CS 229 Lecture Two
Supervised Learning: Regression

Chris Ré

April 2, 2023

Disclaimers

- I'm trying a new format with slides (vs. whiteboard).
- The course notes maintained by Tengyu are your *best source*, the lecture is to give you the overall sense and highlight issues.
- The slides are new (copied from old hand-written notes), so apologies for any bugs. Please flag them!
- I'm worried that the lecture pacing will be too fast. Please slow me down with questions.
- I talk fast, please watch on slower speed.

Supervised Learning and Linear Regression

- Definitions
- Linear Regression
- Batch and Stochastic Gradient
- Normal Equations

Supervised Learning

- A **hypothesis** or a prediction function is function $h : \mathcal{X} \to \mathcal{Y}$

Supervised Learning

- A **hypothesis** or a prediction function is function $h : \mathcal{X} \to \mathcal{Y}$
  - $\mathcal{X}$ is an image, and $\mathcal{Y}$ contains "cat" or "not."
  - $\mathcal{X}$ is a text snippet, and $\mathcal{Y}$ contains "hate speech" or "not."
  - $\mathcal{X}$ is house data, and $\mathcal{Y}$ could be the price.

Supervised Learning

- A **hypothesis** or a prediction function is function $h : \mathcal{X} \rightarrow \mathcal{Y}$
  - $\mathcal{X}$ is an image, and $\mathcal{Y}$ contains "cat" or "not."
  - $\mathcal{X}$ is a text snippet, and $\mathcal{Y}$ contains "hate speech" or "not."
  - $\mathcal{X}$ is house data, and $\mathcal{Y}$ could be the price.
- A **training set** is a set of pairs $\left\{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\right\}$ s.t. $x^{(i)} \in \mathcal{X}$ and $y^{(i)} \in \mathcal{Y}$ for $i = 1, \ldots, n$.

Supervised Learning

- A **hypothesis** or a prediction function is function $h : \mathcal{X} \to \mathcal{Y}$
  - $\mathcal{X}$ is an image, and $\mathcal{Y}$ contains "cat" or "not."
  - $\mathcal{X}$ is a text snippet, and $\mathcal{Y}$ contains "hate speech" or "not."
  - $\mathcal{X}$ is house data, and $\mathcal{Y}$ could be the price.
- A **training set** is a set of pairs $\left\{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\right\}$ s.t. $x^{(i)} \in \mathcal{X}$ and $y^{(i)} \in \mathcal{Y}$ for $i = 1, \ldots, n$.
- Given a training set our goal is to produce a *good* prediction function h
  - Defining "good" will take us a bit. It's a modeling question!
  - We will want to use h on *new* data not in the training set.

Supervised Learning

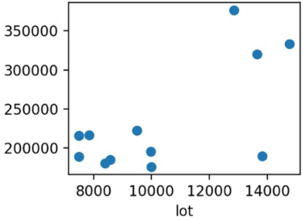
- A **hypothesis** or a prediction function is function $h : \mathcal{X} \to \mathcal{Y}$
  - $\mathcal{X}$ is an image, and $\mathcal{Y}$ contains "cat" or "not."
  - $\mathcal{X}$ is a text snippet, and $\mathcal{Y}$ contains "hate speech" or "not."
  - $\mathcal{X}$ is house data, and $\mathcal{Y}$ could be the price.
- A **training set** is a set of pairs $\{(x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)})\}$ s.t. $x^{(i)} \in \mathcal{X}$ and $y^{(i)} \in \mathcal{Y}$ for $i = 1, \dots, n$.
- Given a training set our goal is to produce a *good* prediction function h
  - Defining "good" will take us a bit. It's a modeling question!
  - We will want to use h on *new* data not in the training set.

- If $\mathcal{Y}$ is continuous, then called a *regression problem*.
- If $\mathcal{Y}$ is discrete, then called a *classification problem*.

Our first example: Regression using Housing Data.

Example Data (Housing Prices from Ames Dataset from Kaggle)

	SalePrice	Lot.Area
4	189900	13830
5	195500	9978
9	189000	7500
10	175900	10000
12	180400	8402
22	216000	7500
36	376162	12858
47	320000	13650
55	216500	7851
56	185088	8577



How do we represent h? (One popular choice)

$h(x) = \theta_0 + \theta_1 x_1$ is an *affine function*

What is logistic Regression?

Linear vs Logistic Regression ?

Linear regression regularization

How do we represent h? (One popular choice)

$h(x) = \theta_0 + \theta_1 x_1$ is an *affine function*

	size
$x^{(1)}$	2104
$x^{(2)}$	2500

	Price
$y^{(1)}$	400
$y^{(2)}$	900

How do we represent h? (One popular choice)

$$h(x) = \theta_0 + \theta_1 x_1 \text{ is an } \textit{affine function}$$

	size
$x^{(1)}$	2104
$x^{(2)}$	2500

	Price
$y^{(1)}$	400
$y^{(2)}$	900

An example prediction?

How do we represent h? (One popular choice)

$$h(x) = \theta_0 + \theta_1 x_1 \text{ is an } \textit{affine function}$$

	size
$x^{(1)}$	2104
$x^{(2)}$	2500

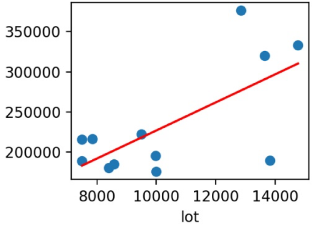
	Price
$y^{(1)}$	400
$y^{(2)}$	900

An example prediction?

Notice the prediction is defined by the *parameters* θ_0 and θ_1. This is a huge reduction in the space of functions!

Simple Line Fit

	SalePrice	Lot.Area
4	189900	13830
5	195500	9978
9	189000	7500
10	175900	10000
12	180400	8402
22	216000	7500
36	376162	12858
47	320000	13650
55	216500	7851
56	185088	8577
58	222500	9505



Slightly More Interesting Data

We add *features* (bedrooms and lot size) to incorporate more information about houses.

	size	bedrooms	lot size
$x^{(1)}$	2104	4	45k
$x^{(2)}$	2500	3	30k

	Price
$y^{(1)}$	400
$y^{(2)}$	900

Slightly More Interesting Data

We add *features* (bedrooms and lot size) to incorporate more information about houses.

	size	bedrooms	lot size
$x^{(1)}$	2104	4	45k
$x^{(2)}$	2500	3	30k

	Price
$y^{(1)}$	400
$y^{(2)}$	900

What's a prediction here?

$$h(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3.$$

Slightly More Interesting Data

We add *features* (bedrooms and lot size) to incorporate more information about houses.

	size	bedrooms	lot size		Price
$x^{(1)}$	2104	4	45k	$y^{(1)}$	400
$x^{(2)}$	2500	3	30k	$y^{(2)}$	900

What's a prediction here?

$$h(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3.$$

With the convention that $x_0 = 1$ we can write:

$$h(x) = \sum_{j=0}^{3} \theta_j x_j$$

Vector Notation for Prediction

	size	bedrooms	lot size
$x^{(1)}$	2104	4	45k
$x^{(2)}$	2500	3	30k

	Price
$y^{(1)}$	400
$y^{(2)}$	900

We write the vectors as (important notation)

$$\theta = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{pmatrix} \text{ and } x^{(1)} = \begin{pmatrix} x_0^{(1)} \\ x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \end{pmatrix} = \begin{pmatrix} 1 \\ 2104 \\ 4 \\ 45 \end{pmatrix} \text{ and } y^{(1)} = 400$$

Vector Notation for Prediction

	size	bedrooms	lot size		Price
$x^{(1)}$	2104	4	45k	$y^{(1)}$	400
$x^{(2)}$	2500	3	30k	$y^{(2)}$	900

We write the vectors as (important notation)

$$\theta = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{pmatrix} \text{ and } x^{(1)} = \begin{pmatrix} x_0^{(1)} \\ x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \end{pmatrix} = \begin{pmatrix} 1 \\ 2104 \\ 4 \\ 45 \end{pmatrix} \text{ and } y^{(1)} = 400$$

We call θ **parameters**, $x^{(i)}$ is the input or the **features**, and the output or **target** is $y^{(i)}$. To be clear,

$$(x, y) \text{ is a training example and } (x^{(i)}, y^{(i)}) \text{ is the } i^{th} \text{ example.}$$

Vector Notation for Prediction

	size	bedrooms	lot size		Price
$x^{(1)}$	2104	4	45k	$y^{(1)}$	400
$x^{(2)}$	2500	3	30k	$y^{(2)}$	900

We write the vectors as (important notation)

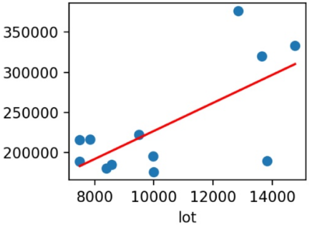
$$\theta = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{pmatrix} \text{ and } x^{(1)} = \begin{pmatrix} x_0^{(1)} \\ x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \end{pmatrix} = \begin{pmatrix} 1 \\ 2104 \\ 4 \\ 45 \end{pmatrix} \text{ and } y^{(1)} = 400$$

We call θ **parameters**, $x^{(i)}$ is the input or the **features**, and the output or **target** is $y^{(i)}$. To be clear,

(x, y) is a training example and $(x^{(i)}, y^{(i)})$ is the i^{th} example.

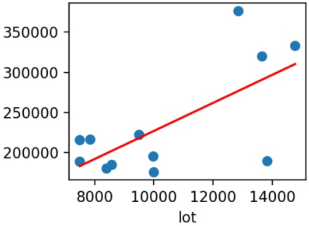
We have n examples (i.e., $i = 1, \ldots, n$). There are d features so $x^{(i)}$ and θ are $d + 1$ dimensional (since $x_0 = 1$)

Visual version of linear regression



Let $h_\theta(x) = \sum_{j=0}^{d} \theta_j x_j$ want to choose θ so that $h_\theta(x) \approx y$.

Visual version of linear regression



Let $h_\theta(x) = \sum_{j=0}^{d} \theta_j x_j$ want to choose θ so that $h_\theta(x) \approx y$. One popular idea called **least squares**

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

Choose

$$\theta = \underset{\theta}{\operatorname{argmin}}\, J(\theta).$$

Linear Regression Summary

- We saw our first hypothesis class *affine* or *linear* functions.
- We refreshed ourselves on notation and introduced terminology like **parameters**, **features**, etc.
- We saw this paradigm that a “good” hypothesis is some how one that *is close to* the data (objective function J).

Linear Regression Summary

- We saw our first hypothesis class *affine* or *linear* functions.
- We refreshed ourselves on notation and introduced terminology like **parameters**, **features**, etc.
- We saw this paradigm that a "good" hypothesis is some how one that *is close to* the data (objective function J).
- Next, we'll see how to solve these equations.

Solving the least squares optimization problem.

Gradient Descent

$$\begin{aligned} \theta^{(0)} &= 0 \\ \theta_j^{(t+1)} &= \theta_j^{(t)} - \alpha \frac{\partial}{\partial \theta_j} J(\theta^{(t)}) \qquad \text{for } j = 0, \dots, d. \end{aligned}$$

Gradient Descent Computation

$$\theta_j^{(t+1)} = \theta_j^{(t)} - \alpha \frac{\partial}{\partial \theta_j} J(\theta^{(t)}) \text{ for } j = 0, \ldots, d.$$

Note that α is called the **learning rate** or **step size**.

Let's compute the derivatives...

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} J(\theta^{(t)}) &= \sum_{i=1}^{n} \frac{1}{2} \frac{\partial}{\partial \theta_j} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 \\
&= \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \frac{\partial}{\partial \theta_j} h_\theta(x^{(i)})
\end{aligned}$$

Gradient Descent Computation

$$\theta_j^{(t+1)} = \theta_j^{(t)} - \alpha \frac{\partial}{\partial \theta_j} J(\theta^{(t)}) \text{ for } j = 0, \ldots, d.$$

Note that α is called the **learning rate** or **step size**.

Let's compute the derivatives...

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} J(\theta^{(t)}) &= \sum_{i=1}^{n} \frac{1}{2} \frac{\partial}{\partial \theta_j} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 \\
&= \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \frac{\partial}{\partial \theta_j} h_\theta(x^{(i)})
\end{aligned}$$

For our *particular* h_θ we have:

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_d x_d \text{ so } \frac{\partial}{\partial \theta_j} h_\theta(x) = x_j$$

Gradient Descent Computation

$$\theta_j^{(t+1)} = \theta_j^{(t)} - \alpha \frac{\partial}{\partial \theta_j} J(\theta^{(t)}) \text{ for } j = 0, \ldots, d.$$

Note that α is called the **learning rate** or **step size**.

Let's compute the derivatives...

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} J(\theta^{(t)}) &= \sum_{i=1}^{n} \frac{1}{2} \frac{\partial}{\partial \theta_j} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 \\
&= \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \frac{\partial}{\partial \theta_j} h_\theta(x^{(i)})
\end{aligned}$$

For our *particular* h_θ we have:

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_d x_d \text{ so } \frac{\partial}{\partial \theta_j} h_\theta(x) = x_j$$

Gradient Descent Computation

Thus, our update rule for component j can be written:

$$\theta_j^{(t+1)} = \theta_j^{(t)} - \alpha \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)}.$$

Gradient Descent Computation

Thus, our update rule for component j can be written:

$$\theta_j^{(t+1)} = \theta_j^{(t)} - \alpha \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)}.$$

We write this in *vector notation* for $j = 0, \ldots, d$ as:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x^{(i)}.$$

Saves us a lot of writing! And easier to understand ...eventually.

Batch Versus Stochastic Minibatch: Motivation

Consider our update rule:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x^{(i)}.$$

- A single update, our rule examines *all* n data points.

Batch Versus Stochastic Minibatch: Motivation

Consider our update rule:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x^{(i)}.$$

- A single update, our rule examines *all* n data points.
- In some modern applications (more later) n may be in the billions or trillions!
  - E.g., we try to "predict" every word on the web.

Batch Versus Stochastic Minibatch: Motivation

Consider our update rule:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x^{(i)}.$$

- A single update, our rule examines *all* n data points.
- In some modern applications (more later) n may be in the billions or trillions!
  - E.g., we try to "predict" every word on the web.
- **Idea** Sample a few points (maybe even just one!) to *approximate* the gradient called **Stochastic Gradient** (SGD).
  - SGD is the workhorse of modern ML, e.g., pytorch and tensorflow.

Stochastic Minibatch

- We randomly select a **batch** of $B \subseteq \{1, \ldots, n\}$ where $|B| < n$.
- We approximate the gradient using just those B points as follows (vs. gradient descent)

$$\frac{1}{|B|} \sum_{j \in B} \left(h_\theta(x^{(j)}) - y^{(j)} \right) x^{(j)} \text{ v.s. } \frac{1}{n} \sum_{j=1}^{n} \left(h_\theta(x^{(j)}) - y^{(j)} \right) x^{(j)}.$$

Stochastic Minibatch

- We randomly select a **batch** of $B \subseteq \{1, \ldots, n\}$ where $|B| < n$.
- We approximate the gradient using just those B points as follows (vs. gradient descent)

$$\frac{1}{|B|} \sum_{j \in B} \left(h_\theta(x^{(j)}) - y^{(j)} \right) x^{(j)} \text{ v.s. } \frac{1}{n} \sum_{j=1}^{n} \left(h_\theta(x^{(j)}) - y^{(j)} \right) x^{(j)}.$$

- So our update rule for SGD is:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha_B \sum_{j \in B} \left(h_\theta(x^{(j)}) - y^{(j)} \right) x^{(j)}.$$

- NB: scaling of $|B|$ versus n is "hidden" inside choice of α_B.

Stochastic Minibatch vs. Gradient Descent

- Recall our rule B points as follows:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha_B \sum_{j \in B} \left(h_\theta(x^{(j)}) - y^{(j)} \right) x^{(j)}.$$

- If $|B| = \{1, \ldots, n\}$ (the whole set), then they coincide.
- Smaller B implies a lower quality approximation of the gradient (higher variance).
- Nevertheless, it may actually converge faster! (Case where the dataset has many copies of the same point–extreme, but lots of redundancy)

Stochastic Minibatch vs. Gradient Descent

- Recall our rule B points as follows:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha_B \sum_{j \in B} \left(h_\theta(x^{(j)}) - y^{(j)} \right) x^{(j)}.$$

- If $|B| = \{1, \ldots, n\}$ (the whole set), then they coincide.
- Smaller B implies a lower quality approximation of the gradient (higher variance).
- Nevertheless, it may actually converge faster! (Case where the dataset has many copies of the same point–extreme, but lots of redundancy)
- In practice, choose B proportional to what works well on modern parallel hardware (GPUs).

Summary of this Subsection of Optimization

- Our goal was to optimize a loss function to find a good predictor.
- We learned about gradient descent and the workhorse algorithm for ML, **Stochastic Gradient Descent** (SGD).
- We touched on the tradeoffs of choosing the right batch size.

Normal Equations

- Least squares with linear hypothesis is *really* special, we can solve it exactly (algebraically)!
  - We'll derive the *Normal Equations* for least squares.

Notation for Least Squares with Linear h_θ

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n}(h_\theta(x^{(i)}) - y^{(i)})^2.$$

Let's get some convenient notation

$$X = \begin{pmatrix} x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(n)} \end{pmatrix} \in \mathbb{R}^{n\times(d+1)} \text{ and } y = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{pmatrix} \in \mathbb{R}^n$$

We may call X the **Design Matrix**.

Notation for Least Squares with Linear h_θ

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n}(h_\theta(x^{(i)}) - y^{(i)})^2.$$

Let's get some convenient notation

$$X = \begin{pmatrix} x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(n)} \end{pmatrix} \in \mathbb{R}^{n\times(d+1)} \text{ and } y = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{pmatrix} \in \mathbb{R}^{n}$$

We may call X the **Design Matrix**.

With this notation for linear $h_\theta(x)$, matrix multiplication is evaluation of h_θ, that is

$$X\theta = \begin{pmatrix} h(x^{(1)}) \\ h(x^{(2)}) \\ \vdots \\ h(x^{(n)}) \end{pmatrix} \text{ and so } J(\theta) = \frac{1}{2}(X\theta - y)^T(X\theta - y)$$

Vector Derivatives

Recall that for a real-valued matrix function $f : \mathbb{R}^{n \times d} \to \mathbb{R}$, we mean

$$\nabla_A f(A) = \begin{pmatrix} \frac{\partial}{\partial_{a_{11}}} f(A) & \frac{\partial}{\partial_{a_{12}}} f(A) & \dots & \frac{\partial}{\partial_{a_{1d}}} f(A) \\ \frac{\partial}{\partial_{a_{21}}} f(A) & \frac{\partial}{\partial_{a_{22}}} f(A) & \dots & \frac{\partial}{\partial_{a_{2d}}} f(A) \\ \vdots & \vdots & & \vdots \\ \frac{\partial}{\partial_{a_{n1}}} f(A) & \frac{\partial}{\partial_{a_{n2}}} f(A) & \dots & \frac{\partial}{\partial_{a_{nd}}} f(A) \end{pmatrix}$$

Here $A \in \mathbb{R}^{n \times d}$.

Vector Derivatives

Recall that for a real-valued matrix function $f : \mathbb{R}^{n \times d} \to \mathbb{R}$, we mean

$$\nabla_A f(A) = \begin{pmatrix} \frac{\partial}{\partial_{a_{11}}} f(A) & \frac{\partial}{\partial_{a_{12}}} f(A) & \dots & \frac{\partial}{\partial_{a_{1d}}} f(A) \\ \frac{\partial}{\partial_{a_{21}}} f(A) & \frac{\partial}{\partial_{a_{22}}} f(A) & \dots & \frac{\partial}{\partial_{a_{2d}}} f(A) \\ \vdots & \vdots & & \vdots \\ \frac{\partial}{\partial_{a_{n1}}} f(A) & \frac{\partial}{\partial_{a_{n2}}} f(A) & \dots & \frac{\partial}{\partial_{a_{nd}}} f(A) \end{pmatrix}$$

Here $A \in \mathbb{R}^{n \times d}$.

- With this notation, to find the minimum of $J(\theta)$ we compute $\nabla_\theta J(\theta) = 0$.
- Note that $\nabla_\theta J(\theta) \in \mathbb{R}^{d+1}$ since $\theta \in \mathbb{R}^{d+1}$.

The normal equation

From our previous derivation,

$$\nabla_\theta J(\theta) = \nabla_\theta \frac{1}{2}(X\theta - y)^T(X\theta - y).$$

multiplying out we have:

$$\nabla_\theta J(\theta) = X^T X\theta - X^T y.$$

Setting $\nabla_\theta J(\theta) = 0$, solving for θ assuming $(X^T X)^{-1}$ exists, we obtain:

$$\theta = \left(X^T X\right)^{-1} X^T y.$$

We have the optimal solution for θ!

Some slight cheating...

$$\theta = \left(X^T X\right)^{-1} X^T y.$$

- We've assumed $(X^T X)^{-1}$ exists. What happens if not? Is θ uniquely defined? Up to what?
- Why was $\nabla_\theta J(\theta) = 0$ a minimum? Notice that

$$\nabla_\theta^2 J(\theta) = \nabla_\theta \left(X^T X\theta - X^T y\right) = X^T X \succeq 0$$

  that is, it's second derivative is positive (semi)definite.
- We did some quick vector calculus, if this isn't familiar practice on Friday!

Summary from Today

- We saw a *lot of notation*
  - The TAs can help you practice on Friday!
- We learned about linear regression: the model, how to solve, and more.
- We learned the workhorse algorithm for ML called **SGD**.
- Next time: Classification!