ML & Optimization

Lecture 1 8/30/2021

- Logistics
- What is optimization?
- What is ML?
- Example ML problems & issue of scalability

What is Optimization

$$J : \mathcal{X} \rightarrow \mathbb{R}$$

$$x^* = \underset{x \in \mathcal{X}}{\operatorname{argmin}} \; J(x)$$

$\mathcal{X}$ – some set (e.g. hypothesis space)

J – objective function

x^* – "best" element in $\mathcal{X}$

n.b. we can also equivalently find x^* as the solution to a maximization problem

$$x^* = \underset{x \in \mathcal{X}}{\operatorname{argmax}} \; -J(x)$$

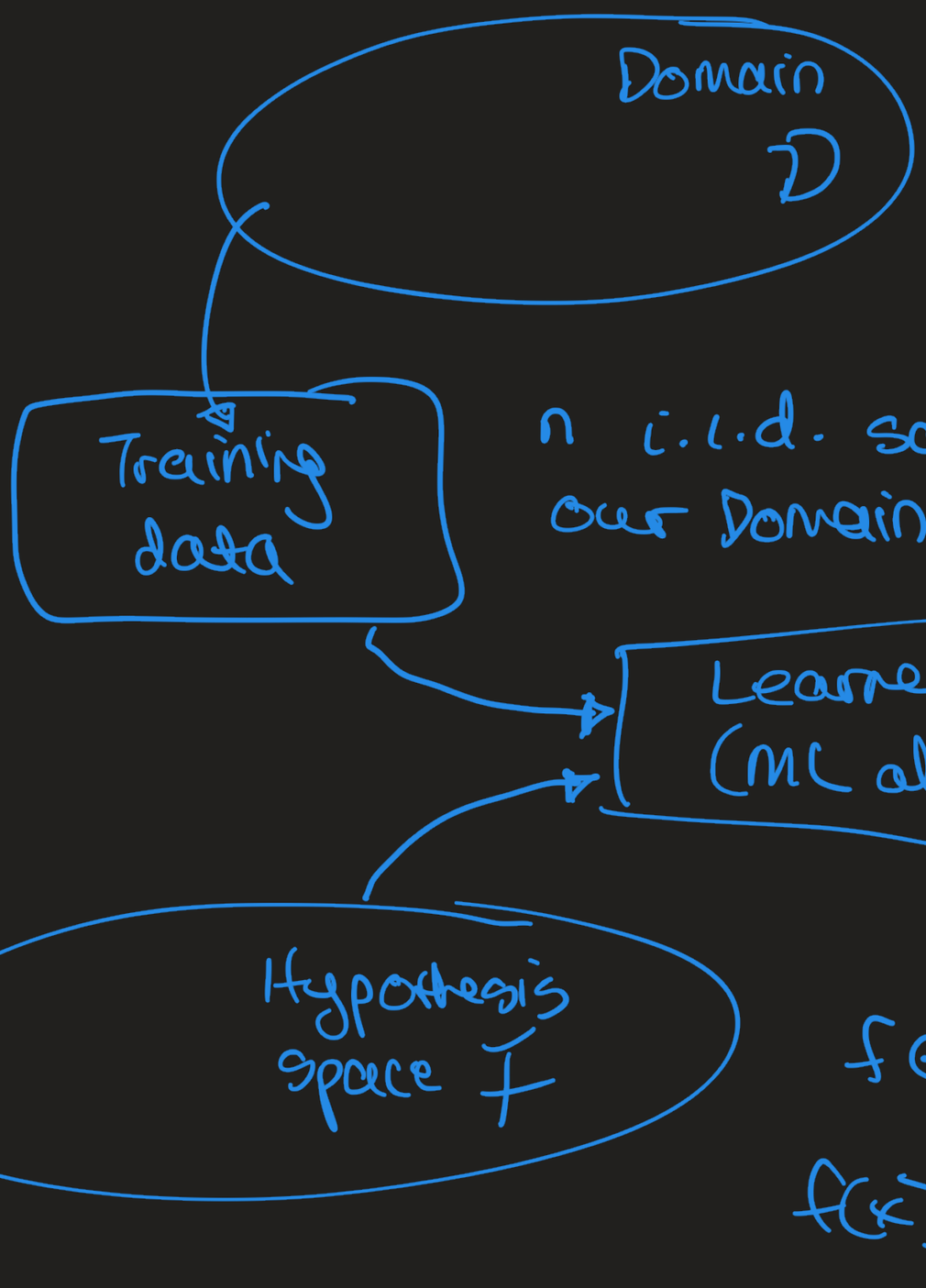
in terms of difficulty, maximization & minimization are the same

Challenges of optimization?

- constraints
- ambiguity in matching your task w/ a specific objective function
- these problems are extremely difficult when there isn't a specific structure we can exploit
- scale of the problem

What is Machine Learning?

"finding a pattern from a set of data"



$(x, y) \in \mathcal{D}$

y ← salary in first job after grad.

x ↑ features of a student

n i.i.d. samples from our Domain

f^* optimal hypothesis that explains our training data (and therefore, we hope, our entire domain)

$f \in \mathcal{F}$ - a hypothesis, regressor, predictor, classifier

$f(x)$ predicts

What do I mean by an optimal hypothesis?

l – loss function $\quad l(\cdot,\cdot)$

$f(x)$ – prediction of a model on an input

y – true label for that input

$$f^* = \underset{f \in \mathcal{F}}{\operatorname{argmin}} \; \mathbb{E}_{\mathbb{P}} \, l(f(x), y)$$

$$\approx \underset{f \in \mathcal{F}}{\operatorname{argmin}} \; \frac{1}{n} \sum_{i=1}^{n} l(f(x_i), y_i)$$

Examples

unsupervised: k-nearest neighbors (knn)

supervised: support vector machines (svms)

A high-level overview of the ML process:

- formulate a task
- collect appropriate data
- choose an appropriate model class for my problem
- formulate an optimization problem to fit the model on my data
- solve the optimization prob (potentially after modifying it to be easier to solve)

Ex: k-nn

Problem: given a data x from $\mathcal{D}$ predict the label $y \in \{-1, 1\}$

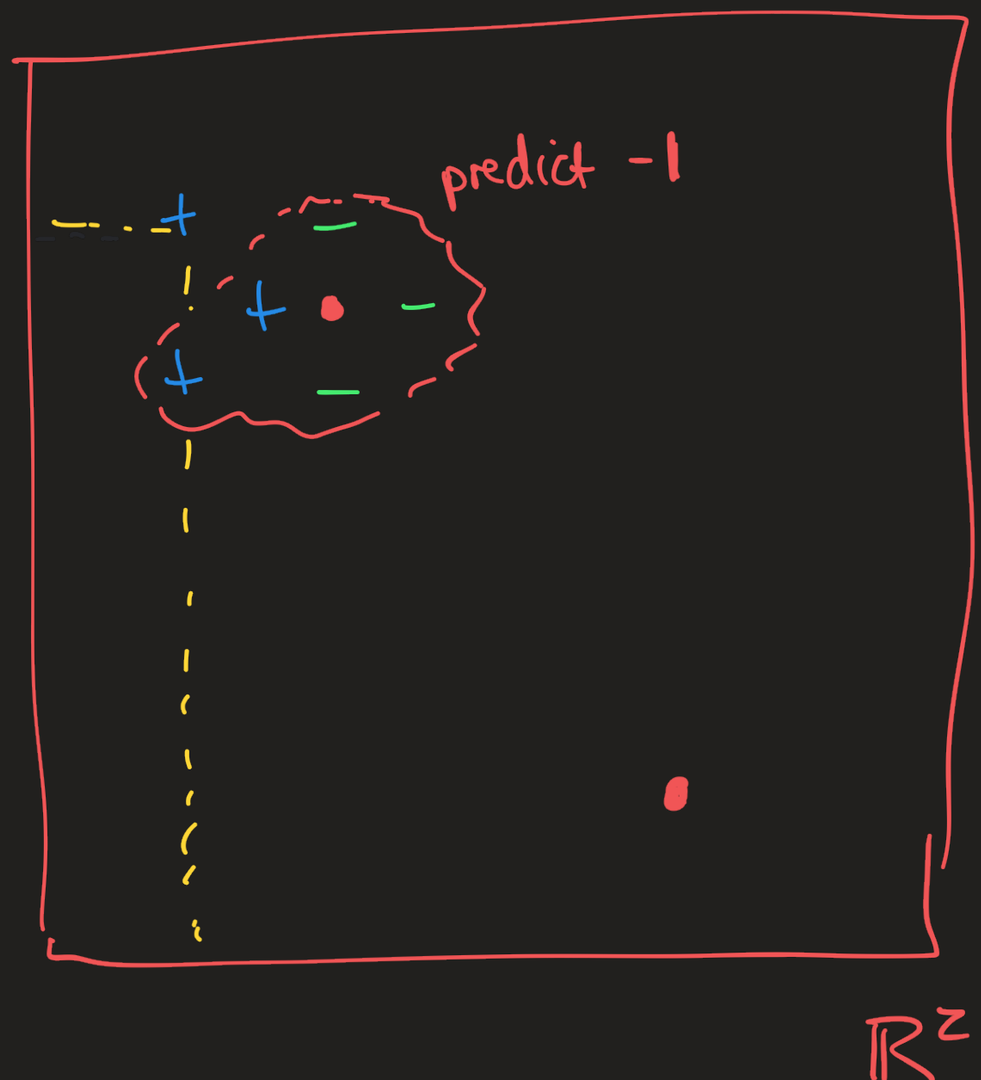
Data: $X = \{(x_i, y_i)\}_{i=1}^{n}$ where $x_i \in \mathbb{R}^d$ and $y_i \in \pm 1$

Model: k-nn

$$f(x) = \text{sign}\left(\sum_{x_j \in N_k(x)} y_j\right) \qquad \text{sign}(x) = \begin{cases} 1 & x > 0 \\ -1 & x < 0 \\ 0 & x = 0 \end{cases}$$

$N_k(x) = \{x_j :$ x_j is one of the k nearest neighbors of x in the training data set $\}$

k-nn $\quad k=5$



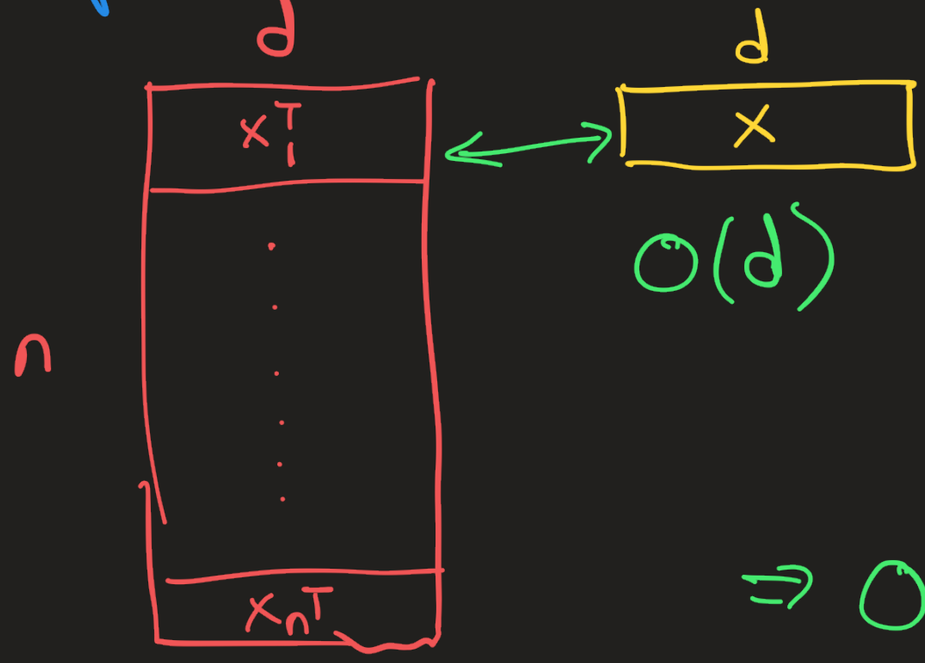
$O(n^d)$

Optimization — given a new point x, find $h_k(x)$ and compute $f(x)$

Cost of naive k-nn

$O(nd + n\log n + k)$ ← very expensive if $n \gg 1$, $d \gg 1$

- compute distance of x from all n training data pts



$\|x - x_i\|_2$

$\Rightarrow O(nd)$ operations to compute distance of x to all training data points

$X \in \mathbb{R}^{n \times d}$

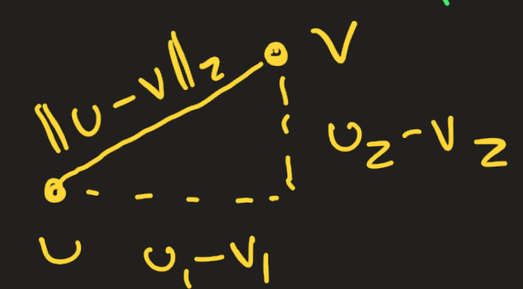
- Find the k smallest distances to determine $\mathcal{N}_k(x)$

  $O(n\log n)$

- Sum over $\mathcal{N}_k(x)$ to compute $f(x)$

  $O(k)$

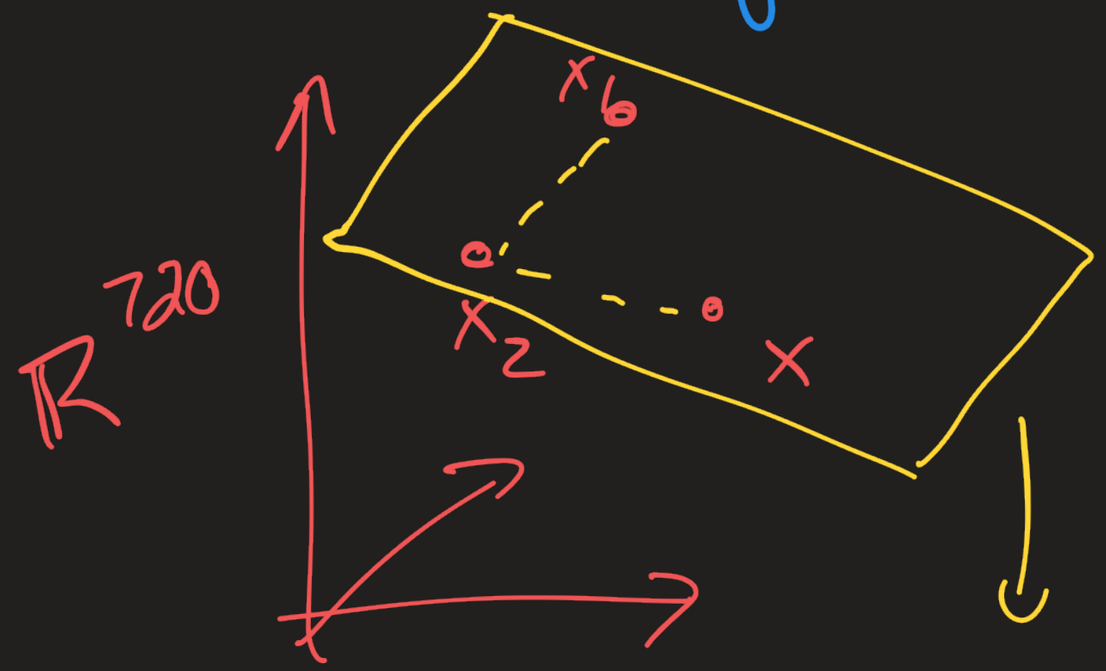
$$\|u - v\|_2^2 = \sum_{i=1}^{d} (u_i - v_i)^2$$

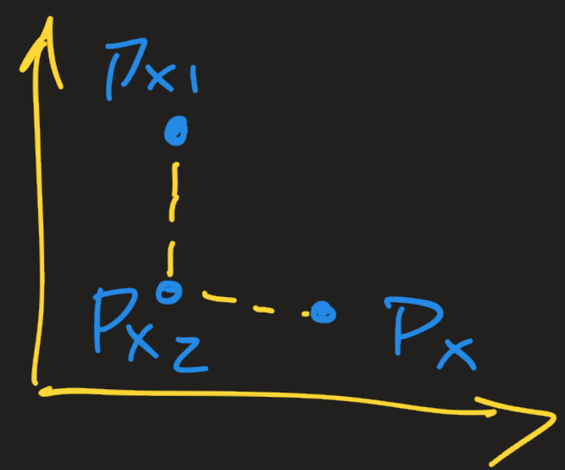


Idea for reducing cost

We use what is called "approximate nearest neighbors"

Project our datapoints from $\mathbb{R}^d$ down to $\mathbb{R}^c$ where $c \ll d$, and then find the nearest neighbors there.





Approximate Nearest Neighbors for k-nn

- Sample a random matrix $P \in \mathbb{R}^{c \times d}$ where $c = O(\log n)$
- Compute the distances $\|Px_i - Px\|_2^2 \approx \|x_i - x\|_2^2$
- Select $N_k(x)$ using the distances $\|Px_i - Px\|_2^2$
- Compute $f(x)$

Offline component: compute P, Px_i for all training data points

$$O(cd) + O(ncd)$$

- $O(cd)$: computing P
- $O(ncd)$: computing Px_i for all training pts

Online : compute P_x, compute $\|P_x - P_{x_i}\|_2$ for all training data, compute $\mathcal{V}_k(x)$, compute $f(x)$

$$O(cd) + O(nc) + O(n \log n) + O(k)$$

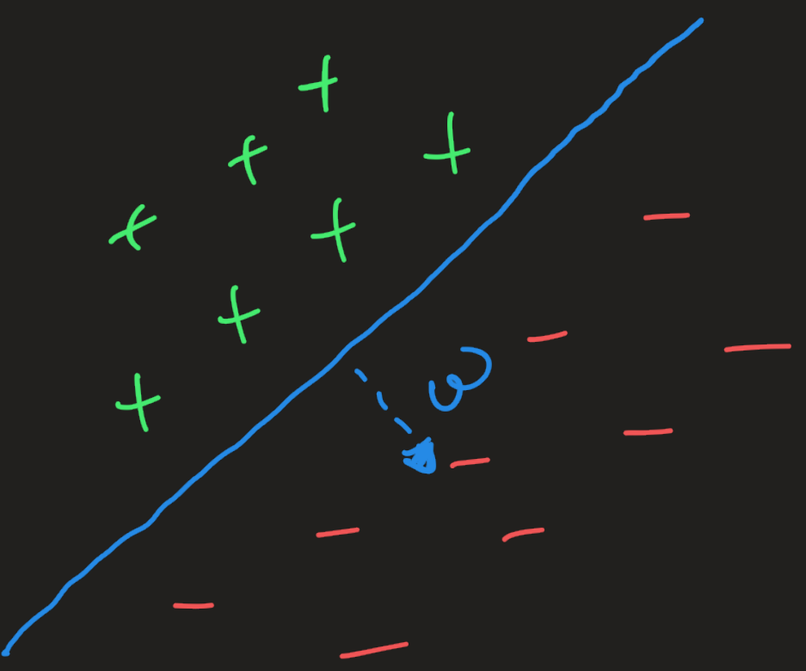
- $O(cd)$: P_x
- $O(nc)$: $\|P_x - P_{x_i}\|_2$
- $O(n \log n)$: $\mathcal{V}_k(x)$
- $O(k)$: $f(x)$

Comparison: (online portion)

Naive knn : $O(nd)$

ANN knn : $O(nc)$ where $c \ll d$

Ex: Binary classification using SVMs



$$\mathcal{F} = \{ f : f(x) = \text{sign}(\omega^T x) \text{ for a } \omega \in \mathbb{R}^d \}$$

To choose our optimization problem for fitting an SVM

- penalize the model if y_i and $\langle \omega, x_i \rangle$ have different signs
- penalize the model if $y_i \langle \omega, x_i \rangle < 1$
- avoid trivially confident models (do not let $\|\omega\|_2 \to \infty$)

Let $\phi(t) = \max(1-t, 0) = (1-t)_+$

let our loss function $l(f(x_i), y_i) = \phi(y_i \langle w, x_i \rangle)$

Two setups for the optimization prob of learning an SVM using the hinge-loss:

1) $w^* = \underset{\|w\|_2 \leq C}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \phi(y_i \langle w, x_i \rangle)$ (Constrained optim)

2) $w^* = \operatorname{argmin} \frac{1}{n} \sum_{i=1}^{n} \phi(y_i \langle w, x_i \rangle) + \lambda \|w\|_2^2$ (Regularized Optim)