Lecture 5: Learning from data.

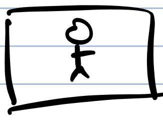
Previously: we have a dataset X, can we learn about X?
e.g. nearest neighbor, heavy hitters.

Today: we have a dataset X that is representative of some underlying process. Can we learn about this underlying process?

$\boxed{\text{This is the basic question underlying ML!}}$

e.g. image classification

$X = \{ ($ $, \text{"cat"}),$

$($  $, \text{"human"}),$

$\vdots$

$($ $, \text{"cat"}) \}$

$\Downarrow$

$f :$ new image $\rightarrow \{\text{"cat"}, \text{"human"}\}$

Binary classification: D is a distribution over $\mathbb{R}^d$
(Unknown) ground truth $f : \mathbb{R}^d \rightarrow \{0,1\}$.
$x_1, \dots, x_n \sim D$ are independent draws from D.
Labels: $y_i = f(x_i)$, $i = 1, \dots, n$.

Input: training set $\{(x_1, y_1), \dots, (x_n, y_n)\}$

Output: A predictor $g : \mathbb{R}^d \rightarrow \{0,1\}$

Goal: $g = f$, or at least, g "looks like" f for typical points in D. $\longrightarrow$ quantified via generalization error.

Def:
$$\text{GenError}(g) = \Pr_{x \sim D}[g(x) \neq f(x)]$$

How to evaluate gen. error? use test set

In addition to the training set, maintain separate test set $\{(x_1', y_1'), \dots, (x_m', y_m')\}$ that the predictor cannot see (i.e. is not trained on).

Then $\forall\, i = 1, \dots, m$

$$\Pr[g(x_i') \neq y_i'] = \Pr_{x \sim D}[g(x) \neq f(x)]$$
$$= \text{GenError}(g)$$

$$\Rightarrow \text{GenError}(g) \overset{\uparrow}{\underset{\text{Chernoff}}{\approx}} \underbrace{\frac{1}{m} \#\{i : g(x_i') \neq y_i,\ i = 1, \dots, m\}}_{\text{"test error"}}.$$

We can also define training error
$$\text{TrainError}(g) = \frac{1}{n} \#\{i : g(x_i) \neq y_i\ , i = 1, \dots, n\}$$

$$\boxed{\text{Important: } \text{TrainError} \neq \text{TestError}}$$

Overfitting: TrainError is small, but TestError is large.

Q: When do models generalize or overfit?

Finite, well-separated case:

$\mathcal{A}$:= finite set of predictors of size k

$\mathcal{A} = \{f_1, \cdots, f_k\}$

Assume: ① (Realizable) ground truth $f \in \mathcal{A}$

② (Well-separated) $\forall f_i \neq f$ in $\mathcal{A}$, we have

GenError$(f_i) \geq \varepsilon$.

$\downarrow$

$$\Pr_{X \sim D}[f_i(X) \neq f(X)] \geq \varepsilon.$$

Let $(x_1, y_1), \cdots, (x_n, y_n)$ be a training set

↳ $x_i \sim D$ independent,

$y_i = f(x_i)$.

Q: How large does n have to be to learn f?

Meta-Algo: Output any $g \in \mathcal{A}$ s.t.

TrainError$(g) = 0$.

Claim: For any $\delta > 0$, let $n \geq \frac{1}{\varepsilon}(\log k + \log 1/\delta)$

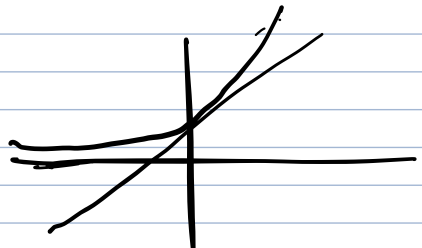
Then $\Pr_{x_1, \cdots, x_n}[g \neq f] \leq \delta$.

$\nearrow$ tight.

pf: fix some $f_j \neq f$

$$\Pr_{x_1, \cdots, x_n}[\text{TrainError}(f_j) = 0] = \prod_{i=1}^{n} \underbrace{\Pr[f_j(x_i) = f(x_i)]}_{\leq (1-\varepsilon)}$$

$\forall x: (1+x) \leq e^x$



$\leq (1-\varepsilon)^n$

$\leq e^{-\varepsilon n}$

$= e^{-\varepsilon \cdot \frac{1}{\varepsilon}(\log k + \log 1/\delta)}$

$$= \delta / k.$$

Now, by union bound:

$$\Pr[\exists f_i \neq f : \text{TrainError}(f_i) = 0]$$
$$\leq \sum_{f_i \neq f} \Pr[\text{TrainError}(f_i) = 0] \leq \frac{\delta}{k} \cdot (k-1)$$
$$\leq \delta.$$

$\Rightarrow$ w.p. $\geq 1-\delta$, only $g \in \mathcal{A}$ w/ $\text{TrainError}(g) = 0$ is $g = f$.

What about not well-separated?
In general, can't hope to recover f!
But, maybe you can get something that acts like f for most points in D.

<u>Probably Approximately Correct (PAC) learning</u> [Valiant 8

Let $\varepsilon, \delta > 0$. Given training data, find $g \in \mathcal{A}$

$$\text{GenError}(g) = \Pr_{x \sim D}[g(x) \neq f(x)] \leq \varepsilon$$

with probability $\geq 1-\delta$.

For this, the same proof works!

$$n \geq \frac{1}{\varepsilon}\left(\log k + \log 1/\delta\right)$$

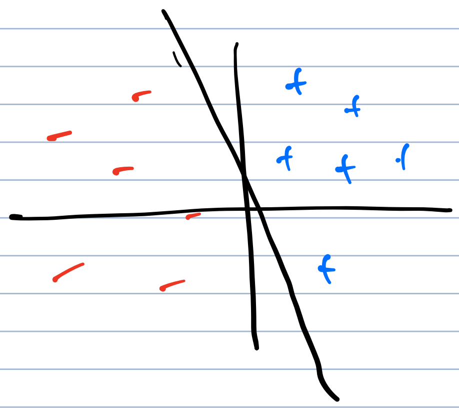
What about infinite families $\mathcal{A}$?

$\mathcal{A}$ = any function? Impossible!

need structure

Linear classifiers: parameter: $\theta \in \mathbb{R}^d$

$$f_\theta(x) = \text{sign}(\langle \theta, x \rangle)$$
$$= \begin{cases} 1 & \text{if } \langle \theta, x\rangle \geq 0 \\ 0 & \text{o.w.} \end{cases}$$



Obs 1: Since we only care about angles, wlog can assume $\|\theta\|_2 = 1$.

But there are still infinitely many unit vectors...

But "only" exponentially many distinct ones!

Recall curse of dimensionality: can fit exponentially many small balls into unit ball.

tight

$k = \#$ "distinct" directions $\approx C^d$

$\Rightarrow \quad n \geq \frac{1}{\varepsilon}(d \log C + \log 1/\delta)$ suffices!

Can slightly improve this:

Vapnik-Chervonenkis theory: (VC theory).

$$n \geq \frac{1}{\varepsilon}(d + \log 1/\delta).$$

Rule of thumb: # samples $\geq$ # free parameters

$\rightarrow$ good generalization

not necessary! e.g. deep networks.

What if $f \notin \mathcal{A}$? "agnostic learning"

i.e. what if optimal error > 0?

Goal: Given $\varepsilon, \delta > 0$, output $g \in \mathcal{A}$ s.t.

$$\Pr_{X \sim D}[g(X) \neq f(X)] \leq \text{OPT} + \varepsilon$$

$$\text{OPT} = \min_{g^* \in \mathcal{A}} \Pr[g^*(X) \neq f(X)]$$

with probability $\geq 1-\delta$.

Meta-Algo: empirical risk minimization (ERM)

output $g \in \mathcal{A}$ with smallest training error

When $\mathcal{A}$ = linear classifiers,

still linear in # params

$$n \geq \Omega\left(\frac{1}{\boxed{\varepsilon^2}}(d + \log 1/\delta)\right)$$

why ε^2? How many coin flips do you need to distinguish a coin w/ bias p vs $p+\varepsilon$?

Realizable: $p=0$ $\rightarrow$ just need to see 1 tails occurs whp after $\geq 1/\varepsilon$ throws.

Agnostic: $p = \frac{1}{2}$ $\rightarrow$ need to see many tails to distinguish. $\rightarrow$ $1/\varepsilon^2$ throws.

Algorithmic aspects: How do we compute the ~~the~~ best fit classifier? say for linear classifiers.

Realizable: poly-time (e.g. support vector machine (SVM), linear programming).

Agnostic: NP-hard :(
use heuristics!
(e.g. linear programming, stochastic gradient descent (SGD))

Generalizing beyond linear?
there are many, many types of $\mathcal{A}$ used in practice.

However many have a same basic structure:
$k: \mathbb{R}^d \to \mathbb{R}^D$, $D \gg d$,
k is "kernel" map, and apply linear classifier in $\mathbb{R}^D$ to lifted data.

e.g. k = polynomials, neural networks.

Intuition: k maps data to a set of useful features, and just apply simple classifier on top.