Machine Lerning

Lecture 2
17-03-2022

Dr. Arif Mahmood
Professor ITU

E-mail: arif.mahmood@itu.edu.pk

[Website](#) [Google Scholar](#)

Office: 6-th Floor CS Department, ITU

Nearest Neighbor Classifiers

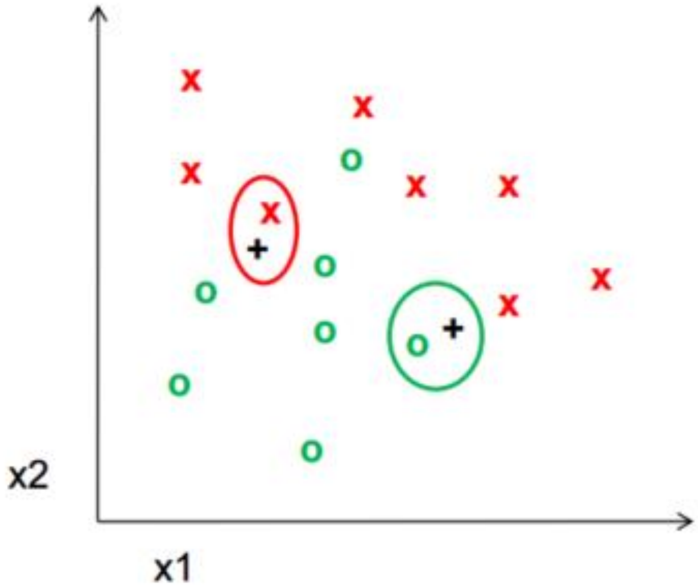
- Give label to an unknown instance as the label of its nearest neighbor
  - 1-Nearest Neighbor
  - 2-Nearest Neighbors
  - 3-Nearest Neighbors

  - K-Nearest Neighbors

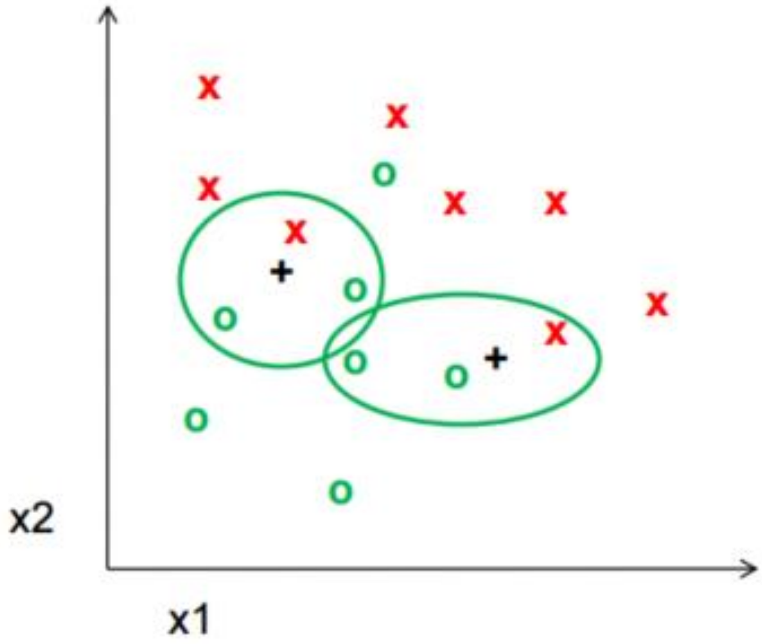
A man is known by the company he keeps.
A man is jugged by the company he keeps.
Birds of a feather flock together.

1-nearest neighbour

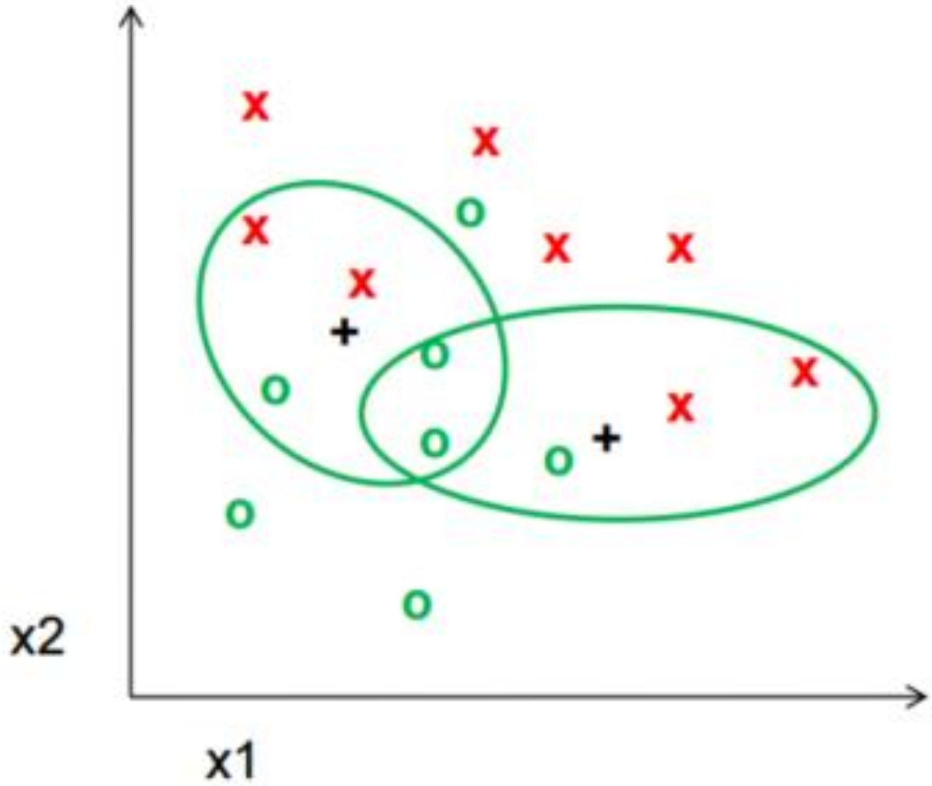
Task: classify the test set of "+"
The labels for the training set are GREEN and RED
The examples are 2-dimensional
Use L2/Euclidean distance



3-nearest neighbour



5-nearest neighbour



The Task: Supervised Learning

- Given a set of labelled examples (the *training set*), determine/predict the labels of a set of unlabelled examples (the *test set*)
  - Training set:

    Train Example 1: $(x_1^{(1)}, x_2^{(1)}, \dots., x_m^{(1)})$ Label: $y^{(1)}$

    Train Example 2: $(x_1^{(2)}, x_2^{(2)}, \dots., x_m^{(2)})$ Label: $y^{(2)}$

    …

    Train Example N: $(x_1^{(N)}, x_2^{(N)}, \dots., x_m^{(N)})$ Label: $y^{(N)}$
  - Test set:

    Test Example 1: $(x_1^{(N+1)}, x_2^{(N+1)}, \dots., x_m^{(N+1)})$ Label: $y^{(N+1)}$

    Test Example 2: $(x_1^{(N+2)}, x_2^{(N+2)}, \dots., x_m^{(N+2)})$ Label: $y^{(N+2)}$

    …

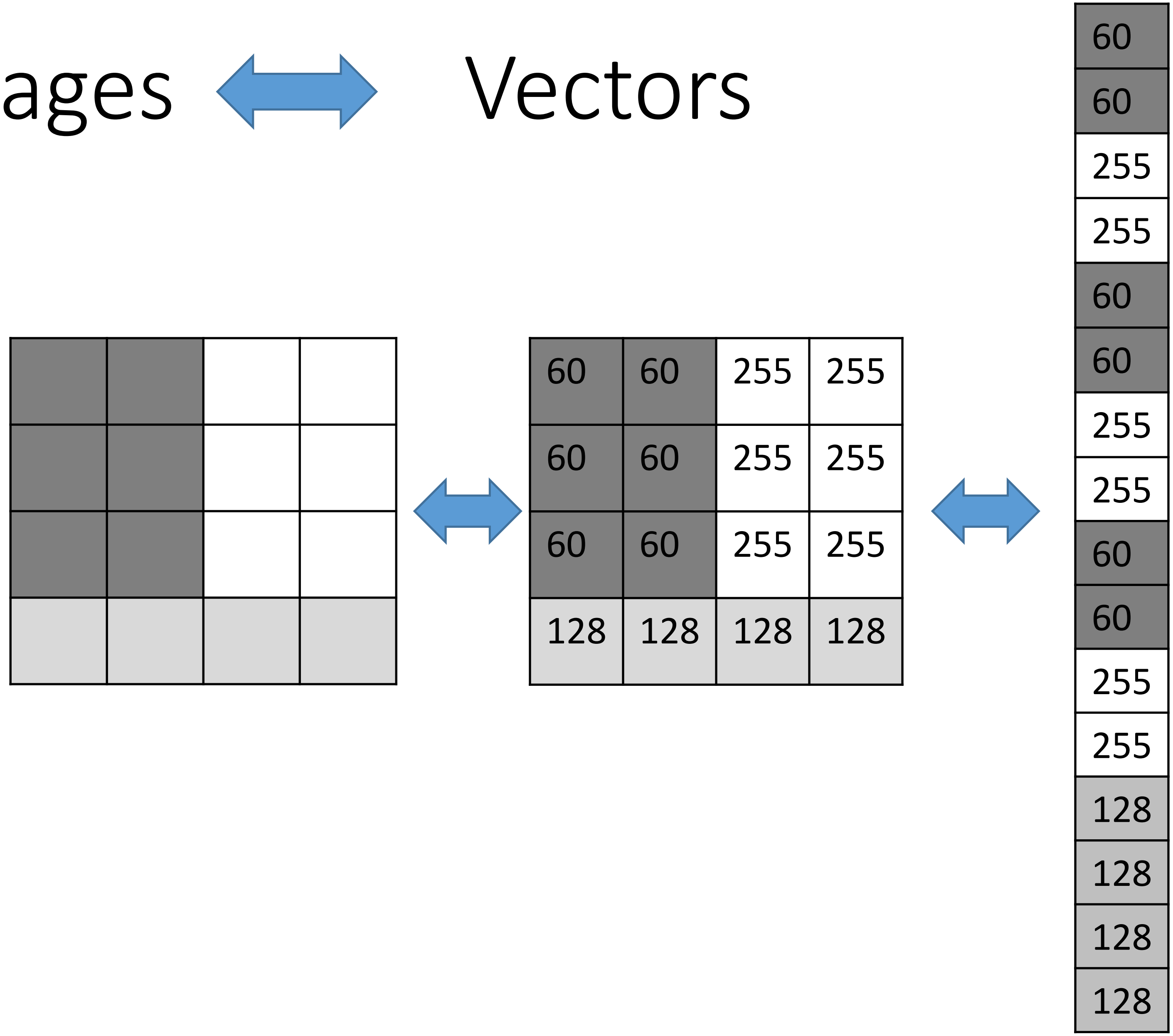
    Test Example K: $(x_1^{(N+K)}, x_2^{(N+K)}, \dots., x_m^{(N+K)})$ Label: $y^{(N+K)}$

Task: Face Recognition

- Training set: photos of musicians with names ("labels")
- Test set: photos of musicians whose name we want to figure out
  - Note: generally, we *will* know the labels for the test set, but we pretend we don't. We can then predict the labels using our algorithm and compare the answers the algorithm gives to the correct answers to figure out the performance of our algorithm.
- An estimate for the performance of the algorithm on *new data*: the proportion of the examples in the test set that were correctly classified

What Justin Bieber Looks like to a Computer

Images ↔ Vectors



The Face Recognition Task

- Training set:
  - $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(N)}, y^{(N)})\}$
    - $x^{(i)}$ is a k-dimensional vector consisting of the intensities of all the pixels in in the i-th photo (20×20 photo $\rightarrow x^{(i)}$ is 400-dimensional)
    - $y^{(i)}$ is the *label* (i.e., name)
- Test phase:
  - We have an input vector x, and want to assign a label y to it
    - Whose photo is it?

Face Recognition using 1-Nearest Neighbors (1NN)

- Training set: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(N)}, y^{(N)})\}$
- Input: x
- 1-Nearest Neighbor algorithm:
  - Find the training photo/vector $x^{(i)}$ that's as "close" as possible to x, and output the label $y^{(i)}$

Input x

Closest training image to the input x

Output: Paul

Are the two images a and b close?

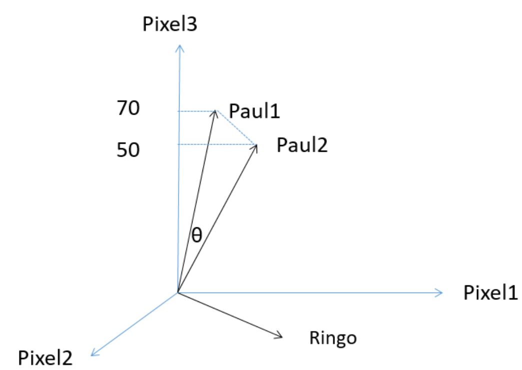
- Key idea: think of the images as *vectors*
  - Reminder: to turn an image into a vector, simply "flatten" all the pixels into a 1D vector
- Is the distance between the endpoints of vectors a and b small?

$$|a - b| = \sqrt{\sum_i \ (a_i - b_i)^2} \text{ small}$$

- Is the cosine of the angle between the vectors a and b large?

$$\cos\theta_{ab} = \frac{a \cdot b}{|a||b|} = \frac{\sum_i \ a_i b_i}{\sqrt{\sum_i \ a_i^2}\sqrt{\sum_i \ b_i^2}} \text{ large}$$

By the law of cosines



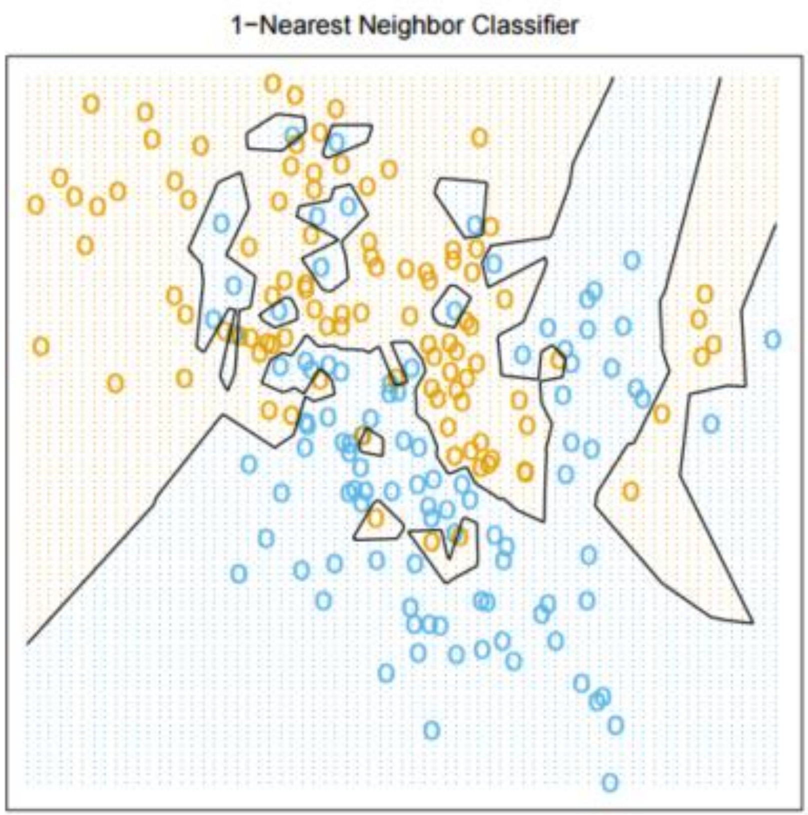
k-Nearest Neighbour Classification

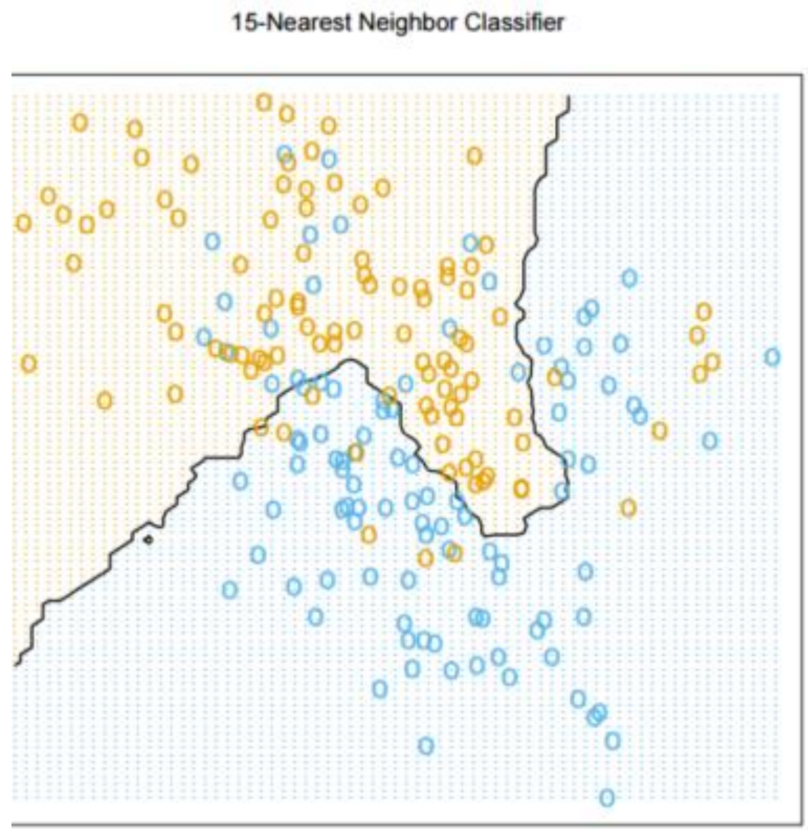
- For an example x
  - Find the k closest examples (neighbours) to x in the training set
  - Output the plurality label for the k closest examples
- Can use various distance functions:
  - Euclidian (L2): $\mathrm{dist}(a, b) = \sqrt{\sum_i (a_i - b_i)^2}$ (default)
  - L-infinity: $\mathrm{dist(a, b)} = \max_i |a_i - b_i|$
  - L-zero: $\mathrm{dist(a, b)} = \#\{a_i \neq b_i\}$
  - Negative cosine: $\mathrm{dist(a, b)} = -\frac{a \cdot b}{|a||b|}$

How do we determine K?

- Try different values, and see which works best on the test set?
  - Could do that, but then we are selecting the best K for our particular test set. This means that the performance on our test set is now an overestimate of how well we'd do on *new data*
- Solution: set aside a *validation set* (which is separate from both the training and the test set), and select the K for the best performance on the validation set, but report the results on the test set
  - Generally, the performance on the validation set will be better than on the test set
  - What about the performance on the *training set*?

What does the best K say about the data?





Large k: relatively simple boundary, no small "islands" in the data. Small changes in x do no generally change the label
Small k: a complex boundary between the labels. Small changes in x often change the labels

Why not let K be very small?

- Great for the performance on the training set!
  - Perfect performance guaranteed for k = 1
- If the test data does not look exactly like the training data, the performance on the test data will be worse for k that is too small
  - The training data could be noisy (e.g., in the orange region, data points are sometimes blue with probability 5%, randomly)
  - This is an example of *overfitting* – building a classifier that works well on the training set, but does not generalize well to the test set

Why not let K be very large?

Distance Functions

- For images, why might the cosine distance make sense?
- For images, why might the Euclidean distance make sense?

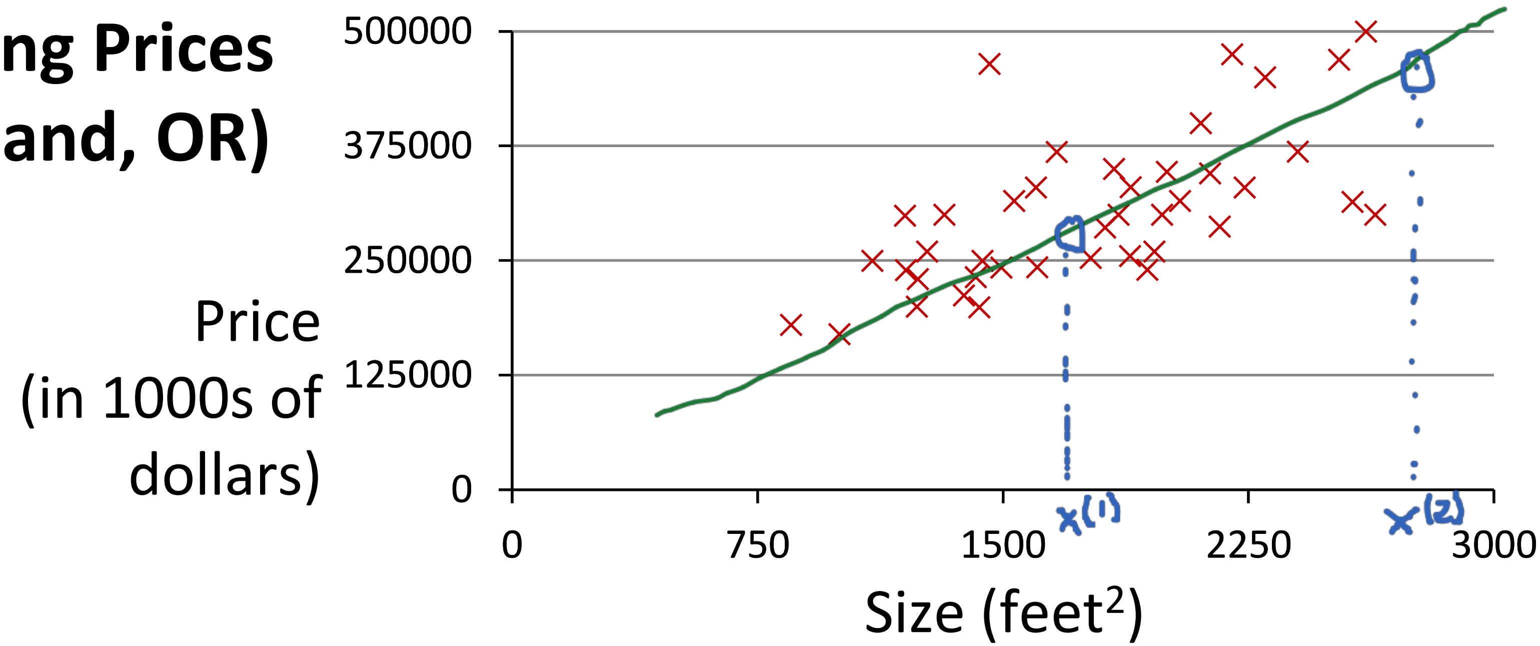
Training set of housing prices (Portland, OR)

Size in feet^2 (x)	**Price ($) in 1000's (y)**
2104	460
1416	232
1534	315
852	178
…	…

Notation:

m = Number of training examples
x’s = “input” variable / features
y’s = “output” variable / “target” variable

Housing Prices (Portland, OR)

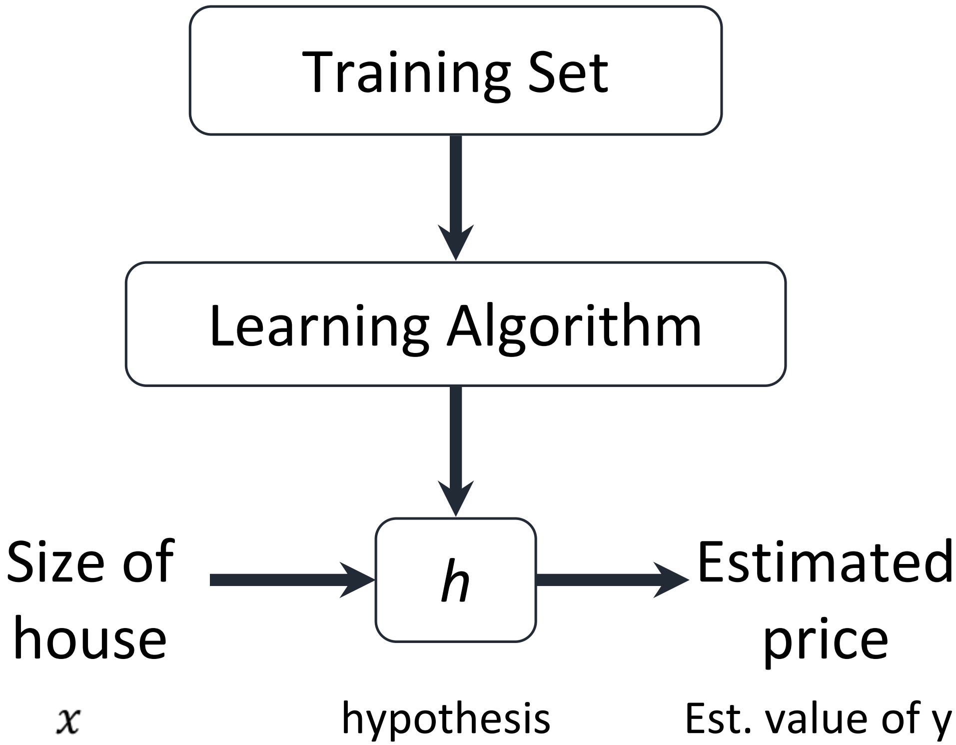


<u>Supervised Learning</u>

Given the “right answer” for each example in the data.

<u>Regression Problem</u>

Predict real-valued output



H maps x's to y's.

How do we represent *h* ?

- We represent hypotheses about the data using the parameters $\theta = (\theta_0, \theta_1)$
- If the data is correctly predicted according to hypothesis h_θ, then $y \approx h_\theta(x) = \theta_0 + \theta_1 x$
- The learning algorithm finds the best hypothesis h_θ for the training set
- We can then estimate the values of y for the test set using that h_θ
- If $h_\theta(x)$ is a linear function of a real number x, this procedure is called linear regression.

Training Set

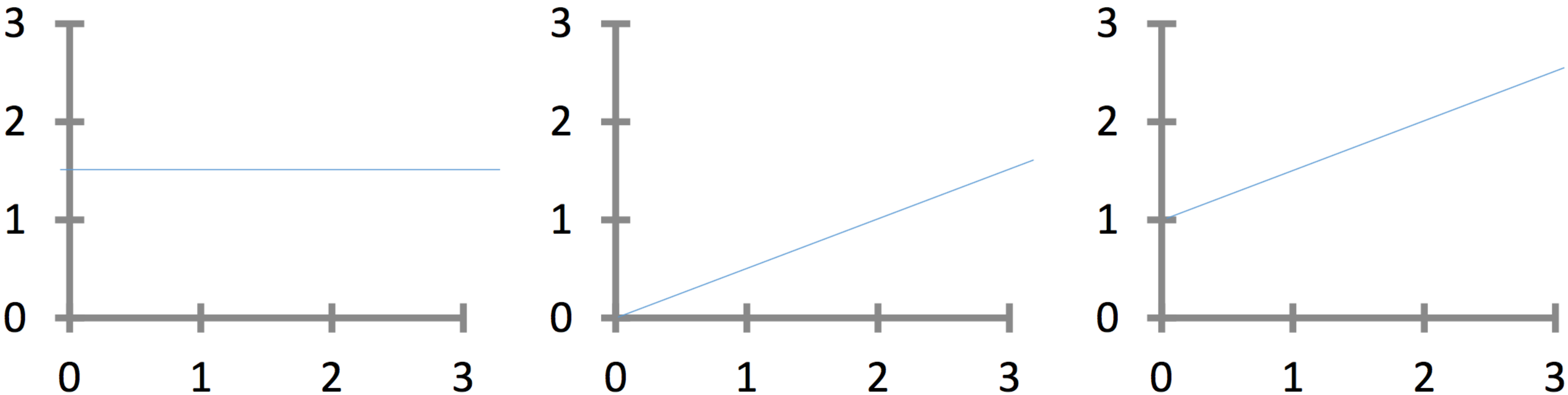
Size in feet^2 (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
…	…

Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$

θ_i's: Parameters

How to choose θ_i's ?

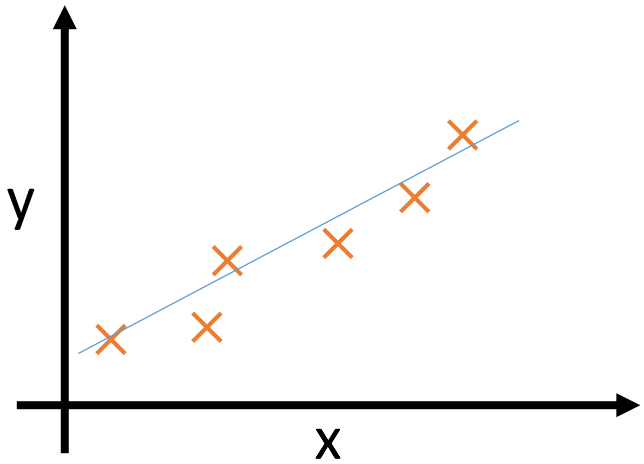
$$h_\theta(x) = \theta_0 + \theta_1 x$$



$\theta_0 = 1.5$
$\theta_1 = 0$

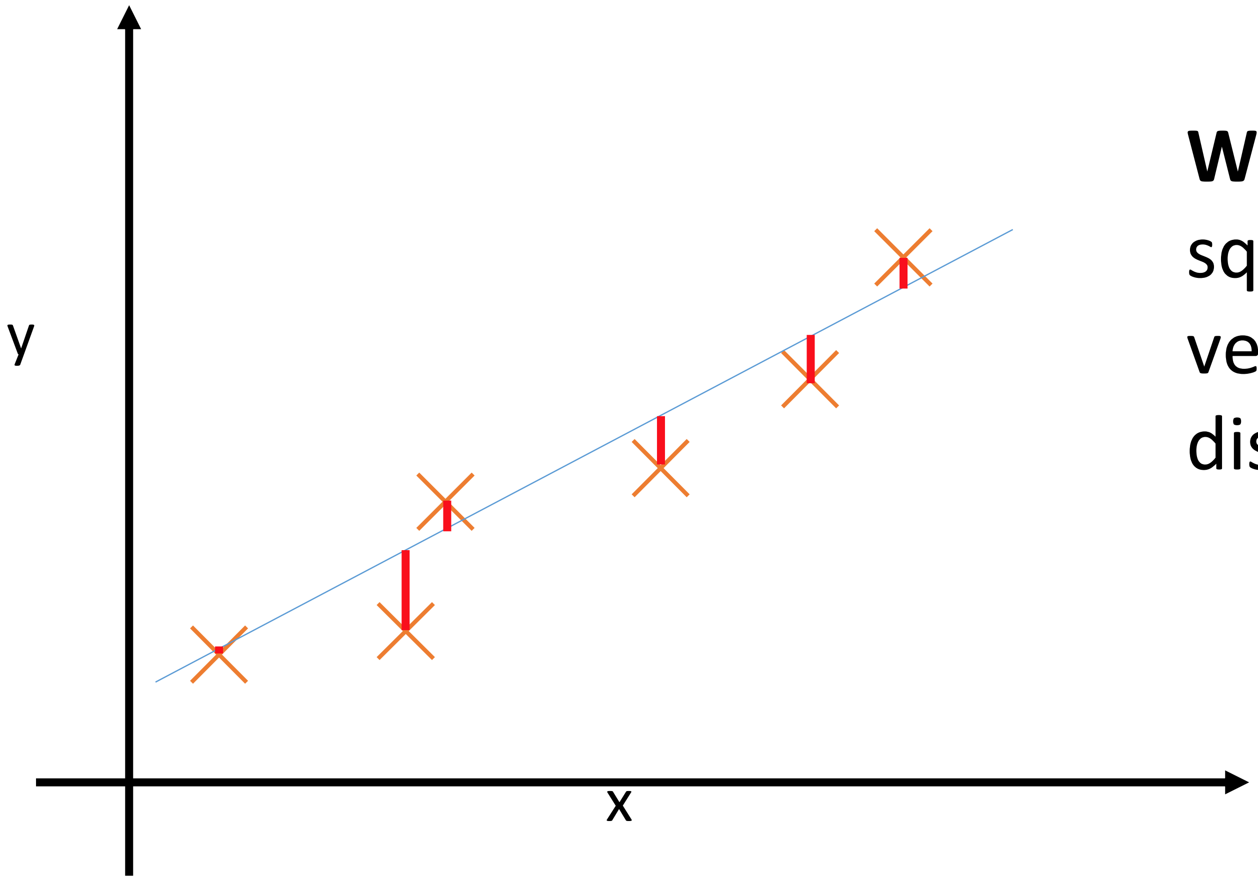
$\theta_0 = 0$
$\theta_1 = 0.5$

$\theta_0 = 1$
$\theta_1 = 0.5$



But what does "close" mean?

Idea: Choose θ_0, θ_1 so that $h_\theta(x)$ is close to y for our training examples (x, y)



We choose:
squared
vertical
distance

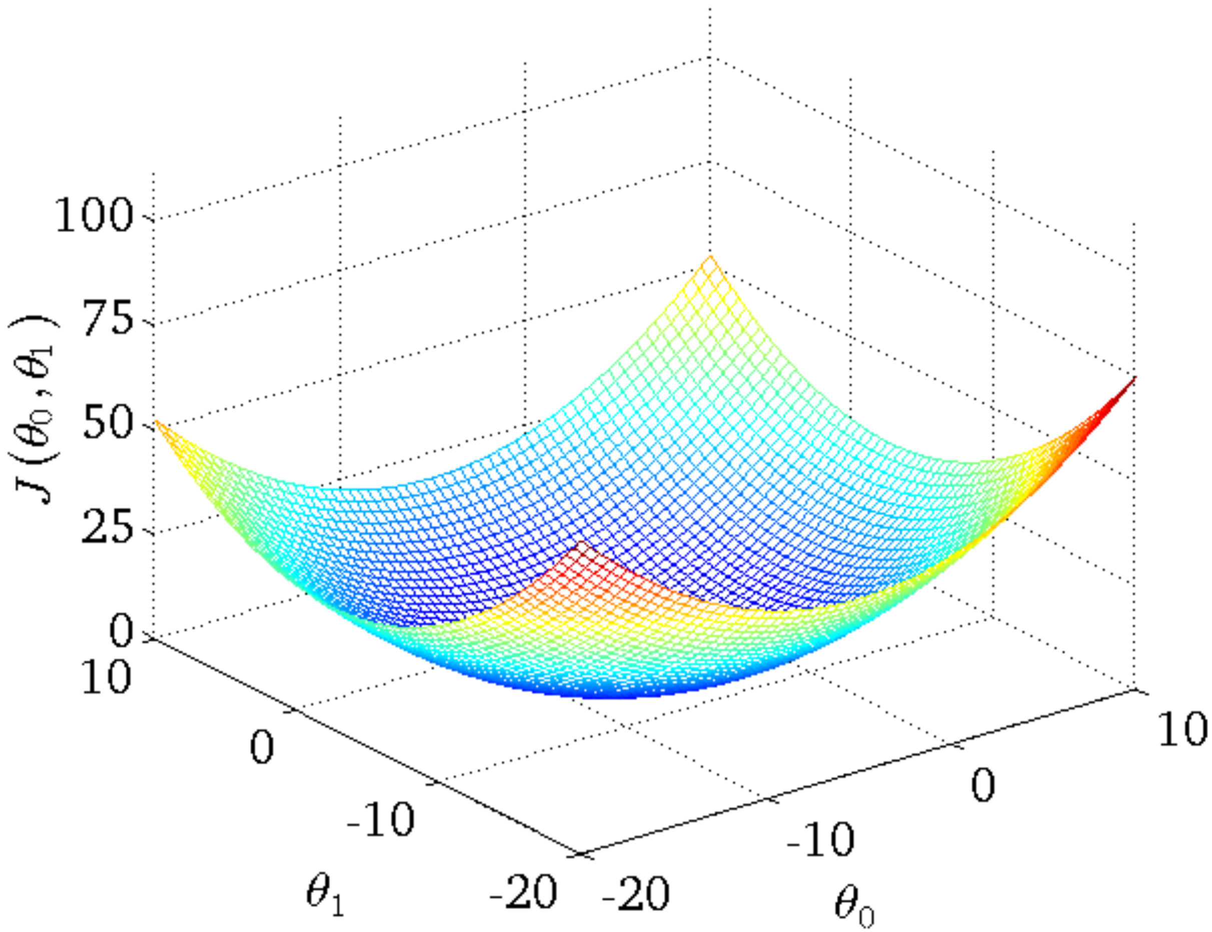
Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$

Parameters: θ_0, θ_1

Cost Function: $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$

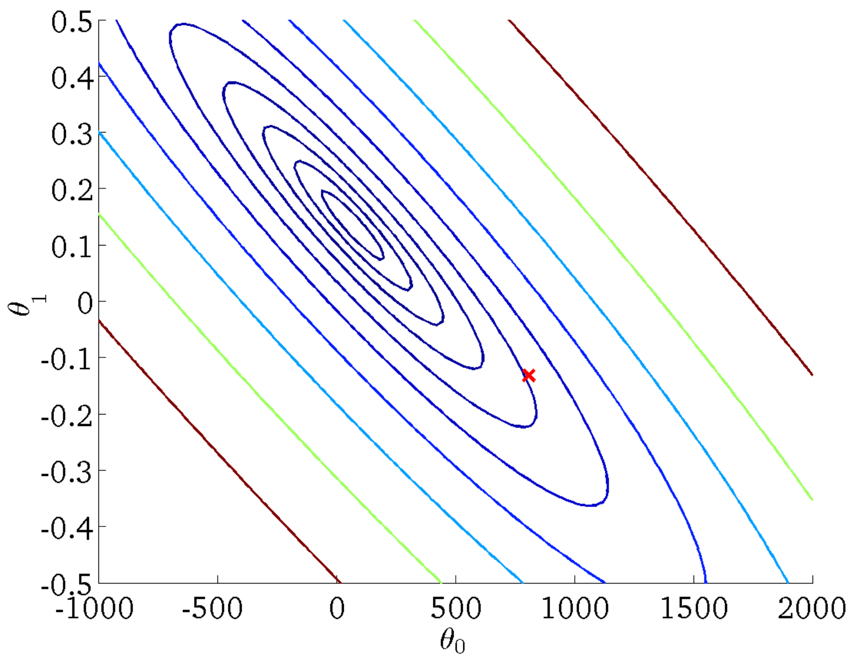
Goal: $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

Cost Function Surface Plot



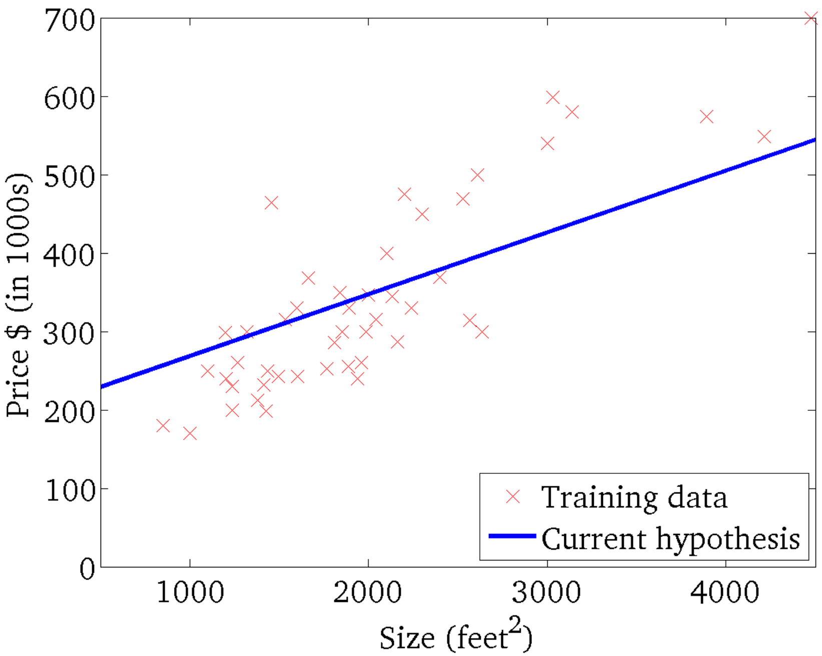
Contour Plots

- For a function F(x, y) of two variables, assigned different colours to different values of F
- Pick some values to plot
- The result will be *contours* – curves in the graph along which the values of F(x, y) are constant



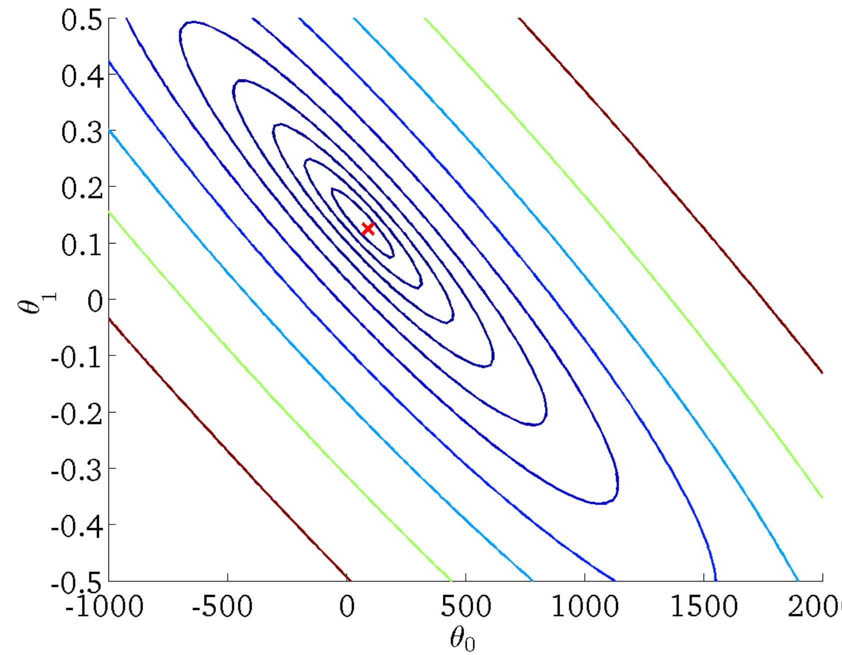
$$h_\theta(x)$$

(for fixed θ_0, θ_1, this is a function of x)



$$J(\theta_0, \theta_1)$$

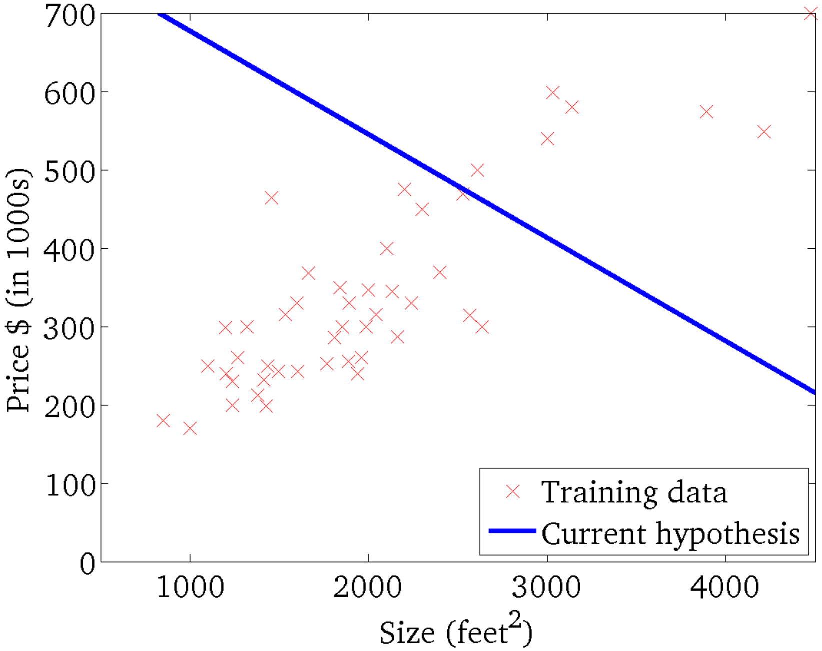
(function of the parameters θ_0, θ_1)



Cost Function Contour Plot

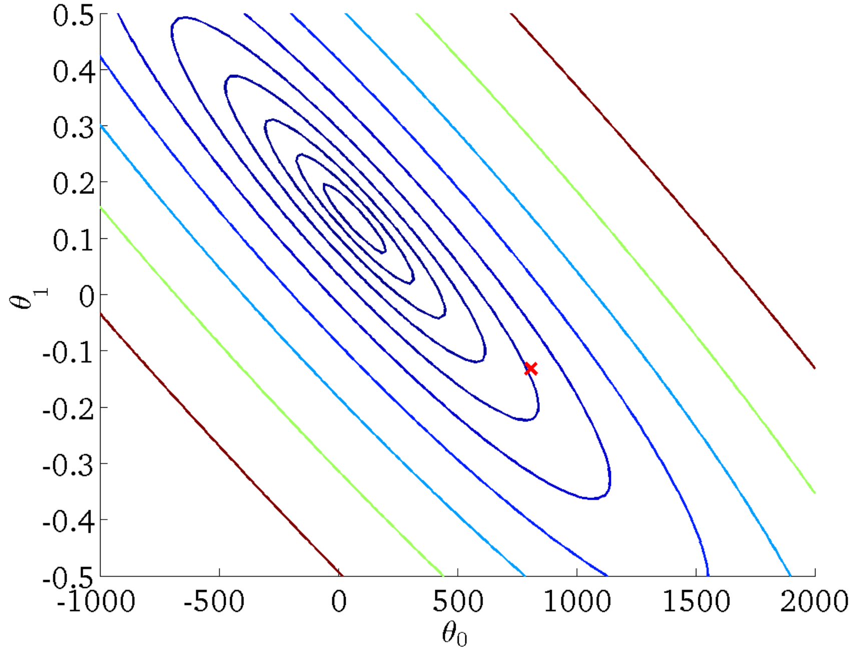
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



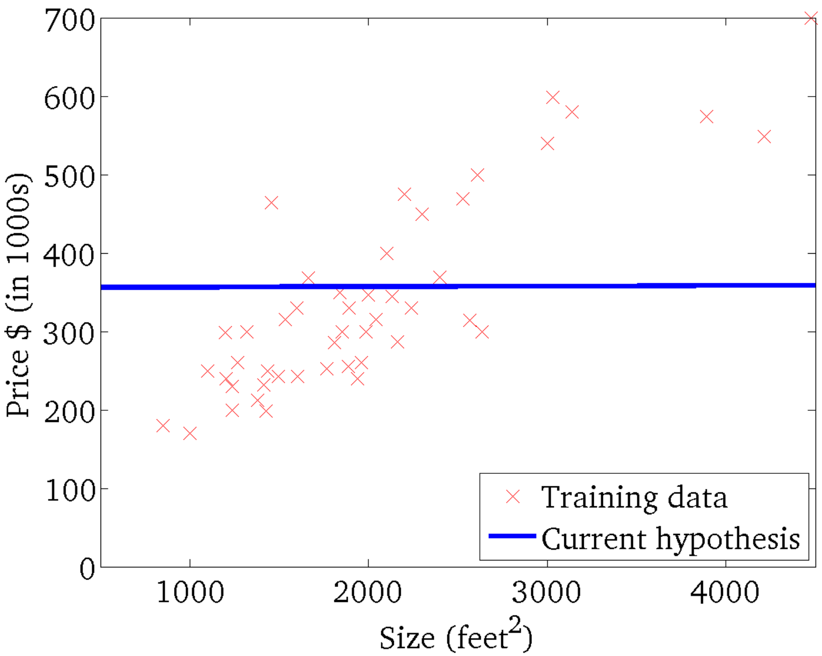
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



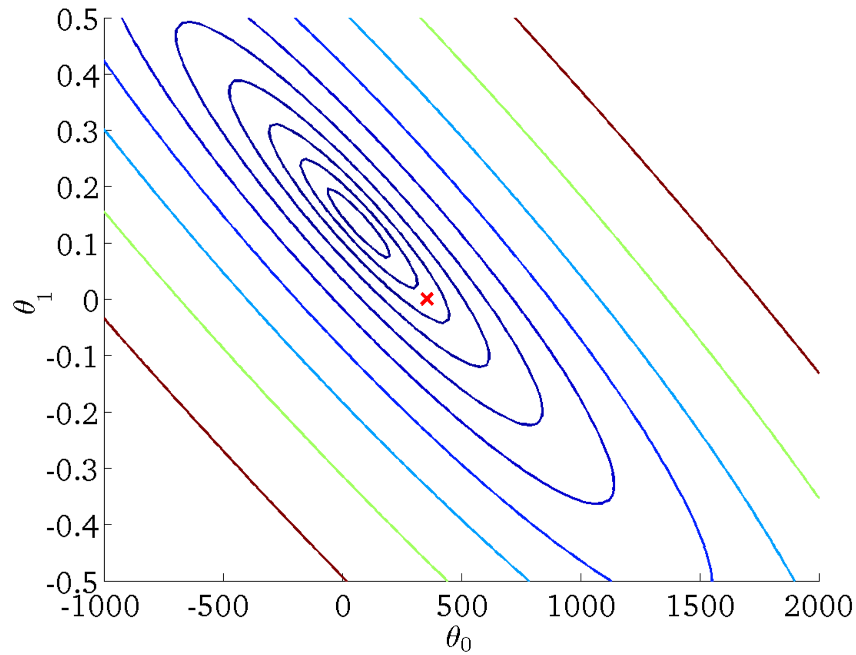
$$h_\theta(x)$$

(for fixed θ_0, θ_1, this is a function of x)



$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



Have some function $J(\theta_0, \theta_1)$

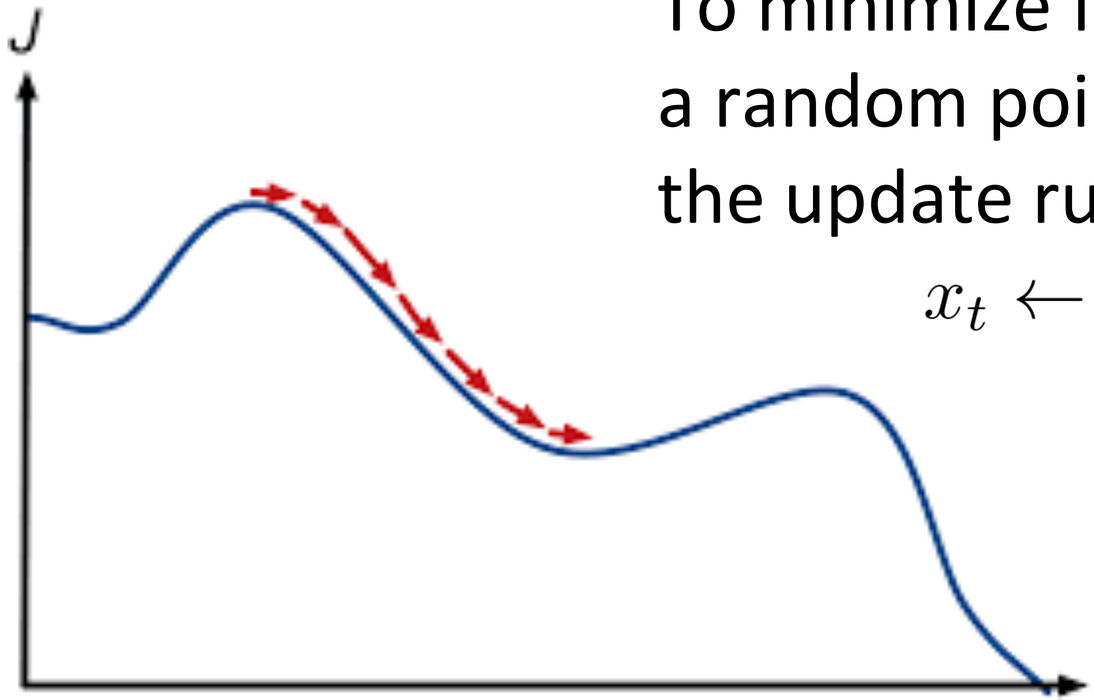
Want $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$

Outline:

- Start with some θ_0, θ_1
- Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$

  until we hopefully end up at a minimum

Gradient Descent in 1D

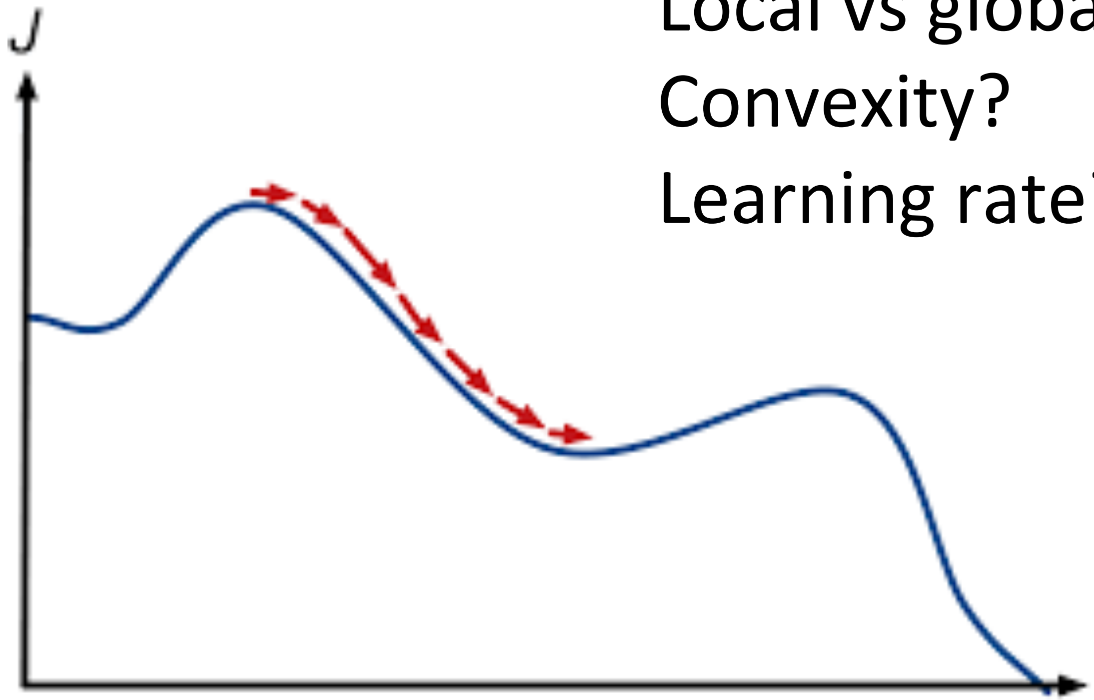


To minimize f(x), we start with a random point and iterate with the update rule:

$$x_t \leftarrow x_{t-1} - \alpha \frac{df}{dx}(x_{t-1})$$

Image Credit: http://www.deepideas.net/deep-learning-from-scratch-iv-gradient-descent-and-backpropagation/

Things to consider:



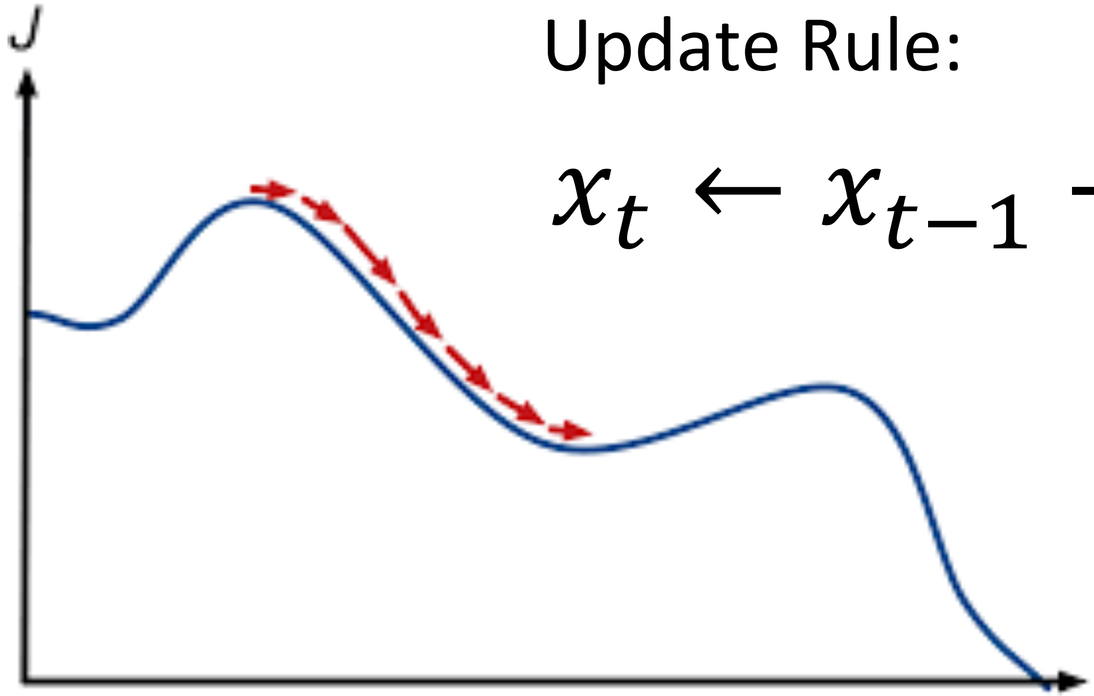
Local vs global minima?

Convexity?

Learning rate? (α)

Image Credit: http://www.deepideas.net/deep-learning-from-scratch-iv-gradient-descent-and-backpropagation/

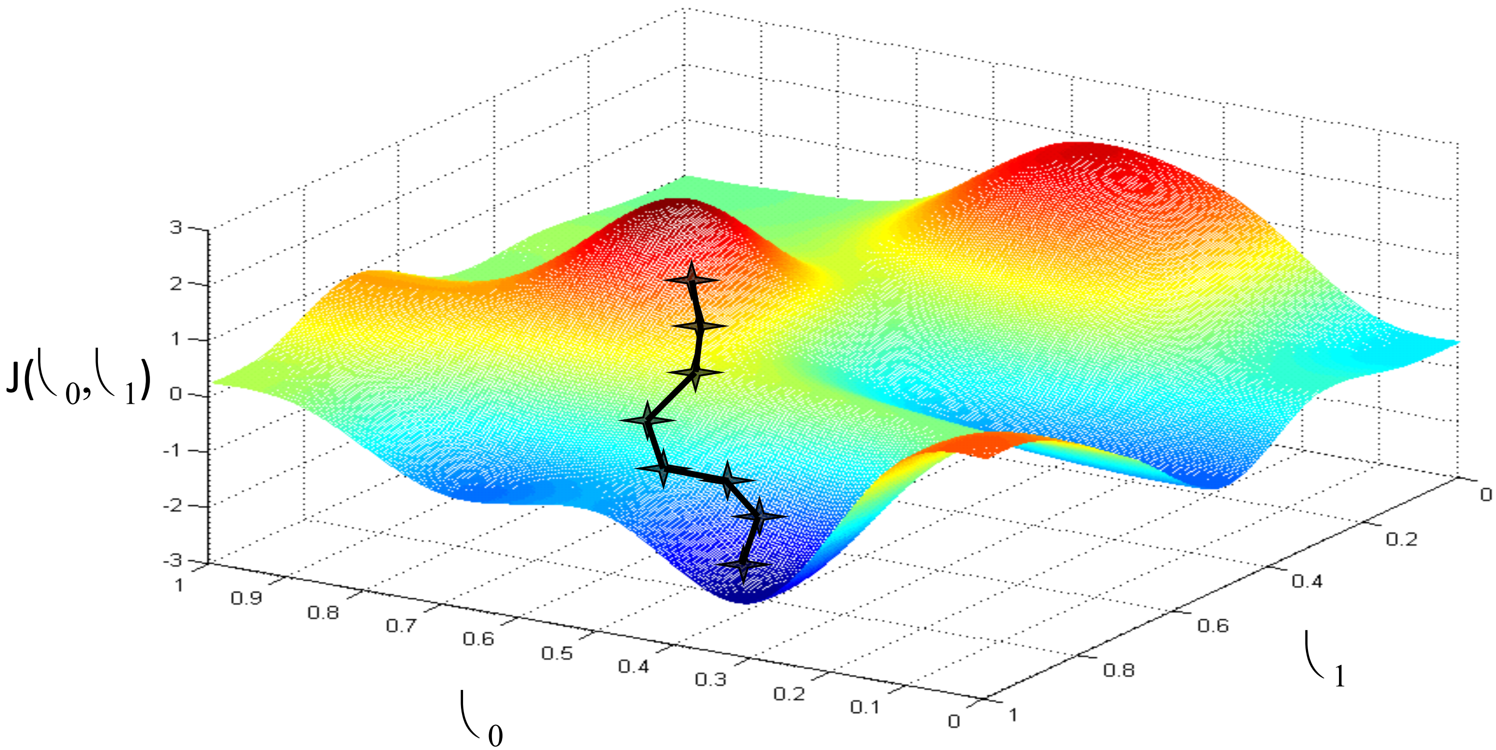
Gradient Descent in Higher Dimensions



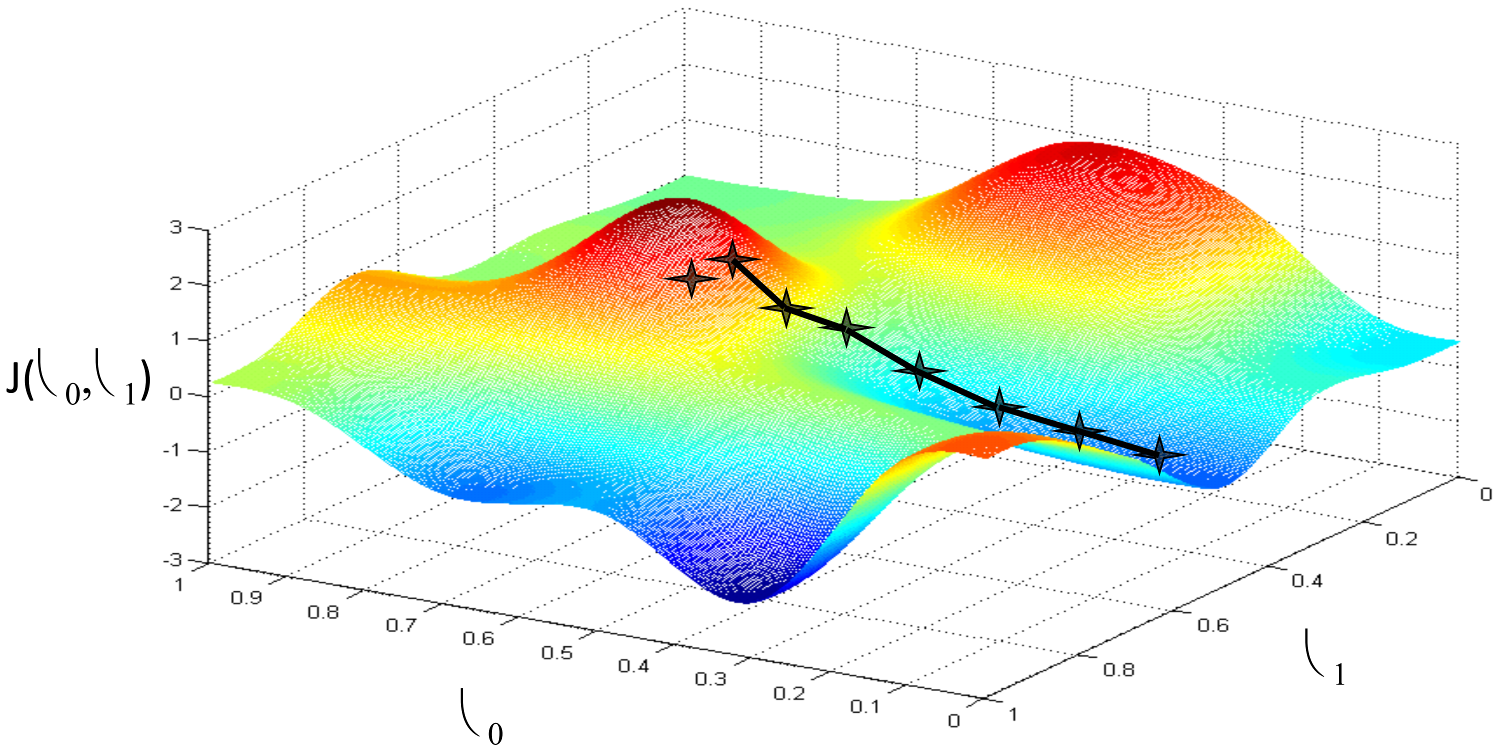
Update Rule:

$$x_t \leftarrow x_{t-1} - \alpha \nabla f(x_{t-1})$$

Image Credit: http://www.deepideas.net/deep-learning-from-scratch-iv-gradient-descent-and-backpropagation/

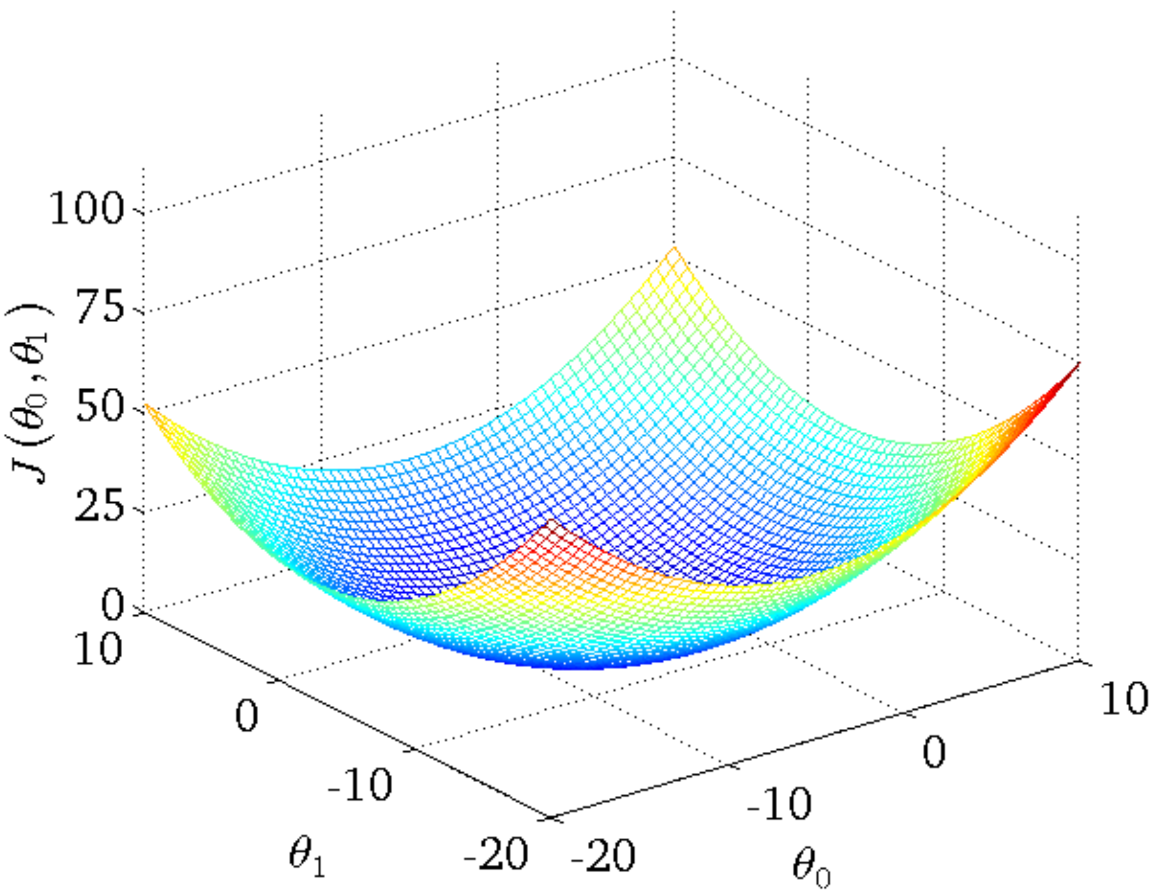


J(θ_0,θ_1)
3
2
1
0
-1
-2
-3
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
θ_0
0
0.2
0.4
0.6
0.8
1
θ_1



3
2
1
0
-1
-2
-3
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.2
0.4
0.6
0.8
1

For Linear Regression, J is bowl-shaped (“convex”)



Gradient Descent Example

Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$

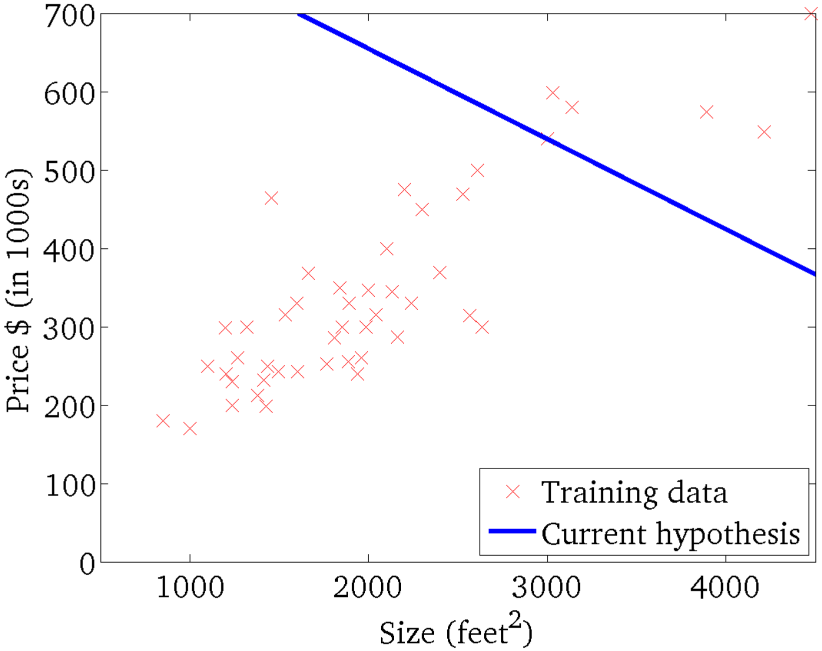
Parameters: θ_0, θ_1

Cost Function: $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$

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

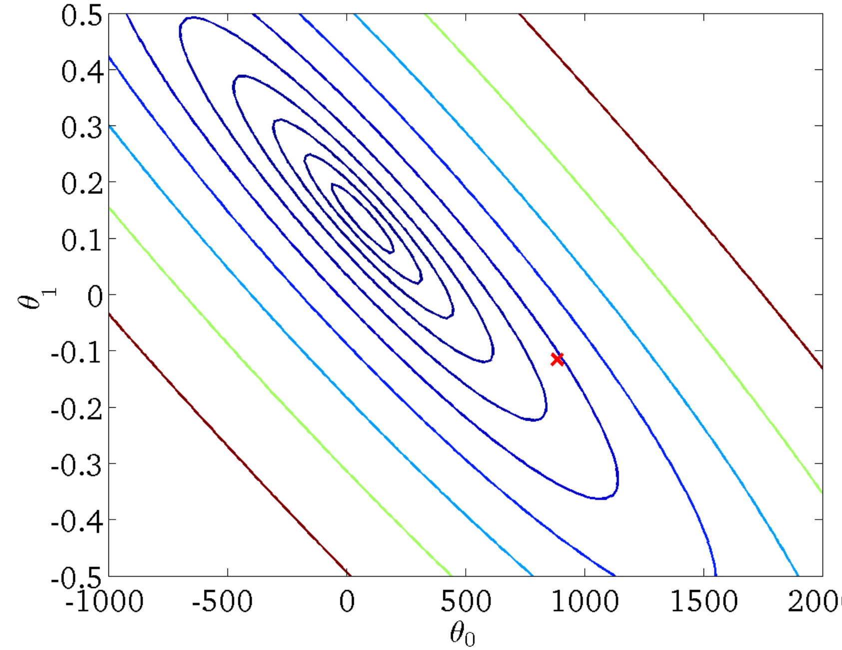
$$h_\theta(x)$$

(for fixed θ_0, θ_1, this is a function of x)



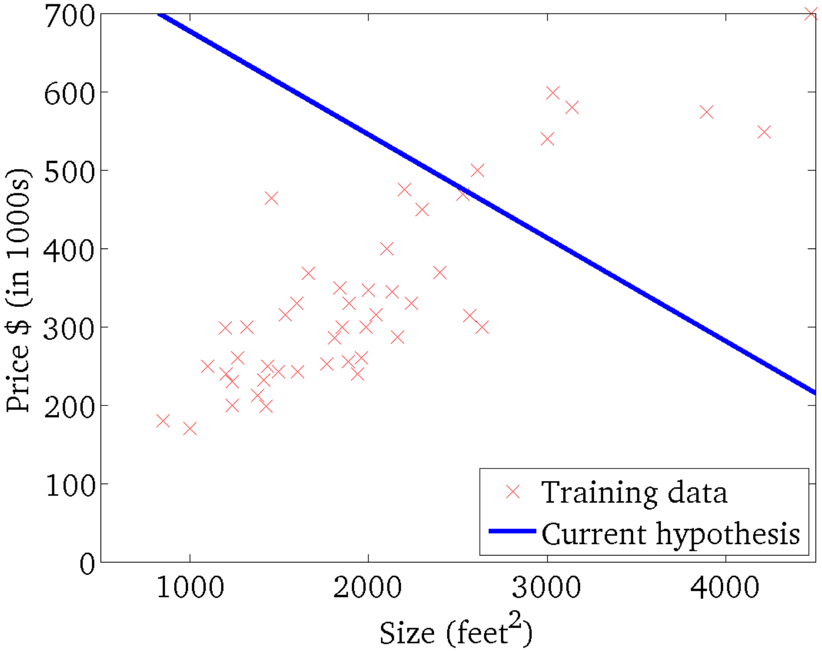
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



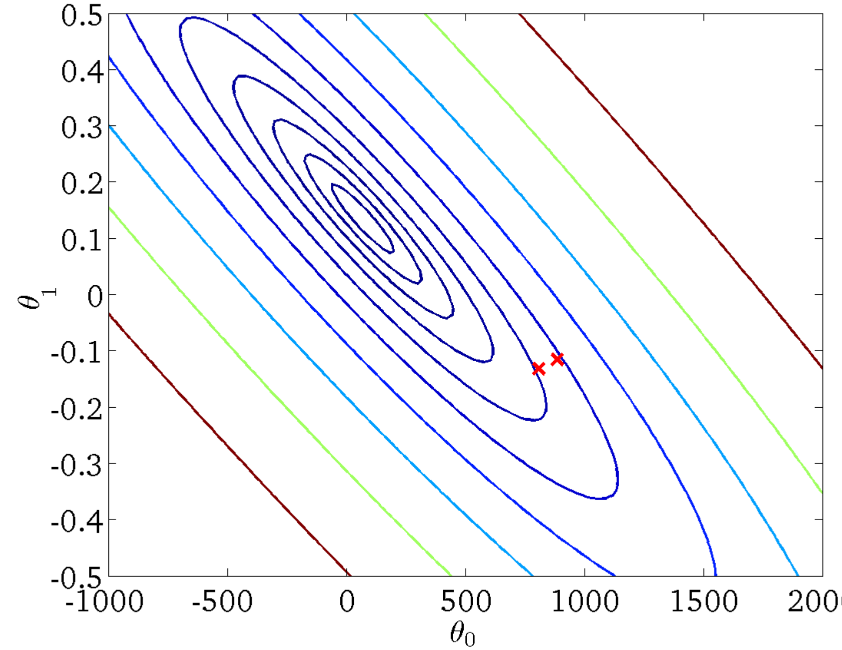
$$h_\theta(x)$$

(for fixed θ_0, θ_1, this is a function of x)



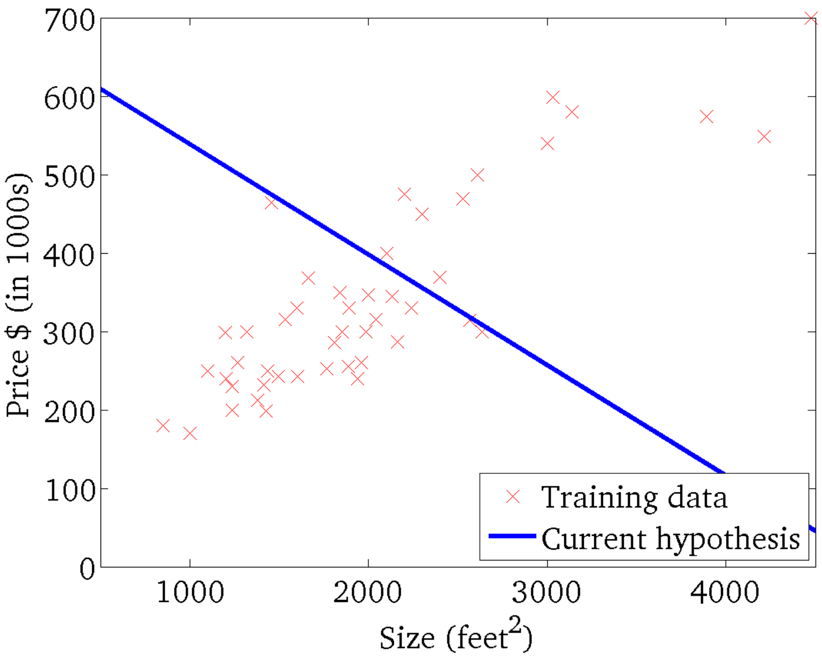
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



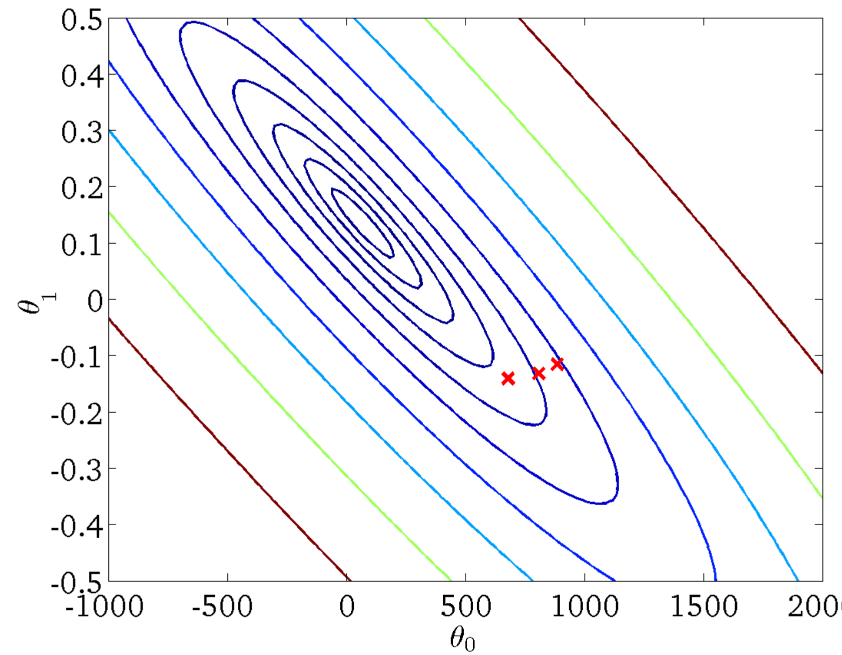
$$h_\theta(x)$$

(for fixed θ_0, θ_1, this is a function of x)



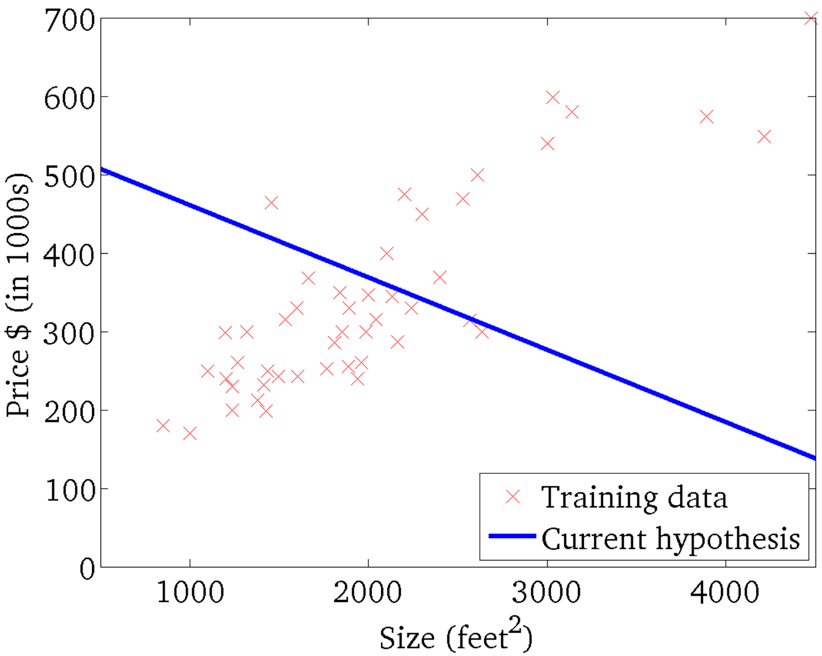
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



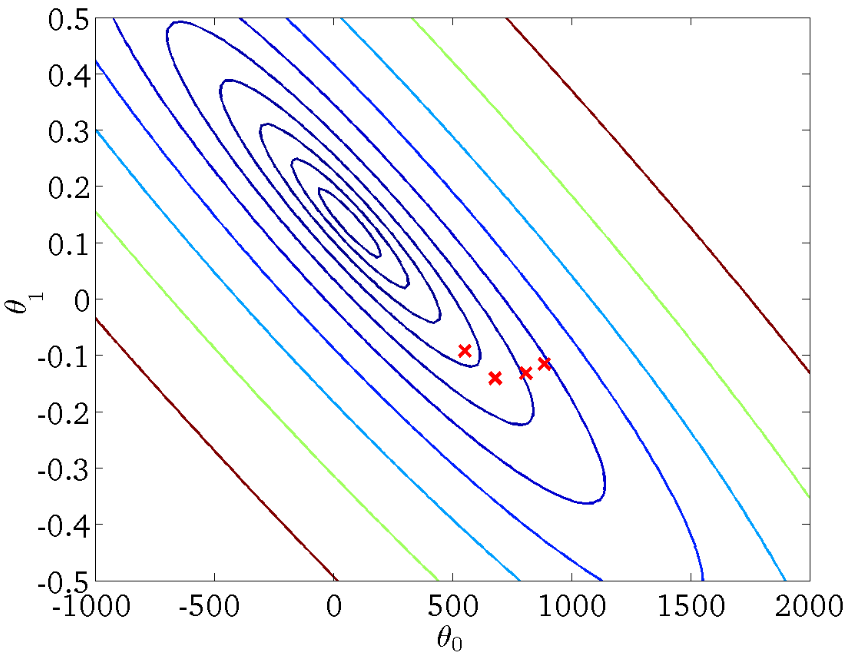
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



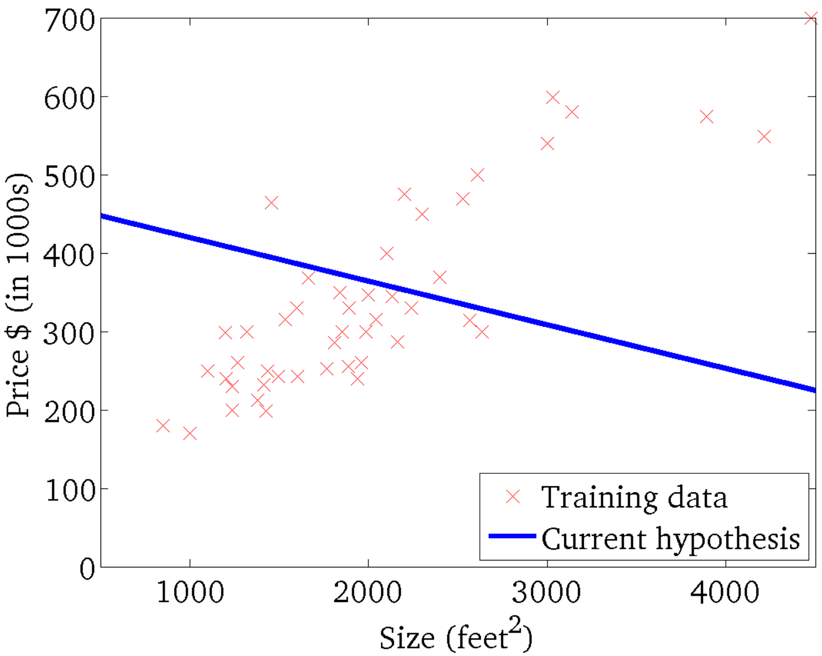
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



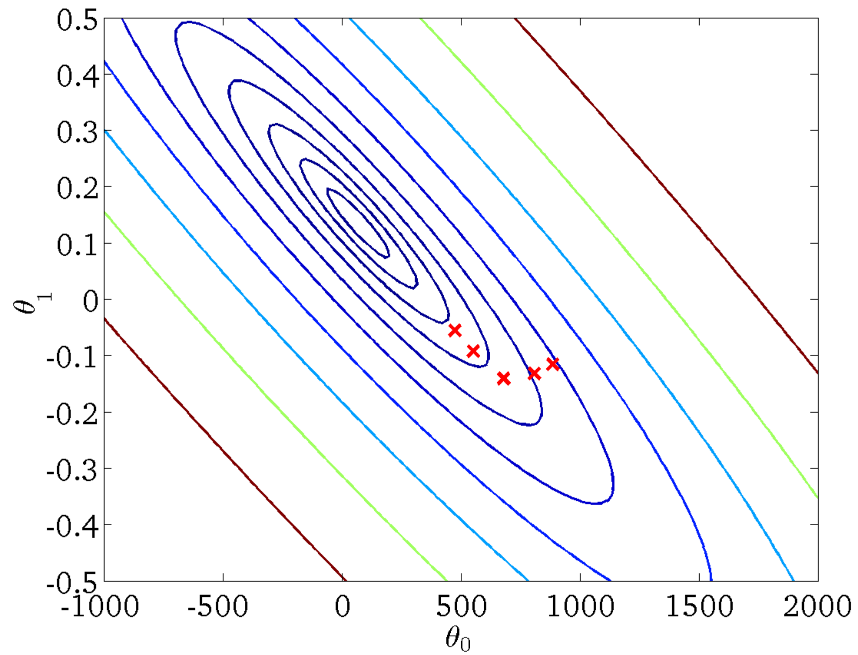
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



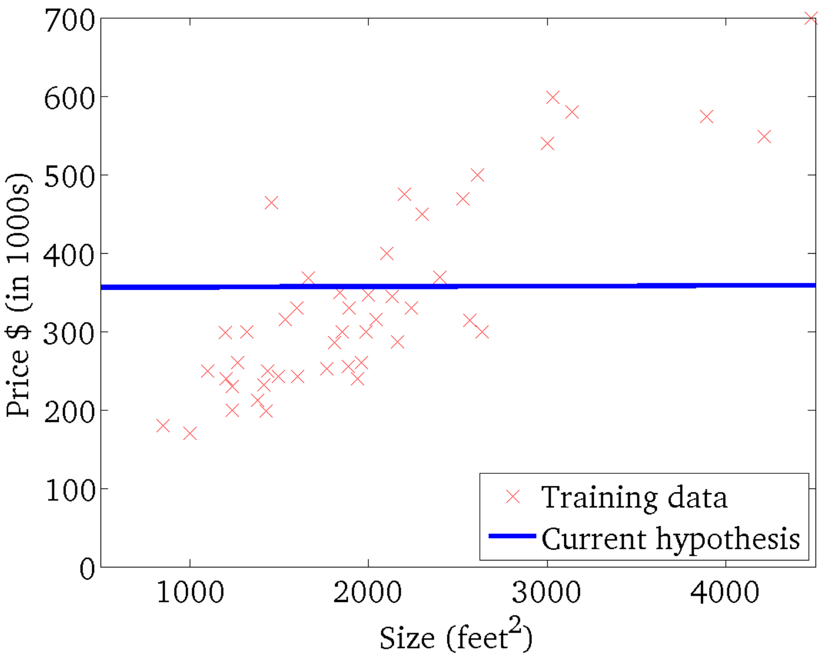
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



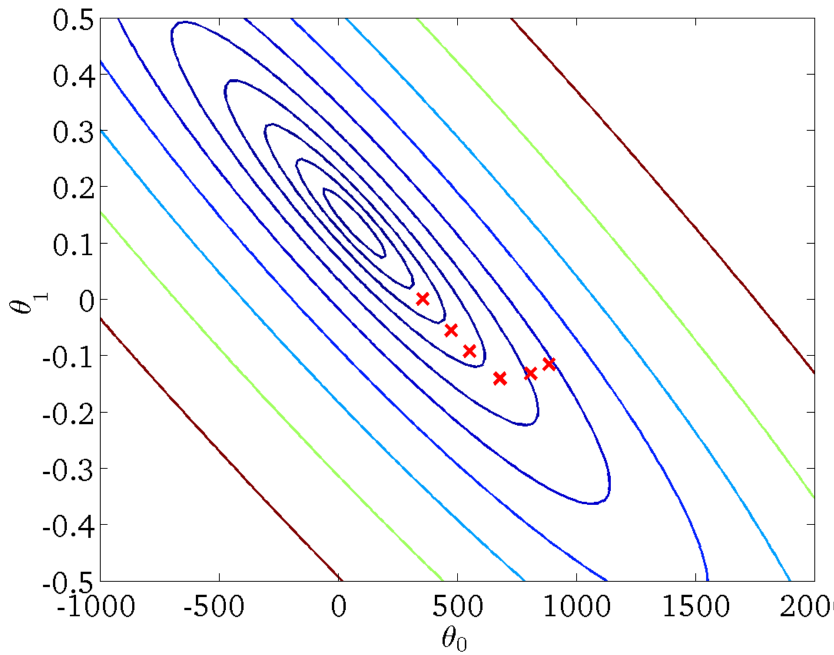
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



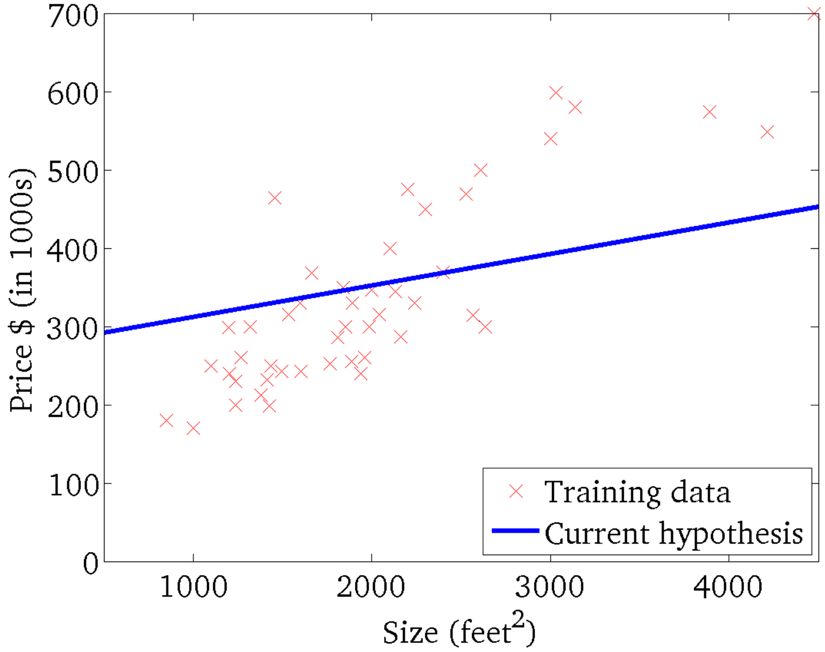
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



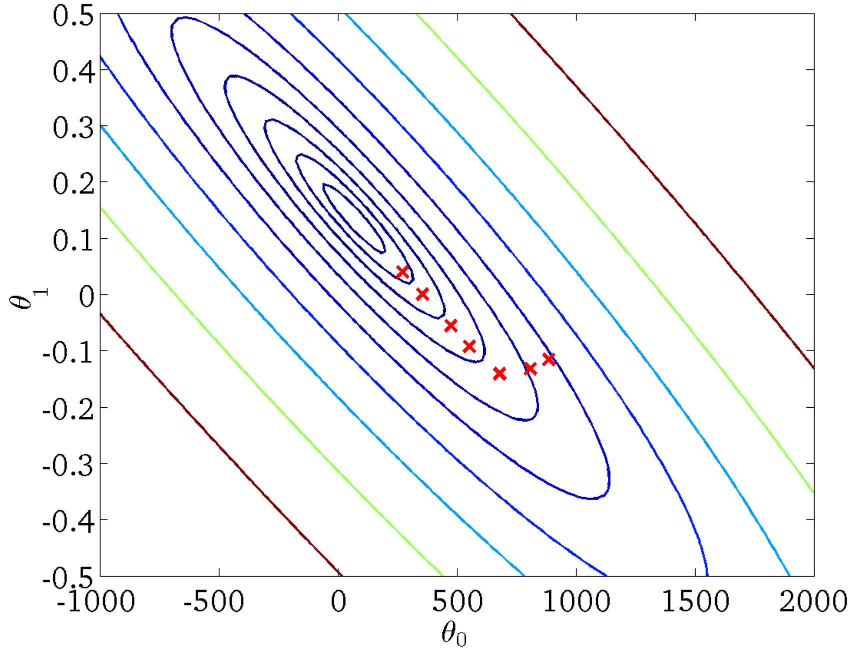
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



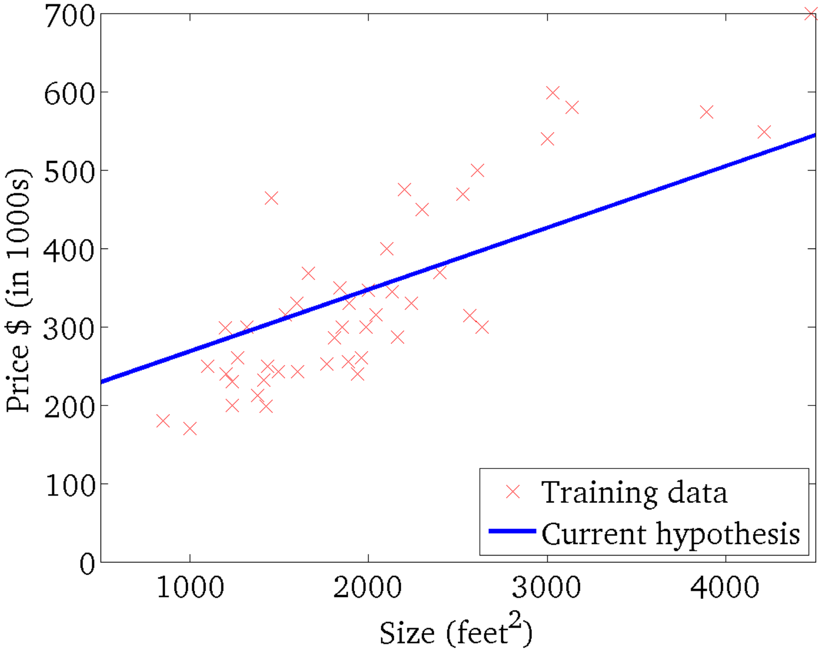
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



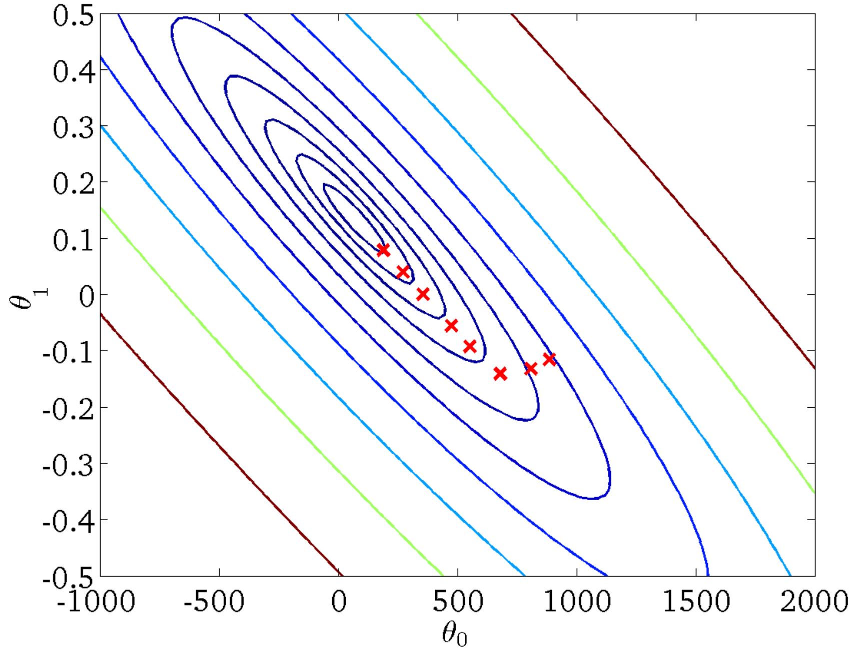
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



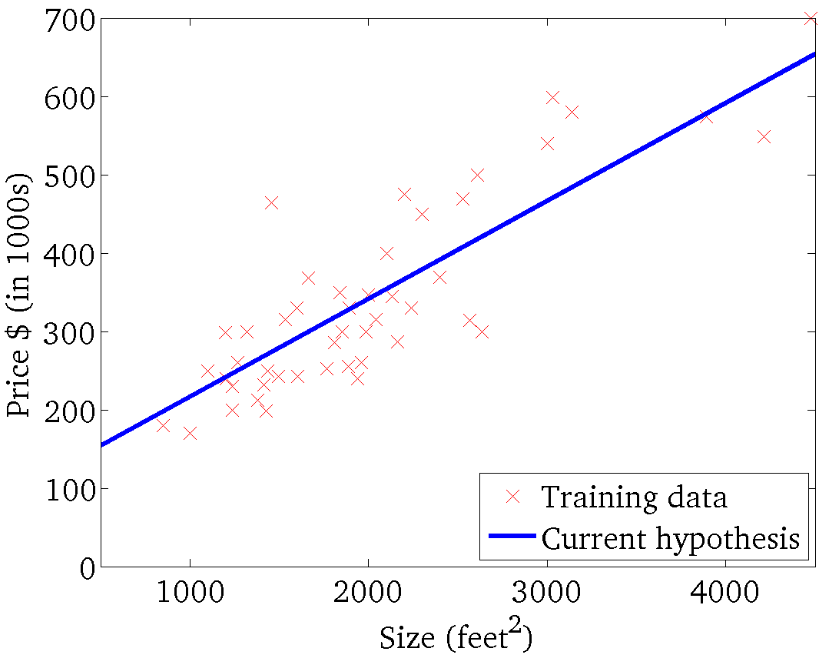
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



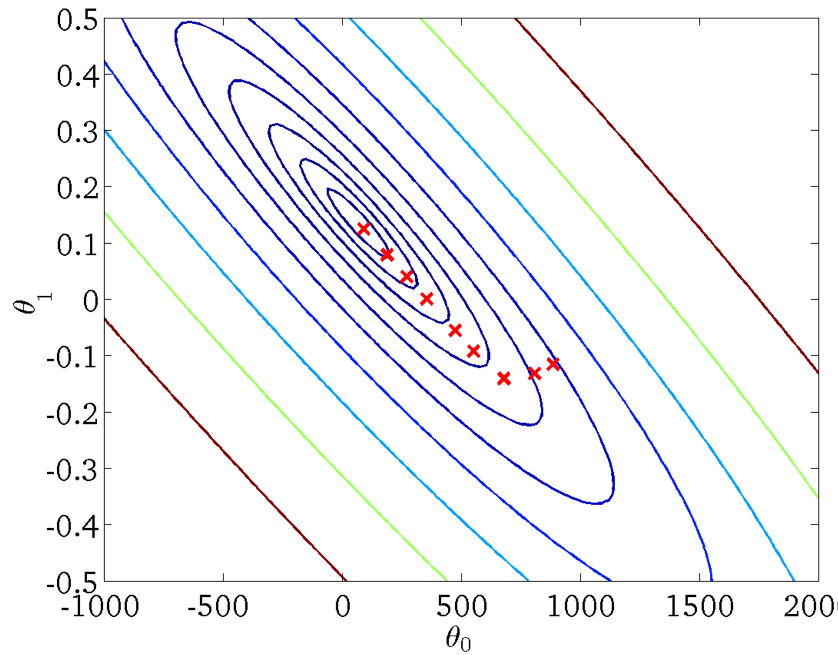
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)

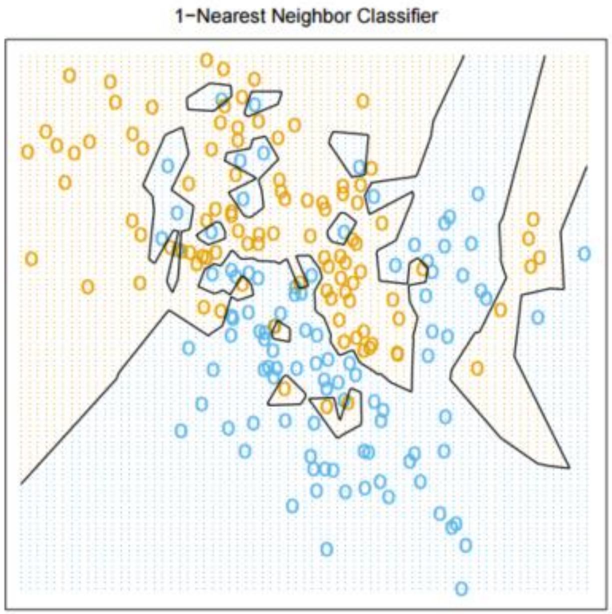


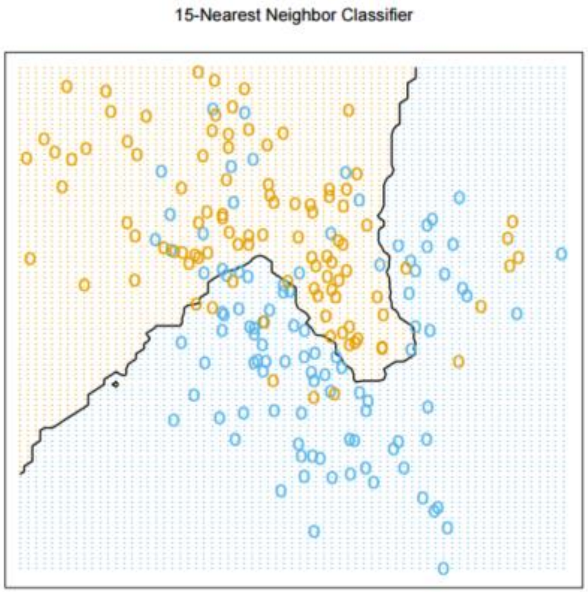
$J(\theta_0, \theta_1)$

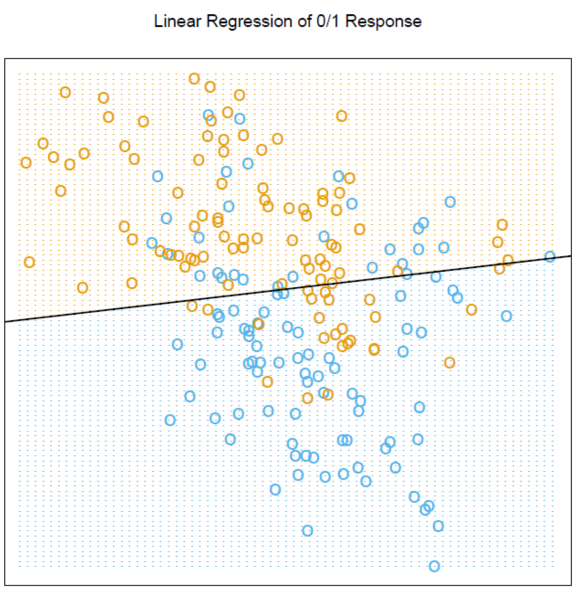
(function of the parameters θ_0, θ_1)



Linear Regression vs. k-Nearest Neighbours







Orange: y = 1
Blue: y = 0

Linear Regression vs. k-Nearest Neighbours

- Linear Regression: the boundary can only be linear
- Nearest Neighbours: the boundary can more complex
- Which is better?
  - Depends on what the *actual boundary* looks like
  - Depends on whether we have enough data to figure out the *correct* complex boundary