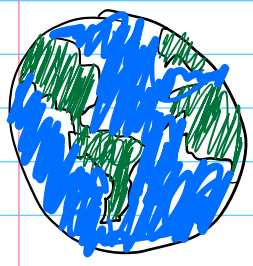
CS 3/5780
Machine Learning Setup

Supervised Learning:



stuff we know

feature extraction → Input $\vec{x} \in \mathbb{R}^d$ → Model/Program/Hypothesis h → Output y

Universe of all data we care about

Represents part of instances we deem relevant for predicting output

label/output we would like to predict

Exercise: What would you use as $\vec{x}$ for the following prediction tasks?

1) Is a given email is spam or not?
2) Price of Apple stock tomorrow
3) Where will Jupiter he in night sky tomorrow
4) Is there a pedestrian in front of the car?

How do we find h?

- Traditional CS Approach: Pay Programmer to program h somehow
- Learning Approach : Learn h from examples (data)

To learn we need the following:

Lateled Data:
$D = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n)\}$

Model class $\mathcal{H}$
a set of functions
$h : \mathcal{X} \to \mathcal{Y}$

Loss function
Tells us how good a model h did on instance (x,y)

Algorithm
To pick good h based on D

1. Data:

a. Multiple scenarios for y:

- Binary classification: $y = \{0, 1\}$ or $y = \{-1, +1\}$

  Eg. spam filtering, An email is either spam $y = +1$ or not spam $y = -1$

- Multiclass classification: $y = \{1, \ldots, K\}$

  Eg: Pet in image:

  1 = "cat" 2 = "dog" 3 = "guniea pig" 4 = "snake"

- Regression $y = \mathbb{R}$

  Eg: Predict price of a house on market

b. $\vec{x_i}$ is a feature vector of d dimensions describing the ith sample

- Text document in bag-of-words format: $d = ?$ properties of $\vec{x}$

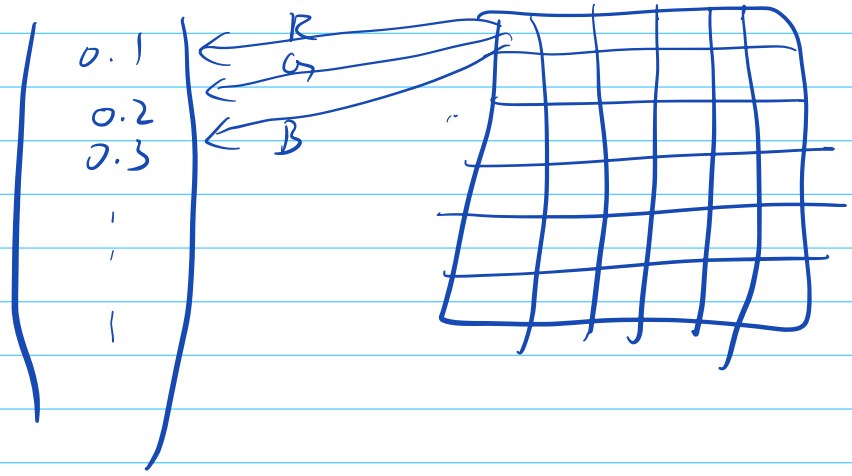
$$\vec{x_i} = \begin{pmatrix} 10 \\ 1 \\ \vdots \\ 0 \end{pmatrix} \begin{array}{l} \leftarrow \text{occurrences of word "the"} \\ \leftarrow \text{occurrences of word "like"} \\ \\ \\ \leftarrow \text{occurrences of word "zebra"} \end{array}$$

- house details

$$\vec{x_i} = \begin{bmatrix} \text{Sq. ft} \\ \#\text{ of bedrooms} \\ \#\text{ Bathrooms} \\ \text{Year} \\ \text{zip code} \end{bmatrix} = \begin{bmatrix} 2000 \\ 4 \\ 2 \\ 2011 \\ 14850 \end{bmatrix}$$

$d = ?$

- Images:

$\vec{x}_i =$ 

3. Loss function : ℓ

- Classification loss: $\ell(h(\vec{x}), y) = \delta_{h(\vec{x}) \neq y}$

  ↳ $\begin{cases} 1 & \text{if } h(\vec{x}) \neq y \\ 0 & \text{o/w} \end{cases}$

- Square loss $\ell(h(\vec{x}), y) = (h(\vec{x}) - y)^2$

- Absolute loss $\ell(h(\vec{x}), y) = |h(\vec{x}) - y|$

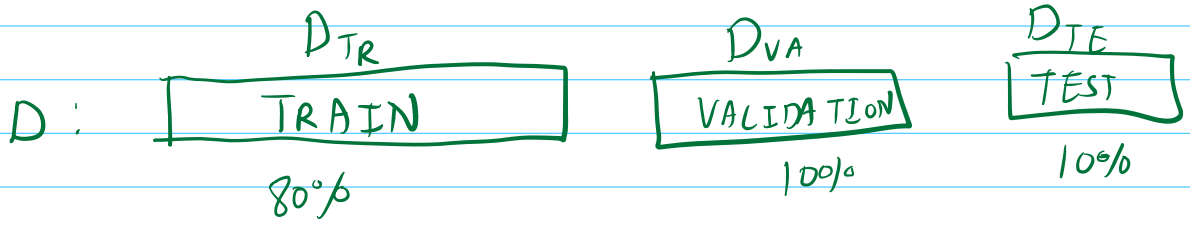
Generalization:

Idea : find a model with low loss on D

Eg. Algorithm, the memorizer ??!

$$h(\vec{x}) = \begin{cases} y_i & \text{if } \exists (\vec{x}_i, y_i) \in D \text{ st. } \vec{x}_i = \vec{x} \\ 0 & \text{otherwise} \end{cases}$$

Train/test split: Split data into train test and validation



Choose h based on D_{Tr} (+ D_{va}) and evaluate on D_{Te}

Why do we need D_{va}?

How to split data:

- Test must simulate Deployment scenario
  - Eg: Data drawn iid (or not temporal)
    split uniformly at random
  - Eg: Spam filter (train sys on past data)
    split train/test temporally
    (only use past to predict next steb)

Non-temporal often modeled as $(\vec{x}, y) \sim P$

Error at Deployment:

Risk: $\mathop{E}_{(x,y)\sim P} \ell(h(\vec{x}), y)$
/Population loss

Goal of ML: Pick h based on D to minimize Risk

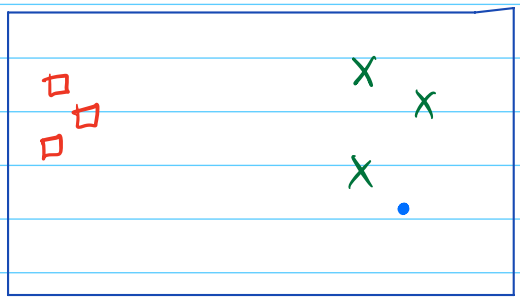
Training Loss: $\frac{1}{|D_{Tr}|} \sum_{(\vec{x},y)\in D_{Tr}} \ell(h(\vec{x}), y) = \mathcal{E}_{Tr}(h)$

Test Loss: $\frac{1}{|D_{Te}|} \sum_{(\vec{x},y)\in D_{Te}} \ell(h(\vec{x}), y) = \mathcal{E}_{Te}(h)$

Validation Loss: $\frac{1}{|D_{va}|} \sum_{(\vec{x},y)\in D_{va}} \ell(h(\vec{x}), y) = \mathcal{E}_{va}(h)$

For a model $\hat{h}$ spit out by learning Algorithm, what is a good proxy for Risk? why?

No free Lunch: You must make assumptions in order to learn. No Algo. Works in all settings



Every ML Algo makes Assumptions!