1 + \dots + x_n)$

"S-negated maj": $\text{sign}(v_1 x_1 + \dots + v_n x_n)$

$v_i = \begin{matrix} +1 \\ -1 \end{matrix}$ $\longleftarrow i \in S$

Perc alg wants $v$ unit vector:

normalize $v_i = \begin{matrix} +1/\sqrt{n} \\ -1/\sqrt{n} \end{matrix}$ ☺

each $x_i = \begin{matrix} +1 \\ \sqrt{n} \end{matrix}$ $\begin{matrix} -1 \\ \sqrt{n} \end{matrix}$ every $\|x\| = 1$, $\|v\| = 1$

PCT: $|v \cdot x| \geq ?$ $\frac{1}{n}$

$|v \cdot x| = |v_1 x_1 + v_2 x_2 + \dots + v_n x_n| =$ sum of $n$ $\pm 1/n$

$\pm 1/\sqrt{n}$ $\pm 1/\sqrt{n}$

$\geq \frac{1}{n}$

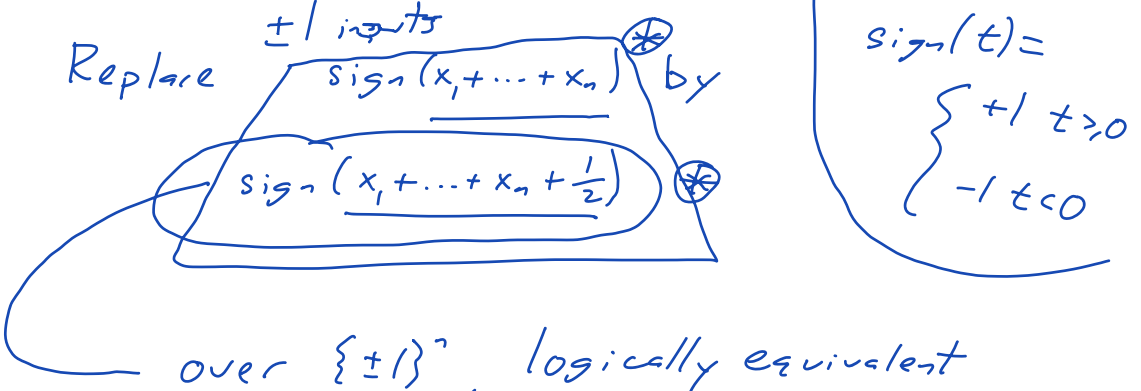
$\delta = \frac{1}{n}$

So PCT $\Rightarrow$ make $\leq \frac{1}{\delta^2} \leq n^2$ mist.

---

What if $n$ even, & MAJ is $\mathbb{I}\left[x_1+\cdots+x_n \geq \frac{n}{2}\right]$ ?

$\pm 1$ $x_1 + \cdots + x_n$ could be 0




over $\{\pm 1\}^n$, logically equivalent

---

## Dual Perceptron & "kernel functions"

Nice thing about Perc: has a "dual form" (exactly equivalent, looks different - will make it possible to use "kernels").

Key idea of dual Perc: to run Perc, only need to be able to compute inner prod. of 2 examples.

Running Perc: init wt vector $w = 0^n$.

After 1st mist (on $x'$), now $w$ is $x'$ or $-x'$.

So $w \cdot x$ is $x^1 \cdot x$ or $-x^1 \cdot x$ (inner prod of 2 ex).

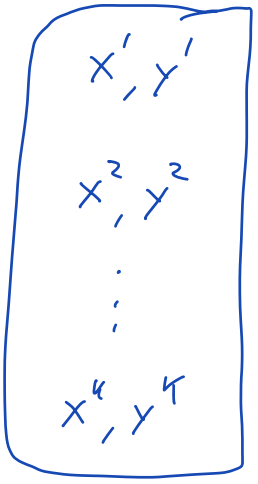
2nd mist, on $x^2$: w becomes, say, $-x^1 + x^2$

To compute $w \cdot x$, compute $-x^1 \cdot x + x^2 \cdot x$

After $k$ updates, can compute $w \cdot x$ as $\pm 1$ sum of $k$ i.p.'s of examples.

More formal descrip. of dual Perc:

Maintain hyp as list of pairs (lab ex's)




$x^i$ = $i^{th}$ ex on which we made mistake

$y^i \in \{\pm 1\}$ = label (correct) of $x^i$.

Given new $x$, $w \cdot x = \sum_{i=1}^{k} y^i (x^i \cdot x)$.

Can Exactly simulate Perceptron this way.

---

Why? Seems ineff — after $k$ mist, hyp. is of size $n \cdot k$.

True!

But... can replace "inner product" with any "kernel function". → (this lets us run dual Perc. over any space for which we have an eff. evaluatable kernel function!)

## Kernel functions & feature expansions.

Let $X$ = original feature space

$\{0,1\}^n$ or $\mathbb{R}^n$

($X$ has an inner product).

Let $X'$ = aspirational feature space, also with an inner product:

$\{0,1\}^N$ or $\mathbb{R}^N$

A feature expansion is a mapping $\Phi : X \to X'$.

Ex:

$X = \{0,1\}^n$, $X' = \{0,1\}^{3^n = N}$

$\Phi : \{0,1\}^n \to \{0,1\}^{3^n}$

$(x_1, \ldots, x_n) \to$ (list of all $3^n$ conj over $x_1, \ldots, x_n$)

$n=2$: $\Phi(x_1, x_2) =$

(empty conj)

$$(1,\ x_1,\ \bar{x}_1,\ x_2,\ \bar{x}_2,\ x_1 \wedge x_2,\ \bar{x}_1 \wedge x_2,\ x_1 \wedge \bar{x}_2,\ \bar{x}_1 \wedge \bar{x}_2)$$

$\Phi(x)$ is "expanded" version of $x$.

---

The Kernel function for feat. exp. $\Phi : X \to X'$

is $K : X \times X \to \mathbb{R}$,

$$K(a, b) = \Phi(a) \cdot \Phi(b)$$

$a$ in $X$; $\Phi(a)$ in $X'$; $\Phi(b)$ in $X'$; $K(a,b)$ in $\mathbb{R}$

→ i.e. inner product in $X'$-space.

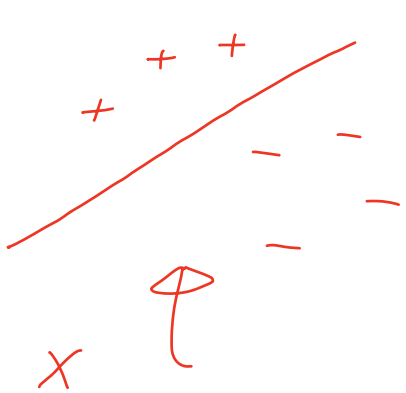
---

Why do feature exp?

$(x_1, x_2)$ orig features

Consider LTFs over $\mathbb{R}^2$

$X = (x_1, x_2)$ all pairs




Consider $X'$ = feat. exp. of $X$

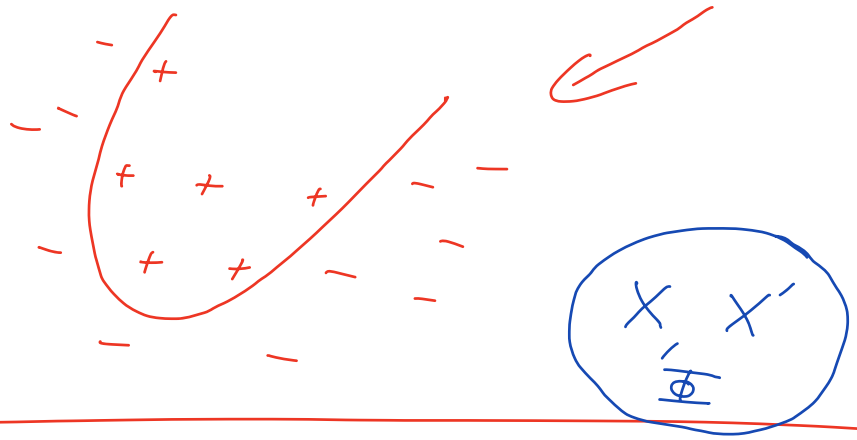
obt. by allowing monom. of degree up to 2:

$X'$ = tuples $(1, x_1, x_2, x_1, x_2, x_1^2, x_2^2)$

$\Phi(x) = \Phi(x_1, x_2) = (1, x_1, x_2, x_1, x_2, x_1^2, x_2^2)$

LTFs over $X'$: ← richer!

$\mathbb{1}\{3x_1^2 - 2x_1x_2 + 5x_1 \geq 3\}$




Suppose you have data in $X$, but your dream is to run Perceptron on the $\Phi$-feature-expanded data in $X'$.

Obvious approach: do it directly.

Each $x$ you get: write down $\Phi(x)$.

Maintain $w'$ in $X'$ domain.

Legit, but could be super-slow: time $N$ to write down $\Phi(x)$.

---

Better way: use kernel $K$ for $\Phi$, run dual Perc., replacing every $a \cdot b$ (inner prod) by $K(a,b)$. This exactly simulates running Perceptron over the $\Phi$-expanded examples! GREAT!!! ....

---

→ if can efficiently compute $K(a,b)$ ....

---

Good news: sometimes (for some interesting feat. exp. $\Phi$), there is a computationally efficient way to compute $K(a,b)$, much faster than writing $\Phi(a)$, $\Phi(b)$ & doing it explicitly.

Ex

Back to $\Phi(x)$ = all-conj feat. exp.

$$X = \{0,1\}^n, \quad X' = \{0,1\}^{3^n} = N$$

Claim: $K(a,b)$ for this can be computed in $O(n)$ time!

Pf: Fix $a, b \in \{0,1\}^n$.

Let $SAME(a,b)$ denote # coords in $\{1,\dots,n\}$ ($[n]$) where $a_i = b_i$

$n = 16$

$a_8 \neq b_8$

$a = 1111111100000000$

$b = 1100000011111000$

$SAME(a,b) = 5$

each coord: 0 or 1

$$\Phi(a) = (1, a_1, \bar{a}_1, a_2, \bar{a}_2, \dots a_1 \wedge a_2, a_1 \wedge \bar{a}_2, \dots\dots)$$

$$\Phi(b) = (1, b_1, \bar{b}_1, b_2, \bar{b}_2, \dots b_1 \wedge b_2, b_1 \wedge \bar{b}_2, \dots\dots)$$

$\Phi(a) \cdot \Phi(b)$ = # of the $3^n$ positions with a 1 in both vectors.

since $a_8 \neq b_8$, any conj involving $x_8$ or $\bar{x}_8$ won't be sat. by both.

Only conj that could give 1 in inner prod. must be over vars where $a_i = b_i$:

in above ex, these are conj over only

$x_1, x_2, x_{14}, x_{15}, x_{16}$.

$S \subseteq \{1, 2, 14, 15, 16\}$

$S = \{1, 14\}$: $x_1 \wedge \bar{x}_{14}$ (a + b both eval. to 1)

$S = \{14, 15, 16\}$: $\bar{x}_{14} \wedge \bar{x}_{15} \wedge \bar{x}_{16}$

So $K(a,b) = 2^{SAME(a,b)}$

So can run (dual) Perc. over $3^n$-dim space of all conj!

→ If Perc makes $k$ mistakes, runtime per trial is $\leq$ poly$(n, k)$

Next time: generic C's.