FoSML: Lecture 1

- ML: using data (either with or without labeled exemplars) to do some kind of labeling or prediction. (Close to statistics & C.S., and arguably a subset of both.

- Typical problems:

1) Regression: Given some pairs of inputs and outputs to some unknown function $f(x)$, call it $\{(x_i, y_i)\}_{i=1}^n$, $x_i \in \mathbb{R}^D$, $y_i \in \mathbb{R}^m$, compute a "good" prediction function $\hat{f}: \mathbb{R}^D \to \mathbb{R}^m$ s.t. $\hat{f}(x) \approx f(x)$ on new $x \neq x_i$, $i = 1, \dots, n$.

2) Classification: Replace $\mathbb{R}^m$ with $\{1, \dots, K\}$ above. Here, there is no topology on the range of f, i.e. x_i, x_j are not necessarily close if $f(x_i), f(x_j)$ are close but not equal.

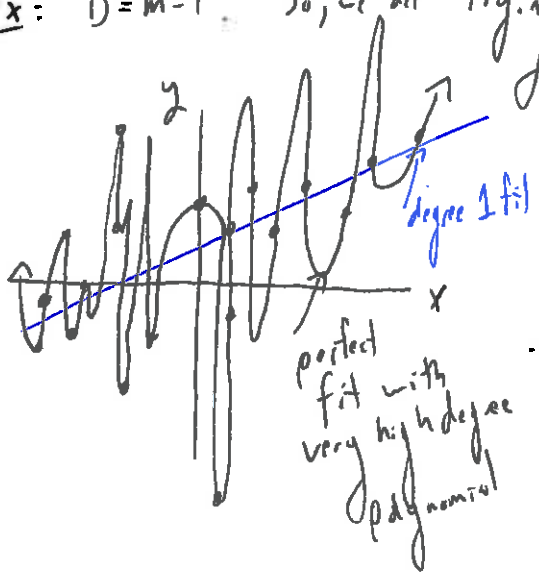
3) Clustering: Determine the labels to data $\{x_i\}_{i=1}^n$ with no y_i's to help. Basically the classification problem, but with no $\{y_i\}_{i=1}^n$ to help!

- When the $\{y_i\}_{i=1}^n$ (i.e. outputs to the hidden function f or labels) are available, the problem is supervised. When no labels are available, unsupervised.

- The bulk of the class will focus on supervised learning, which is better understood and easier to formalize mathematically.

- One approach might be to simply define the regression function in such a way that it memorizes the training data. There are infinitely many such functions that do this.

ex: $D = m = 1$. So, we are trying to fit a curve $f: \mathbb{R} \to \mathbb{R}$ to data $\{(x_i, y_i)\}_{i=1}^n$.



- Clearly I can fit a polynomial of sufficiently large degree $(n-1)$ to these points exactly.
- But, does that seem like it really captures the underlying pattern? To phrase in a more "organic" manner: are we learning the latent function, or simply overfitting?
- This suggests that may using high degree polynomials is "too flexible", in the sense that we doubt that the underlying trend requires high degree polynomials to parsimoniously capture.
- Mathematically, this leads to a notion known as the hypothesis class for the learning problem, namely, where do our candidate functions f live?
- In the above examples, the "best fit" (to be formally defined soon) differs depending on which hypothesis class is used. Of course, the high degree polynomial performs perfectly on the observed data $\{(x_i, y_i)\}_{i=1}^n$, i.e. it has low training error.
- But, what we really care about is how the learned function performs on unseen testing data. We want to learn something, not just memorize.

Quantitatively, we care not about training error, but generalization to new data

· A typical (and simple) set-up for learning a regressor/classifier would be to specify a hypothesis class $\mathcal{H}$ and a loss function L and compute

$$\hat{f} = \underset{f \in \mathcal{H}}{\arg\min}\ L\left(\{(f(x_i), y_i)\}\right), \quad (\star)$$

i.e. choose the best $f \in \mathcal{H}$ for minimizing the error/loss between the predicted outputs $(f(x_i))$ and the observed outputs (y_i).

ex: · $\mathcal{H}$ = degree 1 polynomials, i.e. $f \in \mathcal{H} \Leftrightarrow \exists\, a, b \in \mathbb{R}$ s.t. $f(x) = ax + b$. ($D = m = 1$)

· $L\left(\{f(x_i), y_i)\}\right) = \frac{1}{n}\sum_{i=1}^{n} (f(x_i) - y_i)^2$.

Then, $(\star)$ becomes $\hat{f} = \underset{f(x)=ax+b}{\arg\min}\ \frac{1}{n}\sum_{i=1}^{n} (f(x_i) - y_i)^2$

$$= \underset{a,b \in \mathbb{R}}{\arg\min}\ \frac{1}{n}\sum_{i=1}^{n} (ax_i + b - y_i)^2$$

This is the classical least squares problem. It is (relatively) easy to compute a, b explicitly from $\{(x_i, y_i)\}$ (try it!) We will discuss higher dimensional notions later in the course (they are not much harder).

· So, given data, a hypothesis class, and a loss, what's the best we can do in terms of both training error and (more importantly) generalization error?

- To analyze these notions rigorously, we introduce the notion of "probably approximately correct (PAC)". The idea is we only get some training data which may be a dud (probably), and at best is likely to only get us a good estimate for the "true" predictor (approximately). We must introduce some machinery:

- Let: - X be the set of all possible inputs, distributed according to the distribution D.
  - Y be the set of all possible outputs
  - Let $c: X \to Y$ be a concept
  - A collection of concepts C is a concept class.
  - As above, H is a hypothesis class, which may be understood as a different concept class than C.

Defn (Generalization error): Given a hypothesis $h \in H$, a target concept $c \in C$, and a distribution D over the input space X, the generalization error of h is

$$R(h) = \mathbb{P}_{x \sim D}\left(h(x) \neq c(x)\right) = \mathbb{E}_{x \sim D}\left(\mathbb{1}_{\{h(x) = c(x)\}}\right).$$

- Here, $\mathbb{1}_A(x) = \begin{cases} 1, & x \in A \\ 0, & x \notin A \end{cases}$ is the indicator function of the set A.

- Let us assume for simplicity that $Y = \{0,1\}$, i.e. we are doing binary classification.

- The quantity $R(h)$ is measuring how likely h is to differ from the (hidden) concept c. We consider $x \sim D$ to account for the latent distribution over X.

Remark: Some concepts c may be easier than others. That is, as c changes, the "best" $h \in H$ may yield worse (i.e. higher) error.

· In practice, D is unknown, so $R(h)$ is unknowable exactly. Instead, the supervised ML framework provides empirical data $\{(x_i, y_i)\}_{i=1}^n$. We will re-phrase this as $\{(x_i, c(x_i)\}_{i=1}^n$ for a given concept class $c: X \to Y$.

Defn (Empirical Error): Given a hypothesis $h \in H$, a target concept $c \in C$, and a sample $S = \{x_i\}_{i=1}^n$, the empirical error of h is $\hat{R}_S(h) = \frac{1}{n} \sum_{i=1}^n \mathbb{1}_{h(x_i) \neq c(x_i)}$

$$= \frac{1}{n} \cdot \left| \{ x_i \mid h(x_i) \neq c(x_i) \} \right|.$$

· We hope that the empirical error is a good surrogate for the generalization error.

Remark: Suppose we sample from X according to D. Then,

$$\mathop{\mathbb{E}}_{S \sim D} \left(\hat{R}_S(h) \right) = \mathop{\mathbb{E}}_{S \sim D} \left(\frac{1}{n} \sum_{i=1}^n \mathbb{1}_{h(x_i) \neq c(x_i)} \right)$$

$$= \frac{1}{n} \sum_{i=1}^n \mathop{\mathbb{E}}_{S \sim D} \left(\mathbb{1}_{h(x_i) \neq c(x_i)} \right)$$

$$= \frac{1}{n} \sum_{i=1}^n \mathop{\mathbb{E}}_{S \sim D} \left(\mathbb{1}_{h(x) \neq c(x)} \right)$$

$$= \mathop{\mathbb{E}}_{S \sim D} \left(\mathbb{1}_{h(x) \neq c(x)} \right)$$

$$= R(h).$$

So, when sampling from X, the average (i.e. expected) empirical error equals the generalization error. The challenge is, we only get one sample from X.

- To introduce PAC learning, we need a bit more machinery.
- Let N be s.t. the comp. cost of storing any $x \in \bar{X}$ is $O(N)$
- Let size(c) be the maximal cost of storing any $c \in \mathcal{C}$.
- h_S is the hypothesis returned by an algorithm given samples S

Defn (PAC Learning): A concept class $\mathcal{C}$ is PAC-learnable if $\exists$ an algorithm $\mathcal{A}$ and a polynomial $\text{poly}(\cdot,\cdot,\cdot,\cdot)$ s.t. $\forall \varepsilon, \delta > 0$, $\forall$ distributions D on X and $\forall c \in \mathcal{C}$,

$$m \geq \text{poly}\left(\frac{1}{\varepsilon}, \frac{1}{\delta}, N, \text{size}(c)\right)$$

$$\Downarrow$$

$$\mathbb{P}_{S \sim D}\left[\underbrace{R(h_S) \leq \varepsilon}_{\text{approximately}}\right] \geq \underbrace{1-\delta}_{\text{probably}},$$

Remark: This needs to hold for any distribution D on $\bar{X}$, and note that both the training (to compute h_S) and testing ($\mathbb{P}_{S \sim D}$) are based on the same distribution D.