Download the example code and follow along with lecture!

https://stanfordpython.com/res/lecture-code/Lecture9.zip

And if you don't have `scikit-learn` installed, run:

```
$ pip3 install sklearn
```

Machine Learning

February 24, 2020

Week 8 of CS 41

- Supervised Learning
  - The classic problem setup
- Least Squares – Review
  - An old problem through a new lens
- Regression and Classification
  - A Tale of Two Problems
- Generalizing Least Squares – Gradient Descent
  - A more general paradigm
- Neural Networks
  - The cutting edge!
- Unsupervised Learning
  - Learning without labels!

Today's Goals

You should leave today knowing how to translate some basic ML principles into Python.

$\neq$

You should leave today with an in-depth intuition for machine learning theory.

Today's Goals

You should leave today knowing how to translate some basic ML principles into Python.

- We will review some theory – and *this stuff gets hard!*
- We will review theory, to build intuition, then spend much of our time with code examples.
- Disclaimer: if you'd like a rigorous treatment of the math, take CS229 (which, for the record, is a great class!).
  - Here, we're focused on the high-level and implementations in Python.

A Guiding Pattern

Complicated Math, Simple Code

Supervised Learning

Problem setup!

What is Machine Learning?

- **Machine Learning**: Computer learning a task *without* being explicitly programmed.
  - Selecting pattern from a family of patterns that best matches the data.
- Two categories (of many more!) we will cover today:
  - **Supervised Learning**: making predictions on labeled data.
  - **Unsupervised Learning**: finding interesting patterns in unlabeled data.
- A **regression task** is one where our predictions are real numbers; a **classification task** is one where our predictions are categories.

Supervised Learning

- Given a dataset, X, and labels, Y, predict labels for unseen examples.
- Let's try to predict the price of diamonds given certain features of the diamond.

	Carat	Depth	...	Z-Axis Measurement
Diamond 0	0.23	61.5	...	2.43
Diamond 1	0.21	59.8	...	2.31
⋮	⋮	⋮	⋱	⋮
Diamond $m-1$	0.29	62.4	...	2.63

Diamond 0	326
Diamond 1	326
⋮	⋮
Diamond $m-1$	334

Supervised Learning

- Given a dataset, X, and labels, Y, predict labels for unseen examples.
- Let's try to predict the price of diamonds given certain features of the diamond.

X :

	Carat	Depth	...	Z-Axis Measurement
Diamond 0	0.23	61.5	...	2.43
Diamond 1	0.21	59.8	...	2.31
⋮	⋮	⋮	⋱	⋮
Diamond $m-1$	0.29	62.4	...	2.63

Diamond 0	326
Diamond 1	326
⋮	⋮
Diamond $m-1$	334

Supervised Learning

- Given a dataset, X, and labels, Y, predict labels for unseen examples.
- Let's try to predict the price of diamonds given certain features of the diamond.

X :

	Carat	Depth	...	Z-Axis Measurement
Diamond 0	0.23	61.5	...	2.43
Diamond 1	0.21	59.8	...	2.31
⋮	⋮	⋮	⋱	⋮
Diamond $m-1$	0.29	62.4	...	2.63

Y :

Diamond 0	326
Diamond 1	326
⋮	⋮
Diamond $m-1$	334

Supervised Learning

X :

	Carat	Depth	...	Z-Axis Measurement
Diamond 0	0.23	61.5	...	2.43
Diamond 1	0.21	59.8	...	2.31
⋮	⋮	⋮	⋱	⋮
Diamond $m-1$	0.29	62.4	...	2.63

Y :

Diamond 0	326
Diamond 1	326
⋮	⋮
Diamond $m-1$	334

X_{test} :

New Diamond 0	0.63	55	...	2.48

Y_{test} :

New Diamond 0	?

Supervised Learning

X :

	Carat			
Diamond 0	0.2			
Diamond 1	0.2			
⋮	⋮			
Diamond $m-1$	0.25			

We will use X and Y to refer to the entire dataset and labels, respectively.

We will use x and y to refer to one example from the dataset and one label, respectively.

				326
				326
				⋮
				1

X_{test} :

New Diamond 0	0.63	55	...	2.48

Y_{test} :

New Diamond 0	?

Features and Labels

- **Features:** characteristics of each example that provide information.
  - Features in our diamond price prediction example:
    - Carat
    - Depth
    - Table
    - X, Y, Z measurements
- **Label:** the characteristic we are trying to predict.
  - In the diamond price prediction example, we are trying to predict price.

Models

- **Model:** a mathematical expression that takes in features of an example and returns a label.
  - A model's **parameters** define what prediction a model makes on a given example.

$$h(x;\ \theta) = \hat{y}$$

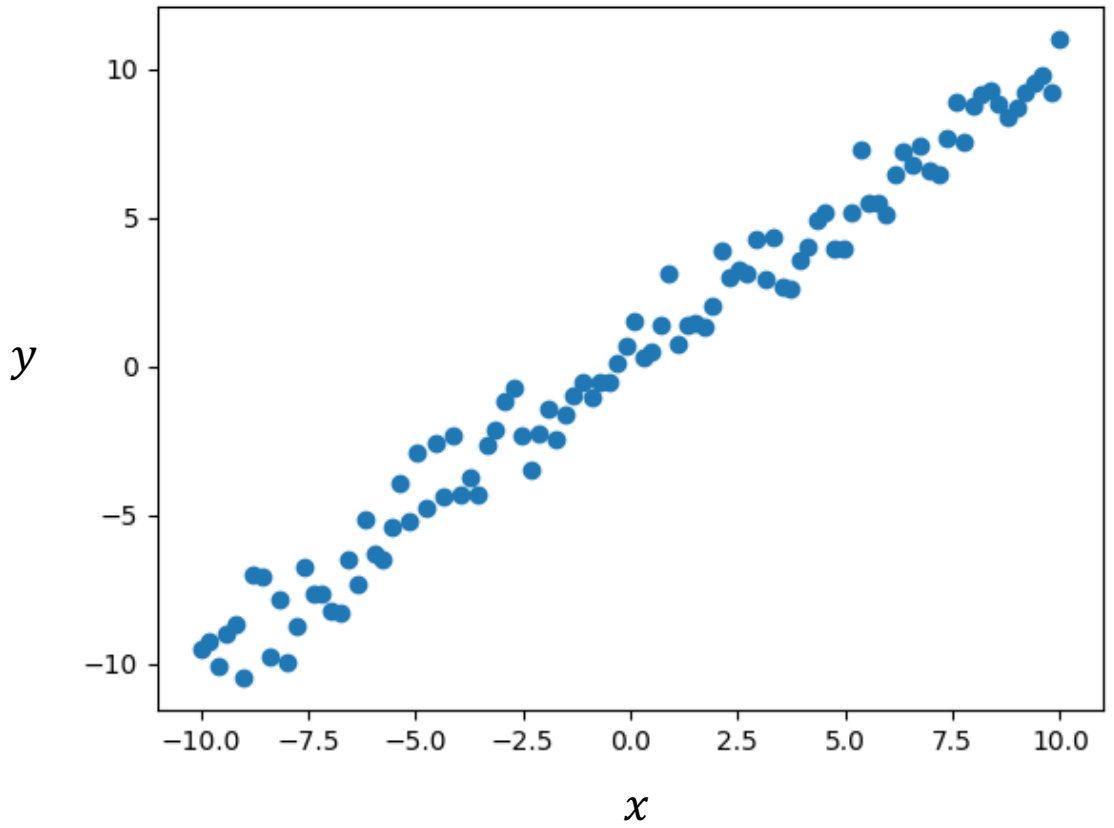
- In machine learning, we try to learn parameters such that our function h which *does a good job* of predicting labels.

Always, always, *always* look at your data!

(Often provides invaluable insight into which types of models will be effective).

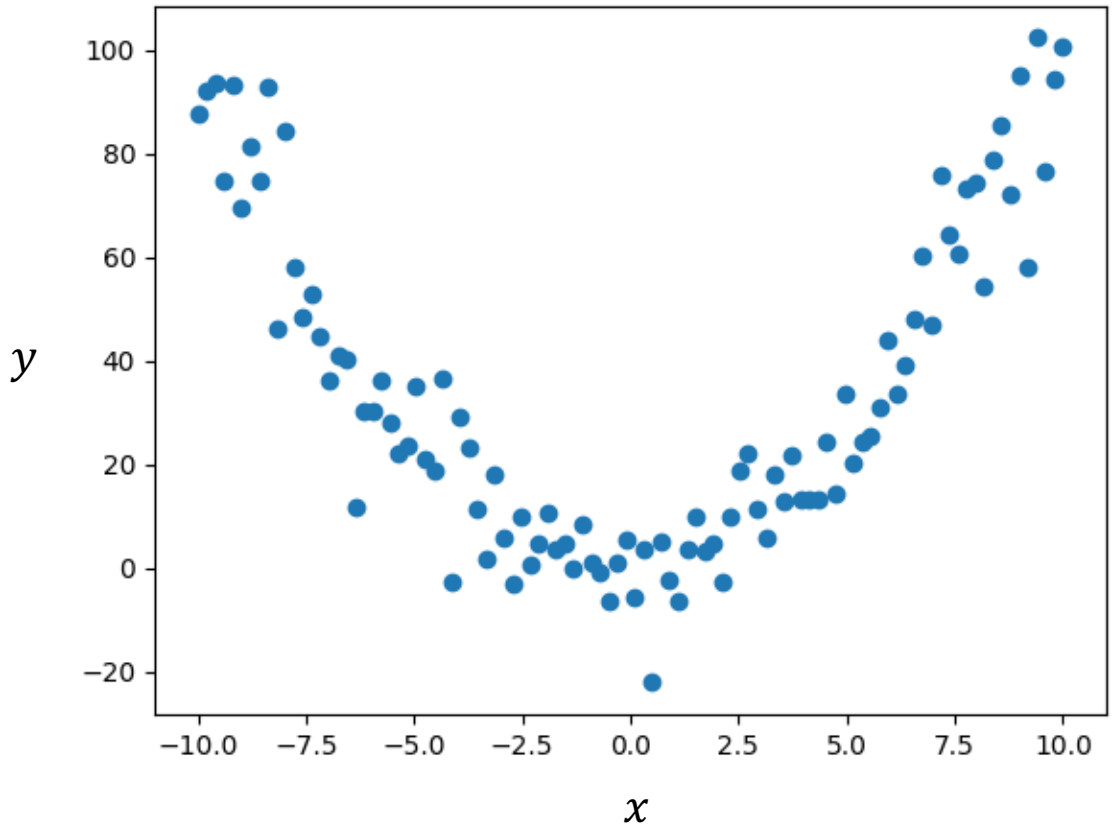
Models

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



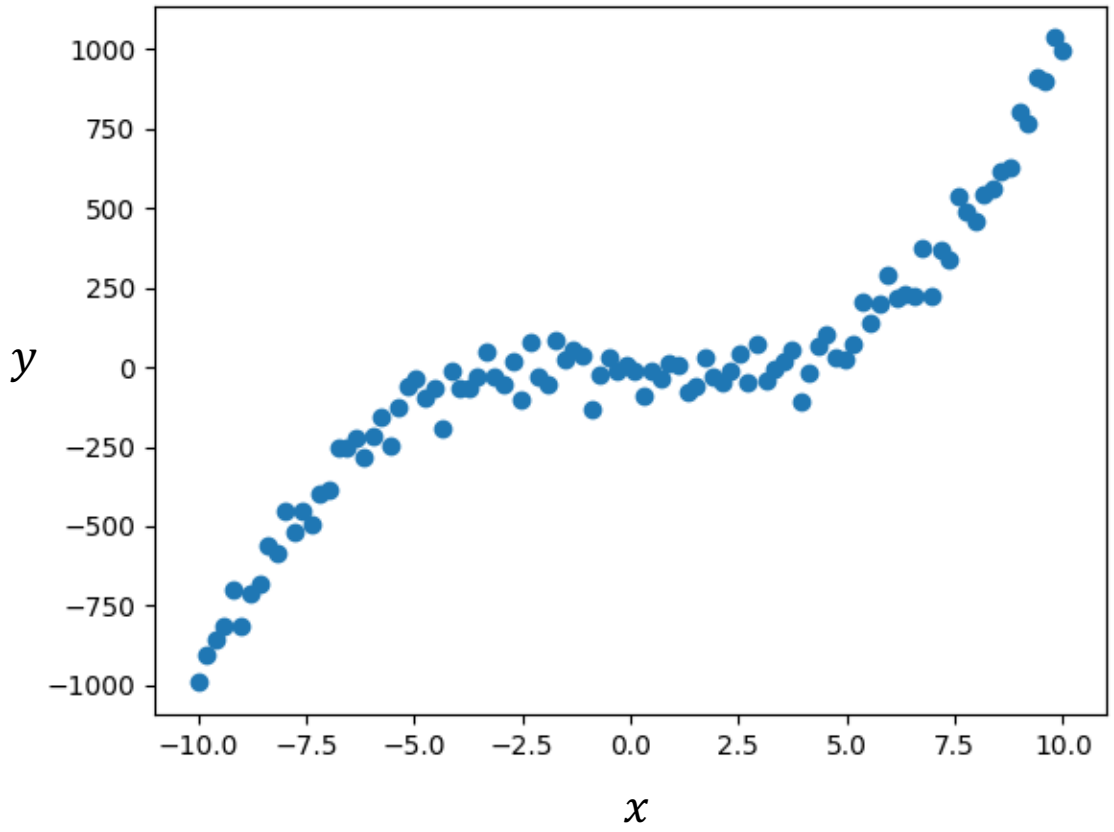
Models

$$h(x) = \theta x^2$$



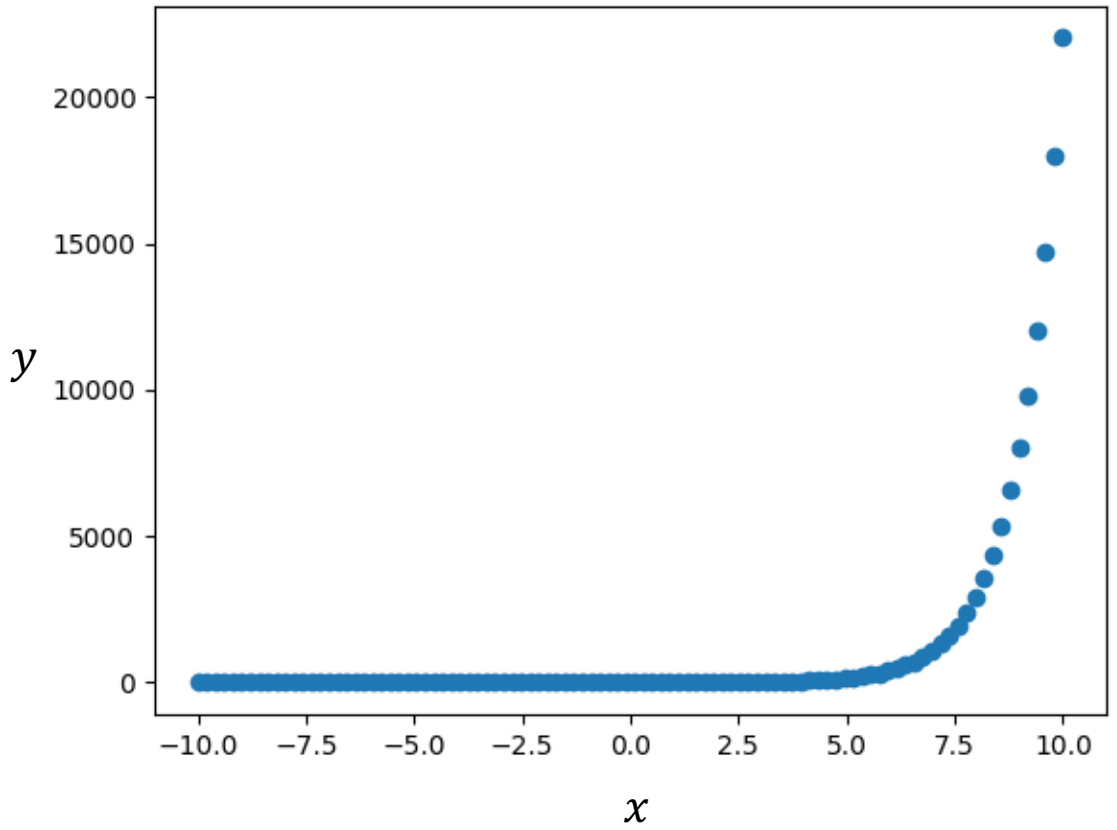
Models

$$h(x) = \theta x^3$$



Models

$$h(x) = \theta e^{x}$$



What is a "Good Job"?

- **Loss Function:** a function which takes in your model's predictions on the dataset and the dataset labels. It returns a *score of how badly you are doing*.

$$L\left(Y, \hat{Y}\right)$$

- In supervised learning, we *set model parameters to minimize the loss function*.
  - The process of finding the optimal parameters is called **training**.

Evaluating Machine Learning Models

X_{train}:

0.23	61.5	…	2.43
0.21	59.8	…	2.31
⋮	⋮	⋱	⋮

Y_{train}:

326
326
⋮

X_{valid}:

0.34	55.1	…	2.1
⋮	⋮	⋱	⋮

Y_{valid}:

508
⋮

X_{test}:

0.25	62	…	2.45
⋮	⋮	⋱	⋮

Y_{test}:

375
⋮

Evaluating Machine Learning Models

80%

X_{train}:

0.23	61.5	…	2.43
0.21	59.8	…	2.31
⋮	⋮	⋱	⋮

Y_{train}:

326
326
⋮

10%

X_{valid}:

0.34	55.1	…	2.1
⋮	⋮	⋱	⋮

Y_{valid}:

508
⋮

10%

X_{test}:

0.25	62	…	2.45
⋮	⋮	⋱	⋮

Y_{test}:

375
⋮

Evaluating Machine Learning Models

80%

X_{train}:

0.23	61.5	…	2.43
⋮	⋮	⋱	⋮

Y_{train}:

326
326
⋮

Fit Model Parameters

10%

X_{valid}:

0.34	55.1	…	2.1
⋮	⋮	⋱	⋮

Y_{valid}:

508
⋮

10%

X_{test}:

0.25	62	…	2.45
⋮	⋮	⋱	⋮

Y_{test}:

375
⋮

Evaluating Machine Learning Models

80%

X_{train}:

0.23	61.5	…	2.43
0.21	59.8	…	2.31
⋮	⋮	⋱	⋮

Y_{train}:

326
326
⋮

10%

X_{valid}:

0.34	55.1	…	2.1
⋮	⋮	⋱	⋮

Y_{valid}:

508
⋮

Make Predictions; Tune Model

10%

X_{test}:

0.25	62	…	2.45
⋮	⋮	⋱	⋮

Y_{test}:

375
⋮

Evaluating Machine Learning Models

80%

X_{train}:

0.23	61.5	...	2.43
0.21	59.8	...	2.31
⋮	⋮	⋱	⋮

Y_{train}:

326
326
⋮

10%

X_{valid}:

0.34	55.1	...	2.1
⋮	⋮	⋱	⋮

Y_{valid}:

508
⋮

10%

X_{test}:

0.25	62	...	2.45
⋮	⋮	⋱	⋮

Y_{test}:

375
⋮

Test Predictive Abilities

Evaluating Machine Learning Models

- Assume we have learned parameters from our training set, and now have an $h(x) = \hat{y}$.
- We can run this on our validation set to obtain $\hat{Y}$ (which we obtain from $h(x_i)$ for all x_i in the validation set). Also have Y (comes with validation set)
- In classification tasks, we can use raw accuracy ("how many times did $y = \hat{y}$ in the validation set?"), or the weighted f1 score.

Least Squares

Another lens on an old problem!

Last Time's Setup...

- Assume that we want to build a model to predict the price of a diamond given certain characteristics.
- X is the *feature matrix*. The rows represent diamonds the value of which we know, the columns represent features.
- Y is the *labels vector*. This is the vector in which we list out the prices of the diamonds from our feature matrix.

	Carat	Depth	...	Z-Axis Measurement
Diamond 0	0.23	61.5	...	2.43
Diamond 1	0.21	59.8	...	2.31
⋮	⋮	⋮	⋱	⋮
Diamond $m-1$	0.29	62.4	...	2.63

	Diamond Price
Diamond 0	326
Diamond 1	326
⋮	⋮
Diamond $m-1$	334

Last Time's Setup…

- We want to learn some set of $\theta_1, \theta_2, \ldots \theta_n$ such that:

$$\text{(Row 1): } \theta_1(0.23) + \theta_2(61.5) + \cdots + \theta_n(2.43) \approx 0.9$$
$$\text{(Row 2): } \theta_1(0.95) + \theta_2(59.8) + \cdots + \theta_n(2.31) \approx 0.85$$
$$\vdots$$

Where did these come from?

	Carat	Depth	…	Z-Axis Measurement
Diamond 0	0.23	61.5	…	2.43
Diamond 1	0.21	59.8	…	2.31
⋮	⋮	⋮	⋱	⋮
Diamond $m-1$	0.29	62.4	…	2.63

	Diamond Price
Diamond 0	326
Diamond 1	326
⋮	⋮
Diamond $m-1$	334

Last Time We Made An Assumption…

- We want to learn some set of $\theta_1, \theta_2, \ldots \theta_n$ such that:

(Row 1): $\theta_1(0.23) + \theta_2(61.5) + \cdots + \theta_n(2.43) \approx 0.9$

(Row 2): $\theta_1(0.95) + \theta_2(59.8) + \cdots + \theta_n(2.31) \approx 0.85$

Where did these come from?

$$h(x) = \theta_1 x_1 + \theta_2 x_2 + \cdots$$

	Carat	Depth	…	Z-Axis Measurement
Diamond 0	0.23	61.5	…	2.43
Diamond 1	0.21	59.8	…	2.31
⋮	⋮	⋮	⋱	⋮
Diamond $m-1$	0.29	62.4	…	2.63

	Diamond Price
Diamond 0	326
Diamond 1	326
⋮	⋮
Diamond $m-1$	334

We say our function h is *parameterized by* $\theta_1, \theta_2 \ldots \theta_n$.

Our Approach

- Step 1: Define a loss function.

$$L(Y, \hat{Y}) = \sqrt{\sum (y_i - \hat{y}_i)^2}$$

Y	$\hat{Y}$	$(y_i - \hat{y}_i)^2$
512	641	$(512 - 641)^2$
634	420	$(634 - 420)^2$
⋮	⋮	⋮
1391	1450	$(1391 - 1450)^2$

- Step 2: Find $\theta_1, \theta_2, \ldots \theta_n$ to minimize the loss function. (Python does this for you!)

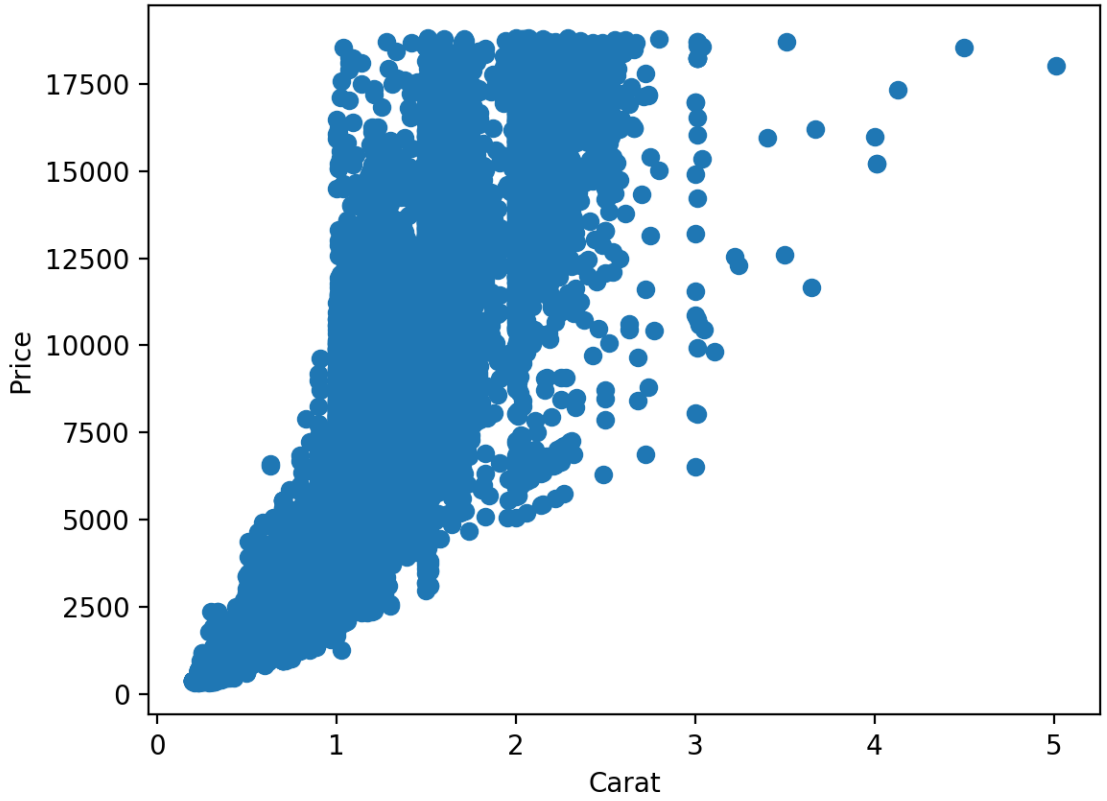
Last Time's Conclusion

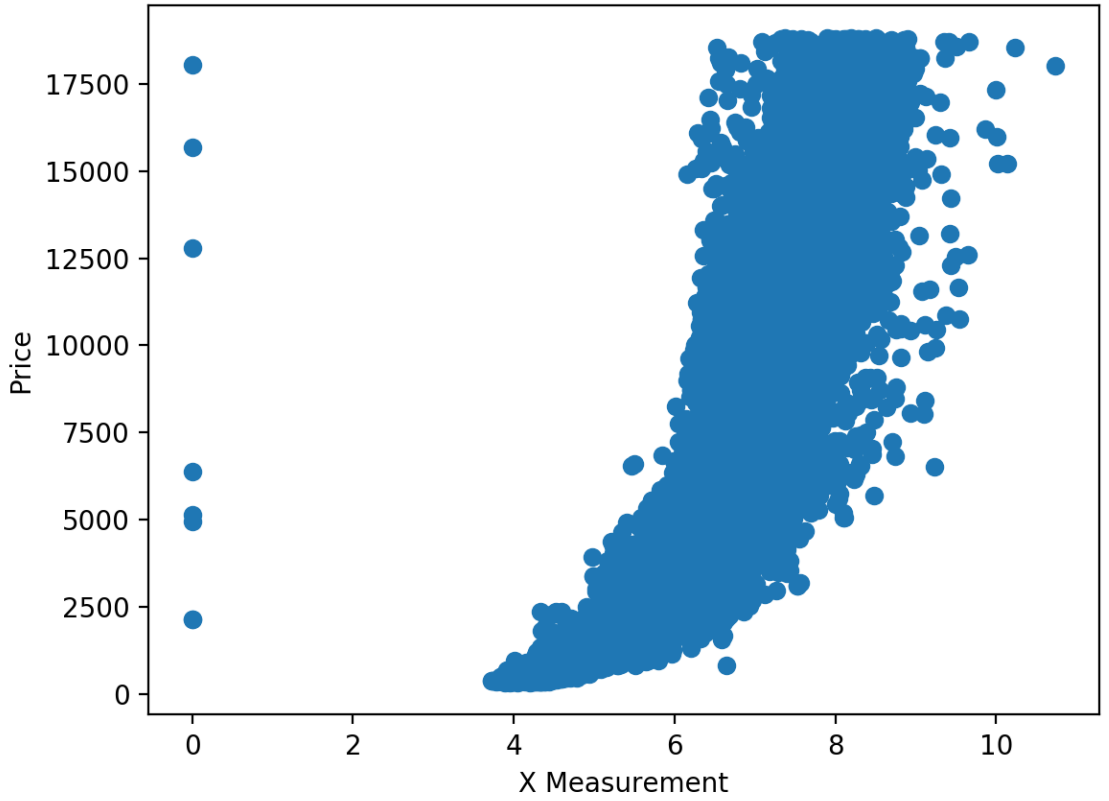
- NumPy has a builtin function that allows us to compute this! Woohoo!

```
# Returns solution vector, θ
>>> theta = np.linalg.lstsq(X, Y)
array([9538.82, 25.43, ... -554.25])
```

Adding Complexity…

- Let's take another look at our data… do these look linear?





Adding Complexity…

- Add in additional features to help us perform better. This process is called **feature selection.**
  - We did this already in `triangulate.py` in the quest for the unicorn!
  - Our features consisted of latitude and longitude coordinates. We added squared features to create a quadratic which fit the data better.
  - Features are generally derived from a visual glance at various data parameters.

The More The Merrier… Right?

- Features add more information.
- More complex models can express greater varieties of data.
- So… why not add a ton of features and make super complicated models?
- **The Overfitting Problem:** if your model can approximate your data too well, it loses predictive abilities on future data.
  - During training, we prioritize fitting the training data well. We haven't seen the validation or test data, so can't take that into account during training!

From Regression to Classification

One more layer!

Similar Problem Setup…

- I'm buying diamonds, but… do I have to sell a kidney to afford them?
  - Yes, if the diamond costs more than $4,000.
  - No, if the diamond costs less than $4,000.
- Given features of a diamond, do I have to sell my kidney?

	Carat	Depth	…	Z-Axis Measurement
Diamond 0	0.23	61.5	…	2.43
Diamond 1	0.21	59.8	…	2.31
⋮	⋮	⋮	⋱	⋮
Diamond $m-1$	1.5	62.4	…	2.63

	Diamond Price
Diamond 0	0
Diamond 1	0
⋮	⋮
Diamond $m-1$	1

Similar Problem Setup…

- I'm buying diamonds, but… do I have to sell a kidney to afford them?
  - Yes, if the diamond costs more than $4,000.
  - No, if the diamond costs less than $4,000.
- Given features of a diamond, do I have to sell my kidney?

	Carat	Depth	…	Z-Axis Measurement
Diamond 0	0.23	61.5	…	2.43
Diamond 1	0.21	59.8	…	2.31
⋮	⋮	⋮	⋱	⋮
Diamond $m-1$	1.5	62.4	…	2.63

	Diamond Price
Diamond 0	0
Diamond 1	0
⋮	⋮
Diamond $m-1$	1

And Similar Assumptions…

- I'm buying diamonds, but… do I have to sell a kidney to afford them?
  - Yes, if the diamond costs more than $10,000.
  - No, if the diamond costs less than $10,000.
- Given features of a diamond, do I have to sell my kidney?

$$h(x) = \theta_1 x_1 + \theta_2 x_2 + \cdots$$

	Carat	Depth	…	Z-Axis Measurement
Diamond 0	0.23	61.5	…	2.43
Diamond 1	0.21	59.8	…	2.31
⋮	⋮	⋮	⋮	⋮
Diamond $m-1$	1.5	62.4	…	2.63

	Diamond Price
Diamond 0	0
Diamond 1	0
⋮	⋮
Diamond $m-1$	1

$$L(Y, \hat{Y}) = \sqrt{\sum (y_i - \hat{y}_i)^2}$$

And Similar Assumptions…

- I'm buying diamonds, but… do I have to sell a kidney to afford them?
  - Yes, if the diamond costs more than $10,000.
  - No, if the diamond costs less than $10,000.

Given features of a diamond, do I have to sell my kidney?

What is this σ? Where's it coming from?

$$h(x) = \sigma(\theta_1 x_1 + \theta_2 x_2 + \cdots)$$

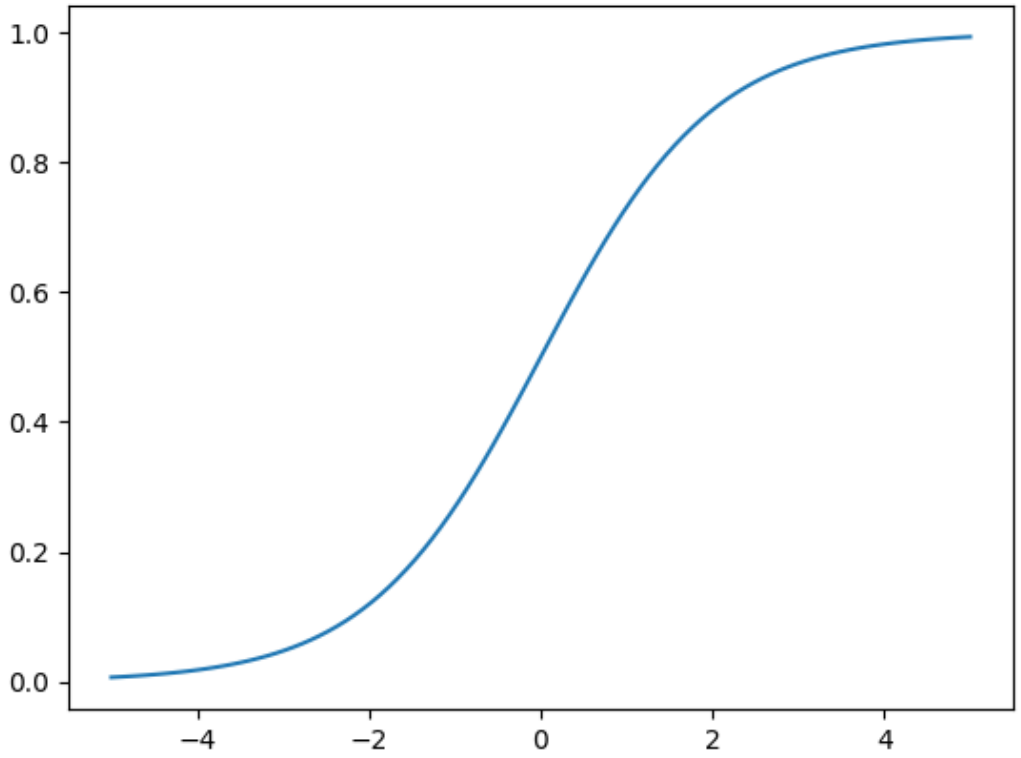
	Carat	Depth	…	Z-Axis Measurement
Diamond 0	0.23	61.5	…	2.43
Diamond 1	0.21	59.8	…	2.31
⋮	⋮	⋮	⋱	⋮
Diamond $m-1$	1.5	62.4	…	2.63

	Diamond Price
Diamond 0	0
Diamond 1	0
⋮	⋮
Diamond $m-1$	1

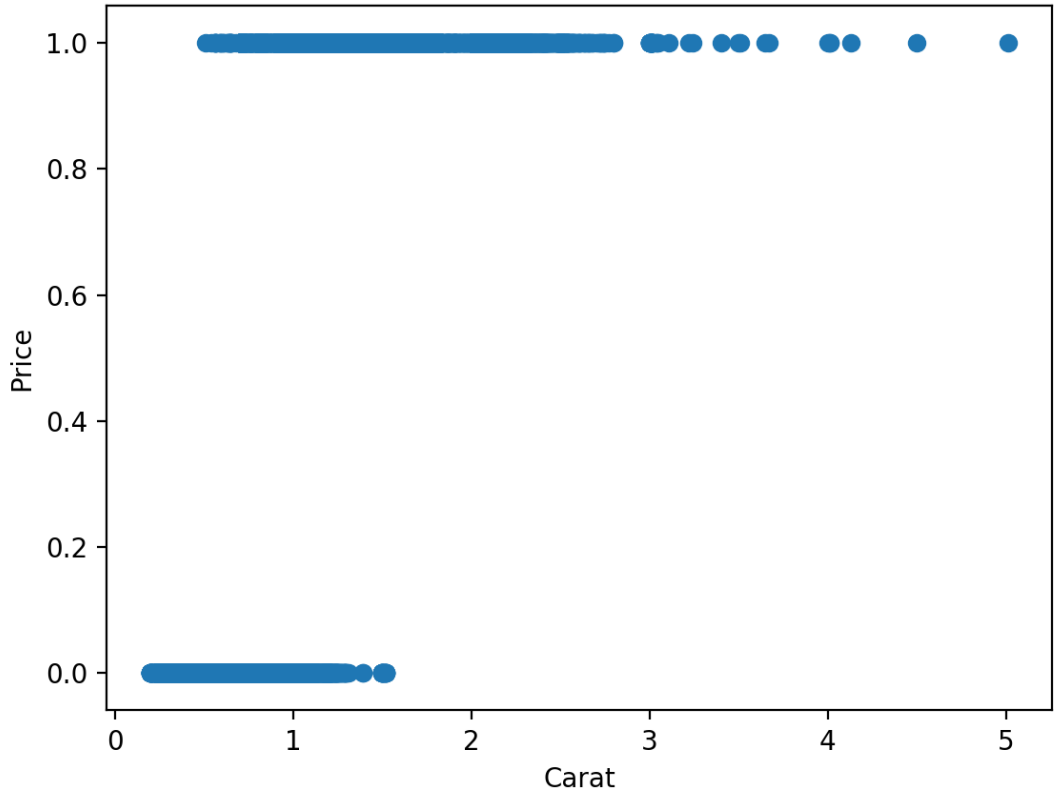
$$L(Y, \hat{Y}) = \sqrt{\sum (y_i - \hat{y}_i)^2}$$

The Logistic Function

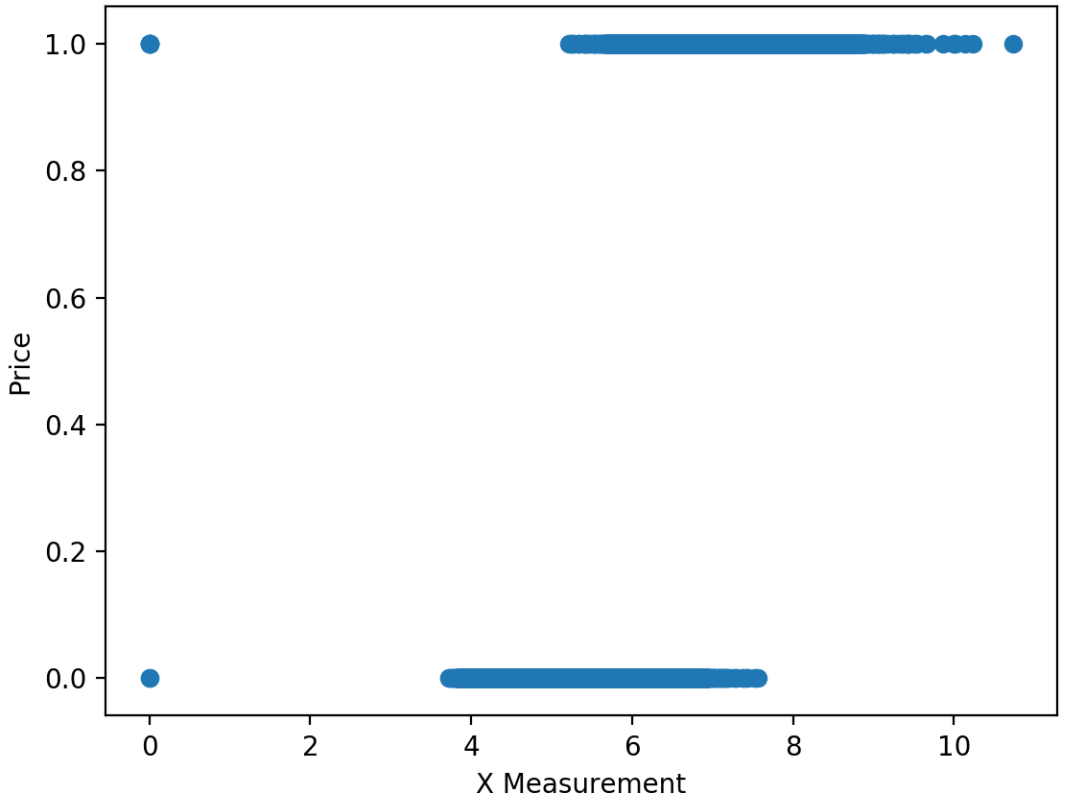
- Take all numbers and squeeze them between 0 and 1.
- Calculated by $y = \frac{1}{1+e^{-x}}$
- Take previous prediction, and pass it through the logistic function.
  - Predict 1 if the result is greater than 0.5, 0 otherwise.



Looking at the Data...



Looking at the Data…



Generalizing Least Squares

More magic!

Recall Our Approach

- Step 1: Define a loss function.

$$L(Y, \hat{Y}) = \sqrt{\sum (y_i - \hat{y}_i)^2}$$

Y	$\hat{Y}$	$(y_i - \hat{y}_i)^2$
512	641	$(512 - 641)^2$
634	420	$(634 - 420)^2$
⋮	⋮	⋮
1391	1450	$(1391 - 1450)^2$

- Step 2: Find $\theta_1, \theta_2, \ldots \theta_n$ to minimize the loss function. (Python does this for you!)

Recall Our Approach

- Step 1: Define a loss function.

$$L(Y,\hat{Y}) = \sqrt{\sum (y_i - \hat{y}_i)^2}$$

Y	$\hat{Y}$	$(y_i - \hat{y}_i)^2$
512	641	$(512-641)^2$
634	420	$(634-420)^2$
⋮	⋮	⋮
1391	1450	$(1391-1450)^2$

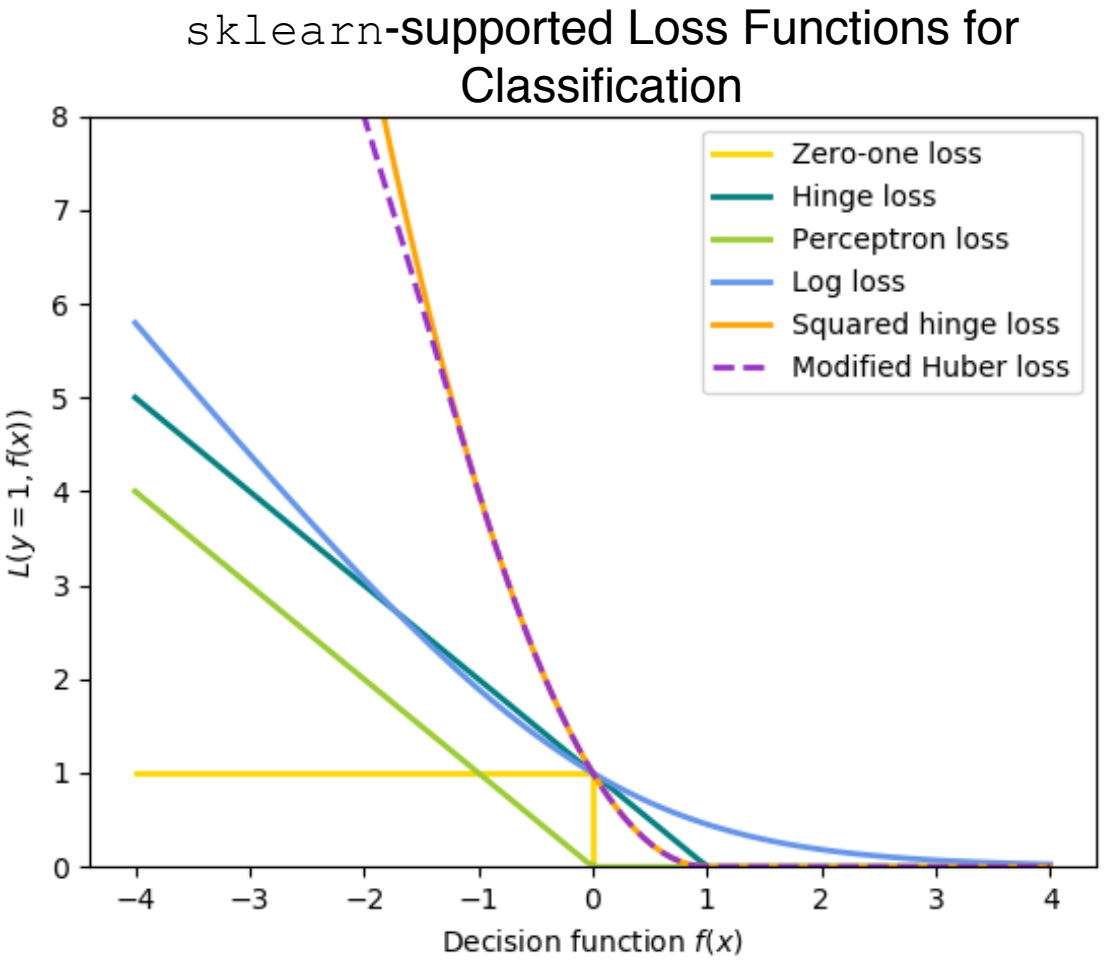
Why… least *squares*?

- Sensitive to outlier values…

- Step 2: Find $\theta_1, \theta_2, \ldots \theta_n$ to minimize the loss function. (Python does this for you!)

Other Loss Functions

- Regression:
  - Squared Loss (which we've seen!)
  - Huber Loss
  - Epsilon-Insensitive Loss
- Classification:
  - Loads more that are supported!



Least Squares Is Nice

- There is an equation to solve least-squares:

$$(X^T X)^{-1} X^T Y$$

- With other loss functions, we can't guarantee a closed-form solution in the same way.
  - So we need some way of taking an arbitrary convex function, and finding its minimum.

Gradient Descent

- Stuck on a mountain with some fog.
- How would you get down from the mountain?

Look at your current position, move in direction of descending slope. Repeat the process.

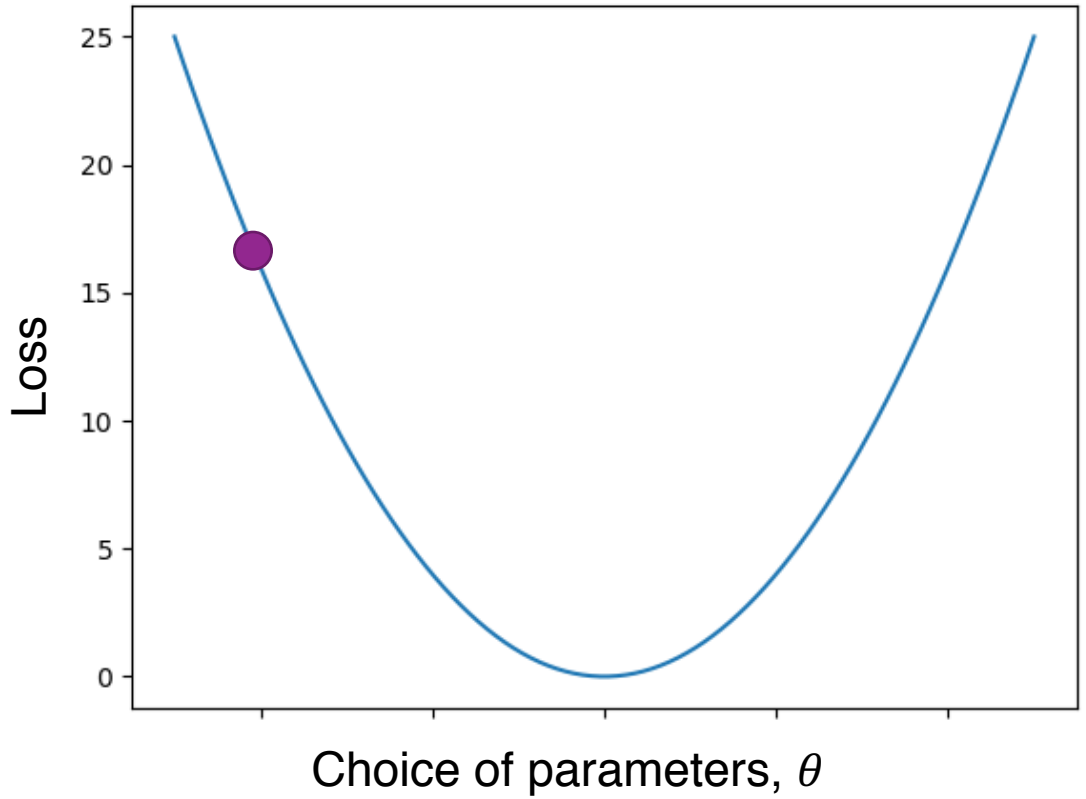
The Loss Function, Revisited

$$L\left(Y, \hat{Y}\right)$$

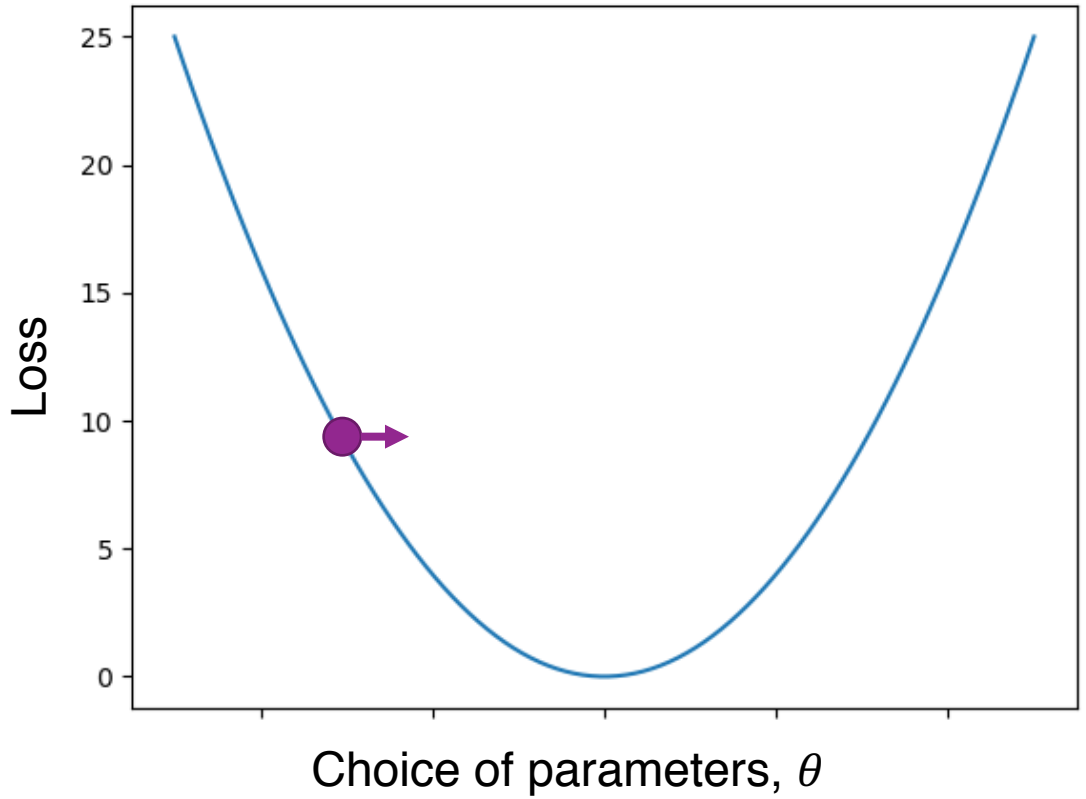
- Loss depends on predictions, and predictions depend on parameters.
- Can equivalently write, $L(\theta)$.

The Gradient Descent Procedure

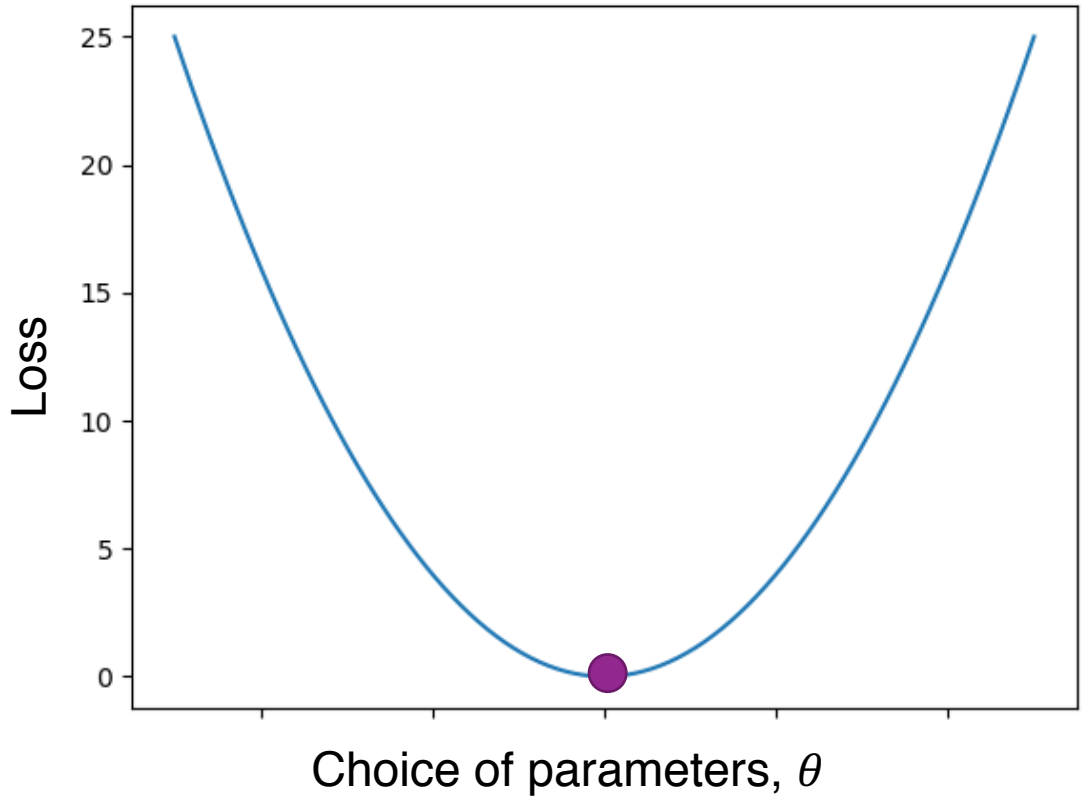
- Is this a good choice?



The Gradient Descent Procedure



The Gradient Descent Procedure



A Little Bit of Randomness…

- This is a slow procedure, in part because we need to calculate the gradient of the loss function (read: really large sum) for each choice of parameters.
- Instead, it's common to pick a random point, calculate the gradient of the loss at that point, and update the direction of movement accordingly.
  - Less predictable, but way faster!
- This is known as **stochastic gradient descent**.

The Gradient Descent Procedure

- This is what's happening behind the scenes! (And why ML models take awhile to train).
- To run in Python, we're going to use the `scikit-learn` package.

```
import sklearn.linear_model

# Fit the model to the training set, using L2 (or "squared") penalty/loss
clf = sklearn.linear_model.LogisticRegression(penalty="l2").fit(X_train, Y_train)

# Predict using a collection of new data points
predictions = clf.predict(X_valid)
```

Assignment 2 - Debrief

- Fantastic job everyone!
- Grades will be out soon – expecting within the next ~1-2 days.
  - More late days and more manual additions on our part, such as the bonus for students who found the unicorn.
- General Feedback:
  - Functionality was high – nice work!
  - Most of the style is there – but we've added individual feedback, so please take the time to check it out once it's released!

Assignment 3 – Final Project

- Project proposals are out!
  - Due **Wednesday at 11:59PM.**
  - We will be giving you feedback on your proposals, so watch out for that!
- Projects will be due **March 9, at 11:59PM**.
  - All details should be on the course site!
- Come chat with us or your TA!
  - We'd love to act as a sounding board for ideas or help point you in the best direction for your project.

A Brief Rewind…

What is Python, anyway?

Guido Van Rossum
Former Benevolent Dictator for Life (BDFL)

- Python 1: 1994
  - A middle ground between C and shell scripts.
- Python 2: 2000
- Python 3: 2008
  - We're using Python 3.8

The Road Ahead

- Wednesday 2/26 – Standard Libraries and Machine Learning Lab!
- Monday 3/2 – Truncated Matplotlib Lecture.
- Wednesday 3/4 – Project Work Time.
- Monday 3/9 – Project Presentations.
- Wednesday 3/11 – Super secret special surprise with unicorn magic! 🦄

Attendance Form

Link: http://iamhere.stanfordpython.com

Code: **UNICORNS_THE_NEW_ELECTRICITY**

If you're not here, fill out this alternative form:

http://bit.ly/32lSx4h

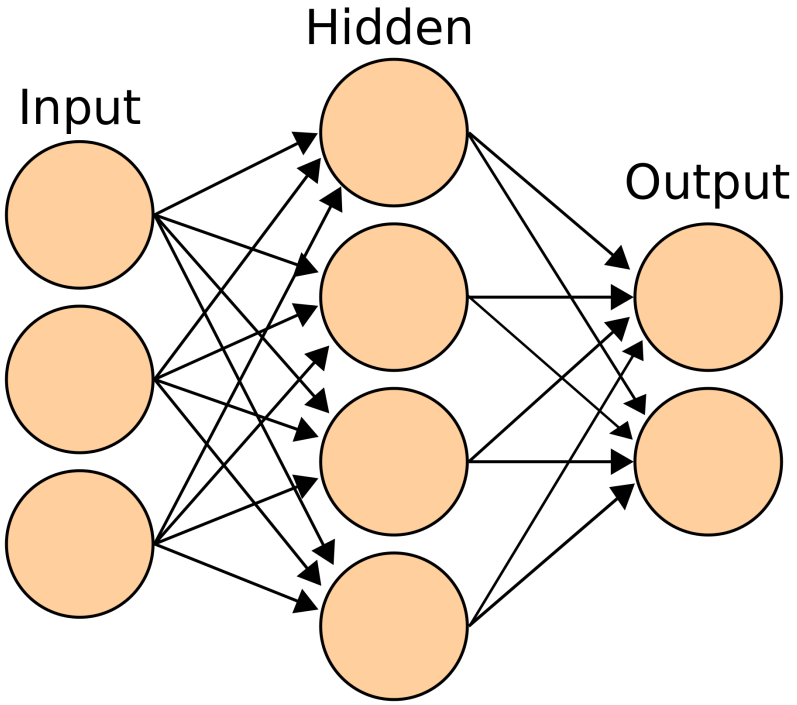
Halftime

Neural Networks

The Cutting Edge!

What is a Neural Network?

- A neural network is a chained sequence of regressions designed to model neurons in the brain.
- Each node, like a neuron, has an "activation threshold" above which it sends a signal to the subsequent node.
- **Input layers** directly receive data. **Hidden layers** are intermediaries which receive inputs from other nodes. The **output layer** returns the result of the prediction.



Inside the Neuron...

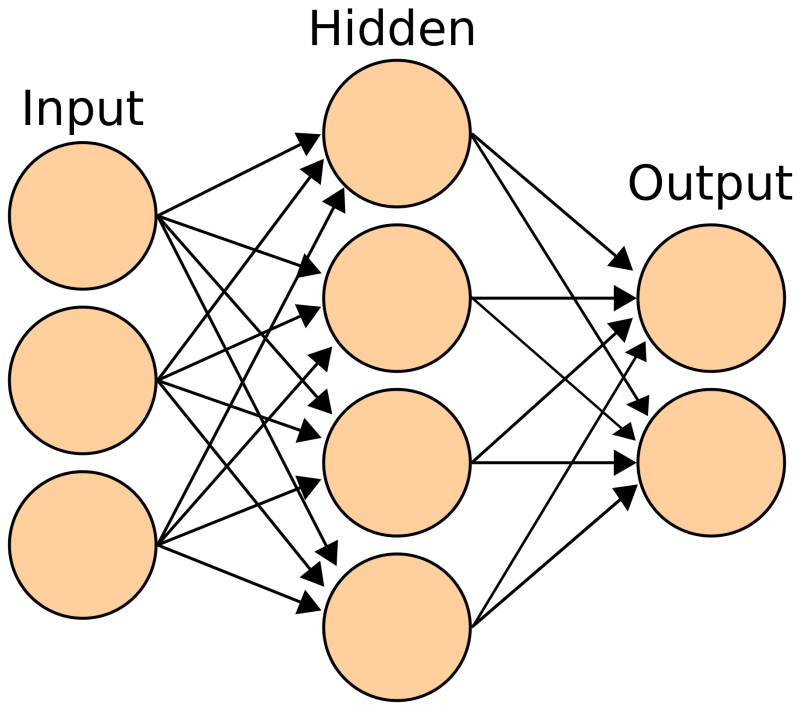
- Each node in a neural network performs linear regression of the form with which we are familiar: learns weights, produces output by:

$$h(x) = f(\theta_1 x_1 + \theta_2 x_2 + \cdots)$$

- Here, f refers to the **activation function**, through which the regression is passed before passing the result on to the next node.

Where the Magic Happens…

- Many nodes, many connections, make neural networks extremely expressive forms of machine learning.
  - This is a 3-layer neural network, with layers of size 3, 4, 2, respectively.



Where the Magic Happens…

- The number of nodes and the pattern in which they are connected are called hyperparameters.
  - Hyperparameters feature in different models, too – in gradient descent, the step size is a hyperparameter.
- **Hyperparameter**: property of the model, the value of which is set before the learning process begins.
- Hyperparameters are often manually tuned – this is why the validation set is so important!
  - Tune the hyperparameters to maximize validation accuracy, then run on the test set as the final exercise.
  - Avoids introducing bias into the process.

Optimizing Chaos!

- Neural networks are so complicated!! How do we find the right weights in each node?
  - Stochastic gradient descent… with heavy modifications.
  - **Backpropagation** is the process by which we take the gradient across an entire neural network to determine the direction in which to update the nodes.
  - *It is very computationally expensive*, hence neural networks take time to train.

Our First Neural Network

- Luckily, `scikit-learn` simplifies the process:

```
from sklearn.neural_network import MLPClassifier

clf = MLPClassifier(
    hidden_layer_sizes=(32, 64, 128, 64, 32),
    learning_rate_init=0.0008, max_iter=400
    ).fit(X_train, Y_train)

predictions = clf.predict(X_valid)
```

- Where do we get all these hyperparameters?
  - I… guessed? If I was doing this for a research paper, I'd try something more methodical like iterate (but to do so exhaustively could take days/weeks). 😝

Unsupervised Learning

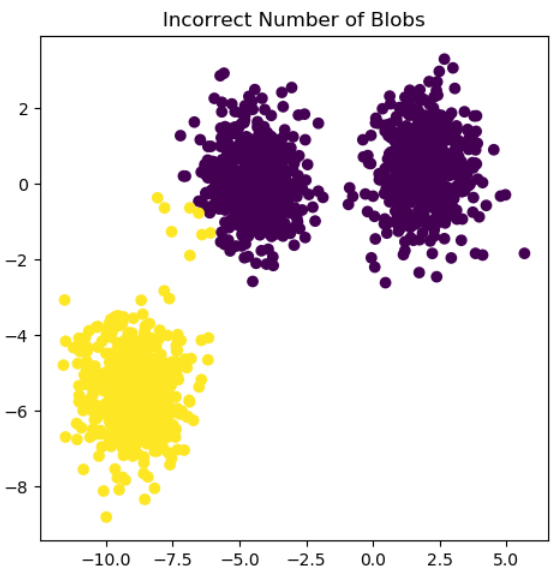
What to do in the absence of labels?

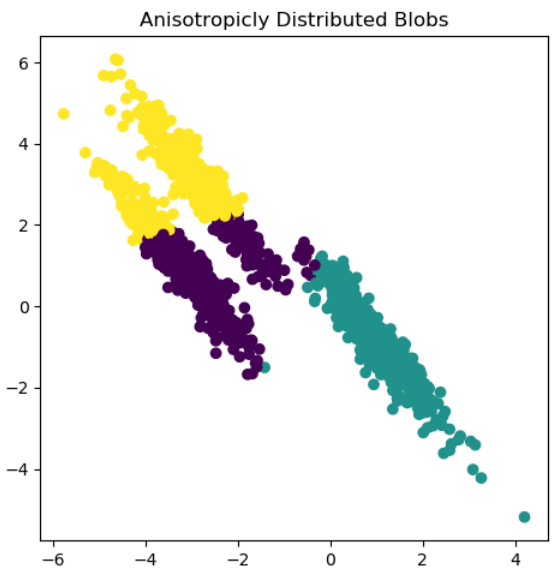
What is Unsupervised Learning?

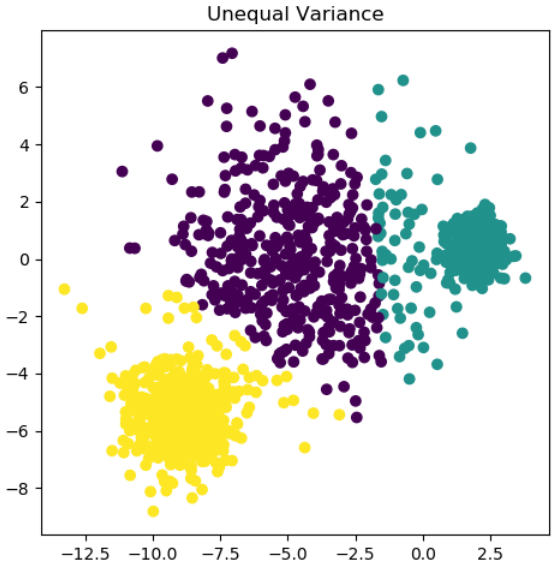
- Machine learning without data labels – instead, it revolves around finding interesting patterns in our data.
- Data clustering is often useful – *does our data naturally form into groupings? And what can we infer about the groupings?*

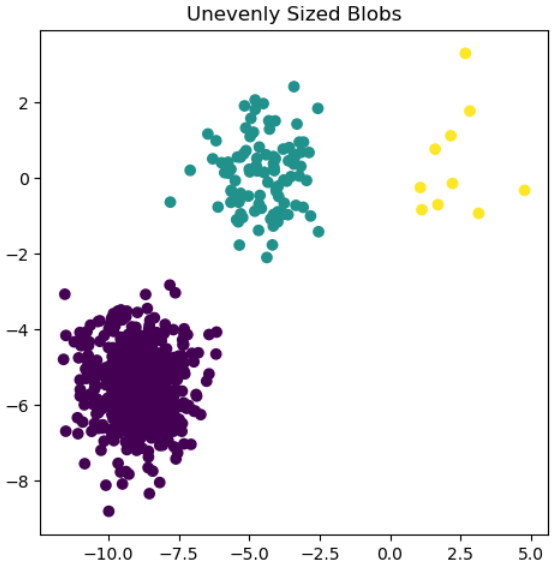
The K-Means Algorithm

- A classic algorithm for clustering datapoints!
  - Visualization of the algorithm and some of its modes of failure.









The K-Means Algorithm

- A classic algorithm for clustering datapoints!

```
from sklearn.cluster import Kmeans

kmeans = KMeans(n_clusters=5).fit(X)

kmeans.labels_ # => Array where x[i] is the cluster to which the ith data point
belongs

kmeans.predict(new_datapoints) # => Array of clusters to which the new data
points would belong

kmeans.cluster_centers_ # => Print out the mean value of each cluster
```

Review

- Supervised Learning
  - The classic problem setup
- Least Squares – Review
  - An old problem through a new lens
- Regression and Classification
  - A Tale of Two Problems
- Generalizing Least Squares – Gradient Descent
  - A more general paradigm
- Neural Networks
  - The cutting edge!
- Unsupervised Learning
  - Learning without labels!