https://introml.mit.edu/

6.390 Intro to Machine Learning

Lecture 2: Linear regression and regularization

Shen Shen

Sept 6, 2024

(many slides adapted from Tamara Broderick)

Instructors


Ike Chuang

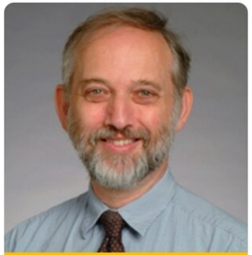

Alexandre Megretski

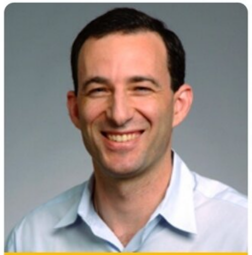

Vince Monardo


Mardavij Roozbehani


Shen Shen


Tess Smidt


Pete Szolovits


Bruce Tidor

Course Assistant


Taylor Braun

6.390-personal@mit.edu

Logistical issues? Personal concerns?
We'd love to help out!

TAs


Mauricio Barba


Abhay Basireddy

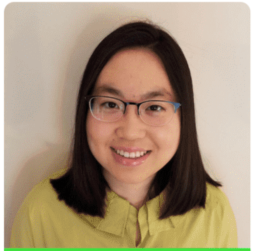

Kevin Bunn


Shaunticlair Ruiz


Yogi Sragow


Yan Wu


Audrey Douglas


Song Kim


Kartikesh Mishra


Elisa Xia


Haley Nakamura

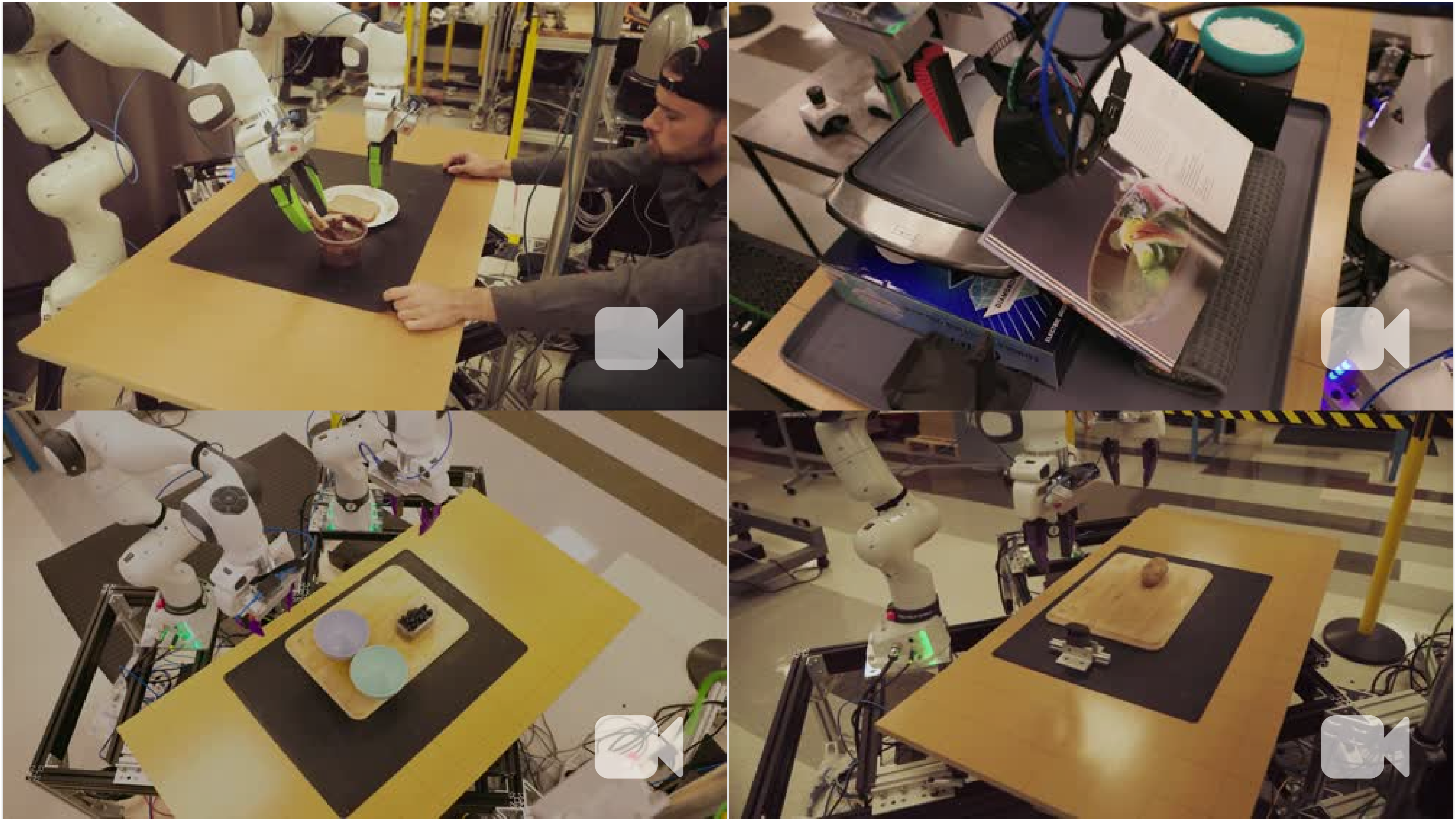

Anh Nguyen


Linh Nguyen

plus ~40 awesome LAs

https://shenshen.mit.edu/demos/gifs/atlas_darpa_overall.gif

Optimization + first-principle physics



https://www.youtube.com/embed/fn3KWM1kuAw?start=1&enablejsapi=1

Outline

- Recap: ML set up, terminology
- Ordinary least-square regression
    - Closed-form solutions (when exists)
    - Cases when closed-form solutions don't exist
        - mathematically, practically, visually
- Regularization
- Hyperparameter and cross-validation

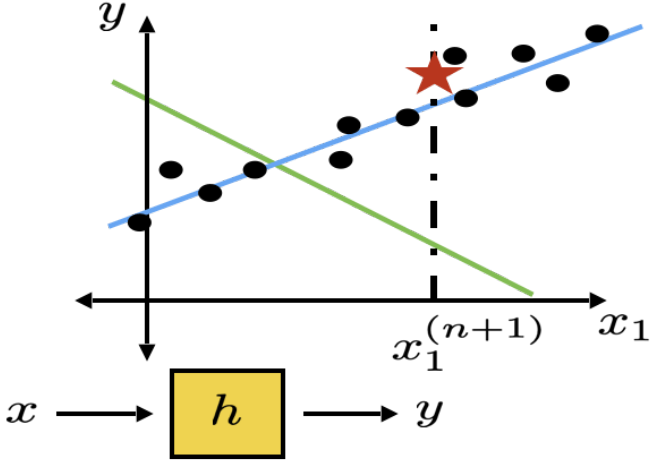
Outline

- Recap: ML set up, terminology
- Ordinary least-square regression
  - Closed-form solutions (when exists)
  - Cases when closed-form solutions don't exist
    - mathematically, practically, visually
- Regularization
- Hyperparameter and cross-validation

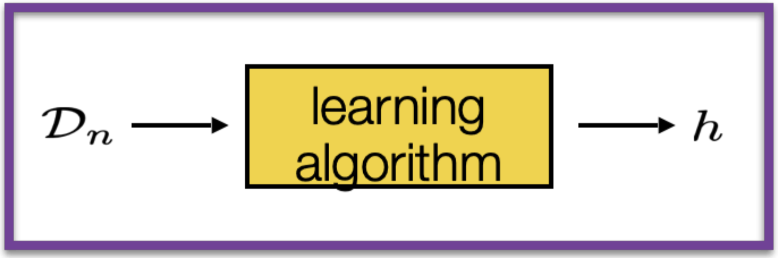
Recall lab1 intro

How do we learn?

- Have data; have hypothesis class
- Want to choose (learn) a good hypothesis h (or more concretely, a set of parameters)



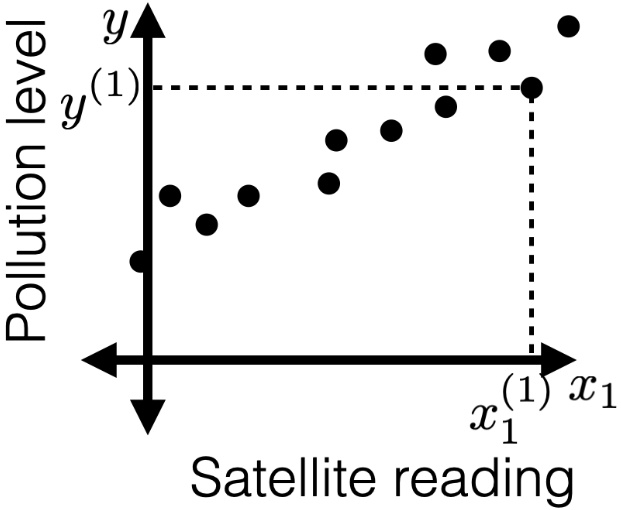
How to get it:
(Next time!)



Example: predict pollution level

(Training) data

- *n* training data points
- For data point $i \in \{1, \dots, n\}$
  - Feature vector $x^{(i)} = (x_1^{(i)}, \dots, x_d^{(i)})^\top \in \mathbb{R}^d$
  - Label $y^{(i)} \in \mathbb{R}$
- Training data $\mathcal{D}_n = \{(x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)})\}$



What do we want? A good way to label new points

How to label? Hypothesis $h : \mathbb{R}^d \to \mathbb{R}$

$x \rightarrow \boxed{h} \rightarrow y$

Is this a ***good*** hypothesis?

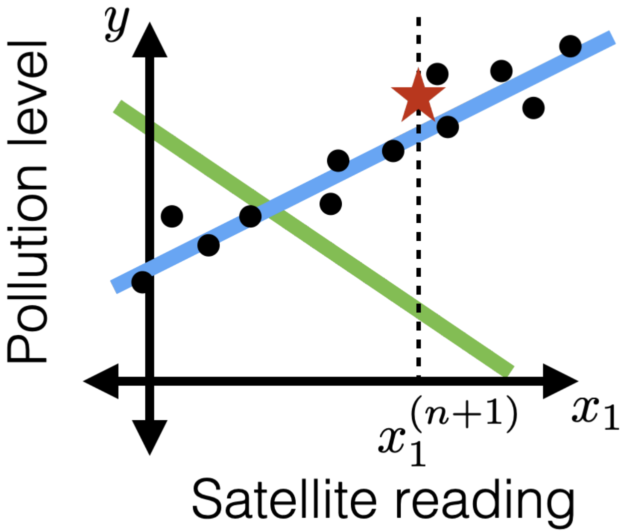
- Example *h*: For any *x*, *h*(*x*) = 1,000,000

How good is a regression hypothesis?

- Should predict well on future data
- How good is a regressor at one point?
- Loss $L(g, a)$
  - Ex: squared loss

g: guess,
a: actual

$$L(g, a) = (g - a)^2$$



- Training error: $\mathcal{E}_n(h) = \frac{1}{n}\sum_{i=1}^{n} L(h(x^{(i)}), y^{(i)})$
- Test error (n' new points): $\mathcal{E}(h) = \frac{1}{n'}\sum_{i=n+1}^{n+n'} L(h(x^{(i)}), y^{(i)})$
- One idea: prefer h to $\tilde{h}$ if $\mathcal{E}_n(h) < \mathcal{E}_n(\tilde{h})$

Recall lab1 Q1

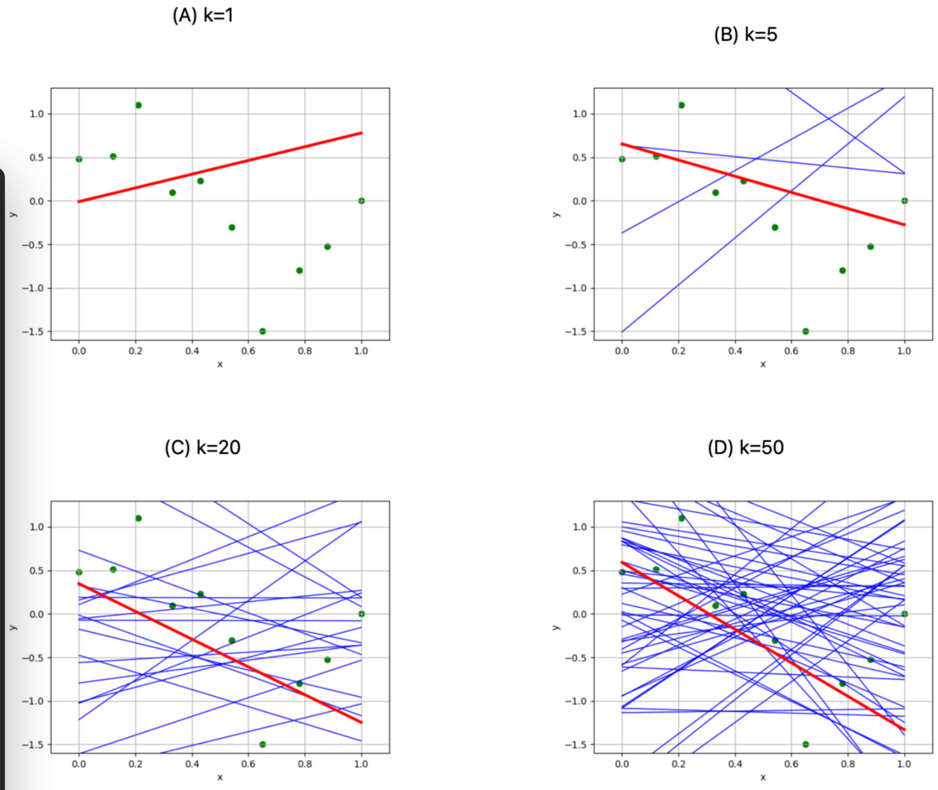
```
def random_regress(X, Y, k):
    d, n = X.shape

    # generate k random hypotheses
    ths = np.random.randn(d, k)
    th0s = np.random.randn(1, k)

    # compute the mean squared error of each
hypothesis on the data set
    errors = lin_reg_err(X, Y, ths, th0s.T)

    # Find the index of the hypotheses with the
lowest error
    i = np.argmin(errors)

    # return the theta and theta0 parameters
that define that hypothesis
    theta, theta0 = ths[:,i:i+1], th0s[:,i:i+1]
    return (theta, theta0), errors[i]
```



- Will this method eventually get arbitrarily close to the best solution? What do you think about the efficiency of this method?

Outline

- Recap: ML set up, terminology
- Ordinary least-square regression
  - Closed-form solutions (when exists)
  - Cases when closed-form solutions don't exist
    - mathematically, practically, visually
- Regularization
- Hyperparameter and cross-validation

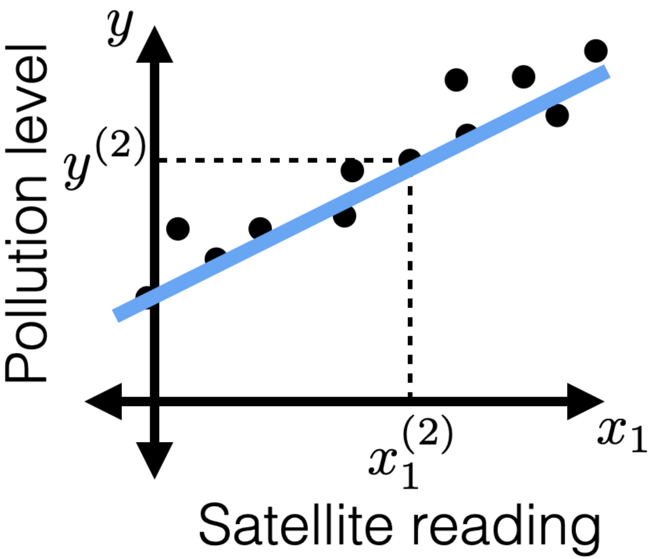
Linear regression: the analytical way

- How about we just consider all hypotheses in our class and choose the one with lowest training error?
- We'll see: not typically straightforward
- But for linear regression with square loss: can do it!
- In fact, sometimes, just by plugging in an equation!

Linear regressors

- Hypothesis class $\mathcal{H}$: set of h
- A linear regression hypothesis when d=1:

$$h(x) = \theta x + \theta_0$$



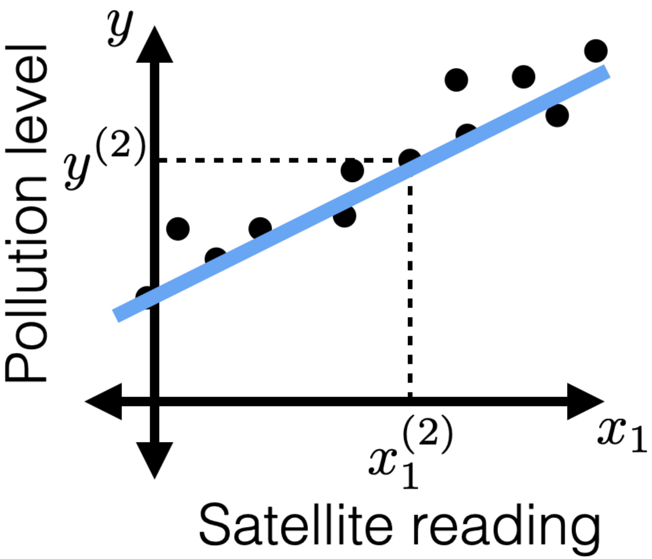
Linear regressors

- Hypothesis class $\mathcal{H}$: set of h

- A linear regression hypothesis when *d*=1:

$$h(x;\theta,\theta_0) = \theta x + \theta_0$$

parameters



Linear regressors

- Hypothesis class $\mathcal{H}$: set of h
- A linear regression hypothesis when *d*=1:

$$h(x; \theta, \theta_0) = \theta x + \theta_0$$

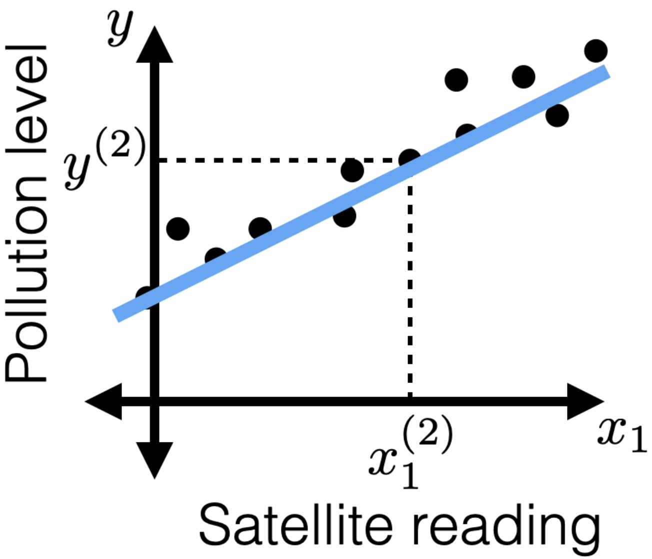
parameters

- A linear reg. hypothesis when *d*≥1:

$$h(x; \theta, \theta_0) = \theta_1 x_1 + \cdots + \theta_d x_d + \theta_0$$
$$= \theta^\top x + \theta_0$$

OR

$$h(x) = \theta_1 x_1 + \cdots + \theta_d x_d + (\theta_0)(1)$$
$$= \theta^\top x$$



- A linear reg. hypothesis when $d \geq 1$:

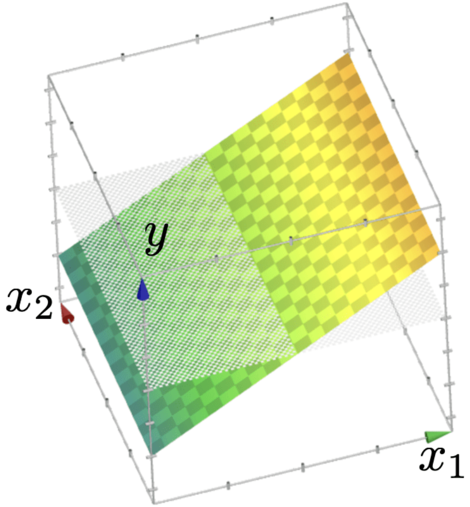
$$h(x;\theta,\theta_0) = \theta_1 x_1 + \cdots + \theta_d x_d + \theta_0$$
$$= \theta^\top x + \theta_0$$

1x2,2x1

OR

$$h(x) = \theta_1 x_1 + \cdots + \theta_d x_d + (\theta_0)(1)$$
$$= \theta^\top x$$

1x3,3x1



Notational trick: *not* the same θ & x!

- Our hypothesis class in linear regression will be the set of all such *h*

Hypothesis is a "hyperplane"

- Recall: training loss:

$$\frac{1}{n} \sum_{i=1}^{n} L\left(h\left(x^{(i)}\right), y^{(i)}\right)$$

- With squared loss:

$$\frac{1}{n} \sum_{i=1}^{n}\left(h\left(x^{(i)}\right)-y^{(i)}\right)^{2}$$

- Using linear hypothesis (with extra "1" feature):

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

- With given data, the error only depends on θ, so let's call the loss $J(\theta)$

Now training loss:

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

$$= \frac{1}{n}(\tilde{X}\theta - \tilde{Y})^\top(\tilde{X}\theta - \tilde{Y})$$

Define

$$\tilde{X} = \begin{bmatrix} x_1^{(1)} & \cdots & x_d^{(1)} \\ \vdots & \ddots & \vdots \\ x_1^{(n)} & \cdots & x_d^{(n)} \end{bmatrix}$$

nxd

$$\tilde{Y} = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \end{bmatrix}$$

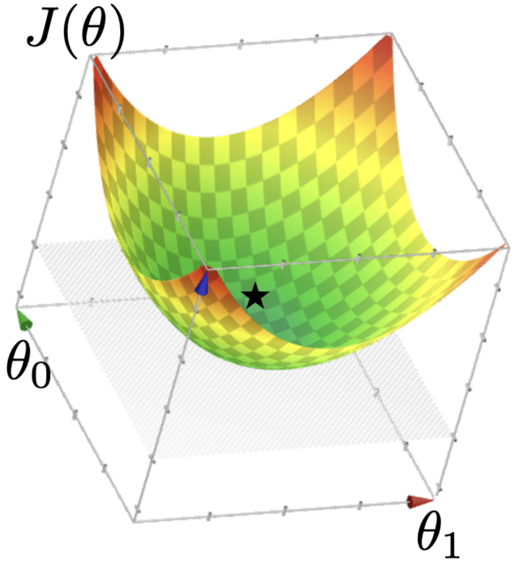
nx1

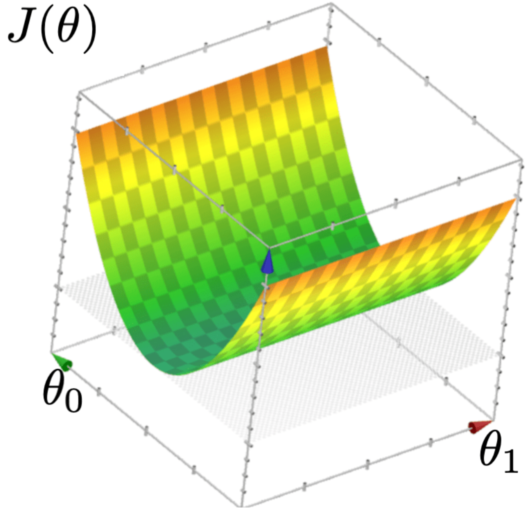
- Goal: find θ to minimize

$$J(\theta) = \frac{1}{n}(\tilde{X}\theta - \tilde{Y})^\top(\tilde{X}\theta - \tilde{Y})$$

- Q: what kind of function is $J(\theta)$ and what does it look like?

- A: Quadratic function. Looks like either a "bowl" or "half-pipe"



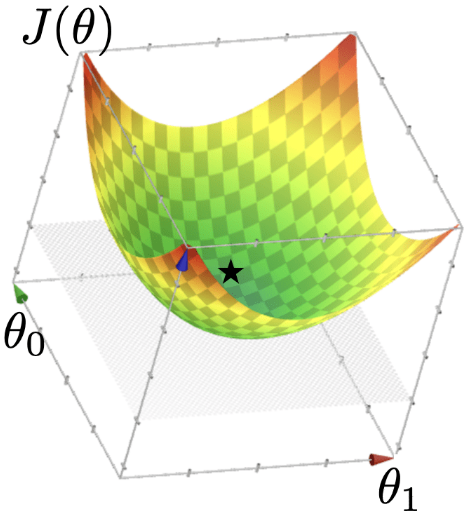


- When

$$J(\theta) = \frac{1}{n}(\tilde{X}\theta - \tilde{Y})^\top(\tilde{X}\theta - \tilde{Y})$$

looks a "bowl" (typically does)

- Uniquely minimized at a point if gradient at that point is zero and function "curves up" [see linear algebra]

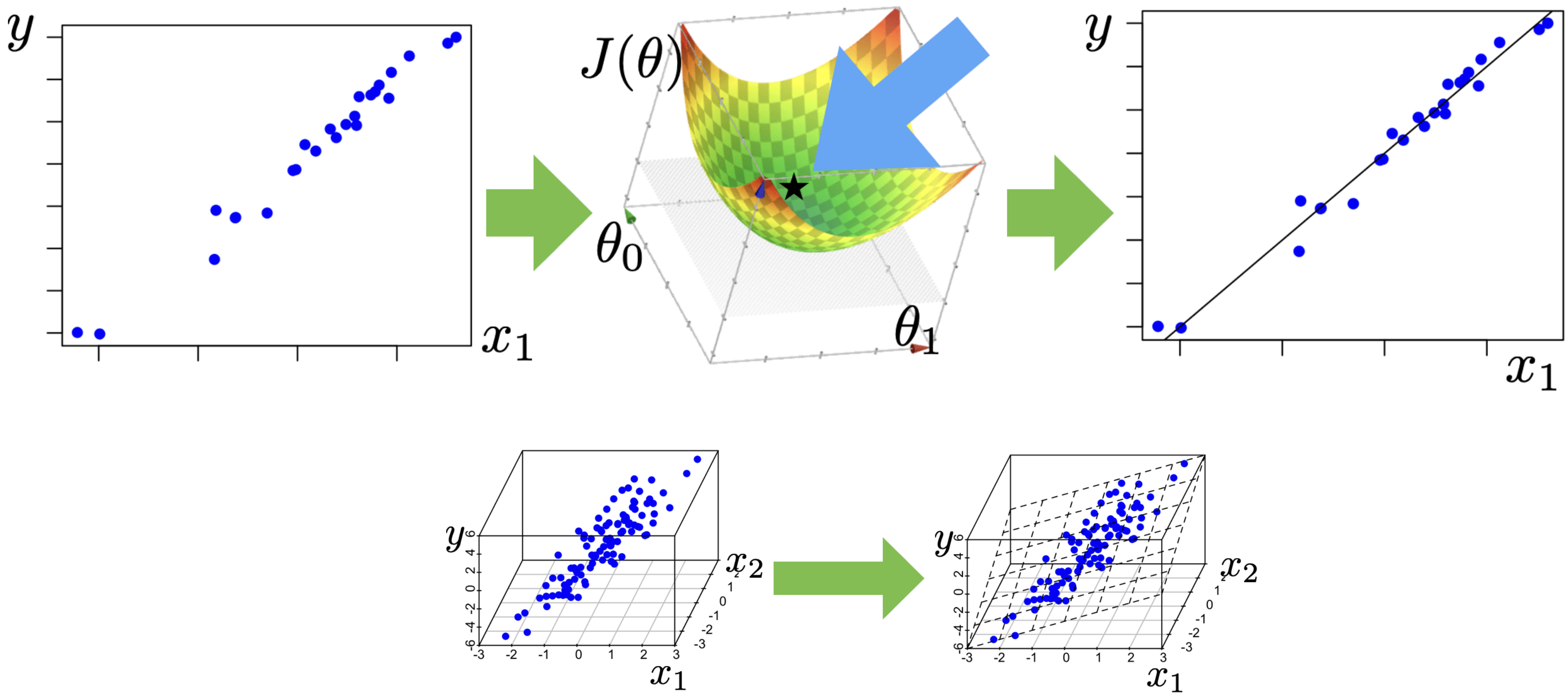


dx1

Set Gradient $\nabla_\theta J(\theta) \overset{\text{set}}{=} 0$

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

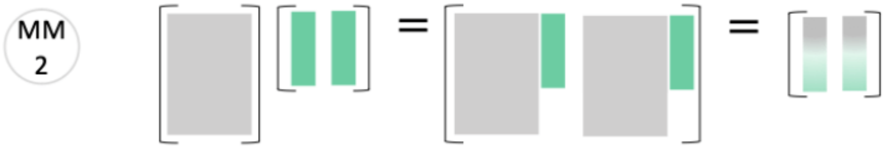
The beauty of $\theta^* = \left(\tilde{X}^\top \tilde{X}\right)^{-1} \tilde{X}^\top \tilde{Y}$: simple, general, unique minimizer



- Now, the catch (we'll see, all lead to half-pipe case)

- $\theta^* = \left(\tilde{X}^\top \tilde{X}\right)^{-1} \tilde{X}^\top \tilde{Y}$ is not well-defined if $\left(\tilde{X}^\top \tilde{X}\right)$ is not invertible

- Indeed, $\left(\tilde{X}^\top \tilde{X}\right)$ is not invertible if and only if $\tilde{X}$ is not full column rank



$A\boldsymbol{x}$ and $A\boldsymbol{y}$ are linear combinations of columns of A.

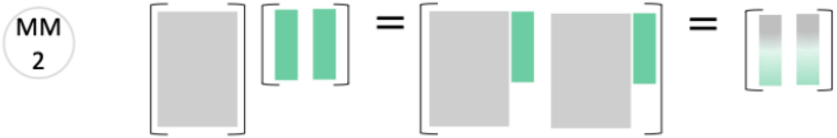
$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \end{bmatrix} = A[\boldsymbol{x} \quad \boldsymbol{y}] = [A\boldsymbol{x} \quad A\boldsymbol{y}]$$

- Indeed, $\left(\tilde{X}^\top \tilde{X}\right)$ is not invertible if and only if $\tilde{X}$ is not full column rank
- Recall

$$\underset{\text{nxd}}{\tilde{X}} = \begin{bmatrix} x_1^{(1)} & \cdots & x_d^{(1)} \\ \vdots & \ddots & \vdots \\ x_1^{(n)} & \cdots & x_d^{(n)} \end{bmatrix}$$

$\tilde{X}$ is not full column rank

1. if $n<d$
2. if columns (features) in $\tilde{X}$ have linear dependency

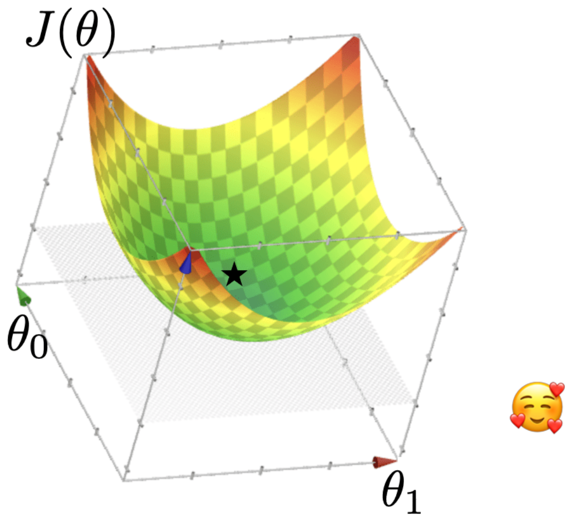


Ax and Ay are linear combinations of columns of A.

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \end{bmatrix} = A[\mathbf{x} \quad \mathbf{y}] = [A\mathbf{x} \quad A\mathbf{y}]$$

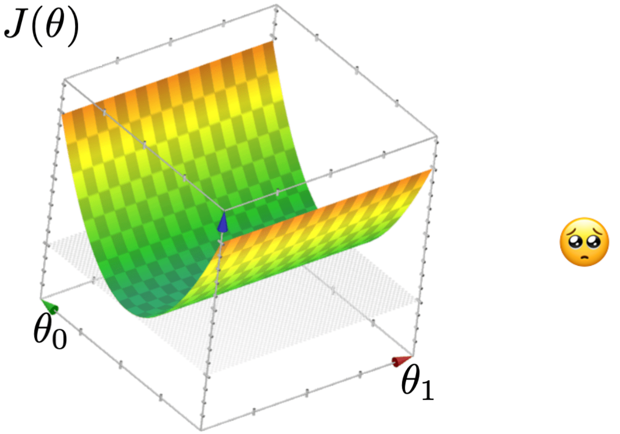
Quick Summary:

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



1. if $n<d$ (i.e. not enough data)
2. if columns (features) in $\tilde{X}$ have linear dependency (aka co-linearity)

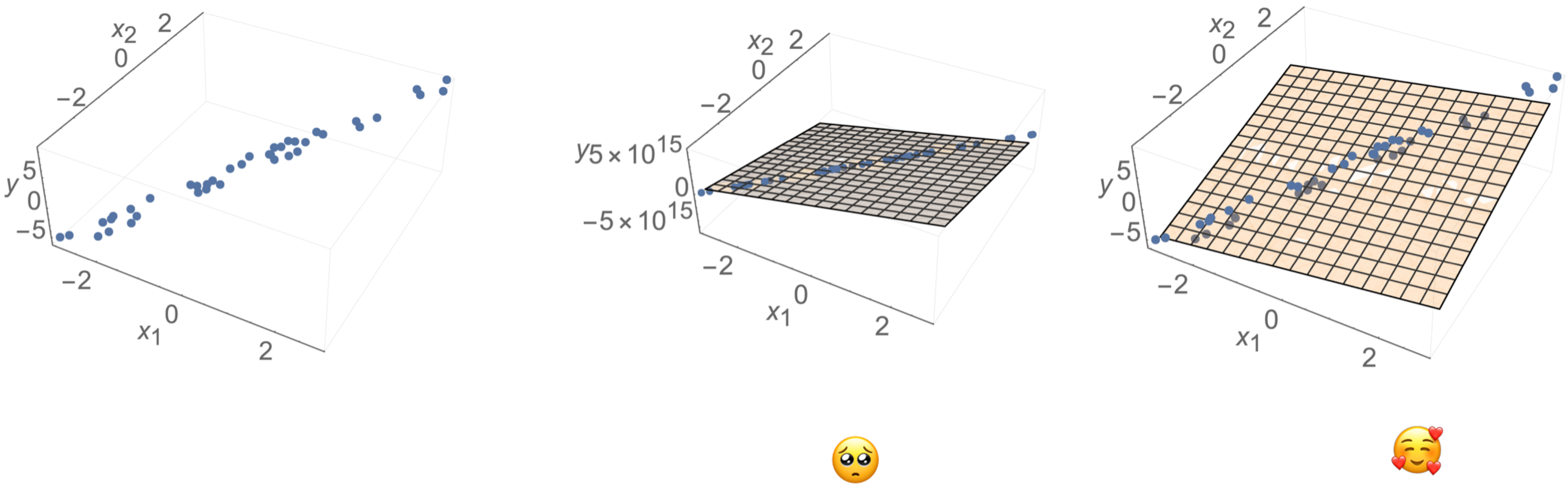
- This formula 👈 is not well-defined
- Infinitely many optimal hyperplanes



Outline

- Recap: ML set up, terminology
- Ordinary least-square regression
  - Closed-form solutions (when exists)
  - Cases when closed-form solutions don't exist
    - mathematically, practically, visually
- Regularization
- Hyperparameter and cross-validation

- Sometimes, noise can resolve the invertibility issue
- but still lead to undesirable results



- How to choose among hyperplanes?
- Prefer θ with small magnitude

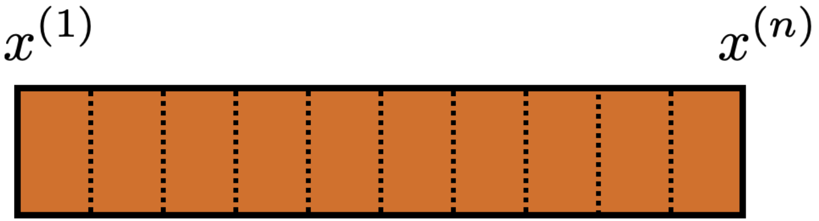
Ridge Regression

- Add a square penalty on the magnitude
- $J_{\text{ridge}}(\theta) = \frac{1}{n}(\tilde{X}\theta - \tilde{Y})^\top(\tilde{X}\theta - \tilde{Y}) + \lambda\|\theta\|^2$ $(\lambda > 0)$
- λ is a so-called "hyperparameter"
- Setting $\nabla_\theta J_{\text{ridge}}(\theta) = 0$ we get
- $\theta^* = \left(\tilde{X}^\top \tilde{X} + n\lambda I\right)^{-1} \tilde{X}^\top \tilde{Y}$
- θ^* always exists, and is always the unique optimal parameters
- (If there's an offset, see recitation/hw for discussion.)

Outline

- Recap: ML set up, terminology
- Ordinary least-square regression
  - Closed-form solutions (when exists)
  - Cases when closed-form solutions don't exist
    - mathematically, practically, visually
- Regularization
- Hyperparameter and cross-validation

Cross-validation

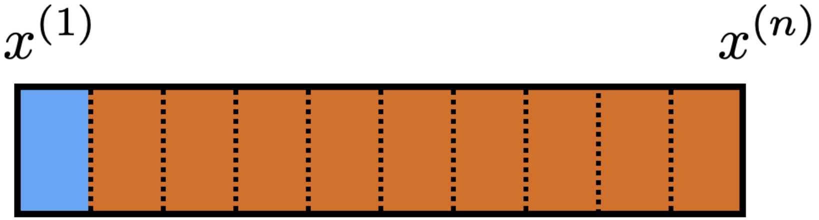



```
Cross-validate( 𝒟_n , k )
  Divide 𝒟_n into k chunks 𝒟_{n,1},...,𝒟_{n,k} (of
  roughly equal size)
```

Cross-validate($\mathcal{D}_n$, k)

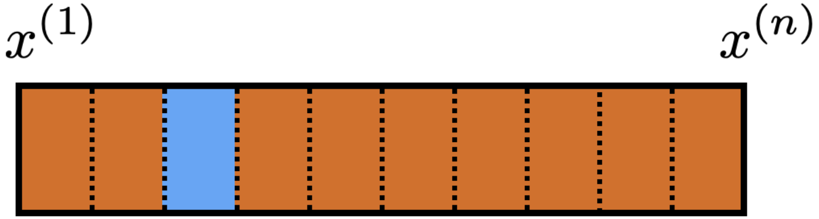
Divide $\mathcal{D}_n$ into k chunks $\mathcal{D}_{n,1}, \ldots, \mathcal{D}_{n,k}$ (of roughly equal size)

Cross-validation




```
Cross-validate( 𝒟_n , k )
  Divide 𝒟_n into k chunks 𝒟_{n,1},...,𝒟_{n,k} (of
  roughly equal size)
  for i = 1 to k
```

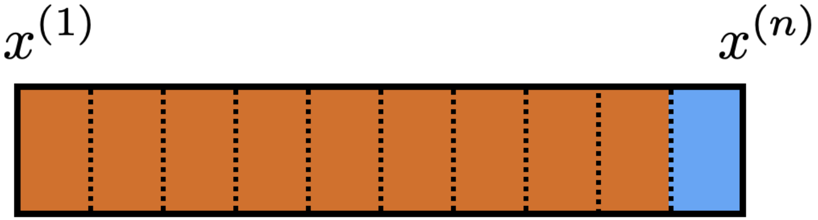
Cross-validation




```
Cross-validate( 𝒟_n , k )
  Divide 𝒟_n into k chunks 𝒟_{n,1},...,𝒟_{n,k} (of
  roughly equal size)
  for i = 1 to k
```

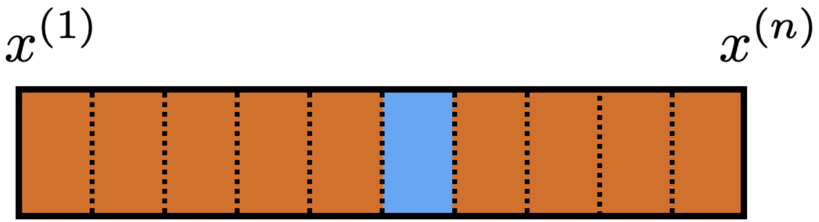
...

Cross-validation



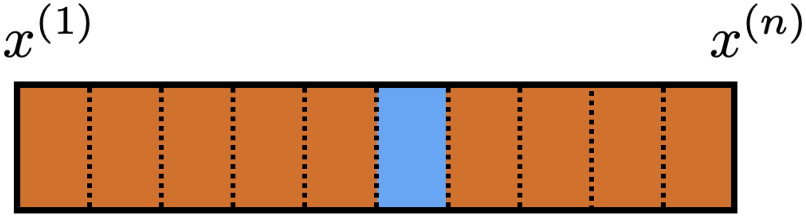

```
Cross-validate( 𝒟ₙ , k )
  Divide 𝒟ₙ into k chunks 𝒟ₙ,₁,...,𝒟ₙ,ₖ (of
  roughly equal size)
  for i = 1 to k
```

Cross-validation



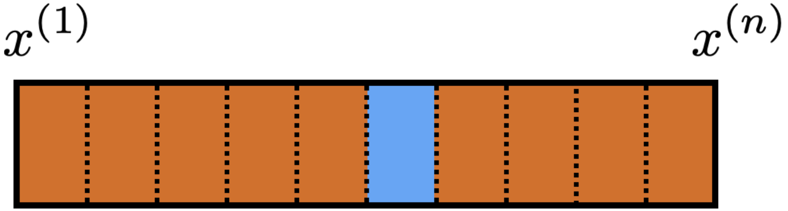

```
Cross-validate( 𝒟_n , k )
  Divide 𝒟_n into k chunks 𝒟_{n,1},...,𝒟_{n,k} (of
  roughly equal size)
  for i = 1 to k
    train h_i on 𝒟_n\𝒟_{n,i} (i.e. except chunk i)
```

Cross-validation




```
Cross-validate(𝒟_n , k )
  Divide 𝒟_n into k chunks 𝒟_{n,1},...,𝒟_{n,k} (of
  roughly equal size)
  for i = 1 to k
    train h_i on 𝒟_n\𝒟_{n,i} (i.e. except chunk i)
    compute "test" error ℰ(h_i, 𝒟_{n,i}) of h_i on 𝒟_{n,i}
```

Cross-validation



```
Cross-validate( D_n , k )
  Divide D_n into k chunks D_{n,1},...,D_{n,k} (of
  roughly equal size)
  for i = 1 to k
    train h_i on D_n\D_{n,i} (i.e. except chunk i)
    compute "test" error E(h_i, D_{n,i}) of h_i on D_{n,i}
  Return (1/k) sum_{i=1}^{k} E(h_i, D_{n,i})
```

Cross-validate($\mathcal{D}_n$, k)

Divide $\mathcal{D}_n$ into k chunks $\mathcal{D}_{n,1},\ldots,\mathcal{D}_{n,k}$ (of roughly equal size)

for i = 1 to k

train h_i on $\mathcal{D}_n \backslash \mathcal{D}_{n,i}$ (i.e. except chunk i)

compute "test" error $\mathcal{E}(h_i, \mathcal{D}_{n,i})$ of h_i on $\mathcal{D}_{n,i}$

Return $\frac{1}{k}\sum_{i=1}^{k}\mathcal{E}(h_i, \mathcal{D}_{n,i})$

Comments on (cross)-validation

- good idea to shuffle data first
- a way to "reuse" data
- it's not to evaluate a hypothesis
- rather, it's to evaluate learning algorithm (e.g. hypothesis class choice, hyperparameters)
- Could e.g. have an outer loop for picking good hyperparameter or hypothesis class

Summary

- One strategy for finding ML algorithms is to reduce the ML problem to an optimization problem.
- For the ordinary least squares (OLS), we can find the optimizer analytically, using basic calculus! Take the gradient and set it to zero. (Generally need more than gradient info; suffices in OLS)
- Two ways to approach the calculus problem: write out in terms of explicit sums or keep in vector-matrix form. Vector-matrix form is easier to manage as things get complicated (and they will!)
- There are some good discussions in the lecture notes.

Summary

- What does it mean for linear regression to be well posed.
- When there are many possible solutions, we need to indicate our preference somehow.
- Regularization is a way to construct a new optimization problem.
- Least-squares regularization leads to the ridge-regression formulation. Good news: we can still solve it analytically!
- Hyperparameters and how to pick them; cross-validation.

https://docs.google.com/forms/d/e/1FAIpQLSftMB5hSccgAbIAFmP_LuZt95w6KFx0x_R3uuzBP8WwjSzZeQ/viewform?embedded=true

We'd love to hear your thoughts.

Thanks!