A (very) brief Introduction to Machine Learning

CS 580
Intro to Artificial Intelligence

What is Machine Learning?

3 Definitions:

Agent flavor
“Machine Learning is the study of agents that improve their performance on a given domain with experience”

Statistics flavor
“Machine Learning [is] a set of methods that can automatically detect patterns in data, and then use the uncovered patterns to predict future data”

Computational flavor
“The study of **algorithms** for making the **best** decisions given **data**”

Defining terms

Data
Pairs of inputs and outputs

Input, Features, X
Candidates: real numbers, integers, vectors, bit-vectors, strings, images, audio sequences...

Output, Target, Y
Candidates: numbers, labels, actions, strings, images…

$$S = \left\{ \left(\mathbf{x}^{(i)}, y^{(i)} \right) \right\}_{i=1}^{N}$$

$$\mathbf{x}^{(i)} \in \mathcal{X}$$

$$y^{(i)} \in \mathcal{Y}$$

Defining terms (2)

Decision
Function from input to output

Hypothesis, concept, model
h(x) = a*x+b, h(x) = 1 if x>0 else -1

Hypothesis class
linear functions, binary, polynomial, smooth, computable...

Target concept
A member of the Hypothesis Class?

$$h : \mathcal{X} \to \mathcal{Y}$$

$$h(\mathbf{x}) = y$$

$$h \in \mathcal{H}$$

$$h \text{ versus } \hat{h}$$

Defining terms (3)

Best

Hypothesis which minimizes some penalty function

Loss, Objective Function, Error

Squared error, misclassification rate

$$\mathcal{J}_S : \mathcal{H} \rightarrow \mathbb{R}$$

$$\text{find } h \in \mathcal{H} \text{ which minimizes } \mathcal{J}_S(h)$$

Some examples

$$\mathcal{J}_S(h) = \frac{1}{N} \sum_{i=1}^{N} \left(h(\mathbf{x}^{(i)}) - y^{(i)} \right)^2$$

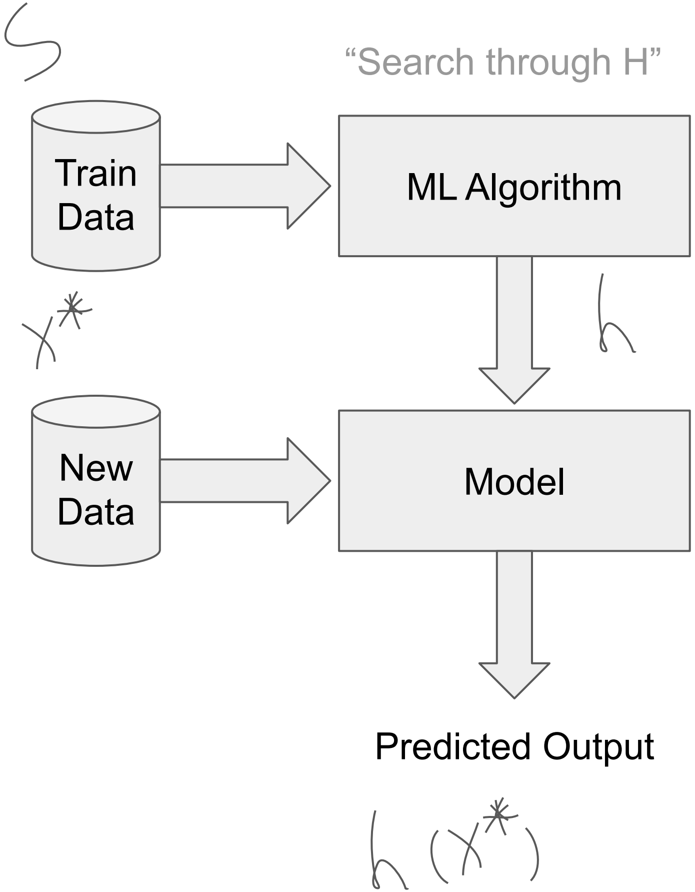
$$\mathcal{J}_S(h) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I} \left[h(\mathbf{x}^{(i)}) \neq y^{(i)} \right]$$

A fourth definition of ML

Least useful flavor
“Searching through the space of functions for the one with minimum loss.”

How can we possibly do this?
Uncountably infinite number of functions!
Ambiguous solutions (local minima)!
Combinatorial nightmare!

ML isn’t just the study of **how** to find h, it’s also the study of **when it’s possible** and **why.**


“Search through H”

Types of Machine Learning

One way of subdividing ML (not the only way)

Supervised Learning
“Given X and Y, find h”, h: X→Y

Our focus for this section of the course

Unsupervised Learning
“Given X, find a good description of X”, h: X→ “description”

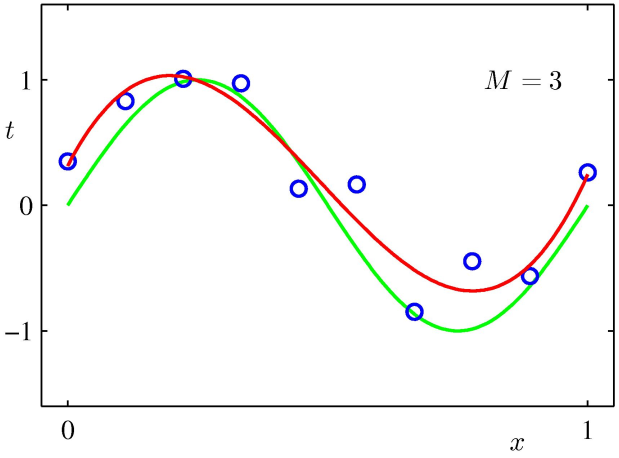
Reinforcement Learning
“Given states described by X, how should you ‘act’?”, h: X→ “actions”

Not necessarily a taxonomy, more a “how you look at a problem”

Supervised Learning

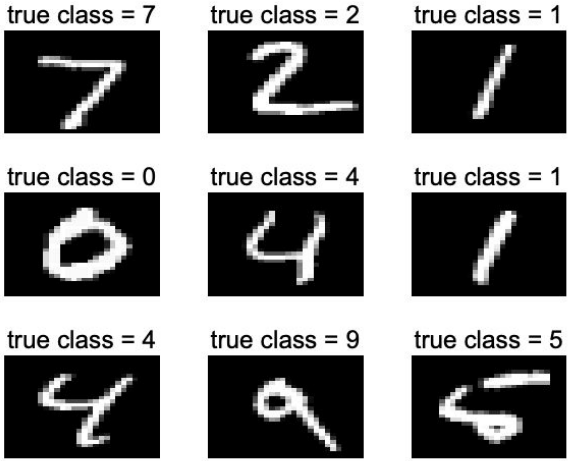
Regression

What's the value of the function at a given input?



Classification

The the correct "label" for a given input?



Motivation

In each of the topics we’ve covered, we’ve made some assumptions about what’s provided to the agent

Search

Successors, action costs, heuristic

MDPs

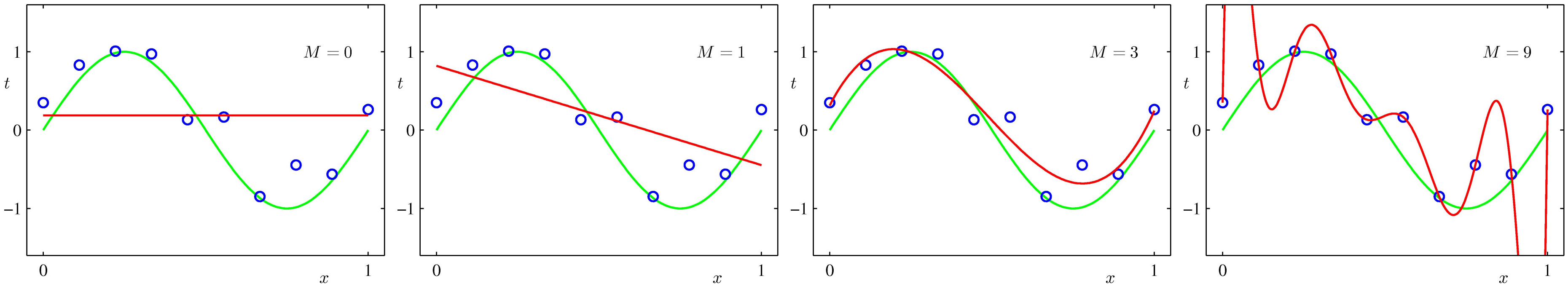
Transition function, reward function

Filtering and Bayes Nets

Probability tables

May be difficult to provide these things for real world problems. Machine Learning allows the agent to “learn” these by collecting data

Choosing among alternatives

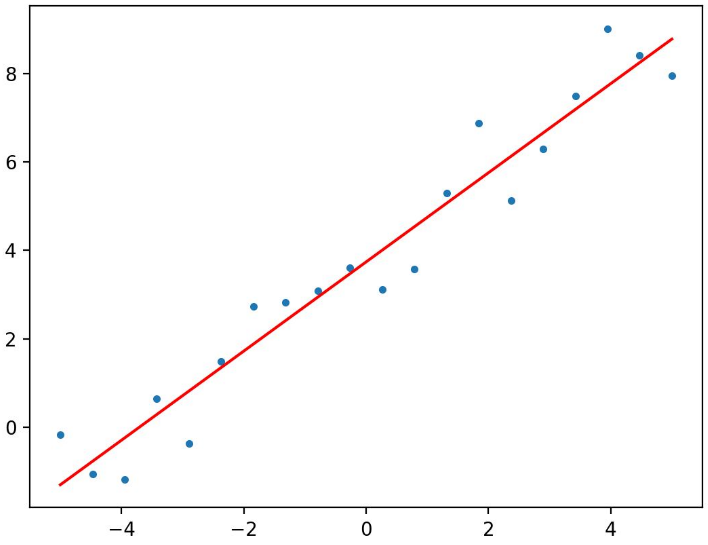


We'd like to find a mapping that does well on data that it's never seen before, one with low **generalization error**.

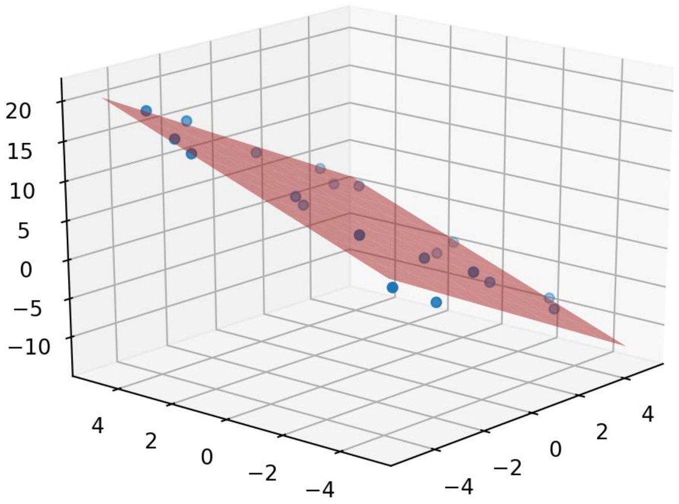
Error can be caused by either **underfitting** or **overfitting**. The first isn't a good model of even the training data. The second is susceptible to noise.

A first ML algorithm - Linear Regression

1D



2D



Notation

Data: (x,y) pairs

Hypothesis class: For now, linear functions: h(x) = $\sum\theta_j{*}x_j + \theta_0$ for some θ's

Loss: Sum of squared errors

$$\mathcal{X} = \mathbb{R}^D, \quad \mathcal{Y} = \mathbb{R}$$

$$\mathcal{H} = \left\{ h_\theta : h_\theta(\mathbf{x}) = \sum_{j=1}^{D} \theta_j x_j + \theta_0, \quad \theta_j \in \mathbb{R} \right\}$$

$$\mathcal{J}_S(h_\theta) = \frac{1}{2N} \sum_{i=1}^{N} \left(h_\theta(\mathbf{x}^{(i)}) - y^{(i)} \right)^2$$

Note! Mysterious 2!

Solving, the lazy way (1)

Rewrite using matrix notation:

Let:

$$\mathbf{x}^{(i)} = \begin{bmatrix} 1 & x_1^{(i)} & x_2^{(i)} & \cdots & x_D^{(i)} \end{bmatrix}^\top$$

$$\theta = \begin{bmatrix} \theta_0 & \theta_1 & \theta_2 & \cdots & \theta_D \end{bmatrix}^\top$$

Then:

$$h_\theta(\mathbf{x}^{(i)}) = \sum_{j=1}^{D} \theta_j x_j^{(i)} = \mathbf{x}^{(i)\top}\theta$$

Solving, the lazy way (2)

We would like it if $h(x^{(i)}) = y^{(i)}$ for all $(x^{(i)}, y^{(i)})$...

Let:

$$X = \begin{bmatrix} \text{---} & \mathbf{x}^{(1)} & \text{---} \\ \text{---} & \mathbf{x}^{(2)} & \text{---} \\ & \vdots & \\ \text{---} & \mathbf{x}^{(N)} & \text{---} \end{bmatrix}, Y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(N)} \end{bmatrix}$$

This part will still be useful

Then?

$$\begin{aligned} X\theta &= Y \\ X^\top X\theta &= X^\top Y \\ (X^\top X)^{-1} X^\top X\theta &= (X^\top X)^{-1} X^\top Y \\ \theta &= (X^\top X)^{-1} X^\top Y \end{aligned}$$

NO

Solving by minimizing the Loss (1)

More useful notation

$$\hat{Y} = \begin{bmatrix} h_\theta(\mathbf{x}^{(1)}) \\ h_\theta(\mathbf{x}^{(2)}) \\ \vdots \\ h_\theta(\mathbf{x}^{(N)}) \end{bmatrix} = \begin{bmatrix} \mathbf{x}^{(1)\top}\theta \\ \mathbf{x}^{(2)\top}\theta \\ \vdots \\ \mathbf{x}^{(N)\top}\theta \end{bmatrix} = X\theta$$

$$\hat{Y} - Y = \begin{bmatrix} h_\theta(\mathbf{x}^{(1)}) - y^{(1)} \\ h_\theta(\mathbf{x}^{(2)}) - y^{(2)} \\ \vdots \\ h_\theta(\mathbf{x}^{(N)}) - y^{(N)} \end{bmatrix}$$

Solving by minimizing the Loss (2)

Rewrite the loss using matrix notation

$$\mathcal{J}_S(h_\theta) = \frac{1}{2N}\sum_{i=1}^{N}\left(h_\theta(\mathbf{x}^{(i)}) - y^{(i)}\right)^2$$

Summing up the squared elements of the vector Ŷ-Y

$$= \frac{1}{2N}\sum_{i=1}^{N}\left(\mathbf{x}^{(i)\top}\theta - y^{(i)}\right)^2$$

$$\sum_{i=1}^{K} v_i^2 = v_1 \cdot v_1 + v_2 \cdot v_2 + \ldots + v_K \cdot v_K$$

$$= \mathbf{v}^\top \mathbf{v}$$

$$= \frac{1}{2N}(X\theta - Y)^\top (X\theta - Y)$$

Useful matrix derivative identities:

$$\boxed{\frac{\partial x^\top A x}{\partial x} = (A + A^\top)x}$$

$$\boxed{\frac{\partial b^\top A x}{\partial x} = b^\top A}$$

Solving by minimizing the Loss (3)

Find the gradient

$$\mathcal{J}_S(h_\theta) = \frac{1}{2N}[\theta^\top X^\top X\theta - Y^\top X\theta - \theta^\top X^\top Y + Y^\top Y]$$

$$\nabla_\theta \mathcal{J}(h_\theta) = \frac{1}{2N}[(X^\top X + X^\top X)\theta - Y^\top X - X^\top Y + 0]$$

$$= \frac{1}{N}[X^\top X\theta - X^\top Y]$$

Solving by minimizing the Loss (3)

Set the gradient to zero and solve

$$\frac{1}{N}\left[X^\top X\theta - X^\top Y\right] = 0$$

$$X^\top X\theta = X^\top Y$$

$$\theta = \left(X^\top X\right)^{-1} X^\top Y$$

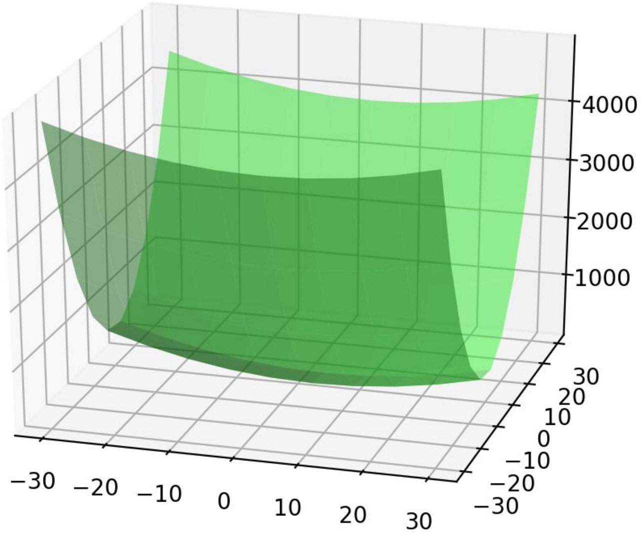
Suspiciously familiar, but this **proves*** that using the above equation for θ **minimizes** the sum of squared errors

Visualizing the Loss

- Loss is quadratic (in θ) → don't have to worry about maxima vs minima vs saddle points
- For any fixed dataset* there's only one minima
- This gives us a way to define "best" in terms of the loss

A common outline for doing ML:

1. Define hypothesis class
2. Define loss
3. Use math to find the hypothesis that minimizes the loss



Determining the performance of the model

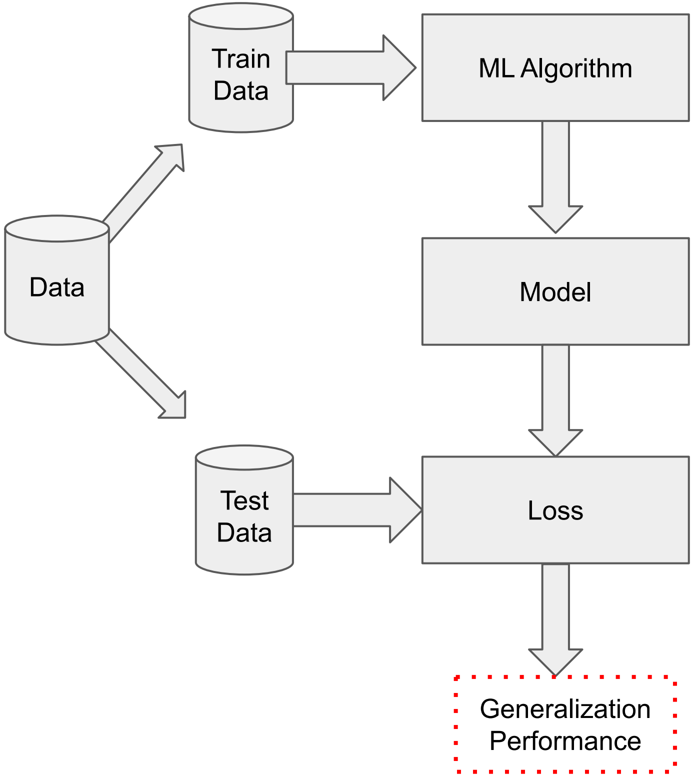
The equation we derived will give you the best linear model (in terms of the loss).

But how can you tell if the “best” linear model is any good?

What we really care about is performance on “unseen data”, what we call **Generalization**

One solution is to split the data into two sets, **training** and **testing**.

Fit model using training data, evaluate performance on testing data



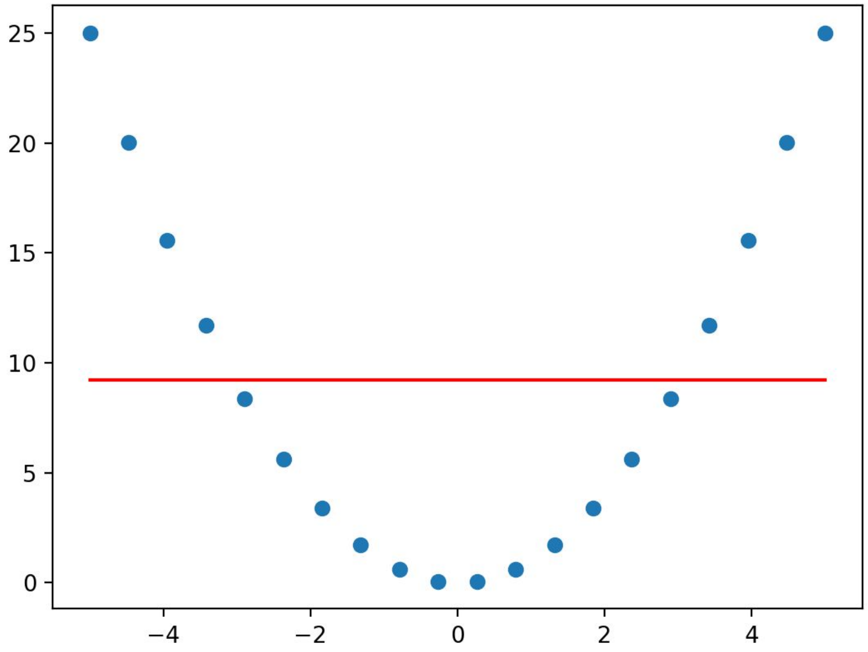
What if it’s not linear?

Some datasets just aren’t linear

Are we going to have to re-derive everything from scratch if we want to change our hypothesis class?

Not necessarily:
What if we just add some columns to our training data?

$$X = \begin{bmatrix} 1 & x_1^{(1)} & (x_1^{(1)})^2 \\ 1 & x_1^{(2)} & (x_1^{(2)})^2 \\ & \vdots & \\ 1 & x_1^{(N)} & (x_1^{(N)})^2 \end{bmatrix}$$



What if X is too big?

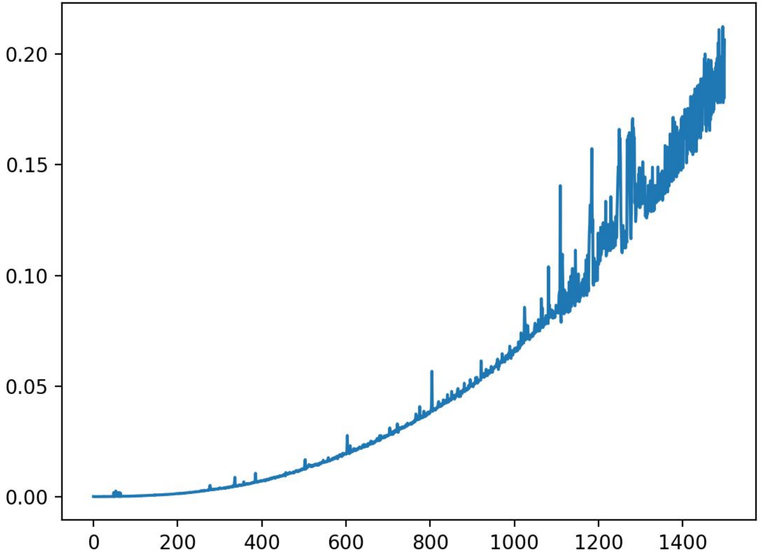
In ML, large amounts of data is “a good problem to have”

- X^TX takes $O(D^2N)$ time
- $(X^TX)^{-1}$ takes $O(D^3)$ time
- X takes O(ND) memory

N, D ~ 10^5 inversion is slow
N ~ 10^9 multiplication is slow
Modern ML datasets can be 100s of GB, probably too large to fit in memory on your laptop.

Need a method that can work iteratively.

Time to invert a random NxN matrix



Summary and preview

Wrapping up

- Supervised Learning is about finding mappings: h(x) = y
- **Linear regression** finds the set of coefficients for a weighted sum of the inputs which produces the minimum sum of squared errors in the data
- We can implement linear regression as a sequence of matrix operations
- The **loss function** is quadratic in the parameters, and so has a single minima

For next time

- Non-linear models, regularization, and gradient descent