Training Algorithm to Learn

CSIT375 AI for Cybersecurity

Dr Wei Zong

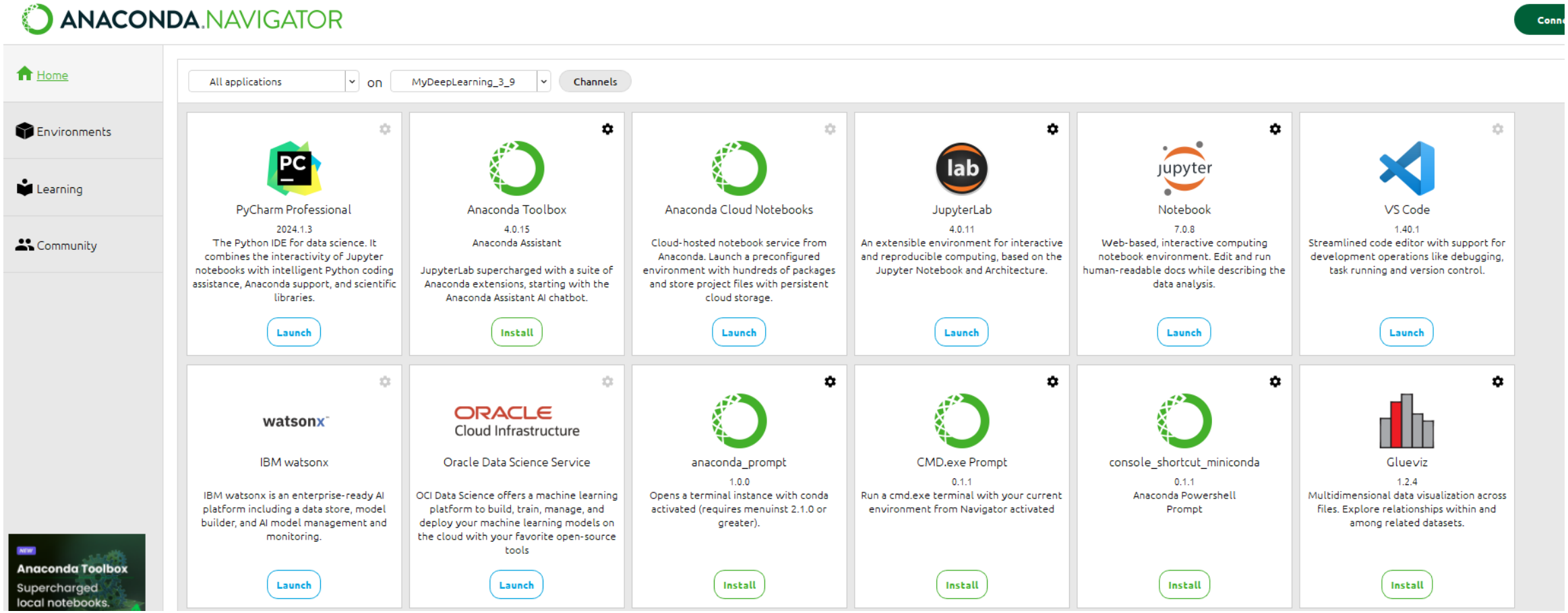
SCIT University of Wollongong

Disclaimer: The presentation materials come from various sources. For further information, check the references section

Outline

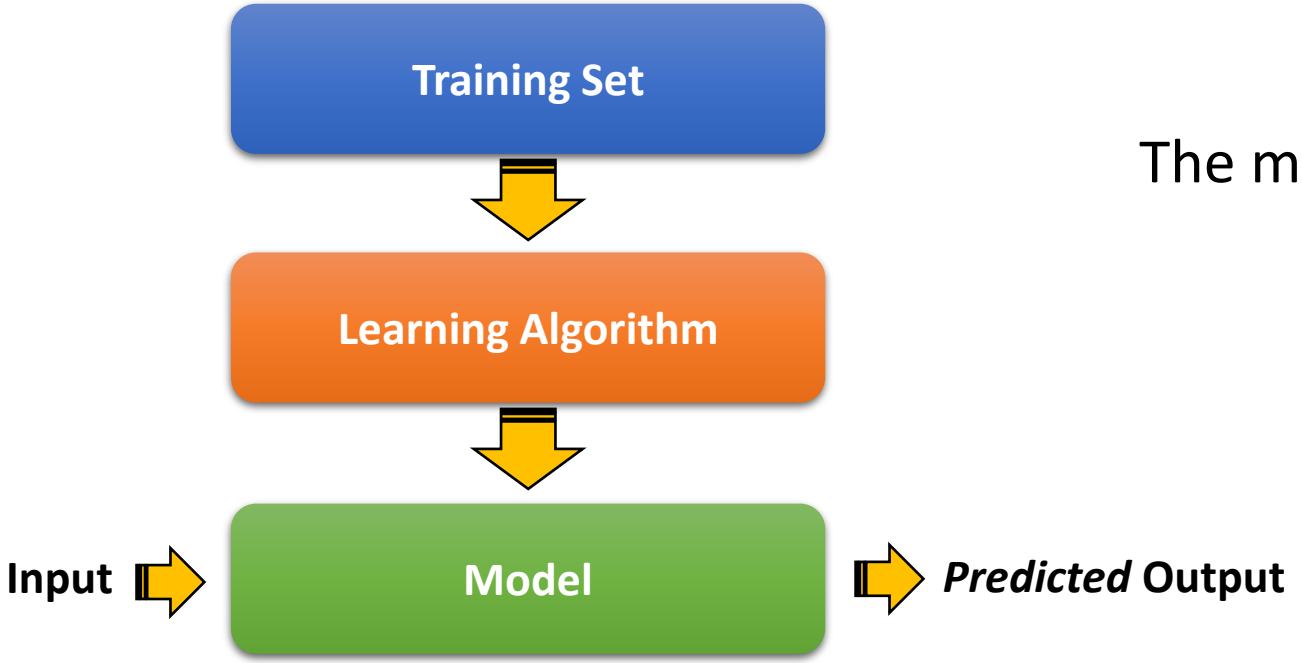
- Setup Jupyter notebook with Anaconda
- Recap: ML Problems and Approaches
- Machine Learning in Practice: a working example
- Training Algorithms to Learn – logistic regression
- Evaluating Models
- Practical Considerations

Setup Jupyter notebook with Anaconda



Recap: What is Machine Learning?

- Machine learning is the process of fitting a mathematical model to a given dataset (called the *training set*), then using this model to predict answers for any new unseen inputs



The model that we learnt is called

Linear Regression

Mean Squared Error (MSE)

- Let us compare their errors

		m = 1 and ***b*** = 0		***m*** = 0.05 and ***b*** = 2		***m*** = 0.052 and ***b*** = 29.21	
# computer (*x*)	Actual cost (*y*)	Predicted cost (y')	(Predicted – Actual)^2 (y' – y)^2	Predicted cost (y')	(Predicted – Actual)^2 (y' – y)^2	Predicted cost (y')	(Predicted – Actual)^2 (y' – y)^2
575	59	59.11	266256	30.75	798.0625	59.11	0.0121
650	55	63.01	354025	34.5	420.25	63.01	64.1601
832	67	72.474	585225	43.6	547.56	72.474	29.964676
850	84	73.41	586756	44.5	1560.25	73.41	112.1481
933	87	77.726	715716	48.65	1470.7225	77.726	86.007076
1001	81	81.262	846400	52.05	838.1025	81.262	0.068644
1111	88	86.982	1046529	57.55	927.2025	86.982	1.036324
1230	92	93.17	1295044	63.5	812.25	93.17	1.3689
1321	101	97.902	1488400	68.05	1085.7025	97.902	9.597604
1370	102	100.45	1607824	70.5	992.25	100.45	2.4025
1390	85	101.49	1703025	71.5	182.25	101.49	271.9201
1422	95	103.154	1760929	73.1	479.61	103.154	66.487716
1480	120	106.17	1849600	76	1936	106.17	191.2689
1487	114	106.534	1885129	76.35	1417.5225	106.534	55.741156
1490	100	106.69	1932100	76.5	552.25	106.69	44.7561
SQUARED ERRORS:		**∑ = 17922958**		**∑ = 14019.9**		**∑ = 936.9**	

Mean Squared Error (MSE)

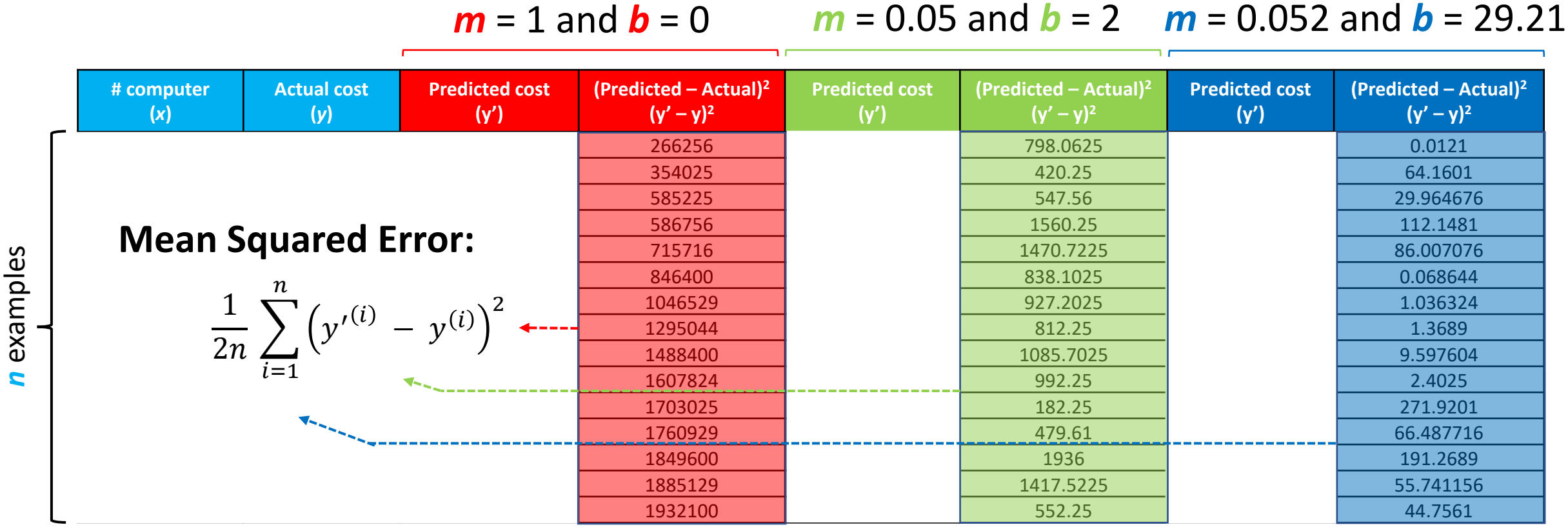
- Let us compare their errors

			m = 1 and ***b*** = 0		***m*** = 0.05 and ***b*** = 2		***m*** = 0.052 and ***b*** = 29.21	
	# computer (*x*)	Actual cost (*y*)	Predicted cost (y')	(Predicted – Actual)^2 (y' – y)^2	Predicted cost (y')	(Predicted – Actual)^2 (y' – y)^2	Predicted cost (y')	(Predicted – Actual)^2 (y' – y)^2
15 examples	575	59	59.11	266256	30.75	798.0625	59.11	0.0121
	650	55	63.01	354025	34.5	420.25	63.01	64.1601
	832	67	72.474	585225	43.6	547.56	72.474	29.964676
	850	84	73.41	586756	44.5	1560.25	73.41	112.1481
	933	87	77.726	715716	48.65	1470.7225	77.726	86.007076
	1001	81	81.262	846400	52.05	838.1025	81.262	0.068644
	1111	88	86.982	1046529	57.55	927.2025	86.982	1.036324
	1230	92	93.17	1295044	63.5	812.25	93.17	1.3689
	1321	101	97.902	1488400	68.05	1085.7025	97.902	9.597604
	1370	102	100.45	1607824	70.5	992.25	100.45	2.4025
	1390	85	101.49	1703025	71.5	182.25	101.49	271.9201
	1422	95	103.154	1760929	73.1	479.61	103.154	66.487716
	1480	120	106.17	1849600	76	1936	106.17	191.2689
	1487	114	106.534	1885129	76.35	1417.5225	106.534	55.741156
	1490	100	106.69	1932100	76.5	552.25	106.69	44.7561

MEAN SQUARED ERRORS: ∑ = 17922958/30 ∑ = 14019.9/30 ∑ = 936.9/30

Mean Squared Error (MSE)

- Let us compare their errors



# computer (x)	Actual cost (y)	Predicted cost (y')	(Predicted – Actual)² (y' – y)²	Predicted cost (y')	(Predicted – Actual)² (y' – y)²	Predicted cost (y')	(Predicted – Actual)² (y' – y)²
			266256		798.0625		0.0121
			354025		420.25		64.1601
			585225		547.56		29.964676
			586756		1560.25		112.1481
			715716		1470.7225		86.007076
			846400		838.1025		0.068644
			1046529		927.2025		1.036324
			1295044		812.25		1.3689
			1488400		1085.7025		9.597604
			1607824		992.25		2.4025
			1703025		182.25		271.9201
			1760929		479.61		66.487716
			1849600		1936		191.2689
			1885129		1417.5225		55.741156
			1932100		552.25		44.7561

Minimizing MSE to Learn a Model

- Let us compare their errors

m = 1 and ***b*** = 0 | ***m*** = 0.05 and ***b*** = 2 | ***m*** = 0.052 and ***b*** = 29.21

# computer (*x*)	Actual cost (*y*)	Predicted cost (y')	(Predicted – Actual)² (y' – y)²	Predicted cost (y')	(Predicted – Actual)² (y' – y)²	Predicted cost (y')	(Predicted – Actual)² (y' – y)²
			266256		798.0625		0.0121
			354025		420.25		64.1601
			585225		547.56		29.964676
			586756		1560.25		112.1481
			715716		1470.7225		86.007076
			846400		838.1025		0.068644
			1046529		927.2025		1.036324
			1295044		812.25		1.3689
			1488400		1085.7025		9.597604
			1607824		992.25		2.4025
			1703025		182.25		271.9201
			1760929		479.61		66.487716
			1849600		1936		191.2689
			1885129		1417.5225		55.741156
			1932100		552.25		44.7561

n examples

Mean Squared Error:

$$\text{minimize} \frac{1}{2n} \sum_{i=1}^{n} \left(y'^{(i)} - y^{(i)} \right)^2$$

The objective is to minimize the mean squared error

Learning a Line via Minimizing MSE

- How to learn a line of equation y’ = mx + b given a *labelled* dataset?
  - By minimizing ***mean squared error***. That is:

$$\text{minimize} \frac{1}{2n} \sum_{i=1}^{n} \left(y'^{(i)} - y^{(i)} \right)^2$$

$$\equiv$$

$$\underset{m,b}{\text{minimize}} \frac{1}{2n} \sum_{i=1}^{n} \left((mx + b)^{(i)} - y^{(i)} \right)^2$$

Learning a Linear Regression Model via Minimizing a Cost Function

- Or, how to learn a *linear regression model* $\boldsymbol{h_\theta(x) = \theta_0 + \theta_1 x}$ given a labelled dataset ($\boldsymbol{\theta_0}$ = b, $\boldsymbol{\theta_1}$= m, and $\boldsymbol{h_\theta(x)}$= y' when using y' = mx + b)?
  - By minimizing ***mean squared error***. That is:

This problem is referred to as an ***optimization problem*** with the objective of:

$$\underset{\theta_0,\theta_1}{\text{minimizing}}\ \boldsymbol{J(\theta_0,\theta_1)}$$

$$\text{minimize}\ \frac{1}{2n}\sum_{i=1}^{n}\left(y'^{(i)} - y^{(i)}\right)^2$$

$$\equiv$$

$$\underset{\theta_0,\theta_1}{\text{minimize}}\ \frac{1}{2n}\sum_{i=1}^{n}\left((\theta_0+\theta_1 x)^{(i)} - y^{(i)}\right)^2$$

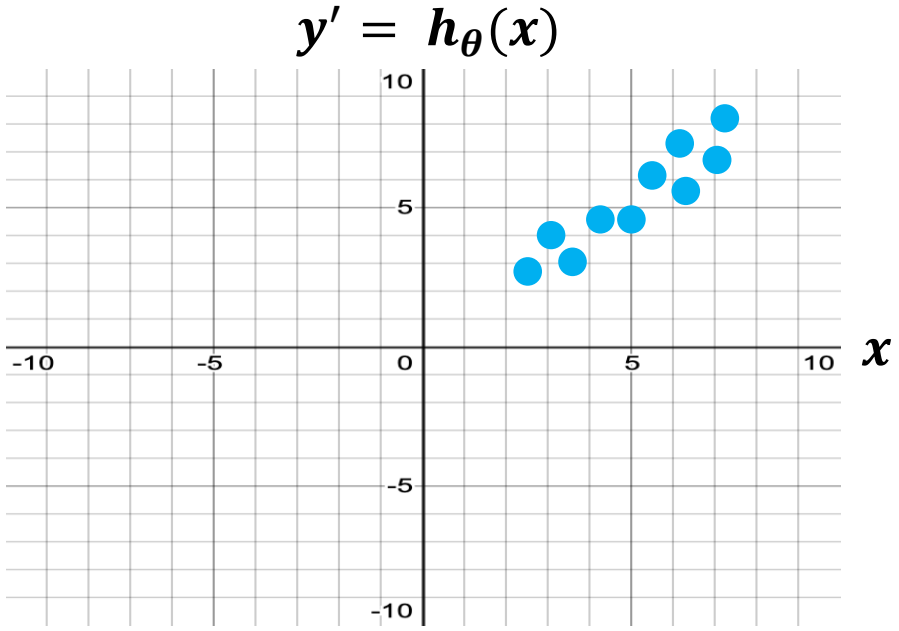
Known as ***cost function*** $\boldsymbol{J(\theta_0,\theta_1)}$

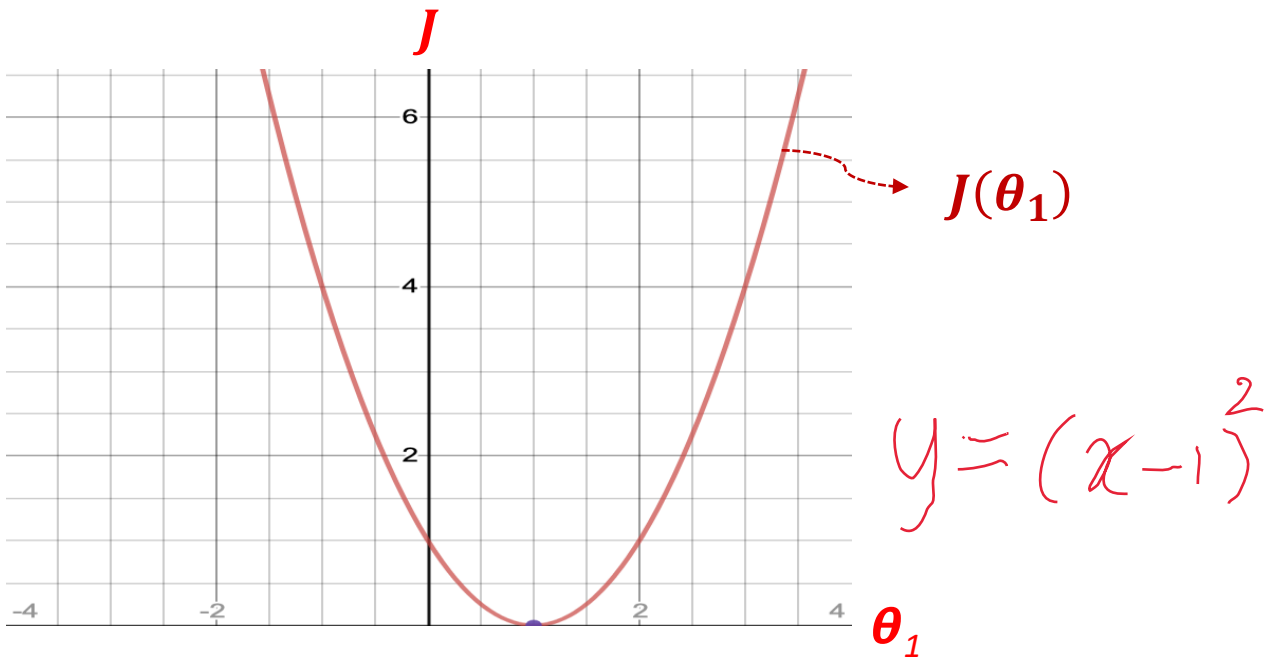
$$\equiv$$

$$\underset{\theta_0,\theta_1}{\text{minimize}}\ \boldsymbol{J(\theta_0,\theta_1)}$$

Learning a Linear Regression Model via Minimizing a Cost Function

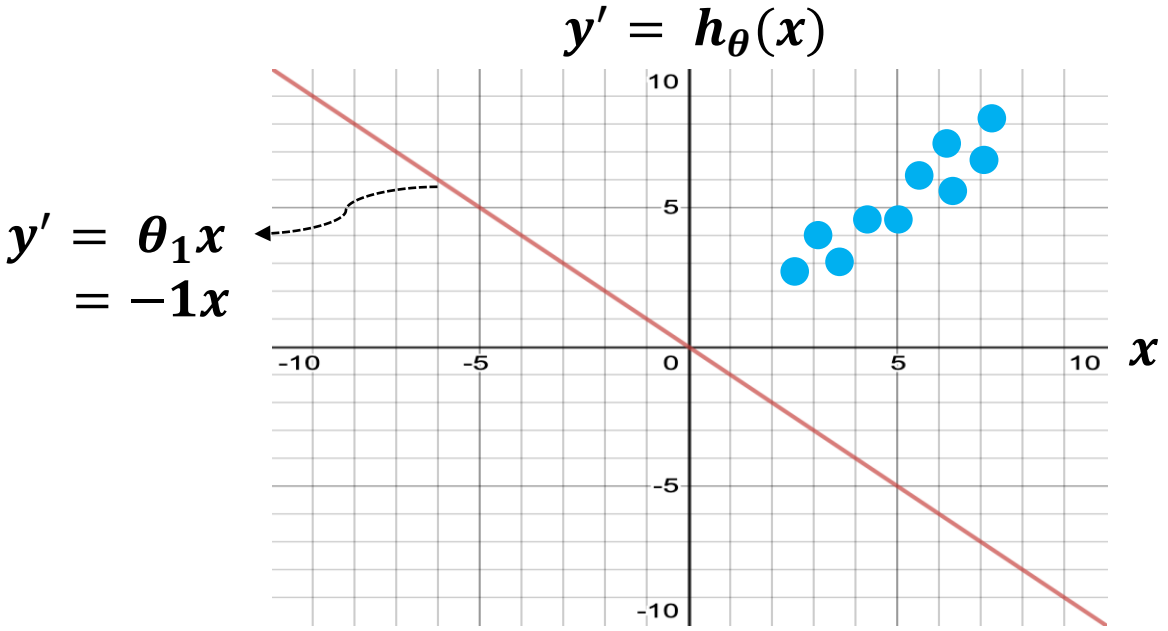
- But, how minimizing a cost function (e.g., a mean squared error) can lead to fitting a given training dataset?
  - Let us assume our optimization objective is to $\underset{\theta_0,\theta_1}{\text{minimize}}\, \boldsymbol{J(\theta_1)}$, thus, $\boldsymbol{\theta_0} = 0$

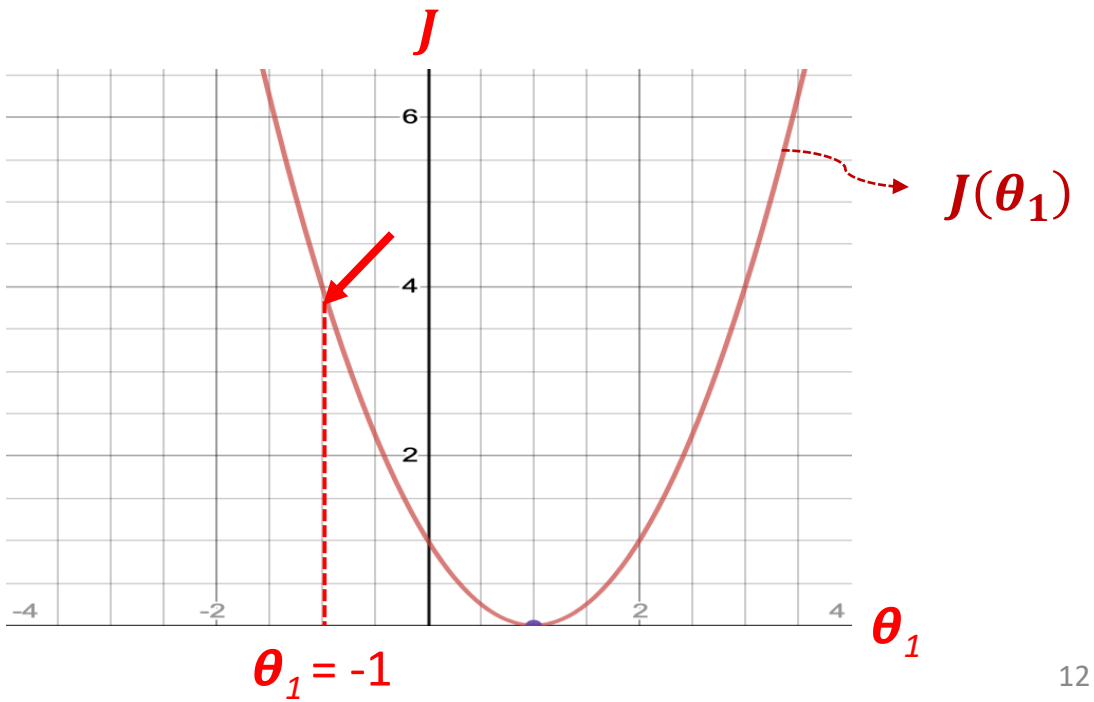




Learning a Linear Regression Model via Minimizing a Cost Function

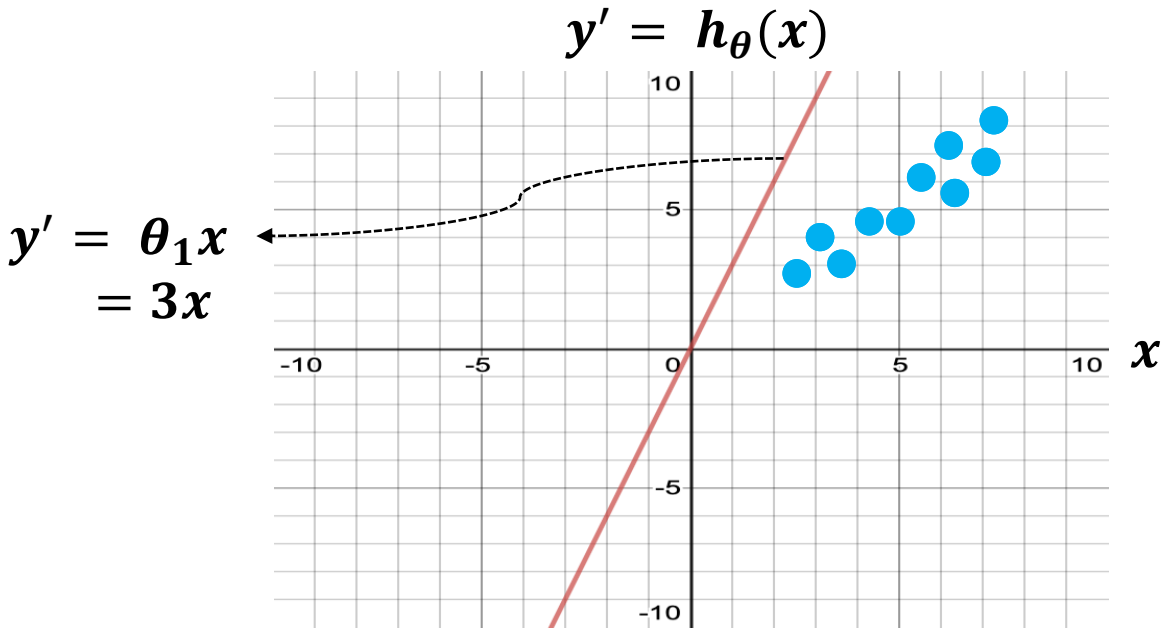
- But, how minimizing a cost function (e.g., a mean squared error) can lead to fitting a given training dataset?
  - Let us assume our optimization objective is to $\underset{\theta_0,\theta_1}{\text{minimize}}\, \boldsymbol{J(\theta_1)}$, thus, $\boldsymbol{\theta_0} = 0$

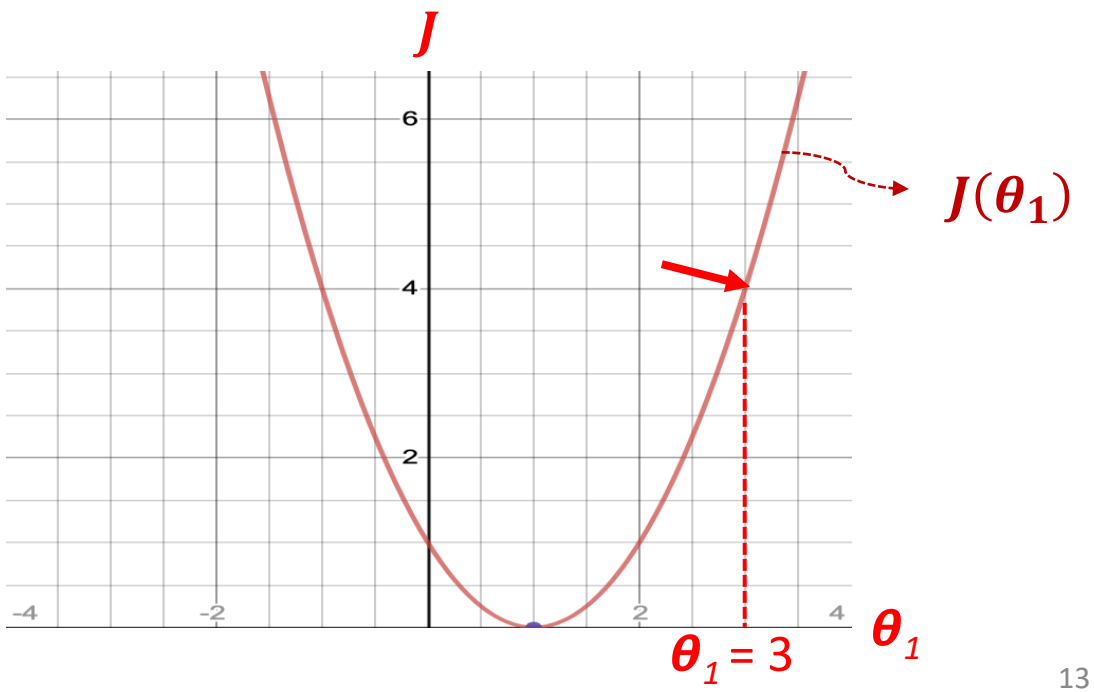




Learning a Linear Regression Model via Minimizing a Cost Function

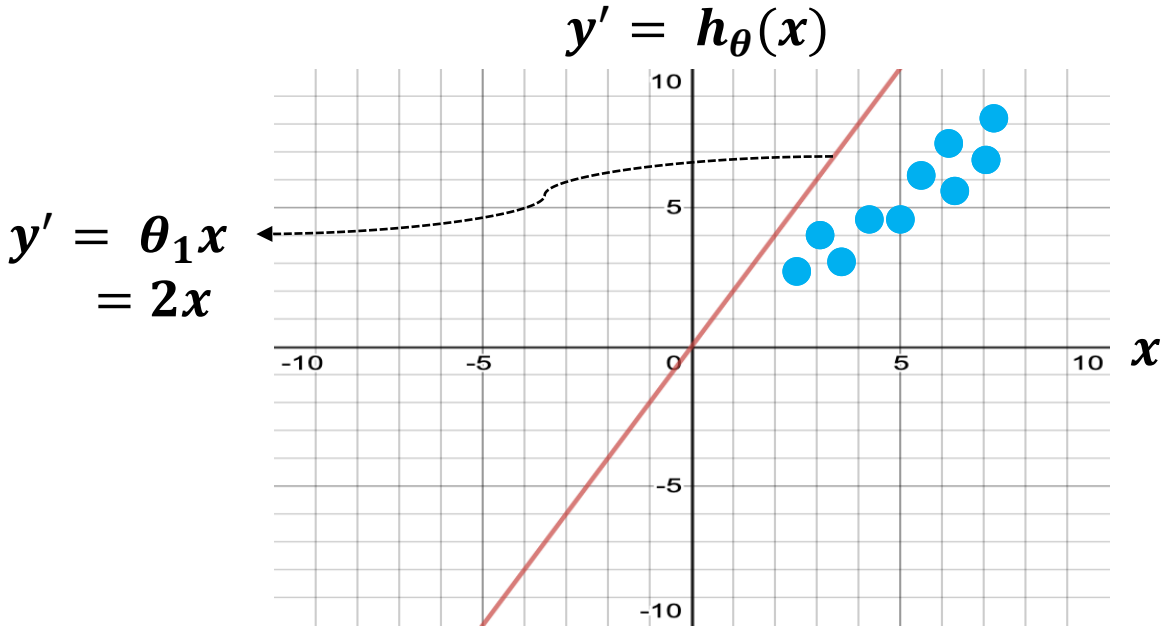
- But, how minimizing a cost function (e.g., a mean squared error) can lead to fitting a given training dataset?
  - Let us assume our optimization objective is to $\underset{\theta_0,\theta_1}{\text{minimize}}\, \boldsymbol{J(\theta_1)}$, thus, $\boldsymbol{\theta_0} = 0$

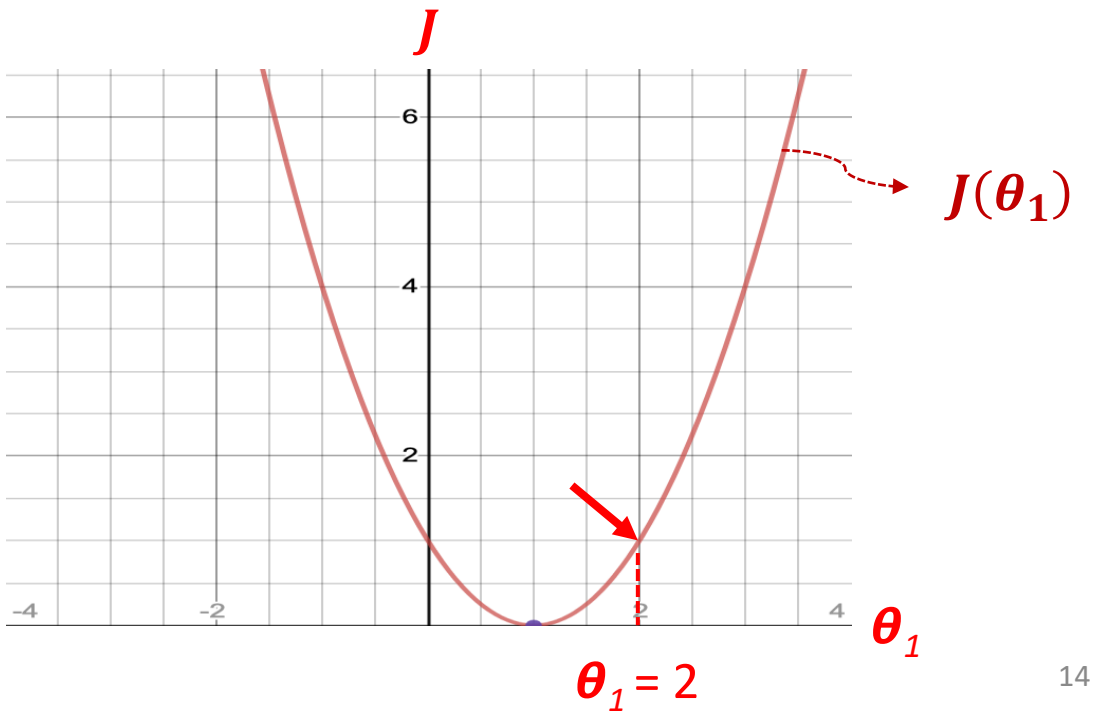




Learning a Linear Regression Model via Minimizing a Cost Function

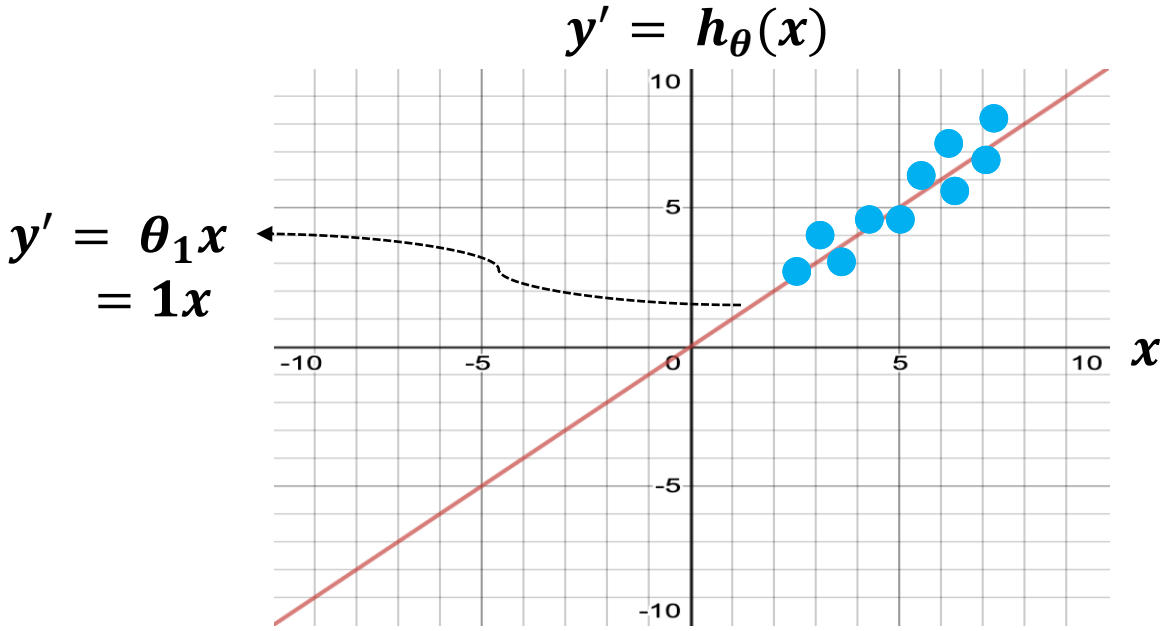
- But, how minimizing a cost function (e.g., a mean squared error) can lead to fitting a given training dataset?
  - Let us assume our optimization objective is to $\underset{\theta_0,\theta_1}{\text{minimize}}\, \boldsymbol{J(\theta_1)}$, thus, $\boldsymbol{\theta_0} = 0$

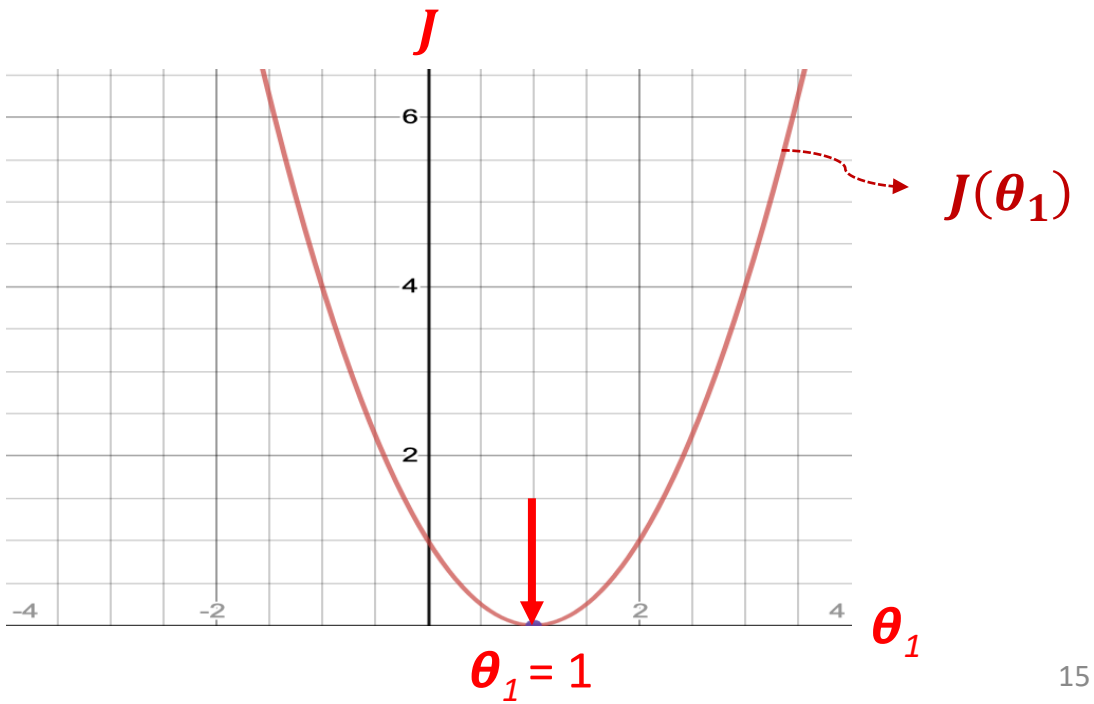




Learning a Linear Regression Model via Minimizing a Cost Function

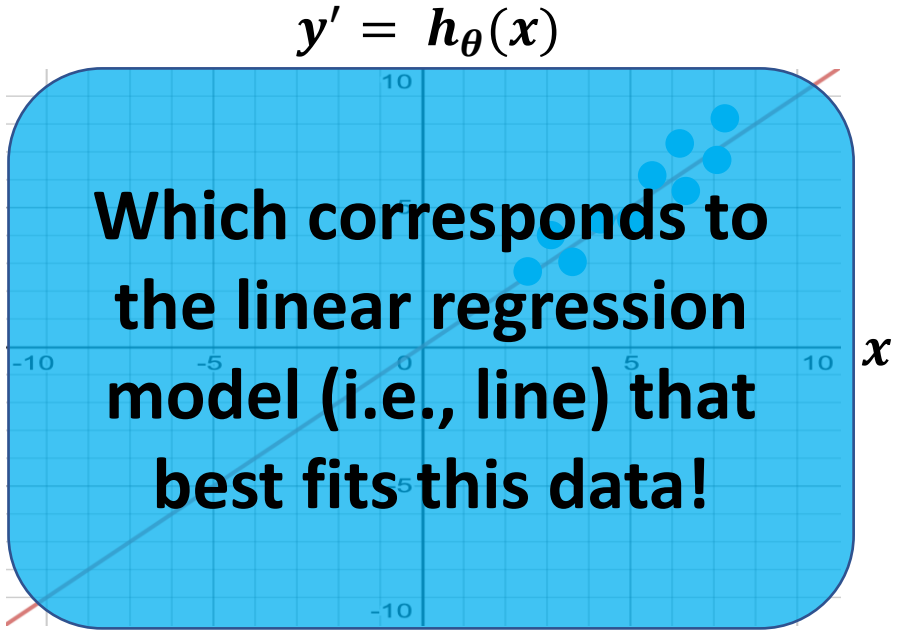
- But, how minimizing a cost function (e.g., a mean squared error) can lead to fitting a given training dataset?
  - Let us assume our optimization objective is to $\underset{\theta_0,\theta_1}{\text{minimize}}\, \boldsymbol{J(\theta_1)}$, thus, $\boldsymbol{\theta_0} = 0$

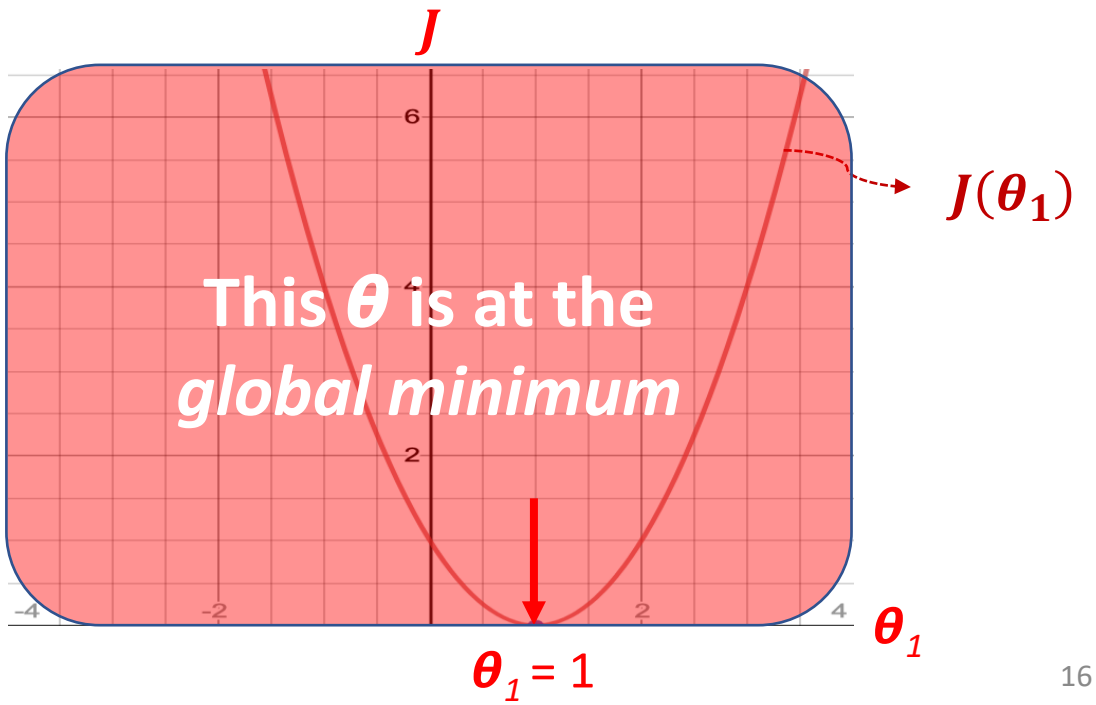




Learning a Linear Regression Model via Minimizing a Cost Function

- But, how minimizing a cost function (e.g., a mean squared error) can lead to fitting a given training dataset?
  - Let us assume our optimization objective is to $\underset{\theta_0,\theta_1}{\text{minimize}}\, J(\theta_1)$, thus, $\theta_0 = 0$





Gradient Descent For Linear Regression

- How to $\underset{\theta_0, \theta_1}{\text{minimize}}\, \boldsymbol{J(\theta_0, \theta_1)}$ and locate the minimum, which corresponds to $h_\theta(x)$ that best fits the given data?

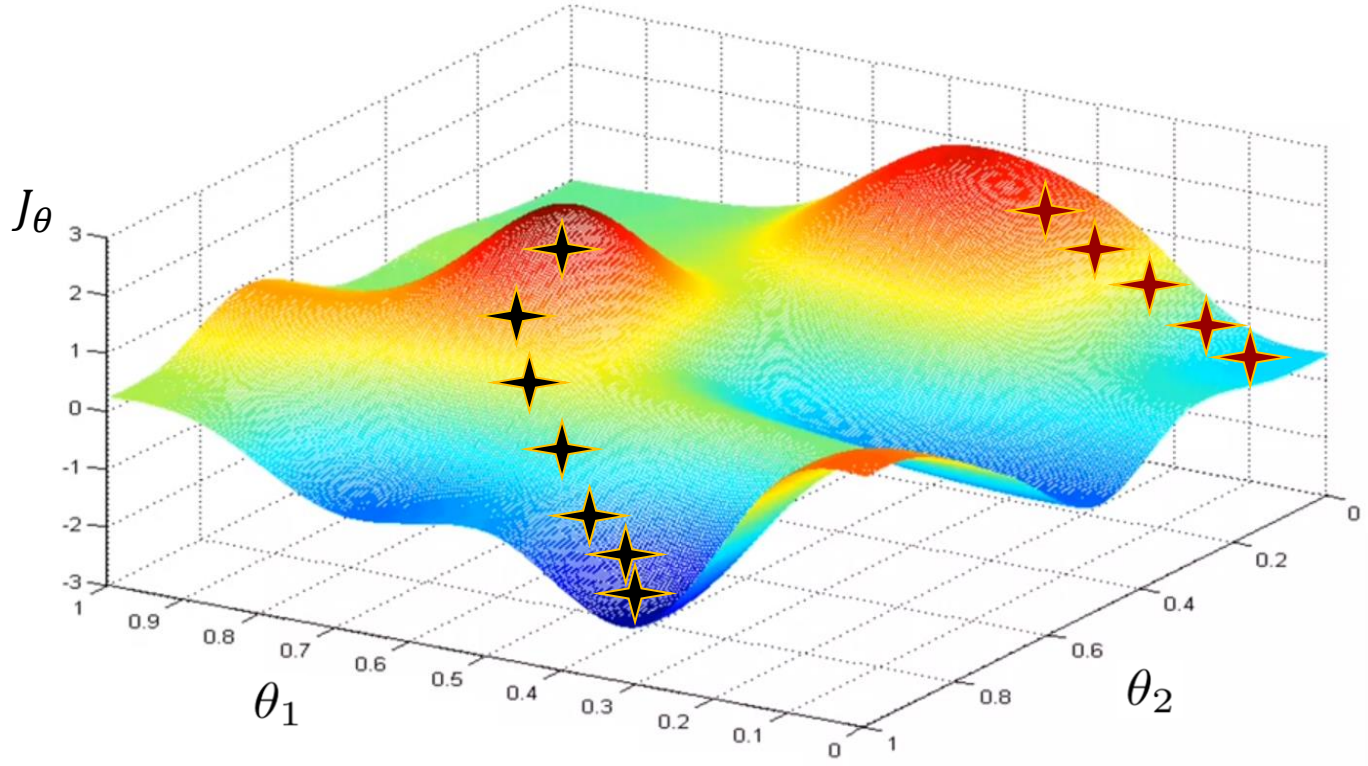
- More generally, how to $\underset{\theta_0, \ldots, \theta_{n-1}}{\text{minimize}}\, \boldsymbol{J(\theta_0, \ldots, \theta_{n-1})}$ and locate the minimum, which corresponds to $h_\theta(x)$ that best fits the given data?
  - By using the *gradient descent algorithm*
  - The gradient descent algorithm can be utilized also to learn many other mathematical models and not only a linear regression model
    - Image classification, speech-to-text, etc.

Gradient Descent For Linear Regression

- **Steps**:
  - Have some cost function $\boldsymbol{J(\theta_0, \ldots, \theta_{n-1})}$
  - Start with some guesses for $\theta_0, \ldots, \theta_{n-1}$
    - a common choice is to set them all initially to zero
    - or to random values
  - Keep changing $\theta_0, \ldots, \theta_{n-1}$ to reduce $\boldsymbol{J(\theta_0, \ldots, \theta_{n-1})}$ until we hopefully end up at a minimum location
    - When you are at a certain position on the surface of $\boldsymbol{J}$, look around, then take a little step in the direction of *the steepest descent*, then repeat

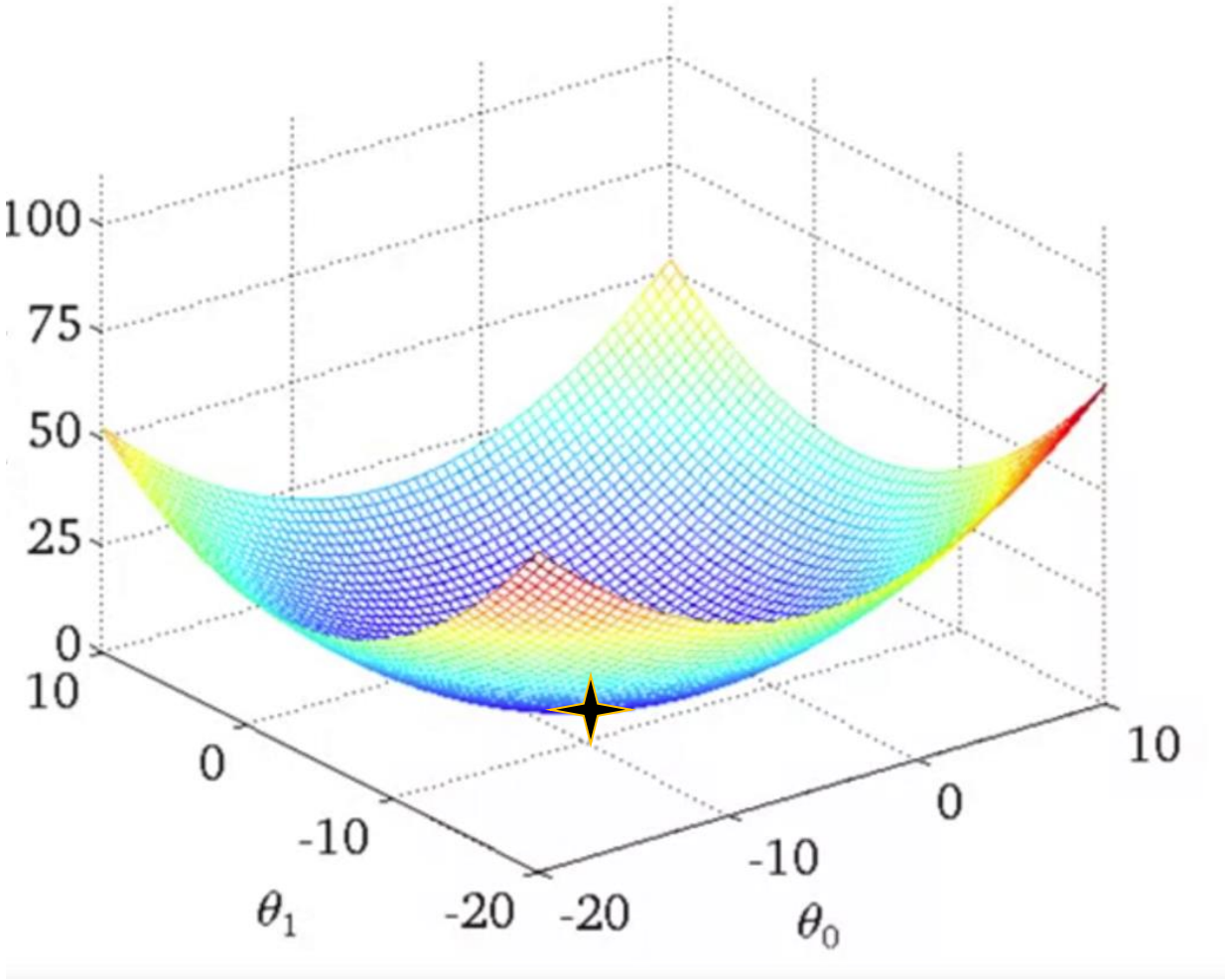
Gradient Descent

Coming down a mountain.



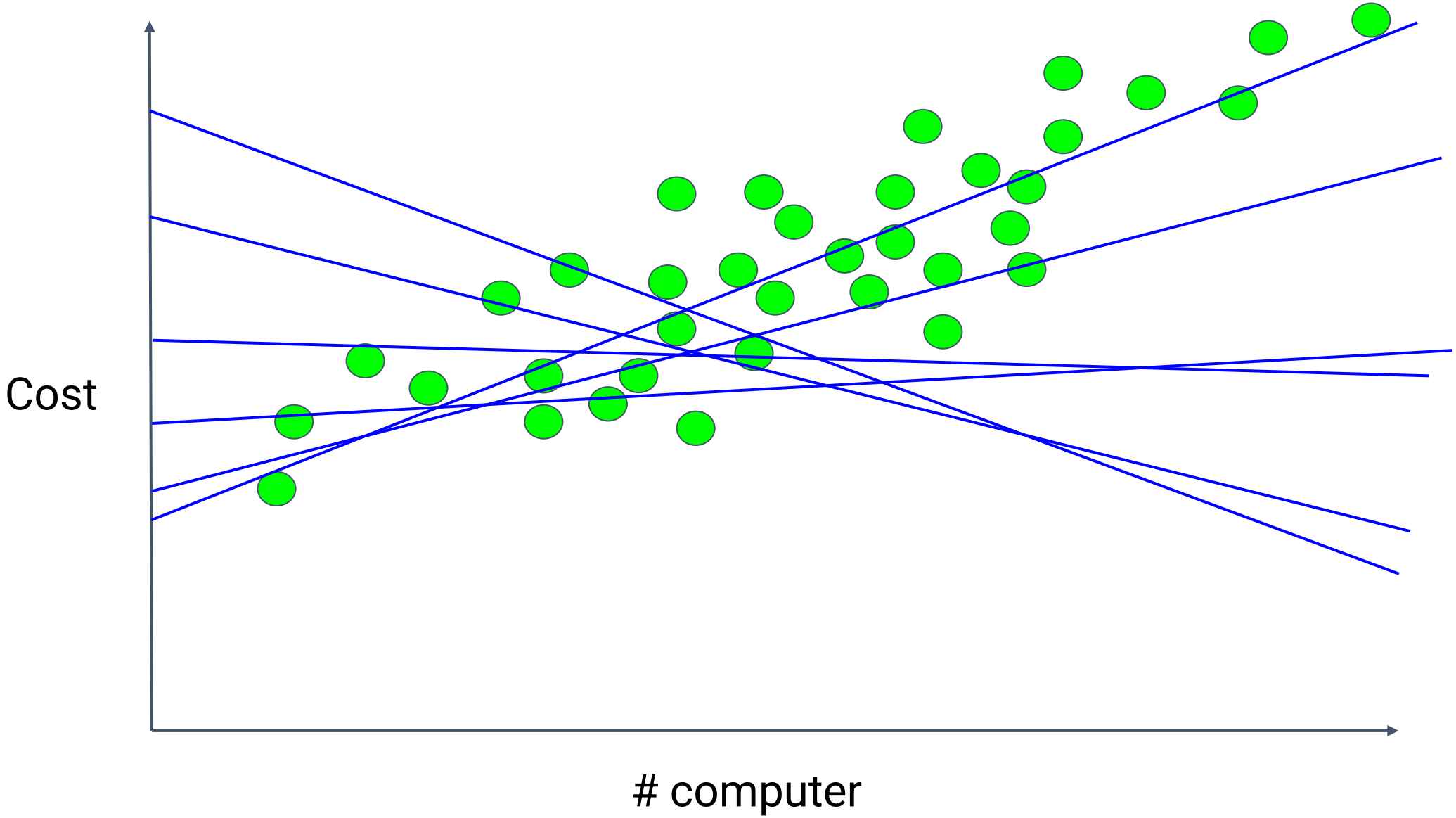
Gradient Descent

MSE is a convex function: a unique global minimum.



https://i.stack.imgur.com/cYAx7.png

Gradient Descent



Gradient Descent For Linear Regression

- **Steps**:
  - Have some cost function $\boldsymbol{J(\theta_0, \ldots, \theta_{n-1})}$
  - Start with some guesses for $\theta_0, \ldots, \theta_{n-1}$
    - a common choice is to set them all initially to zero or random values
  - Repeat until convergence{

$$\theta_j = \theta_j - \alpha \frac{\partial \boldsymbol{J(\theta_0, \ldots, \theta_{n-1})}}{\partial\, \theta_j}$$

Partial derivative

Learning Rate

}

Gradient Descent For Linear Regression

- **Steps**
  - Have some cost function $\boldsymbol{J(\theta_0, \theta_1)}$
  - Start with some guesses for θ_0, θ_1
    - a common choice is to set them both initially to zero
  - Repeat until convergence{

$$temp_0 = \theta_0 - \alpha \frac{\partial\, \boldsymbol{J(\theta_0, \theta_1)}}{\partial\, \theta_0} \rightarrow \frac{1}{n}\sum_{i=1}^{n}\left(h_\theta(x)^{(i)} - y^{(i)}\right)$$

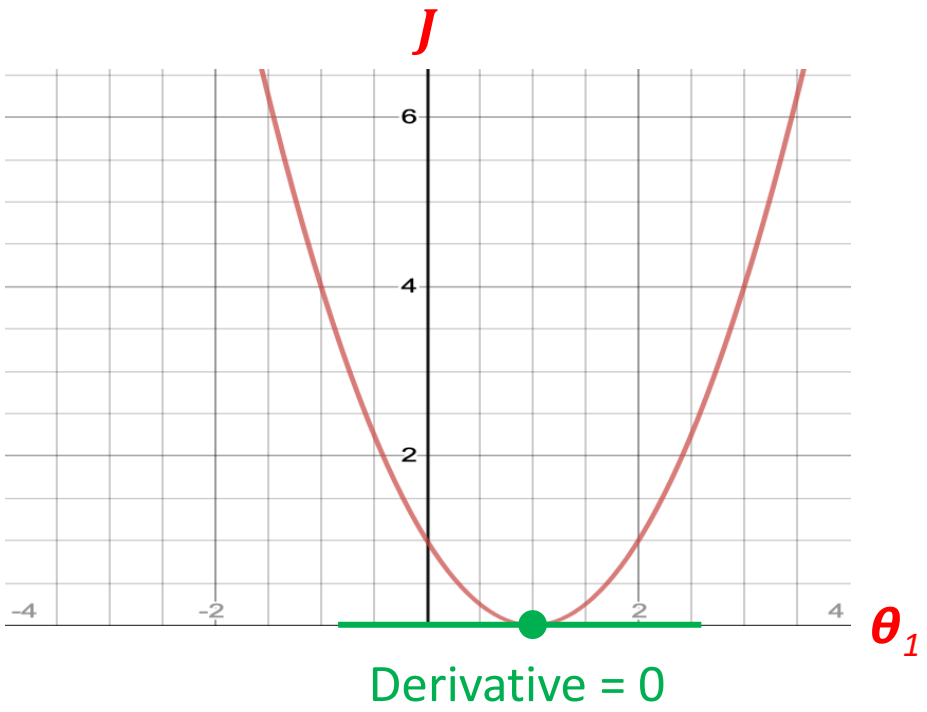
$$temp_1 = \theta_1 - \alpha \frac{\partial\, \boldsymbol{J(\theta_0, \theta_1)}}{\partial\, \theta_1} \rightarrow \frac{1}{n}\sum_{i=1}^{n}\left(h_\theta(x)^{(i)} - y^{(i)}\right) .x^{(i)}$$

$$\theta_0 = temp_0$$
$$\theta_1 = temp_1$$

  }

The Impact of Partial Derviative

- For simplicity, let us assume our optimization objective is to $\underset{\theta_0,\theta_1}{\text{minimize}}\, \boldsymbol{J(\theta_1)}$, thus, $\boldsymbol{\theta}_0 = 0$



$$\theta_1 = \theta_1 - \alpha \frac{\boldsymbol{d}\, \boldsymbol{J(\theta_1)}}{\boldsymbol{d}\, \theta_j}$$

$$= \theta_1 - \alpha\, (Zero)$$

e.g.: when θ_1 remains the same, the descent algorithm converges to a stationary point (i.e., a point where the derivative is zero)

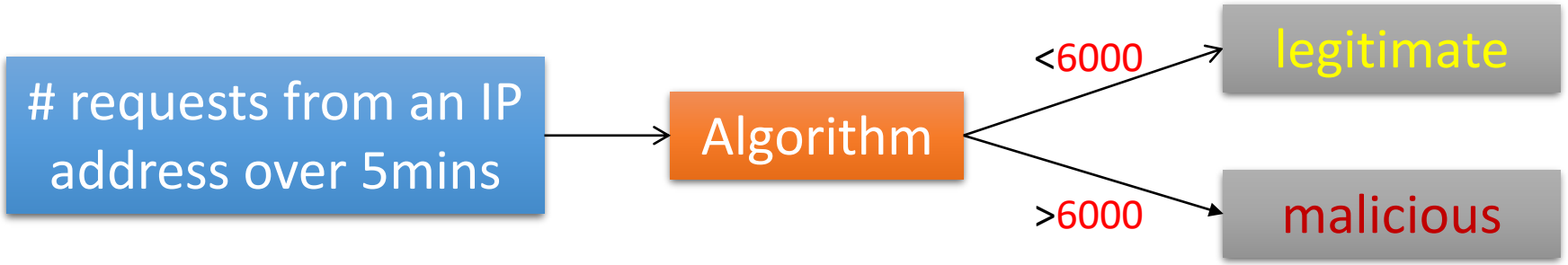
Rule-based Classification

- Suppose that you are in charge of computer security for your company
  - install firewalls, hold security awareness training, ensure secure coding, etc.
- But all your CEO cares about is you don't have a breach
- Systems that can detect and block malicious traffic to any attack surface
  - For every file sent through the network, does it contain malware?
  - For every login attempt, has someone's password been compromised?
  - For every email received, is it a phishing attempt?
  - For every request to your servers, is it a denial-of-service (DoS) attack?
  - For every outbound request from your network, is it a bot calling its command- and-control server?

Classification tasks: Classify all events in your network as malicious or legitimate

Rule-based Classification

- How to classify all traffic? **(“Human learning” not machine learning)**
  - historical logs of binary files, login attempts
  - emails received, and inbound and outbound requests, etc.
- Look for patterns in the past data indicating malicious attacks
  - single IP address making 20 requests/sec over 5 mins => possible DoS attack
- Encode these patterns as an algorithm
  - a model that takes as input data, outputs or ‘legitimate’ or ’malicious’
  - In this example, the algorithm would be very simple:

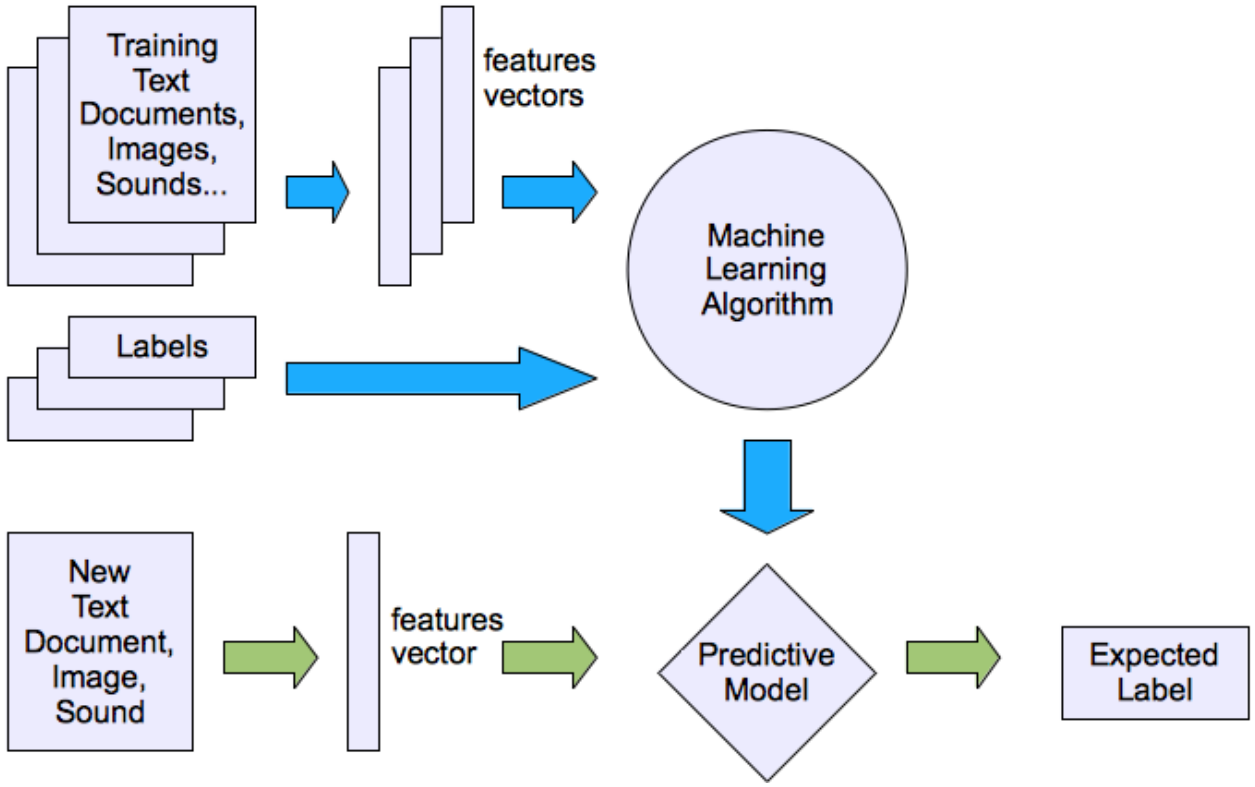


Machine Learning: problems and approaches

- **Machine learning:** use historical data to create a prediction algorithm for future data
  - **Classification**: determine which class a new data point falls into
    - binary. e.g. legitimate or malicious
    - multiclass. e.g. ransomware / a keylogger / remote access trojan
  - **Regression**: predict the value of a real-number variable
    - e.g. predict # phishing emails an employee receives in a given month
  - **Clustering**: given data points, which ones are similar to one another?
    - e.g. internet traffic to your site, botnets / mobile providers / legitimate users

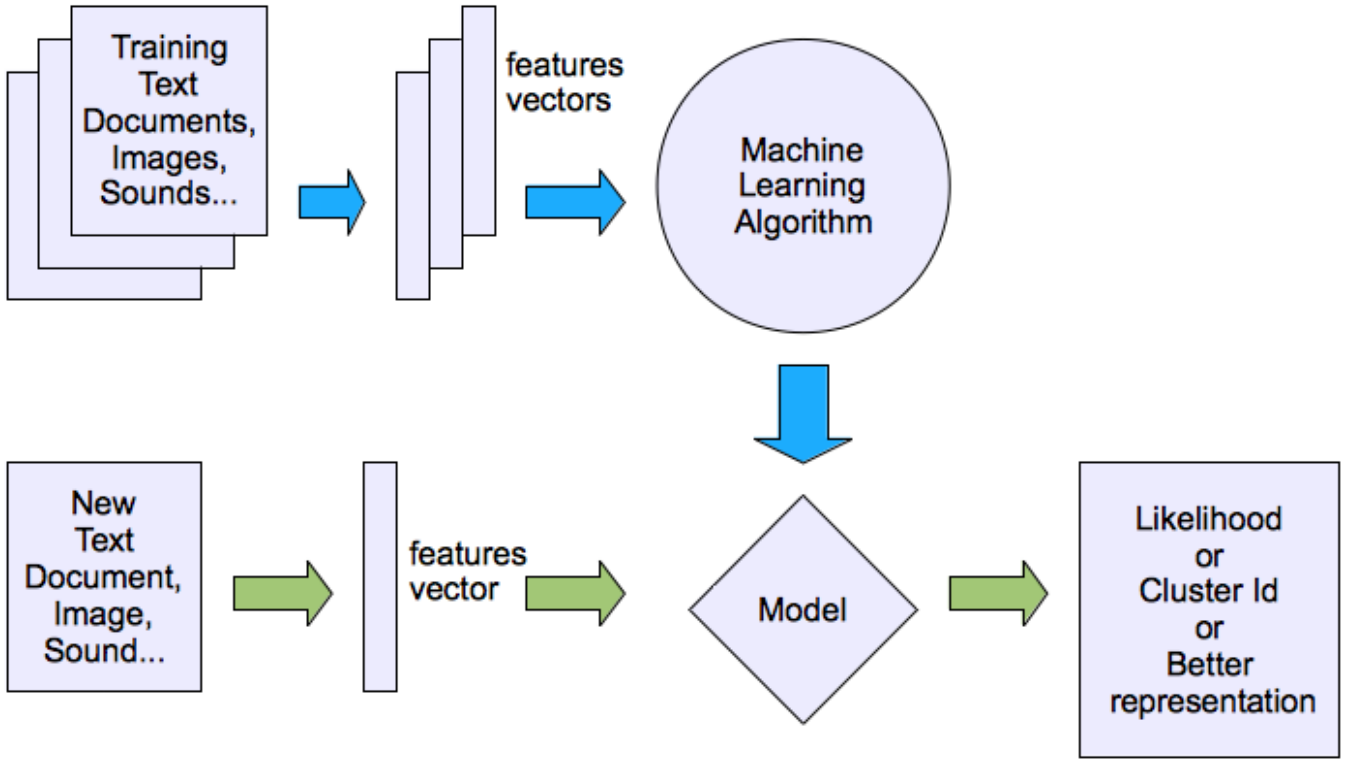
Machine learning structure

- **Supervised learning**
  - given a set of feature/label pairs, find a rule that predicts the label associated with a previously unseen input
  - e.g., given emails labeled as spam or ham, train a spam classifier, predict whether a new incoming message is spam



Machine learning structure

- **Unsupervised learning**
  - given a set of feature vectors (no labels), group them into clusters
  - e.g. unknown botnets attacking your network, disambiguate from one another



ML in Practice: Example

- **Goal**: train a model to learn how to identify a fraudulent transaction from the dataset
- Dataset: transaction data for online purchases
  - 39,221 transactions
    - 5 properties (***features***)
    - and a binary ***label*** as either: 1(fraudulent) or 0(normal)
  - CSV file, first row: the names for each positional value in each subsequent line

```
accountAgeDays,numItems,localTime,paymentMethod,paymentMethodAgeDays,label
...
196, 1, 4.962055, creditcard, 5.10625, 0
```

ML in Practice: Example

Create a prototype system by using machine learning

- The fraud detection system
  - take in ***features*** of a transaction
  - return a ***probability*** indicating how likely to be fraudulent

1. Prepare the data

- read the dataset in the CSV file: `df = pd.read_csv('ch1/payment_fraud.csv')`
- three random rows from df `df.sample(3)`

	accountAgeDays	numItems	localTime	paymentMethod	paymentMethodAgeDays	label
31442	2000	1	4.748314	storecredit	0.000000	0
27232	1	1	4.886641	storecredit	0.000000	1
8687	878	1	4.921349	paypal	0.000000	0

numerical index

categorical variable

ML in Practice: Example

- convert variables from ***categorical*** to ***numeric***

	accountAgeDays	...	paymentMethod_creditcard	paymentMethod_paypal	paymentMethod_storecredit
23393	57	...	1	0	0
3355	1,366	...	0	1	0
34248	19	...	1	0	0

- **one-hot encoding**
  - each feature is a binary feature (either 0 or 1)
  - each row has exactly one of these features set to 1

- Why do we use one-hot instead of simply assigning 1, 2, and 3?

ML in Practice: Example

- convert variables from ***categorical*** to ***numeric***

	accountAgeDays	...	paymentMethod_creditcard	paymentMethod_paypal	paymentMethod_storecredit
23393	57	...	1	0	0
3355	1,366	...	0	1	0
34248	19	...	1	0	0

- Why do we use one-hot instead of simply assigning 1, 2, and 3?
  - **Numerical values implicitly assume mathematical relationships: larger, smaller or equal.**
    - **They are meaningless.**
  - **Distance to each other is the same for every pair of categories.**

ML in Practice: Example

- divide the dataset into **training** and **test** sets randomly

```
X_train, X_test, y_train, y_test = train_test_split(
    df.drop('label', axis=1), df['label'],
    test_size=0.33, random_state=17)
```

  - Split into X_train and X_test to the ratio of 0.67:0.33
  - The labels will then be split in the same ratio into y_train and y_test

2. Apply a supervised learning algorithm to this data:

```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression()
clf.fit(X_train, y_train)
```

- This classifier takes the training data and uses logistic regression to distill some generalizations about fraudulent/nonfraudulent transactions into a ***model***

ML in Practice: Example

3. Make predictions using this model

```
y_pred = clf.predict(X_test)
```

- at training time the classifier did **not** have any access to y_test at all
- y_pred are a result of the generalizations learned from the training set

4. Evaluate these predictions

```
print(confusion_matrix(y_test, y_pred))
```

0	Predicted NOT FRAUD	Predicted FRAUD
Actual NOT FRAUD	12,753	0
Actual FRAUD	1	189

- 1 misclassification in the test set
  - 1 fraudulent transaction that was not detected
- 189 transactions are correctly flagged as fraud
- 12,753 transactions are correctly identified as not fraud

ML in Practice: Example

- **Model usage**
  - Given any incoming transaction, we can apply this model and get a probability score for how likely this transaction is to be fraudulent

```
clf.predict_proba(df_real)

  # Array that represents the probability of the transaction
  # having a label of 0 (in position 0) or 1 (in position 1)
> [[  9.99999994e-01   5.87025707e-09]]
```

- The part where we learn the prediction algorithm has been abstracted into the scikit-learn API call `LogisticRegression.fit()`
  - now open the box and find out what goes on

Logistic regression

- **Logistic regression** is a supervised learning algorithm
  - can be used to classify input data
  - e.g., classify transactions into fraud or normal

- Major questions about logistic regression:
  - What is the ***model*****?**
  - What is the ***cost function***?
  - How can we learn the ***parameters*** of the model?

Logistic regression

- The Logistic Regression Model

$$\boldsymbol{h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}}$$

  - $\boldsymbol{x}$ is the ***feature vector***: $\boldsymbol{x} = [\boldsymbol{x_0}, \boldsymbol{x_1}, \ldots, \boldsymbol{x_m}]^T$
  - $\boldsymbol{\theta}$ is the ***parameter vector***: $\boldsymbol{\theta} = [\boldsymbol{\theta_0}, \boldsymbol{\theta_1}, \ldots, \boldsymbol{\theta_m}]^T$
  - $\boldsymbol{g(z)}$ is the sigmoid function: $g(z) = \frac{1}{1+e^{-z}}$

- ***After*** learning the model, we can then apply thresholding to predict the binary output as follows:

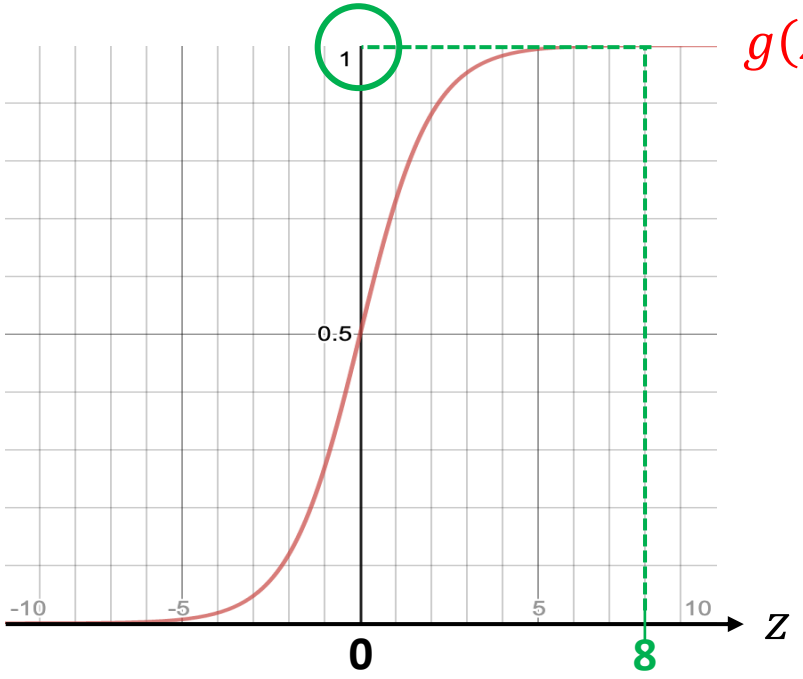
$$if\ h_\theta(x) < 0.5 \text{ predict } 0$$
$$if\ h_\theta(x) \geq 0.5 \text{ predict } 1$$

Logistic regression

Logistic regression takes as input numerical feature vectors and attempts to predict the probabilities by using the **sigmoid function** $g(z) = \frac{1}{1+e^{-z}}$

$$g(z) = \frac{1}{1 + e^{-z}}$$

$z \in \mathbb{R}, but$
$g(z) \in [0,1]$



Assume a labeled example ($\boldsymbol{x}$, y):

If y = **1**, we want $g(z) \approx 1$ (i.e., we want a correct prediction)

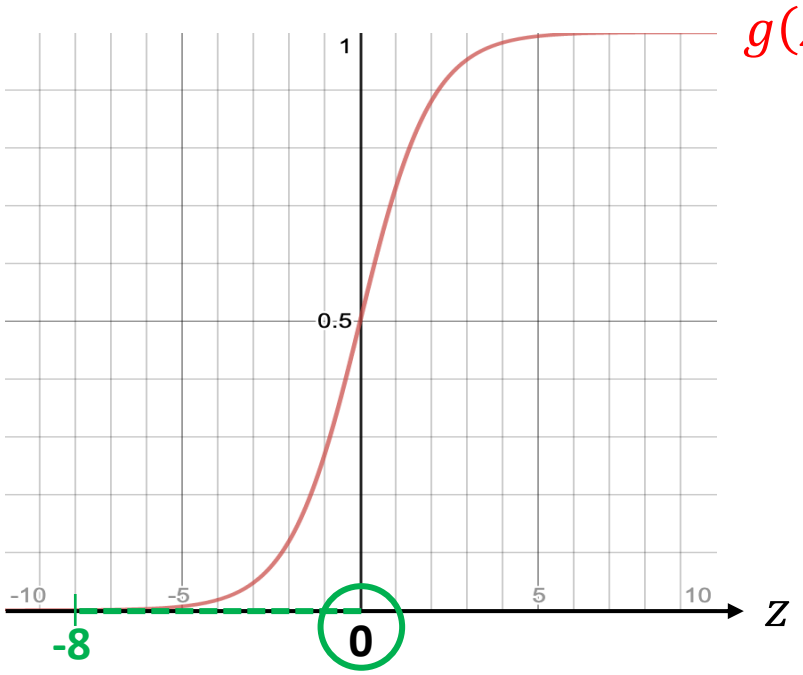
For this to happen, $\boldsymbol{z} \gg \mathbf{0}$

Logistic regression

Logistic regression takes as input numerical feature vectors and attempts to predict the probabilities by using the **sigmoid function** $g(z) = \frac{1}{1+e^{-z}}$

$$g(z) = \frac{1}{1 + e^{-z}}$$

$z \in \mathbb{R}, but$
$g(z) \in [0,1]$



Assume a labeled example ($\boldsymbol{x}$, y):

If y = **0**, we want $g(z) \approx 0$ (i.e., we want a correct prediction)

For this to happen, $\boldsymbol{z \ll 0}$

Logistic regression: Calculating a Probability

- Logistic regression takes as input numerical feature vectors and attempts to predict the probabilities by using the **sigmoid function**

$$\boldsymbol{h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}}$$

- Suppose we had a logistic regression model with three features that learned the following weights:

$$\boldsymbol{\theta} = [1, 2, -1, 5]^T$$

- Further suppose the following feature values for a given example:

$$\boldsymbol{x} = [1, 0, 10, 2]^T$$

- Then the logistic regression prediction for this example will be

$$\boldsymbol{h_\theta(x) = ?}$$

Cost function

- Now that we have restricted our choice of prediction algorithms to logistic regression, we need to choose the best one for the given training data

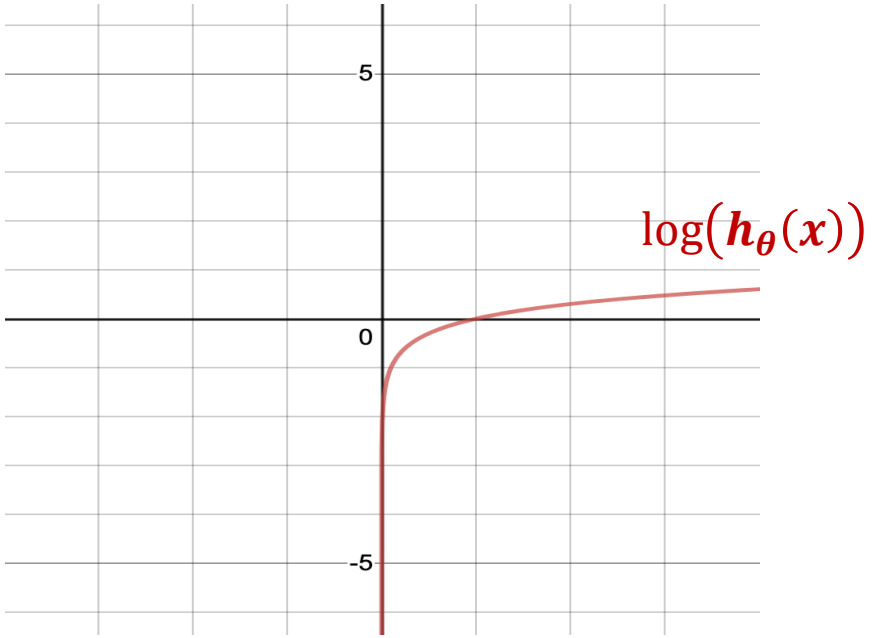
- How do we know when we have found the best algorithm
  - **cost function**
  - maps a set of pairs of (predicted label, truth label) to a real number

- The goal is to find the model parameters that produce predicted labels for the training set that minimize the cost function

The logistic regression cost function

- How to learn a *logistic regression model* $\boldsymbol{h_\theta(x) = g(\theta^T x)}$, where $\boldsymbol{\theta} = [\boldsymbol{\theta_0}, \ldots, \boldsymbol{\theta_m}]^T$ and $\boldsymbol{x} = [\boldsymbol{x_0}, \ldots, \boldsymbol{x_m}]^T$?
- Cost function

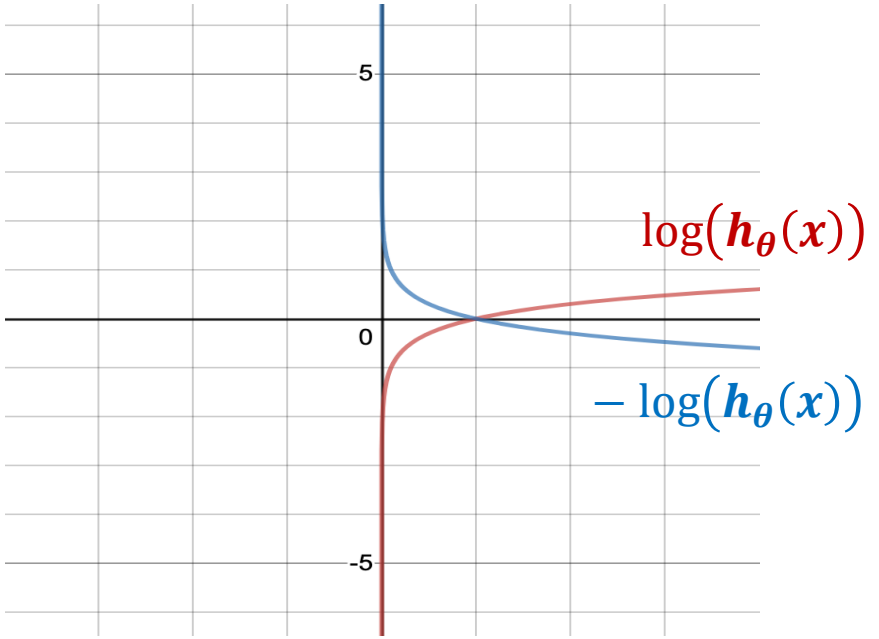
$$\text{Cost}(\boldsymbol{h_\theta(x)}, \text{y}) = \begin{cases} -\log\big(\boldsymbol{h_\theta(x)}\big)\ \boldsymbol{if}\ \boldsymbol{y = 1} \\ -\log\big(\boldsymbol{1 - h_\theta(x)}\big)\ \boldsymbol{if}\ \boldsymbol{y = 0} \end{cases}$$



The logistic regression cost function

- How to learn a *logistic regression model* $\boldsymbol{h_\theta(x) = g(\theta^T x)}$, where $\boldsymbol{\theta} = [\boldsymbol{\theta_0}, \ldots, \boldsymbol{\theta_m}]^T$ and $\boldsymbol{x} = [\boldsymbol{x_0}, \ldots, \boldsymbol{x_m}]^T$?
- Cost function

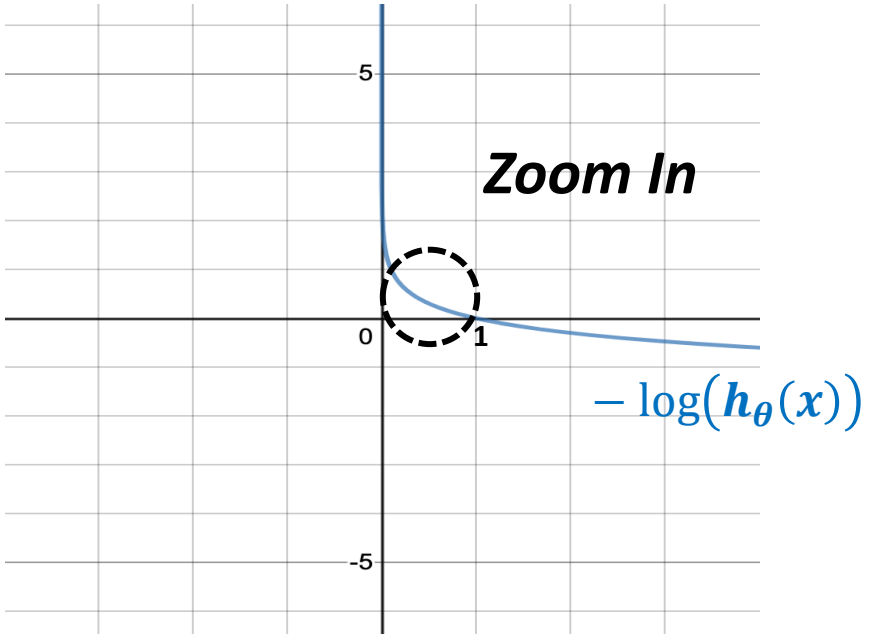
$$\text{Cost}(\boldsymbol{h_\theta(x)}, \text{y}) = \begin{cases} -\log\big(\boldsymbol{h_\theta(x)}\big) \; \boldsymbol{if} \; \boldsymbol{y = 1} \\ -\log\big(\boldsymbol{1 - h_\theta(x)}\big) \; \boldsymbol{if} \; \boldsymbol{y = 0} \end{cases}$$



The logistic regression cost function

- How to learn a *logistic regression model* $\boldsymbol{h_\theta(x) = g(\theta^T x)}$, where $\boldsymbol{\theta}$ = $\boldsymbol{[\theta_0, \dots, \theta_m]^T}$ and $\boldsymbol{x = [x_0, \dots, x_m]^T}$?
- Cost function

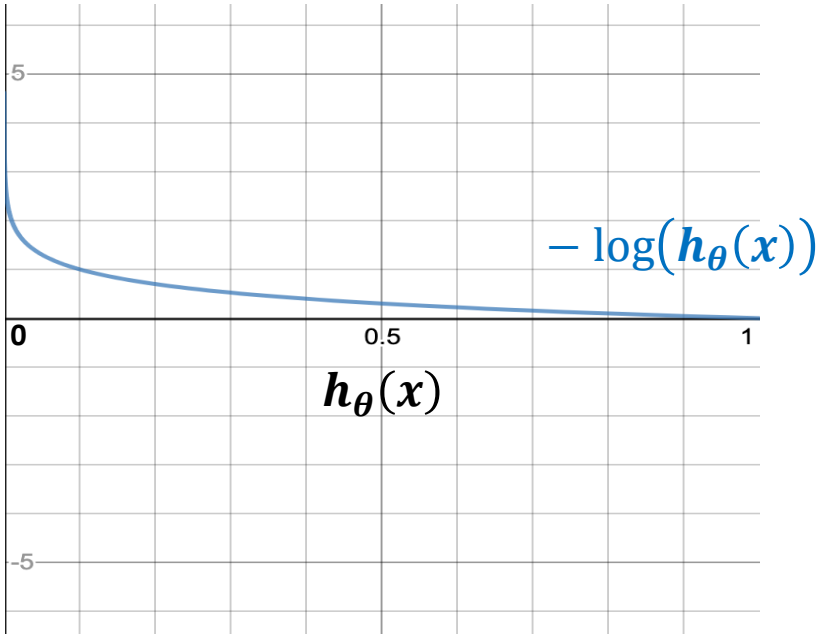
$$\text{Cost}(\boldsymbol{h_\theta(x)}, \text{y}) = \begin{cases} -\log\big(\boldsymbol{h_\theta(x)}\big) \; \boldsymbol{if} \; \boldsymbol{y = 1} \\ -\log\big(\boldsymbol{1 - h_\theta(x)}\big) \; \boldsymbol{if} \; \boldsymbol{y = 0} \end{cases}$$



The logistic regression cost function

- How to learn a *logistic regression model* $\boldsymbol{h_\theta(x) = g(\theta^T x)}$, where $\boldsymbol{\theta} = [\boldsymbol{\theta_0}, \ldots, \boldsymbol{\theta_m}]^T$ and $\boldsymbol{x} = [\boldsymbol{x_0}, \ldots, \boldsymbol{x_m}]^T$?
- Cost function

$$\text{Cost}(\boldsymbol{h_\theta(x)}, \text{y}) = \begin{cases} -\log\big(\boldsymbol{h_\theta(x)}\big) \; \boldsymbol{if} \; \boldsymbol{y = 1} \\ -\log\big(\boldsymbol{1 - h_\theta(x)}\big) \; \boldsymbol{if} \; \boldsymbol{y = 0} \end{cases}$$

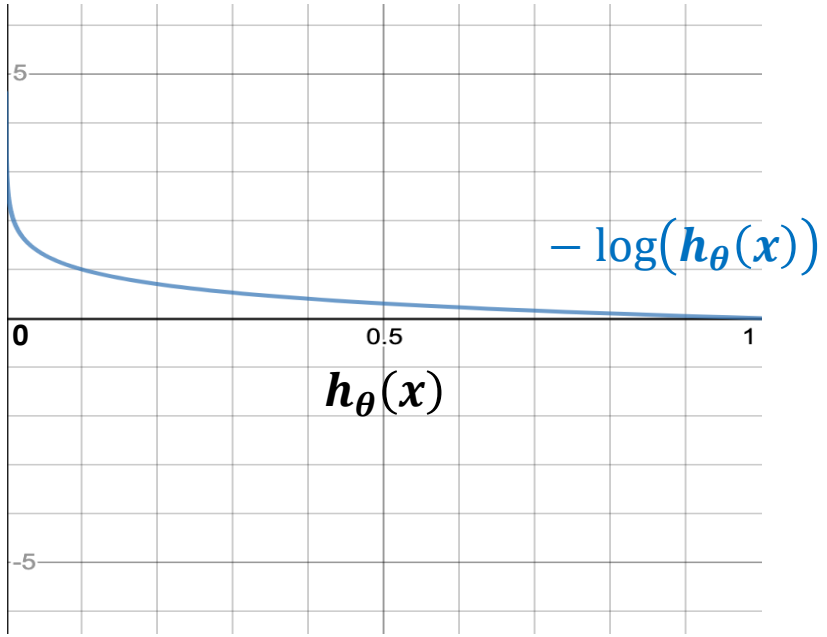


$$\boldsymbol{If \; y = 1} \begin{cases} \text{As } \boldsymbol{h_\theta(x)} \rightarrow 0, \; -\log\big(\boldsymbol{h_\theta(x)}\big) \rightarrow \infty \\ \text{As } \boldsymbol{h_\theta(x)} \rightarrow 1, \; -\log\big(\boldsymbol{h_\theta(x)}\big) \rightarrow 0 \text{ (i.e., cost } \rightarrow 0) \end{cases}$$

The logistic regression cost function

- How to learn a *logistic regression model* $\boldsymbol{h_\theta(x) = g(\theta^T x)}$, where $\boldsymbol{\theta} = [\boldsymbol{\theta_0}, \ldots, \boldsymbol{\theta_m}]^T$ and $\boldsymbol{x} = [\boldsymbol{x_0}, \ldots, \boldsymbol{x_m}]^T$?
- Cost function

$$\text{Cost}(\boldsymbol{h_\theta(x)}, \text{y}) = \begin{cases} -\log\big(\boldsymbol{h_\theta(x)}\big) \; \boldsymbol{if} \; \boldsymbol{y = 1} \\ -\log\big(\boldsymbol{1 - h_\theta(x)}\big) \; \boldsymbol{if} \; \boldsymbol{y = 0} \end{cases}$$

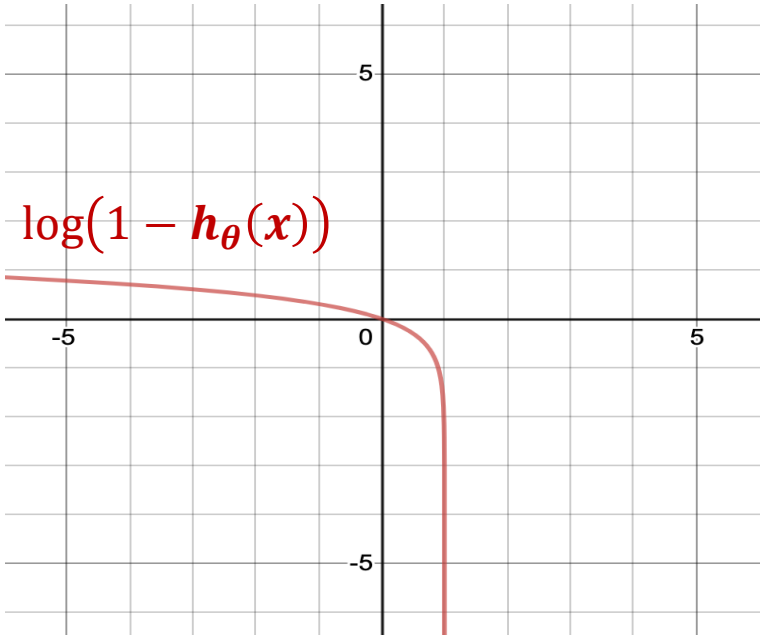


This captures the intuition that if $\boldsymbol{h_\theta(x) = 0}$ (i.e., what we will predict is $\boldsymbol{0}$), but $\boldsymbol{y = 1}$ (hence, we are mispredicting!), we shall penalize the learning algorithm by a very large cost!

The logistic regression cost function

- How to learn a *logistic regression model* $\boldsymbol{h_\theta(x) = g(\theta^T x)}$, where $\boldsymbol{\theta} = [\boldsymbol{\theta_0}, \ldots, \boldsymbol{\theta_m}]^T$ and $\boldsymbol{x} = [\boldsymbol{x_0}, \ldots, \boldsymbol{x_m}]^T$?
- Cost function

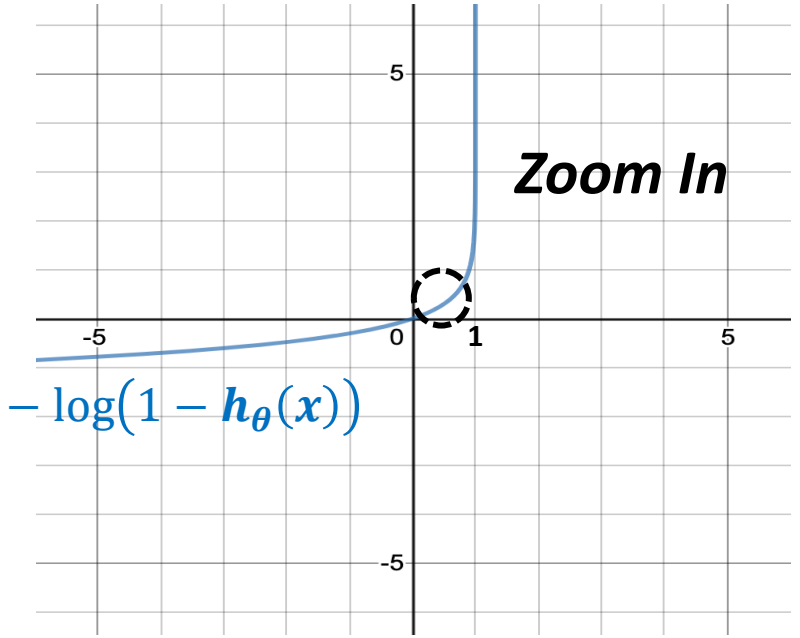
$$\text{Cost}(\boldsymbol{h_\theta(x)}, \text{y}) = \begin{cases} -\log\big(\boldsymbol{h_\theta(x)}\big) \; \boldsymbol{if} \; \boldsymbol{y = 1} \\ -\log\big(\boldsymbol{1 - h_\theta(x)}\big) \; \boldsymbol{if} \; \boldsymbol{y = 0} \end{cases}$$



The logistic regression cost function

- How to learn a *logistic regression model* $\boldsymbol{h_\theta(x) = g(\theta^T x)}$, where $\boldsymbol{\theta} = [\boldsymbol{\theta_0}, \ldots, \boldsymbol{\theta_m}]^T$ and $\boldsymbol{x} = [\boldsymbol{x_0}, \ldots, \boldsymbol{x_m}]^T$?
- Cost function

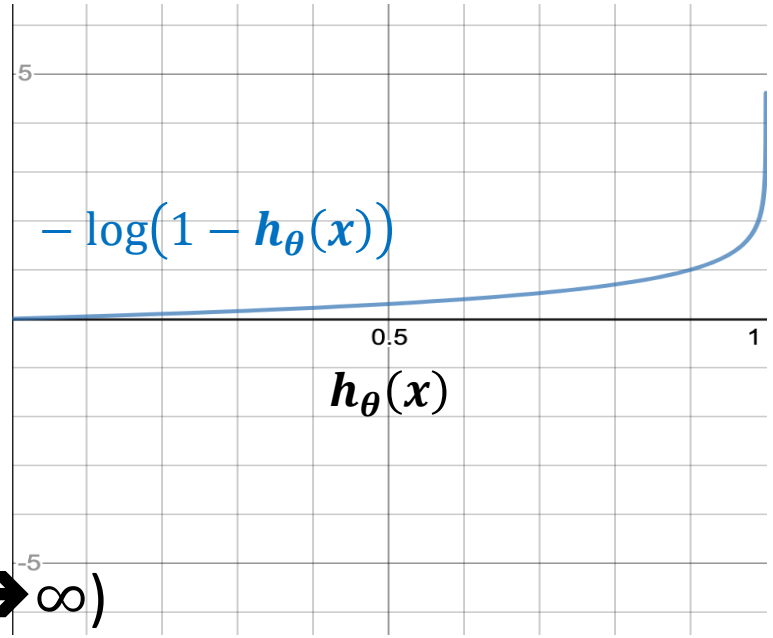
$$\text{Cost}(\boldsymbol{h_\theta(x)}, \text{y}) = \begin{cases} -\log\big(\boldsymbol{h_\theta(x)}\big) \; \boldsymbol{if} \; \boldsymbol{y = 1} \\ -\log\big(\boldsymbol{1 - h_\theta(x)}\big) \; \boldsymbol{if} \; \boldsymbol{y = 0} \end{cases}$$



The logistic regression cost function

- How to learn a *logistic regression model* $\boldsymbol{h_\theta(x) = g(\theta^T x)}$, where $\boldsymbol{\theta} = [\boldsymbol{\theta_0}, \ldots, \boldsymbol{\theta_m}]^T$ and $\boldsymbol{x} = [\boldsymbol{x_0}, \ldots, \boldsymbol{x_m}]^T$?
- Cost function

$$\text{Cost}(\boldsymbol{h_\theta(x)}, \text{y}) = \begin{cases} -\log\big(\boldsymbol{h_\theta(x)}\big) \; \boldsymbol{if} \; \boldsymbol{y = 1} \\ -\log\big(\boldsymbol{1 - h_\theta(x)}\big) \; \boldsymbol{if} \; \boldsymbol{y = 0} \end{cases}$$

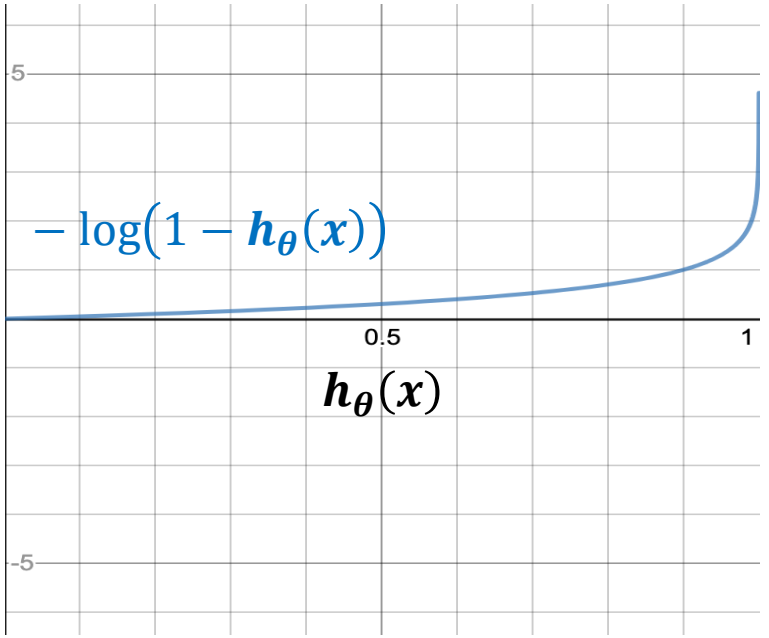


$\boldsymbol{If\ y = 0}$
- As $\boldsymbol{h_\theta(x)}$ ➔ 0, $-\log\big(1 - \boldsymbol{h_\theta(x)}\big)$ ➔ 0
- As $\boldsymbol{h_\theta(x)}$ ➔ 1, $-\log\big(1 - \boldsymbol{h_\theta(x)}\big)$ ➔ ∞ (i.e., cost ➔ ∞)

The logistic regression cost function

- How to learn a *logistic regression model* $\boldsymbol{h_\theta(x) = g(\theta^T x)}$, where $\boldsymbol{\theta} = [\boldsymbol{\theta_0}, \ldots, \boldsymbol{\theta_m}]^T$ and $\boldsymbol{x} = [\boldsymbol{x_0}, \ldots, \boldsymbol{x_m}]^T$?
- Cost function

$$\text{Cost}(\boldsymbol{h_\theta(x)}, \text{y}) = \begin{cases} -\log\big(\boldsymbol{h_\theta(x)}\big) \; \boldsymbol{if} \; \boldsymbol{y = 1} \\ -\log\big(\boldsymbol{1 - h_\theta(x)}\big) \; \boldsymbol{if} \; \boldsymbol{y = 0} \end{cases}$$



This captures the intuition that if $\boldsymbol{h_\theta(x) = 1}$ (i.e., what we will predict is $\boldsymbol{1}$), but $\boldsymbol{y = 0}$ (i.e., we are mispredicting!), we shall penalize the learning algorithm by a very large cost!

The logistic regression cost function

- How to learn a *logistic regression model* $\boldsymbol{h_\theta(x) = g(\theta^T x)}$, where $\boldsymbol{\theta} = [\boldsymbol{\theta_0}, \ldots, \boldsymbol{\theta_m}]^T$ and $\boldsymbol{x} = [\boldsymbol{x_0}, \ldots, \boldsymbol{x_m}]^T$?
- Cost function

$$\text{Cost}(\boldsymbol{h_\theta(x)}, \text{y}) = \begin{cases} -\log\big(\boldsymbol{h_\theta(x)}\big) \; \boldsymbol{if} \; \boldsymbol{y = 1} \\ -\log\big(\boldsymbol{1 - \; h_\theta(x)}\big) \; \boldsymbol{if} \; \boldsymbol{y = 0} \end{cases}$$

Equivalent To

$$\text{Cost}(\boldsymbol{h_\theta(x)}, \text{y}) = -y \log\big(\boldsymbol{h_\theta(x)}\big) - (1 - y) \log\big(\boldsymbol{1 - \; h_\theta(x)}\big)$$

The logistic regression cost function

- How to learn a *logistic regression model* $\boldsymbol{h_\theta(x) = g(\theta^T x)}$, where $\boldsymbol{\theta} = [\boldsymbol{\theta_0}, \ldots, \boldsymbol{\theta_m}]^T$ and $\boldsymbol{x} = [\boldsymbol{x_0}, \ldots, \boldsymbol{x_m}]^T$?
  - By minimizing the following cost function:

$$\text{Cost}(\boldsymbol{h_\theta(x)}, \text{y}) = -y \log\big(\boldsymbol{h_\theta(x)}\big) - (1-y) \log\big(\boldsymbol{1} - \boldsymbol{h_\theta(x)}\big)$$

$$= -y \log\big(\boldsymbol{g(\theta^T x)}\big) - (1-y) \log\big(1 - \boldsymbol{g(\theta^T x)}\big)$$

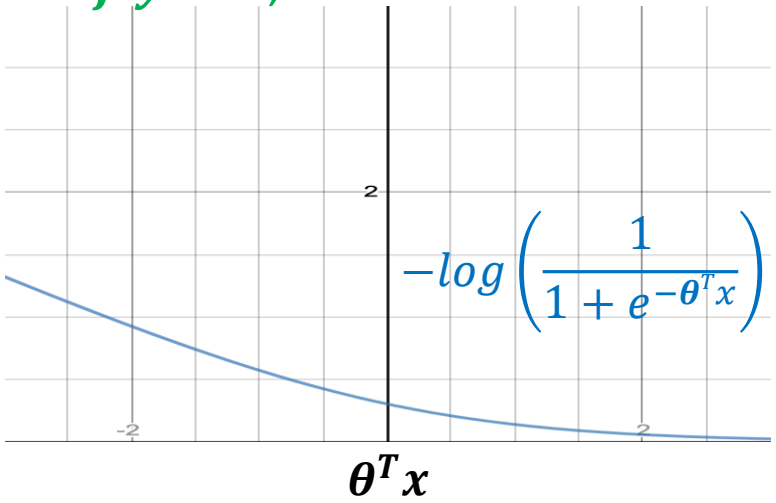
$$= -y \log\left(\frac{1}{1+e^{-\boldsymbol{\theta}^T x}}\right) - (1-y) \log\left(1 - \frac{1}{1+e^{-\boldsymbol{\theta}^T x}}\right)$$

The logistic regression cost function

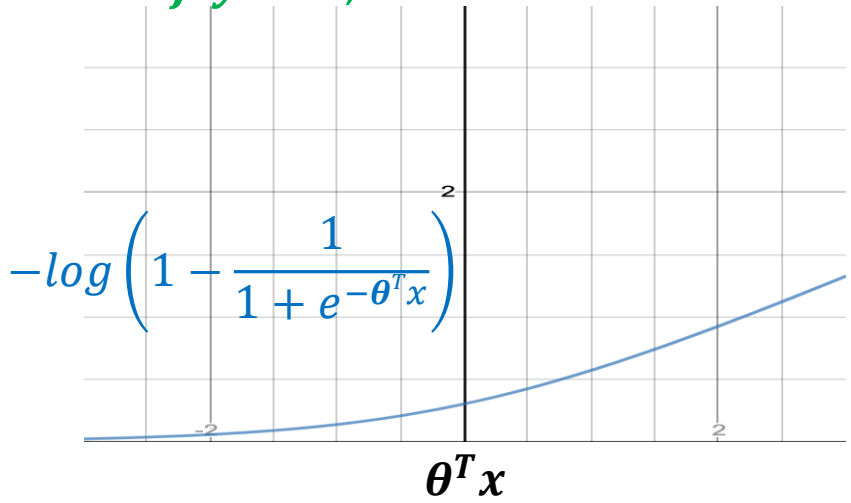
- How to learn a *logistic regression model* $\boldsymbol{h_\theta(x) = g(\theta^T x)}$, where $\boldsymbol{\theta} = [\boldsymbol{\theta_0}, \ldots, \boldsymbol{\theta_m}]^T$ and $\boldsymbol{x} = [\boldsymbol{x_0}, \ldots, \boldsymbol{x_m}]^T$?
  - By minimizing the following cost function:

$$\text{Cost}(\boldsymbol{h_\theta(x)}, \text{y}) = -y \log\left(\frac{1}{1+e^{-\boldsymbol{\theta}^T x}}\right) - (1-y)\log\left(1 - \frac{1}{1+e^{-\boldsymbol{\theta}^T x}}\right)$$

If $\boldsymbol{y = 1}$, we want $\boldsymbol{\theta^T x >> 0}$



If $\boldsymbol{y = 0}$, we want $\boldsymbol{\theta^T x << 0}$



Learning a logistic regression model

- How to learn a *logistic regression model* $\boldsymbol{h_\theta(x) = g(\theta^T x)}$, where $\boldsymbol{\theta} = [\boldsymbol{\theta_0}, \ldots, \boldsymbol{\theta_m}]^T$ and $\boldsymbol{x} = [\boldsymbol{x_0}, \ldots, \boldsymbol{x_m}]^T$?
  - By minimizing the following cost function:

$$\text{Cost}(\boldsymbol{h_\theta(x)}, \text{y}) = -y \log\left(\frac{1}{1+e^{-\boldsymbol{\theta}^T x}}\right) - (1-y)\log\left(1 - \frac{1}{1+e^{-\boldsymbol{\theta}^T x}}\right)$$

- That is:

$$\underset{\theta}{\text{minimize}} \frac{1}{n}\sum_{i=1}^{n} \text{Cost}(\boldsymbol{h_\theta(x)}^{(i)}, y^{(i)})$$

$$\equiv$$

$$\underset{\theta}{\text{minimize}} \frac{1}{n}\sum_{i=1}^{n} -y^{(i)} \log\left(\frac{1}{1+e^{-\boldsymbol{\theta}^T x^{(i)}}}\right) - (1-y^{(i)})\log\left(1 - \frac{1}{1+e^{-\boldsymbol{\theta}^T x^{(i)}}}\right)$$

Cost function $J(\boldsymbol{\theta})$

Gradient descent for logistic regression

- **Outline**:
  - Have cost function $\boldsymbol{J(\theta)}$, where $\boldsymbol{\theta} = [\boldsymbol{\theta_0}, \ldots, \boldsymbol{\theta_m}]^T$
  - Start with some guesses for $\theta_0, \ldots, \theta_m$
    - a common choice is to set them all initially to zero or random values
  - Repeat until convergence{

$$\theta_j = \theta_j - \alpha \frac{\partial \boldsymbol{J(\theta)}}{\partial\, \theta_j}$$

***Learing rate**, which controls how big a step to take when update θ_j*

}

Gradient descent for logistic regression

- **Outline**:
  - Have cost function $\boldsymbol{J(\theta)}$, where $\boldsymbol{\theta} = [\boldsymbol{\theta_0}, \dots, \boldsymbol{\theta_m}]^T$
  - Start with some guesses for $\theta_0, \dots, \theta_m$
    - a common choice is to set them all initially to zero
  - Repeat until convergence{

$$\theta_j = \theta_j - \alpha \sum_{i=1}^{n} \left(\frac{1}{1 + e^{-\boldsymbol{\theta}^T x^{(i)}}} - y^{(i)} \right) x_j{}^{(i)}$$

The final formula after applying partial derivatives

}

Inference after learning

- After learning the parameters $\boldsymbol{\theta} = [\boldsymbol{\theta_0}, \ldots, \boldsymbol{\theta_m}]^T$, we can predict the output of any new unseen $\boldsymbol{x} = [\boldsymbol{x_0}, \ldots, \boldsymbol{x_m}]^{\boldsymbol{T}}$ as follows:

$$\begin{cases} \boldsymbol{if}\ \boldsymbol{h_\theta(x)} = \dfrac{1}{1 + e^{-\boldsymbol{\theta}^T \boldsymbol{x}}} < \mathbf{0.5}\ \mathbf{predict\ 0} \\ \boldsymbol{Else\ if}\ \boldsymbol{h_\theta(x)} = \dfrac{1}{1 + e^{-\boldsymbol{\theta}^T \boldsymbol{x}}} \geq \mathbf{0.5}\ \mathbf{predict\ 1} \end{cases}$$

How to evaluate models?

- Accuracy
- Different types of errors
- Precision and Recall
- F-score

Two example datasets

- We use the metrics to evaluate several models on two toy datasets
  - a medical dataset of patients, some are diagnosed with coronavirus
    - **1000** patients: **10** 'sick' and **990** 'healthy'
    - The goal of a model: predict the diagnosis based on the features of each patient
  - a dataset of emails that have been labeled as spam or not spam(ham)
    - **100** emails: **40** 'spam' and **60** 'ham'
    - The goal of a model: predict the label based on the features of the email
- Note that we are not building models
- Instead, we use the models as black boxes and **evaluate** them
  - how many of the data points they predict correctly or incorrectly

Accuracy

- **Accuracy**: the percentage of times that a model is correct

$$\frac{\textit{number of correctly predicted data points}}{\textit{total number of data points}}$$

- Example
  - if we evaluate a model on a test dataset of 1,000 samples, and the model predicted the correct label of the samples 900 times, then this model has an accuracy of 900/1000 = 90%

- Be careful of “Accuracy”
  - sometimes accuracy doesn’t fully describe the performance of the model

Accuracy

- This measure is dominated by the larger set (of positives or negatives) and may favor trivial classifiers
- Can you think of a completely useless model that predicts coronavirus in our dataset, yet is correct 99% of the time?
  - Recall that the dataset has **1000** patients: **10** 'sick' and **990** 'healthy'
- Simply diagnose every patient as 'healthy'
  - out of 1,000 tries, it's incorrect 10 times and correct 990 times
- Errors are not created equal, and some mistakes are much more expensive than others

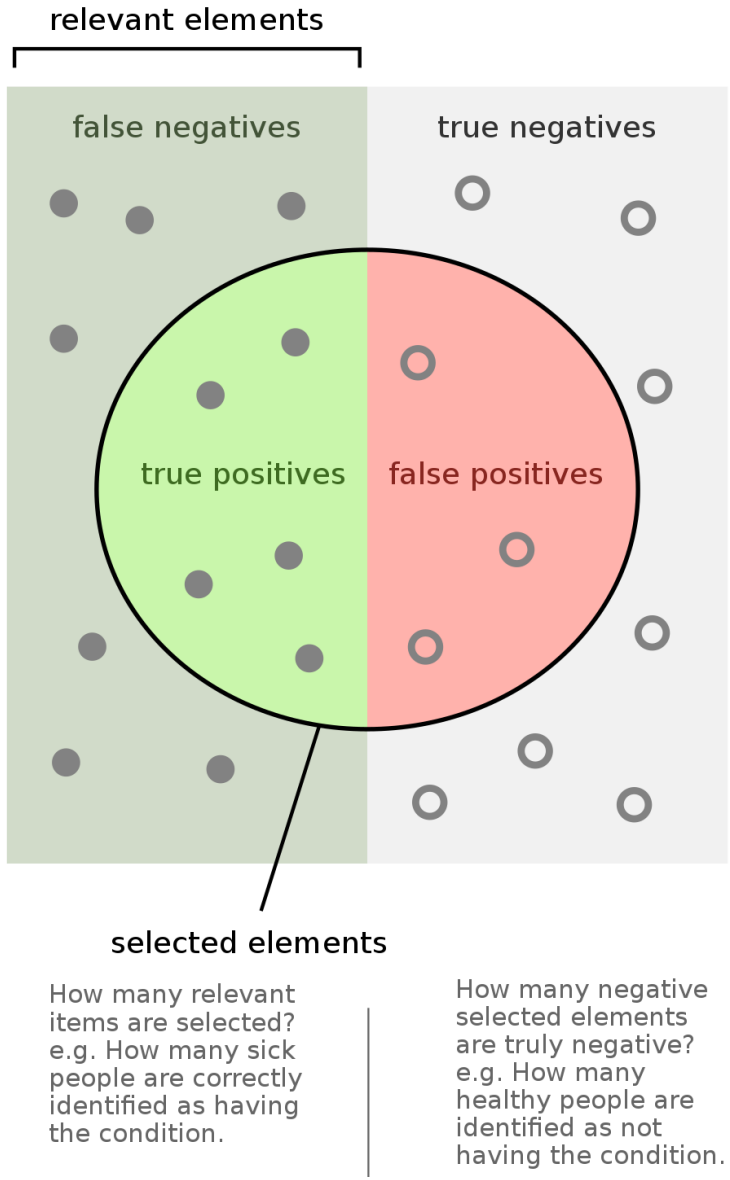
Different types of errors

*In our model, we label the **sick** patients **as positive***

- What are the two types of errors the coronavirus model can make?
  - **False positive**: a healthy person who is **incorrectly** diagnosed as sick
  - **False negative**: a sick person who is **incorrectly** diagnosed as healthy
- Cases that are **correctly** diagnosed also have names
  - **True positive**: a sick person who is diagnosed as sick
  - **True negative**: a healthy person who is diagnosed as healthy
- Question: If consider the **positives** to be the **spam** emails
  - False positive? a ham email that is incorrectly classified as spam
  - False negative? a spam email that is incorrectly classified as ham
  - True positive? a spam email that is correctly classified as spam
  - True negative? a ham email that is correctly classified as ham

Different types of errors

- a true positive is a data point with a positive label that is correctly classified as positive
- a true negative is one with a negative label that is correctly classified as negative
- a false positive is one with a negative label, but the model falsely classifies it as positive
- a false negative is one with a positive label, but the model falsely classified it as negative



Confusion matrix

- A specific table layout to visualize the performance
  - rows represent the label
  - columns represent the prediction
  - elements in the diagonal are classified correctly
  - elements off the diagonal are classified incorrectly
- Question:
  - Accuracy = ?

$$\frac{TP + TN}{TP + TN + FP + FN}$$

		Predicted condition	
	Total population = P + N	Positive (PP)	Negative (PN)
Actual condition	Positive (P)	True positive (TP)	False negative (FN)
	Negative (N)	False positive (FP)	True negative (TN)

Confusion matrix

- Let's consider the 'useless' model
  - For dataset with **1000** patients: **10** 'sick' and **990** 'healthy'
  - Simply diagnose every patient as 'healthy'
  - The confusion matrix:

Coronavirus model 1	Diagnosed sick (predicted positive)	Diagnosed healthy (predicted negative)
Sick (positive)	0 (number of true positives)	10 (number of false negatives)
Healthy (negative)	0 (number of false positives)	990 (number of true negatives)

- The model creates too many false negatives

Recall

- **Recall**: TP/(TP + FN)
  - the proportion of correct predictions among the data points with positive labels

Coronavirus model 1	Diagnosed sick (predicted positive)	Diagnosed healthy (predicted negative)
Sick (positive)	0 (number of true positives)	10 (number of false negatives)
Healthy (negative)	0 (number of false positives)	990 (number of true negatives)

- Accuracy = 99%
- Recall = ?

Precision

- **Precision**: TP/(TP + FP)
  - the proportion of true positives among data points that have been predicted as positive
- spam model 1:
  - precision=85.7%
  - accuracy = 85%

Spam model 1	Predicted spam	Predicted ham
Spam	30 (true positives)	10 (false negatives)
Ham	5 (false positives)	55 (true negatives)

- Spam model 2:
  - precision=77.7%
  - accuracy=85%

Spam model 2	Predicted spam	Predicted ham
Spam	35 (true positives)	5 (false negatives)
Ham	10 (false positives)	50 (true negatives)

F-score

- A measure that naturally combines ***P***recision and ***R***ecall is the F-score:

$$F_\beta = \frac{(1+\beta^2)PR}{\beta^2 P + R}$$

- Setting $\beta = 1$ gives us the ***F_1 – score***. It can also be computed as:

$$F_1 = \frac{2PR}{P+R}$$

  - The F1 score gives equal weight to both measures
  - If we want to create a classification model with a good balance of precision and recall, maximize the F1 score

Practical Considerations

- In theory, applying a machine learning algorithm
  - put training data into a large matrix
  - run training procedure
  - use the resulting model to classify unseen data
- However, the task is not so simple
  - dozens of choices to be made in the model construction process
  - each choice can lead to very different outcomes in your final model
- Now consider some of the most important choices to be made during the modeling process

Selecting a Model Family

- Given a task, how do we decide which algorithm to pick?
- Let the data decide: try different approaches and see what works best
- Some factors to consider:
  - Computational complexity
    - Train the model in a reasonable amount of time
  - Explainability
    - A human might need to read the output of the model and determine why it made the decision
    - e.g. if you block a user, you should educate them as to what they did that was suspicious
- But ultimately, you should experiment to find the best choice for your data

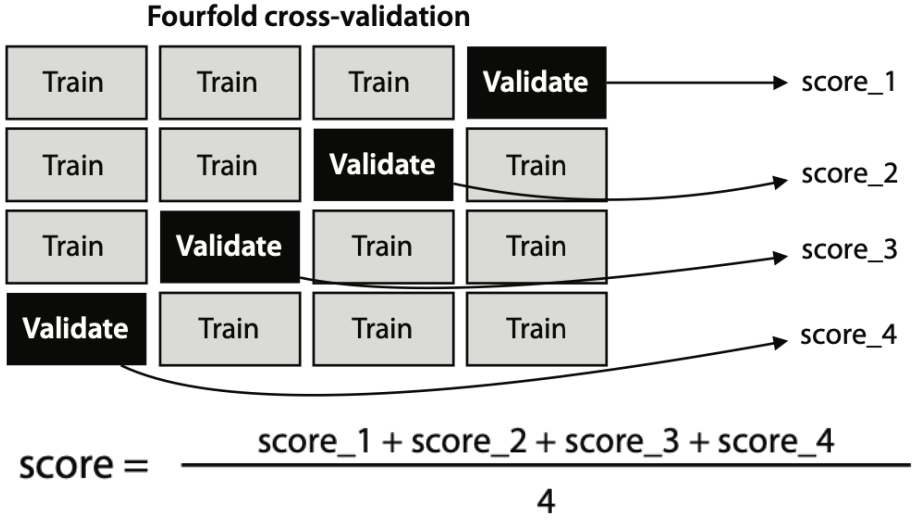
Training data construction

- In a supervised learning problem, we are given labeled examples of the data that we want to classify
  - e.g. account registrations, user logins, email messages
- Goal is to produce a model that can predict the future based on past data, need to reserve some of the labeled data to act as this “future” data so that we can **evaluate** our model
- Different ways to do this
  - Cross-validation
  - Train/validate/test
  - Out-of-time validation

Cross-validation

- K-fold cross-validation is a way to use all the data for training and testing, by recycling it several times
  - Split the labeled data into k equal portions
  - Train the model k times, using the union of k – 1 of the portions as the training set and the remaining one as a validation set
  - The final score of that model is the **average** of the validation scores

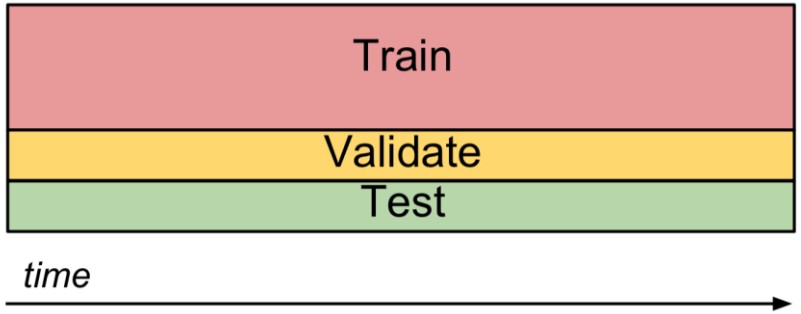
classical training-validation split



$$\text{score} = \frac{\text{score\_1} + \text{score\_2} + \text{score\_3} + \text{score\_4}}{4}$$

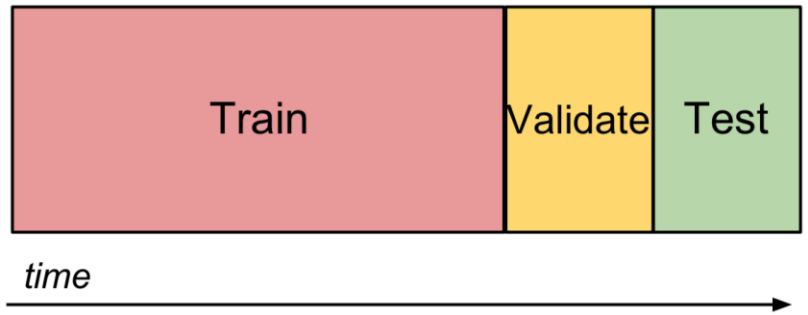
Train/validate/test

- Randomly divide the labeled data into three parts
  - The majority is the *training set:* input to the learning algorithm
  - The second part is the *validation set (a.k.a. development set):* evaluate and iterate on the model output by the learning algorithm
    - tune model parameters based on performance on the validation set
  - Finally, after get an optimal model, use the remaining part as the *test set* to estimate real-world performance on unseen data
    - Deep learning competitions normally keep the test set secret.



Out-of-time validation

- This approach addresses the problem of validating models when the data distribution changes over time
- In this case, sampling training and validation sets at random from the same labeled set is “cheating” in some sense
  - the distributions of the training and validation sets will match closely and not reflect the time aspect
- A better approach is to split the training and validation sets based on a time cutoff:



Unbalanced data

- When classify a rare event (e.g. account hijacking)—the “good” samples can be even 99.9% of the labeled data
  - In this case the “bad” samples might not have enough weight to ‘influence’ the classifier
  - model might perform poorly when trained on this highly unbalanced data
- Some options:
  - ***Oversample*** the minority class: repeat observations in your training data to make a more balanced set
    - Synthesizing minority classes: generative models.
  - ***Undersample*** the majority class: select a random subset of the majority class to produce a more balanced set
  - ***Change the cost function*** to weight performance on the minority class more heavily, so that each minority sample has more influence on the model

Missing features

- In an ideal world every event is logged perfectly, but things go wrong
  - e.g. bugs appear in the logging; some features are delayed
- As a result, some of the samples might have ***missing features***
- If the features are missing due to random failures, remove events with missing features
  - if the data with missing features is clustered around a certain event, throwing out this data will change the distribution
- Impute the value of the missing feature
  - Simplest approach: assign the missing feature the average or median value for that feature

Adaptive attacks

- In an adversarial environment the attackers will rarely give up after we deploy a new defense.
  - they will modify their methods to try to circumvent defenses, we need to respond, and so on so forth
- Thus, not only does the distribution of attacks change over time, but it changes directly in response to our actions
- e.g. to produce a model that is robust against current attacks
  - training set should weight recent data more heavily,
    - either by relying only on the past days
    - or by using some kind of decay function to downsample historical data

Summary

- ML Problems and Approaches
- Machine Learning in Practice: a working example
- Training Algorithms to Learn
- Evaluating Models
- Practical Considerations

References

- *Machine Learning , Analytics & Cyber Security the Next Level Threat Analytics, Manjunath N V*
- *Machine Learning and Security Protecting Systems with Data and Algorithms, Clarence Chio, David Freeman*
- *Machine learning foundational course, google developers*
- *Logistic regression, Mohammad Hammoud, CMU Qatar*
- *Practical Machine Learning in Infosecurity, Clarence Chio and Anto Joseph*
- *scikit-learn: https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression*
- *pandas documentation: https://pandas.pydata.org/docs/*
- *Machine Learning Core Concept, Mohammad Hammoud, CMU in Qatar*
- *Linear regression, Mohammad Hammoud, CMU in Qatar*