What is learning?

It can't just be regurgitation, otherwise you'd need to see everything, making learning useless

Do not begin to create a model until you verify the following:

- A pattern exists.
- We don't know it
- We have data to learn it.

Lecture 2

Classification:

Decision: yes/no

Formal setup:

1. $x \in R^d \leftarrow$ input
2. $y \in \{-1, 1\} \leftarrow$ output
3. $f(x) = y \leftarrow$ tangent function [FIXED UNKNOWN]

Note that the input/output can be pretty much anything

"ThE pAtTerN ExiSts, iT's UnKnoWn"

"The trees are jiggling and wiggling"

Outputs an approximation to f(x) called g(x) that you want to be close to the data

You can check this against historical data in a given dataset (D) which consists of a bunch of data points from x_1 to x_N called "examples", and their outputs (y = f(x)) y_1, y_2, y_N

Once you get a g that approximates f well enough, you can say that g approximates f, and it is good enough

Example: if a bank only gives people approved for credit loans, then feeds the data on how many people default after getting said loans, what happens when someone who has not been approved for credit applies? The model won't know what to do

We can say that data is "as given", like as is

The quality of g MUST be validated OUTSIDE the data

Construct a hypothesis set H consisting of h_1, h_2, h_3, etc which you b

This is called “testing data”, wow. Amazing

Example: credit approval

x = [salary, debt, age, hair color, years in residence] ∈ R^5

If score > threshold: yes (+1)

If score < threshold: no (-1)

So you need to come up with a system for coming up with the score and the threshold

Certain attributes should be assigned less value than others

For example, hair color is an indicator of age, so it should be worth less than age. Debt should remove points (as it’s bad), etc

So you can make smth like 10 * salary - 10 * debt + 2 * age + 0 * hair color + 3 * years in res

w_1 = 10

w_2 = -10

w_3 = 2

…..

+1 if $\sum_{i=1}^{d} w_i x_i$ > threshold

-1 if $\sum_{i=1}^{d} w_i x_i$ < threshold

The above two things are the output (y) which equals the sign of $\sum_{i=1}^{d} w_i x_i - threshold$

We replace the threshold with “- w0” to get $\sum_{i=1}^{d} w_i x_i + w_0$

Define x_0 as 1 to make summing easier

The human should choose the w_0, making $y = sign(\sum_{i=0}^{d} w_i x_i)$

This sum is the product of two vectors: $w^T x$

Note that w is the vector of weights and x is the vector of data points

$$X = \begin{pmatrix} X_0 \\ X_1 \\ \cdot \\ \cdot \\ \cdot \\ X_d \end{pmatrix} \qquad W = \begin{pmatrix} W_0 \\ W_1 \\ \cdot \\ \cdot \\ \cdot \\ W_d \end{pmatrix}$$

So $y = sign(w^T x)$

We can make a hypothesis set like so: $H = \{h \mid h(x) = sign(w^T x)\ w \in R^{d+1}\}$

This is called the perceptron hypothesis set

This set can be wrong for a few reasons. First of all, some w_* is wrong. Alternatively, the true hypothesis set might just be completely wrong, like the true function might not be $sign(w^T x)$

To pick a good g from H, we have to do two things:

1. Figure out on what basis to pick (criteria)
2. Algorithm to pick

When y = 0 (in this case), this determines where the line is that splits what is approved vs what is not approved (above is +, below is -), which is called the **classification boundary**

If the line is closer to more data points, then it's better, AKA a line of best fit

Learning vs Design

Learning:

- based on data

Design:

- does not use datasets
- Problem is well-specified and we can derive *f* without needing to see any data
- The program will be given specifications and data about the object then search for a hypothesis that best fits the data
    - For example, in recognizing coins, we can give the program the size and mass of each type of coin, a well as the number of the said coin in current circulation. The program will then use that to classify the coins.

- We can calculate the time it will take an object to hit the ground without extensive trials and datasets by using the laws of physics

Supervised, Unsupervised, and Reinforcement learning

Supervised:

- We can check the input against the output manually
  - Example: when classifying images of numbers, we can have a set of images and a set of corresponding numbers as answers
- There are multiple ways that a dataset can be represented during the learning process
  1. Active learning:
     - Data is acquired through queries that we make
     - We get to choose a point x in the input space, and the supervisor reports to us the target value for x
     - This lets us strategically use data points, similar to a game of 20 questions
  2. Online learning,
     - An algorithm us given data 1 example at a time
     - This lets the algorithm receive data as a stream and process it in real time
     - For example, new movies can be added to the algorithm as they come in
     - Online learning can also be useful when we have limitations on computing and storage the preclude us from processing all the data at once

Unsupervised:

- The training data does not contain any output info at all
- Only given inputs x_1, x_2, …, x_N
- This is helpful for finding spontaneous patterns in data
  - Example: if our task is to categorize a set of books into topics, and we only use general properties of various books, we can identify books that have similar properties and put them together in one category without naming the category
- Can be used to gain a higher-level representation of data
  - Example: Imagine you are going to Spain, but you don't know the language. All you have access to is a Spanish radio, so you just constantly listen to it,

developing a better representation of the language in your brain by becoming more tuned to its commons sounds and structures

- In some cases, unsupervised learning can be a precursor to supervised learning and in others, a standalone technique

Reinforcement Learning:

- Training data does not explicitly contain the correct output for each input.
    - Example: a toddler learning not to touch a hot cup of tea. The experience of a toddler would comprise a set of occasions when the toddler "confronted" a cup of tea and had to make the decision of whether to touch it or not. Every time she did, there was a high degree of pain. Eventually, the toddler learns not to touch the hot cup
    - The training examples did not explicitly state what the toddler should have done, nit they instead graded different actions that she has taken. Nevertheless, she uses the examples to **reinforce** the better actions and eventually learns what she should do in similar situations
- This style of learning does not contain the target output, but instead some possible output with a measure of how good that output is.
- ★ This contrasts with supervised learning and its *(input, correct output)* form
- ★ The form of reinforcement learning is *(input, some output, grade for this output)*
- Reinforcement learning is especially useful for learning how to play a game
    - Example: In backgammon, you have to choose between different actions and identify the best action
    - It's difficult to tell what the best move would be for a particular move in this game so we can not easily create supervised learning models
    - If we use reinforcement learning, all we need to do is take some action and report how well things went, and you now have a training example
    - The reinforcement learning algorithm is left with the task of sorting out the information coming from different examples and find the best line of play.

Lecture 3: Too hot, steal notes later

What do we do if not linearly separable?

$x \in \{0, 1\} \leftarrow x \in R^9$

How many possible x's? 2^9

How many possible h's? 2^{512}

How many possible explanations are there for this data? == how many possible h's match the data?

Lecture 4

D → outside the data

Bin Model: a simple learning problem where you have a bin that contains red and green marbles. Pick a sample of marbles randomly of size N and compute:

- v = the sample of red
- μ = sample of red in Bin

$P[1 - v - \mu > \epsilon] \leq 2e^{-2\epsilon^2 N}$ ← probabilistic

ϵ is approximate. The larger the sample, the smaller the ϵ

Verification:

Gets input space X from bin model

$x \in X$:

- $red\ x$: $h(x) \neq f(x)$
- $green\ x$: $h(x) = f(x)$

D: iid x_1 x_2 . . . x_N ← P(x)

y_1 y_2 . . . y_N ← y_i = f(x_i)

h(x_1) h(x_2) . . . h(x_N) ← red x_i and green x_i

In sample error: E_{in} = frac of red in data = $\frac{1}{N}\sum_{i=1}^{N}[h(x_i) \neq f(x_i)]$

Out sample error: $E_{out} = P[h(x) \neq f(x)]$

Hoeffding: $E_{out}(h) \approx E_{in}(h)$ with probability

$P[|E_{in} - E_{out}| > \epsilon] \leq 2e^{-2\epsilon^2 N}$

Best case: $E_{in} \approx 0 \rightarrow E_{out} \approx 0$

Worst case: $E_{in} \approx 50 \rightarrow E_{out} \approx 50$

$P[|E_{in} - E_{out}| > \epsilon] \leq 2|H|e^{-2\epsilon^2 N}$ ← H is the price for "choice"

If N is large enough we can verify g

$P[|E_{in}(g) - E_{out}(g)| \leq \epsilon] \geq 1 - 2|H|e^{-2\epsilon^2 N}$

$\epsilon = \sqrt{\frac{1}{2N} ln^2 * \frac{|H|}{\sigma}}$

With prob >= 1 - σ

$E_{out}(g) \leq E_{in}(g) + \sqrt{\frac{1}{2N} ln^2 * \frac{2|H|}{\sigma}}$ ← error bar

Lecture 5

Theory of Learning:

1) $E_{out}(g) \approx E_{in}(g)$ ← theoretically
2) $E_{in}(g) \approx 0$ ← measured

H - fixed before you see D

D - must be IID ~ [P(x), f(x)]

With prob >= 1 - δ

$E_{out}(g) \leq E_{in}(g) + \sqrt{\frac{1}{2N} \frac{ln(2|H|)}{\delta}}$

Limitations: H is finite → $E_{out}(g) < E_{in}(g) + \sqrt{\frac{8}{N} ln(\frac{4m_H}{\delta})}$

General: any learning problem / any learning algorithm

How do we approach infinite H?

|H| is a measure of complexity does not **finish these notes**

Lecture 6

Complexity of H:

$x_1 x_2 \ldots x_n \rightarrow$ all dichotomies

$|H(x_1 x_2 \ldots x_n)| \leftarrow$ # of dichotomies implemented by H in a fixed dataset

$m_H(N) =$ maximum of H dichotomies on ant dataset of size N ($m_H(N)$ is the growth function)

$m_H(N) \leq 2^N$

Does $m_H(N)$ allow us to establish step 1 of learning?

$E_{out}(g) \leq E_{in}(g) + \sqrt{\frac{8}{N} ln(\frac{4m_H(2N)}{\delta})}$ where $\sqrt{\frac{8}{N} ln(\frac{4m_H(2N)}{\delta})} = O(\frac{D \cdot ln(N)}{N})$

Proof:

1) $E_{out} \approx E_{in}$
2) Look at $E_{in}(g)$
    a) If $E_{in}(g)$ small, then you're good
    b) If $E_{in}(g)$ large, then you're done

1) $m_H(N) = 2^N \forall N$ [step 1, BAD]
2) $\exists$ a breakpoint k $\rightarrow m_H(N) \leq N$

Lecture 7

The Linear Model

Credit analysis:

- approve/not approve
    - Classification
        - $y \in \{-1, 1\}$
- Credit level
    - Regression
        - $y \in R$
- Prob default
    - Logistic regression
        - $y \in [0, 1]$

The uhhh….the rest of them are gone and can therefore not be coallated