Last time:
  • Winnow 2 alg. for (certain) LTFs over \( \mathbb{F}_2 \)
  • Perceptron alg. for learning LTFs over \( \mathbb{R} \)
    (with a margin)

Today:
  • review \( w_1, w_2, \hat{p} \); Finish Perceptron application
  • Dual Perceptron, "kernelization"
  • Start generic bounds for online learning
    - Halving alg
    - Randomized HA for oblivious adversaries

Questions?
  Let's answer q. from last time
  Ex: King has \( n \) advisors (\( n \) odd)
  he feels subset \( S \) of them give bad advice.
  King has to decide Y/N: he negates advice of ones in \( S \) & takes maj vote.

  Observer sees this: can observer "learn \( S \)"
  (predict King's votes, given advisors' votes)
  with a reasonable MB?

Recall \( \text{MAJ}: \{0,1\}^n \rightarrow \{0,1\} \)
\[ \text{MAJ}(x) = \begin{cases} 1 & \text{if } x_1 + \ldots + x_n \geq \frac{n}{2} \\ 0 & \text{otherwise} \end{cases} \]
Let's view inputs not as 0/1, but as -1/1.

Now \( \text{MAJ} : \{ -1, 1 \} \rightarrow \{ -1, 1 \} \) is

\[
\text{MAJ}(x) = \text{sign}(x_1 + \ldots + x_n)
\]

Moreover, a bit like \( \text{not}(x_i) \), \( \bar{x}_i \) is not just \(-x_i\).

Target concept: \( \text{MAJ}(b_1, b_2, \ldots, b_n) \) each \( b_i \) either \( x_i \) or \(-x_i\) i.e. \( \text{sign}(v \cdot x) \) \( v \in \{ \pm 1 \}^n = \{-1, 1\}^n \).

Use Perc. alg. : need each ex. \( x \) is unit vector

\[
\text{target } v \quad \text{unit } v
\]

We have each \( x \in \{-1, 1\}^n \) has \( \|x\| = \sqrt{n} \)

Likewise \( \|v\| = \sqrt{n} \).

Rescale: each ex. by \( \frac{1}{\sqrt{n}} \) \( \frac{1}{\sqrt{n}} \cdot \{-1, 1\}^n \) now \( \|x\| = 1 \)

\[
\text{target } v \in \frac{1}{\sqrt{n}} \cdot \{-1, 1\}^n \text{ now } \|v\| = 1.
\]

What's \( \delta ? \) \( \min_{x \in \text{ex}} \|v \cdot x\| \)

\[
v \cdot x = \frac{1}{\sqrt{n}} \cdot \frac{1}{\sqrt{n}} (\pm 1, \pm 1, \ldots, \pm 1)
\]

\( \text{sign}(v \cdot x) = \text{sign}(v \cdot x') \)

if \( v' = C \cdot v \), \( x' = C \cdot x \), \( C, C \geq 0 \) \( \|v \cdot x\| \geq \frac{1}{\sqrt{n}} \). \( \delta = \frac{1}{n} \)

So Perc alg. makes \( \leq \frac{1}{\delta^2} = \frac{n}{\delta^2} \) mistakes.

\( n \) Even: tweak \( v \cdot x = 0 \) "\n
Consider different sep. hyperplane s.t. margin nonzero:

Discussion of Perceptron:
Dual Perceptron & "Kernelization"

Fact: Perceptron has an equivalent "dual form" which is "kernelizable" (can run it over large or no feature space in an efficient way).

Key insight behind Dual Perc: to run Perc, only need to be able to compute $x \cdot x'$ for two input vectors.

Recall how Perc works over $\mathbb{R}^n$:
initial hyp vector $w = 0^- = (0, \ldots, 0)$.
After 1st update: $w \leftarrow w + yx$, $y \in \{+1, -1\}$
  $w$ is either $x'$ or $-x'$ ($x'$ = 1st ex. that caused).
Say $w$ is $x'$, new ex.
Then $w \cdot x$ is $x' \cdot x$, inner prod of 2 ex.

After 2nd update: maybe $w$ is $x' - x^2$, thus
  $w \cdot x$ is $(x' - x^2) \cdot x = x' \cdot x - x^2 \cdot x$
After $k$ updates, can compute $w\cdot x$ by computing $k$ inner prod. between examples.

More formal description of dual percs:

Hyp. stored as list of pairs $(x_i, y_i)$, $y_i \in \{1, -1\}$

$x_i$ = $i^{th}$ ex. on which Perceptron made mistake.

Given new ex $x$, dual perc. computes $w \cdot x$ as

$$\sum_{i=1}^{k} y_i(x_i \cdot x) .$$

This exactly simulates Perceptron.

---

Q: Why? Seems clunkier, more storage than old way.
A: is useful be. can replace usual inner prod. over $\mathbb{R}^n$ with any "kernel function"

---

**Kernel function & feature expansions**

Let $X, X'$ be two spaces with inner products

(think of $X = \mathbb{R}^n$, $X' = \mathbb{R}^N$, where $N \gg n$).

$r^n \rightarrow r^N$

A feature expansion is a map $\Phi : X \rightarrow X'$

Ex: $X = \{0,1\}^n$, $X' = \{0,1\}^3$; $\Phi : \{0,1\}^n \rightarrow \{0,1\}^3$.
$\Phi(x)$ is the feat. exp. with a feature for every poss. conjunction over $\{0,1\}^n$.

"all-conjunctions feature exp."

Ex: $n=2$ : $\Phi(x_1, x_2) = \text{all 2 conj over } \{0,1\}^2$.

$(1, x_1, \overline{x_1}, x_2, \overline{x_2}, x_1 \land x_2, x_1 \land \overline{x_2}, \overline{x_1} \land x_2, \overline{x_1} \land \overline{x_2})$

(We view $\Phi(x)$ as "expanded" version of $x$.)

The kernel function corresponding to $\Phi$ is $K : X \times X \to \mathbb{R}$ defined as

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

Suppose you have examples from $X$, but want to run Perceptron on $X'$ after mapping according to $\Phi$.

Why? LTFs over expanded features can be more powerful/expansive than LTFs over $X$.

$X = \mathbb{R}^2$

$3x_1 + 4x_2 \geq 0$

$\Phi(x) = \Phi(x_1, x_2) = (x_1, x_2, x_1^2, x_2^2, x_1x_2)$

$3x_1^2 - 4x_1x_2 + x_2^2 \geq 0$
Obv. approach: do it directly:

- each $x \in \mathcal{X}$ you get, write down $\overline{\Phi}(x)$, use it as ex. for $\overline{\Phi}(x)$.

Takes time $\geq N$ just to write down $\overline{\Phi}(x)$. 

Better way: run dual Perc, but replace each $x \triangleleft x'$ with $K(x, x')$. This exactly simulates the above.

(running Perc over $X$ with $\overline{\Phi}(x)$ examples) $\Rightarrow$

Sometimes, for some $\overline{\Phi}$'s, can compute $K(x, x')$ much faster than time $N$!

Back to prev ex: "all conjunctions feature exp."

\[ X = \{0,1\}^n, \quad X' = \{0,1\}^n \]

3-dim vectors

Claim: the kernel $K(a, b) = \overline{\Phi}(a) \cdot \overline{\Phi}(b)$

is computable in $O(n)$ time! $\Rightarrow$ \{1, ..., n\}

Given $a, b \in \{0,1\}^n$, define set $\text{SAME}(a, b) \subseteq [n]$

to be \{i \in [n] : a_i = b_i\} (both 1 or both 0).

Claim: $K(a, b) = \gamma \text{SAME}(a, b)$. 


PF: Every subset $S \subseteq \text{SAME}(a, b)$ corresponds to a conj. in which cards in $S$ are 1 in both $\overline{a}(a)$, $\overline{a}(b)$, & these $S$'s are only such conj. So $2^{\text{SAME}(a, b)} \leq \sum$

$\overline{a}(a) = (\ldots - - - - 1 - - - )$
$\overline{a}(b) = (\ldots - - - - 1 - - - )$

Ex: $a = 1111111000000000 \in S_0, b = \overline{11000011111000}$
$\text{SAME}(a, b) = \{1, 2, 14, 15, 16\}$
$S = \{2, 14, 15\}$
$S_2 \wedge \overline{X}_{14}$
$S = \{1, 6\}$
$X_1 \wedge \overline{X}_{16}$
$S = \{14, 15\}$
$\overline{X}_{14} \wedge \overline{X}_{15}$

So, can run Perc. over space of all conj. in time $O(k^n)$ per trial if there were $k$ mistakes so far.

Powerful augmentation of Perc. alg.

Art to choosing kernel function.

Next time: generic bds for online learning.