AIFA: Stochastic Planning MDP [Policy Iteration]

09/04/2024

Koustav Rudra

Policy Iteration

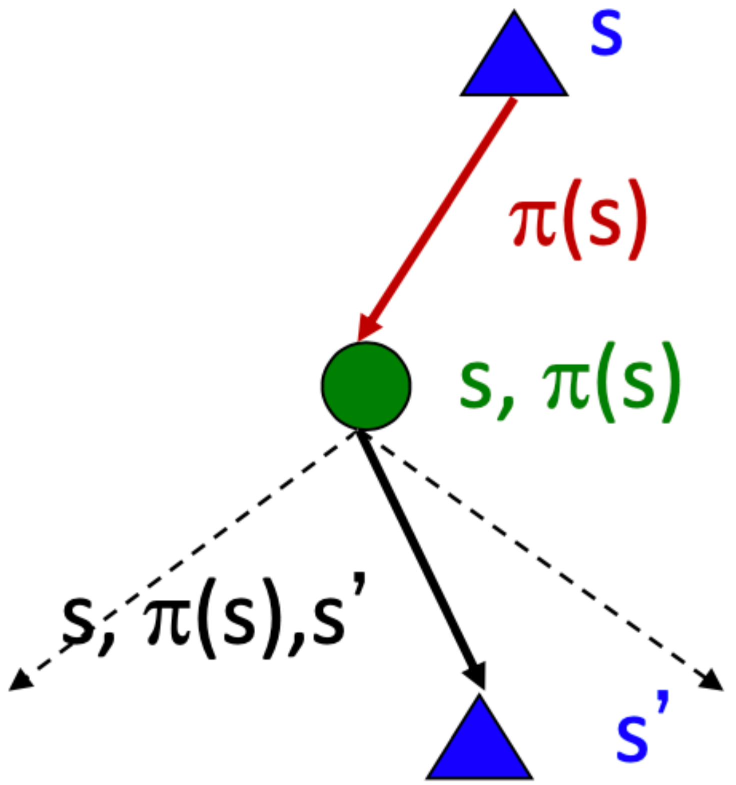
- Alternative approach for optimal values:
  - Step 1: Policy Evaluation:
    - calculate utilities for some fixed policy (not optimal utilities!) until convergence
  - Step 2: Policy Improvement:
    - update policy using one-step look-ahead with resulting converged (but not optimal!) utilities as future values
    - Repeat steps until policy converges

Policy Evaluation

- How do we calculate the V's for a fixed policy π?
- Idea 1: Turn recursive Bellman equations into updates

$$V_0^\pi(s) = 0$$

$$V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V_k^\pi(s')]$$



Policy Iteration

- Evaluation: For fixed current policy π, find values with policy evaluation:
    - Iterate until values converge:

$$V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} T(s, \pi_i(s), s') \left[R(s, \pi_i(s), s') + \gamma V_k^{\pi_i}(s')\right]$$

- Improvement: For fixed values, get a better policy using policy extraction
    - One-step look-ahead:

$$\pi_{i+1}(s) = \arg\max_a \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V^{\pi_i}(s')\right]$$

Comparison

- Both value iteration and policy iteration compute the same thing (all optimal values)

- In value iteration:
  - Every iteration updates both the values and (implicitly) the policy
  - We don't track the policy, but taking the max over actions implicitly recomputes it

- In policy iteration:
  - We do several passes that update utilities with fixed policy (each pass is fast because we consider only one action, not all of them)
  - After the policy is evaluated, a new policy is chosen (slow like a value iteration pass)
  - The new policy will be better (or we're done)

- Both are dynamic programs for solving MDPs

Introduction to Learning

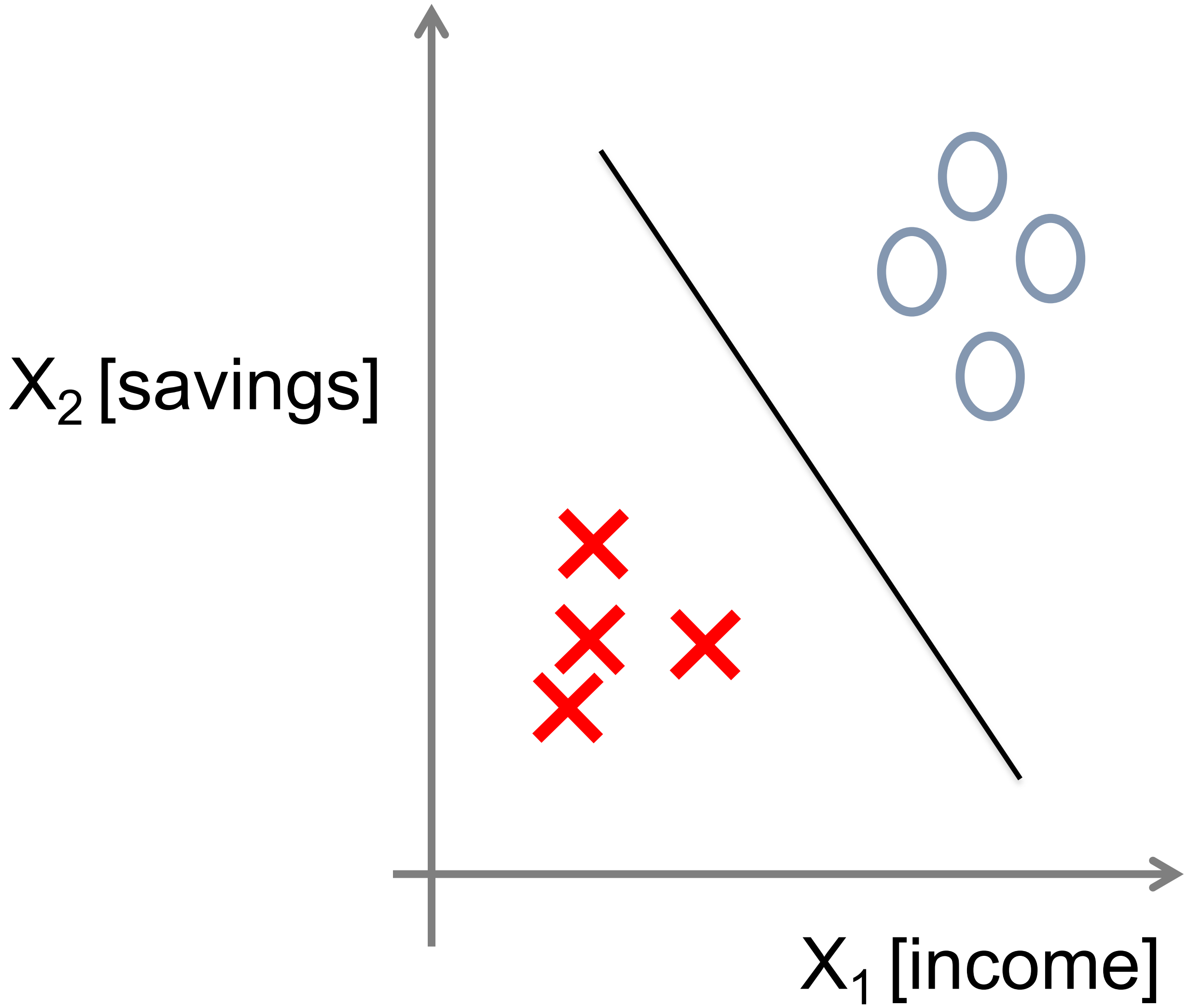
09/04/2024

Koustav Rudra

Paradigms of Learning

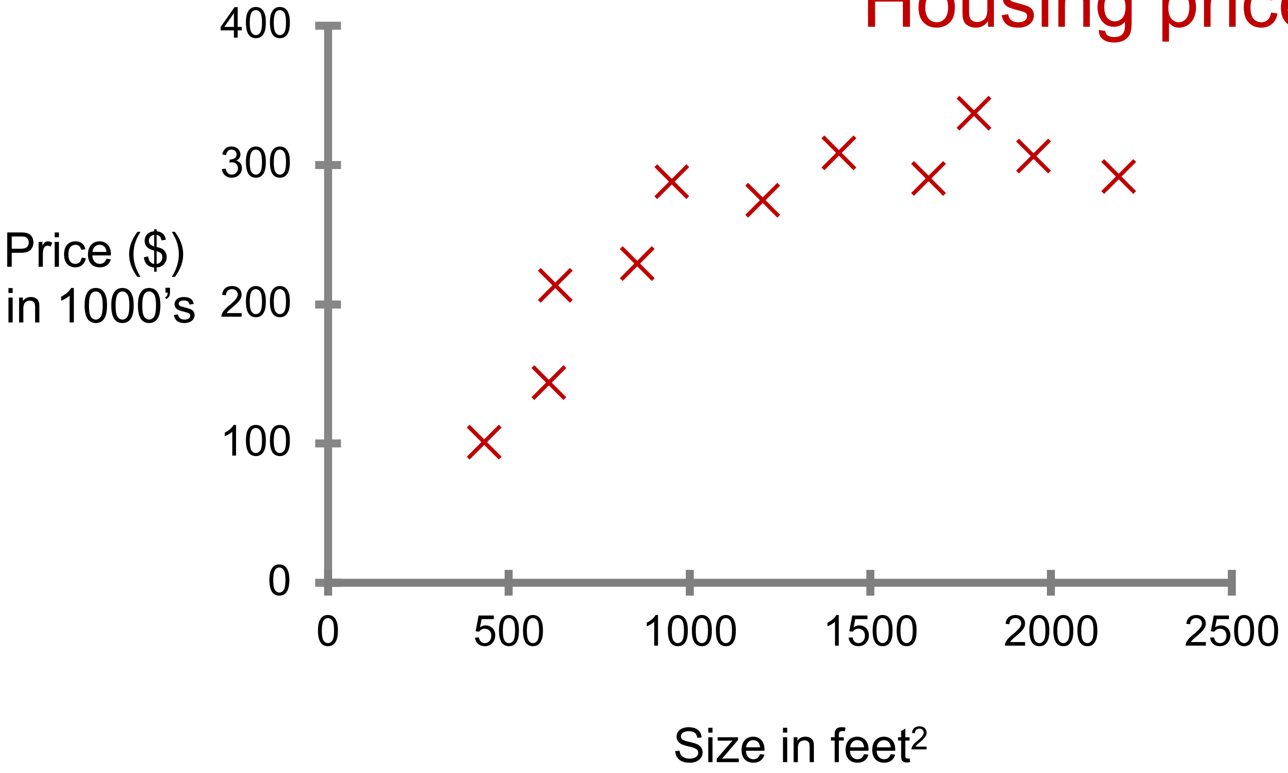
- **Supervised Learning**
  - Both inputs and outputs are given
  - The outputs are provided by experts
- **Reinforcement Learning**
  - The agent receives some evaluation of actions
  - E.g., fine for giving a wrong chess move
- **Unsupervised Learning**
  - No Label, no feedback
  - Have to learn pattern from the inputs

Supervised Learning: Classification



Supervised Learning: Regression

Housing price prediction



<u>Supervised Learning</u>

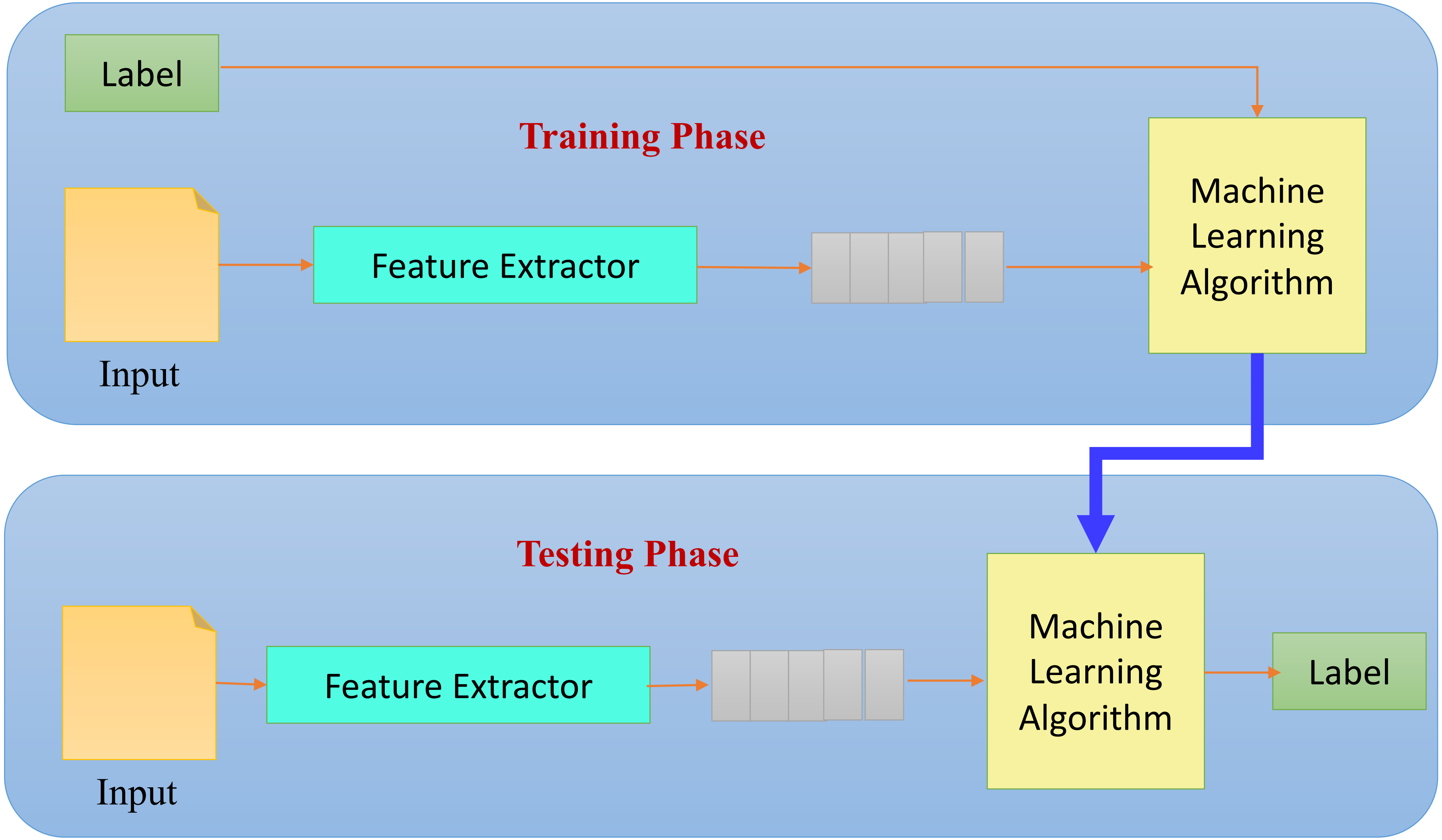
"right answers" given

<u>Regression:</u> Predict continuous valued output (price)

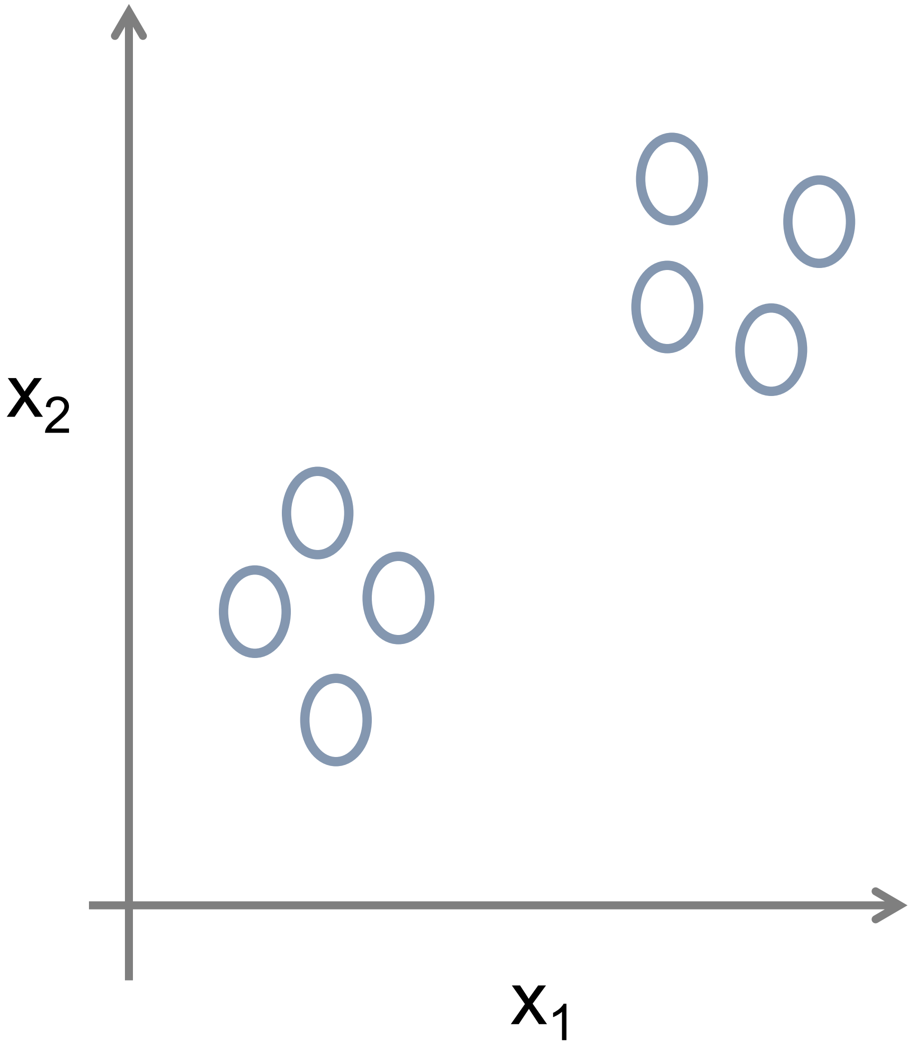
Features

- Often the individual observations are analyzed into a set of quantifiable properties which are called features
  - categorical - well-defined finite set of values
    - nominal (e.g. A, B, AB or O, for blood types)
    - ordinal (e.g. large, medium, small)
    - dichotomous (e.g. male, female)
  - integer-valued (e.g. the number of words in a text)
  - real-valued (e.g. height)

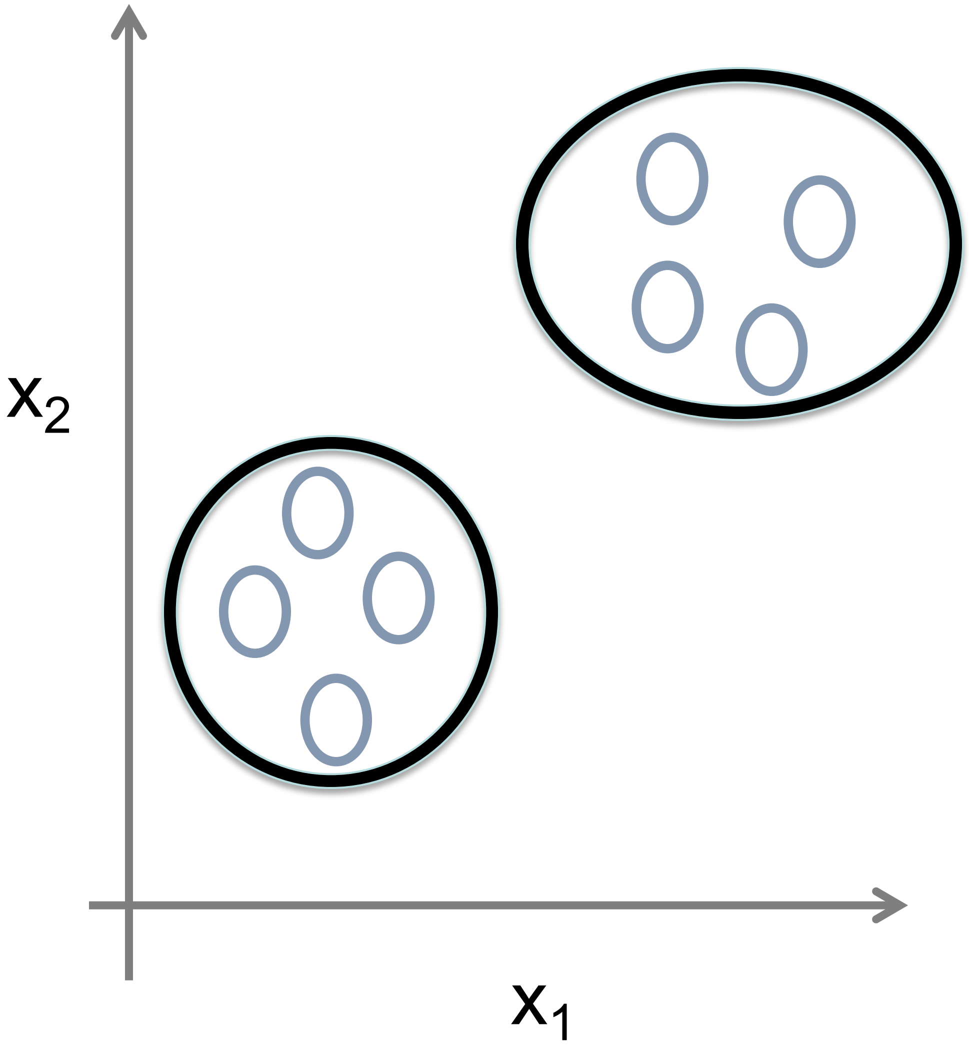
Supervised Learning: Classification



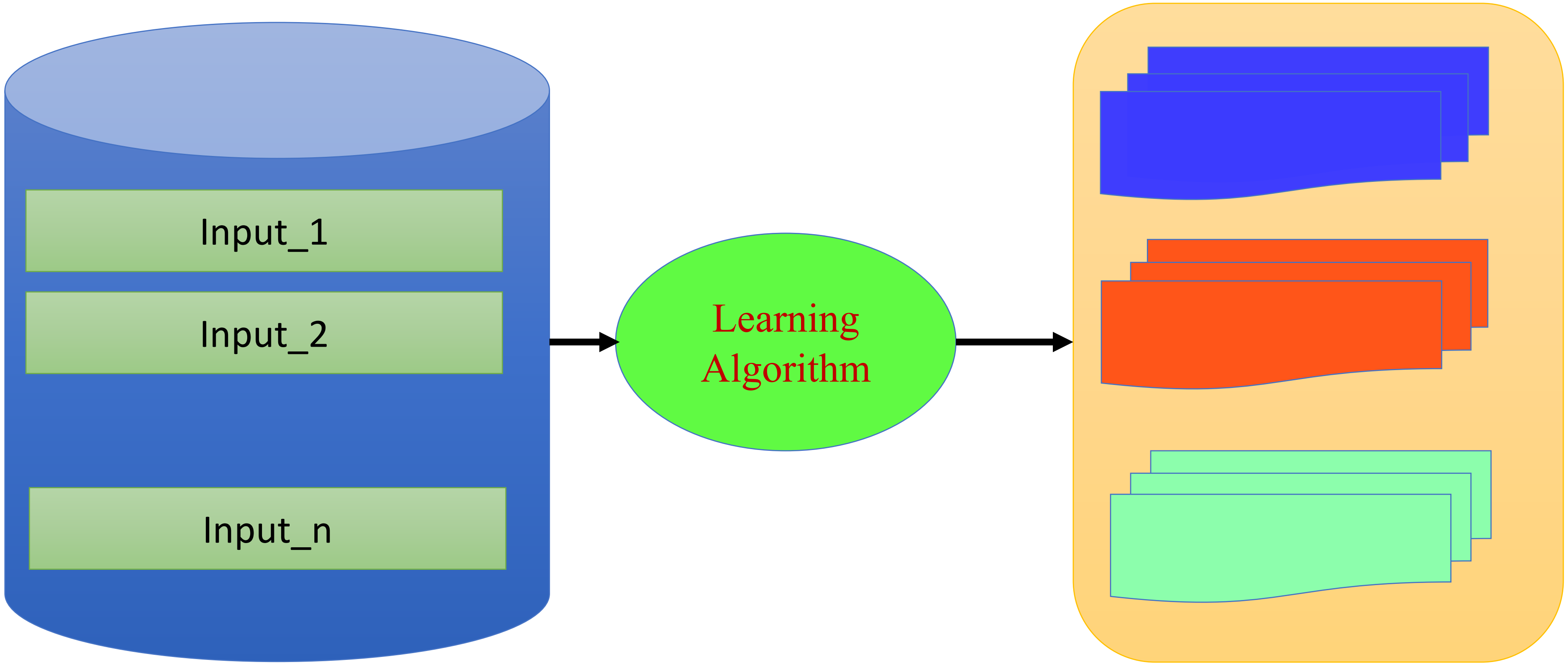
Unsupervised Learning: Clustering



Unsupervised Learning: Clustering



Unsupervised Learning



Terminology

- **Features:** The number of distinct traits that can be used to describe each item in a quantitative manner
- **Feature vector:** n-dimensional vector of numerical features that represent some object
- **Instance Space X:** Set of all possible objects describable by features
- **Example (x,y):** Instance x with label y=f(x)

Concept Learning

Concept Learning Task

- **Target concept:** "days on which Jack should enjoy sports"

	Sky	AirTemp	Humidity	Wind	Water	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Yes
2	Sunny	Warm	High	Strong	Warm	Yes
3	Rainy	Cold	High	Strong	Warm	No
4	Sunny	Warm	High	Strong	Cool	Yes

Concept Learning Task: Example

- X = {1, 2, 3, 4}
  - Set of all possible days
- **Features:** <Sky, AirTemp, Humidity, Wind, Water>
- Target concept (c) is to be learned
  - c(x)=1 if EnjoySport = Yes
  - c(x)=0 if EnjoySport = No
  - c : EnjoySport: X $\rightarrow$ {0, 1}
- **Training data D:** {(1,1), (2,1), (3,0), (4,1)}

Hypothesis Space and Inductive Learning

Concept Learning

- Given examples of a data point D = {(x, c(x)}

- Find out a hypothesis h: X $\rightarrow$ {0, 1}
  - h basically approximates c

- Hypothesis Space: $H = \{h_1, h_2, h_3, \ldots, h_n\}$

- **Objective:**
  - Find out a hypothesis h in H such that $h(x)=c(x)\ \forall x$

Hypothesis Space

- Set of all legal hypothesis defined by the chosen feature set and the chosen hypothesis language

- The space of all hypotheses that can, in principle, be output by a ***learning algorithm***

- One way to think about a supervised learning machine is as a device that explores a **"hypothesis space"**
  - Each setting of the parameters in the machine is a different hypothesis about the function that maps input vectors to output vectors

- Given a set of data points, hypothesis $h \in H$ is the **output of a learning algorithm**

Hypothesis Space: Example

- X = {1, 2, 3, 4}
  - Set of all possible days

- **Features:** <Sky, AirTemp, Humidity, Wind, Water>

- h_1 : AirTemp = "cold" and Humidity = "high"
- h_2 : Sky = "sunny" and Water = "cool"

Inductive Learning

- **Inductive learning:** Inducing a general function from training examples
  - Construct hypothesis h to agree with c on the training examples
  - A hypothesis is consistent if it agrees with **all training examples**
  - A hypothesis said to generalize well if it correctly predicts the value of y for novel example

- Inductive Learning is an ill Posed Problem:
  - Unless we see all possible examples the data is not sufficient for an inductive learning algorithm to find a unique solution

Inductive Learning Hypothesis

- Any hypothesis h found to approximate the target function **c** well over a sufficiently large set of training examples D will *also approximate the target function well over other unobserved examples*

Learning Issues

- What are good hypothesis spaces?
- Algorithms that work with the hypothesis spaces
- How to optimize accuracy over future data points
- How can we have confidence in the result? (How much training data?)

Linear Regression

09/04/2024

Koustav Rudra

Dataset of living area and price of houses in a city

Living area ($feet^2$)	Price (1000$s)
2104	400
1600	330
2400	369
1416	232
3000	540
⋮	⋮

This is a training set.

How can we learn to predict the prices of houses of other sizes in the city, as a function of their living area?

Dataset of living area and price of houses in a city

Living area (feet^2)	Price (1000\$s)
2104	400
1600	330
2400	369
1416	232
3000	540
⋮	⋮

Example of supervised learning problem

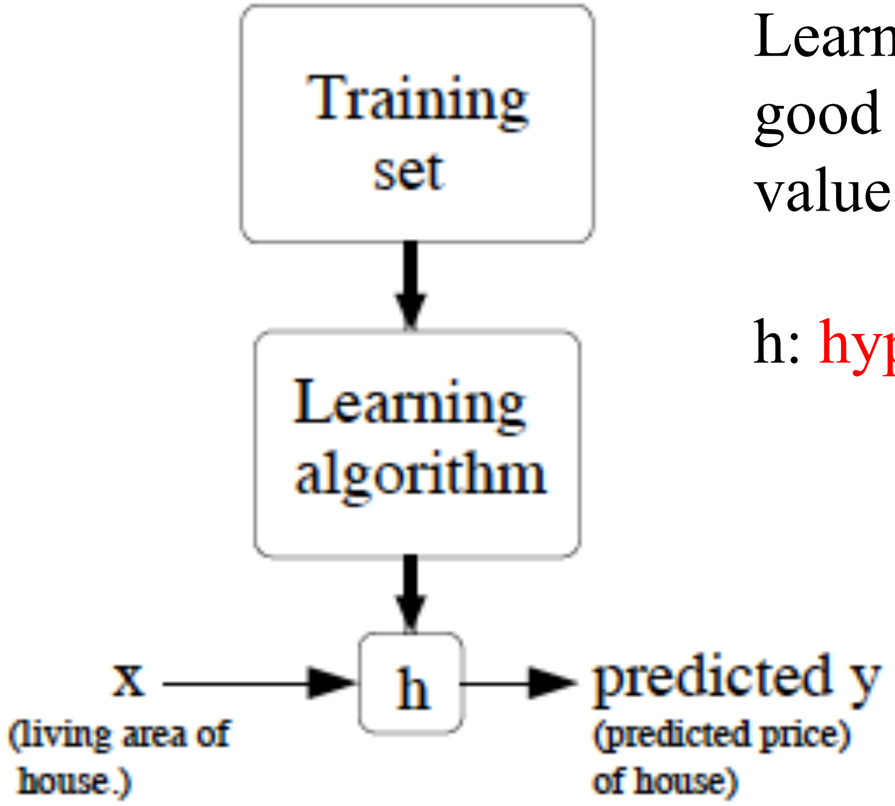
When the target variable we are trying to predict is continuous, regression problem

Dataset of living area and price of houses in a city

Living area ($feet^2$)	Price (1000$s)
2104	400
1600	330
2400	369
1416	232
3000	540
⋮	⋮

m = number of **training examples**
x's = input variables / features
y's = output variables / "target" variables
(x,y) - single training example
(x^i, y^i) - specific example (i^{th} training example)
i is an index to training set

How to use the training set?



Learn a function h(x), so that h(x) is a good predictor for the corresponding value of y

h: hypothesis function

How to represent hypothesis h?

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

θ_i are **parameters**
- θ_0 is zero condition
- θ_1 is gradient

θ: vector of all the parameters

We assume y is a linear function of x

Univariate linear regression

How to learn the values of the parameters?

Digression:
Multivariate linear regression

Living area (feet^2)	#bedrooms	Price (1000$s)
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540
⋮	⋮	⋮

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

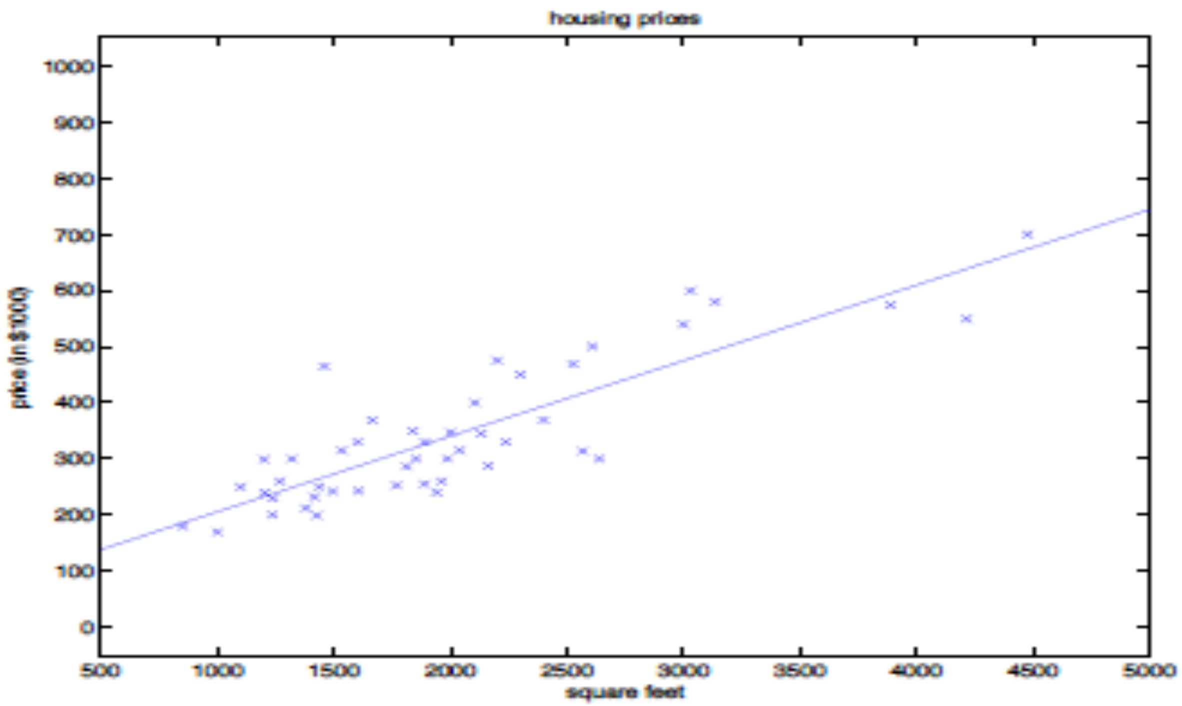
How to represent hypothesis h?

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

θ_i are **parameters**
- θ_0 is zero condition
- θ_1 is gradient

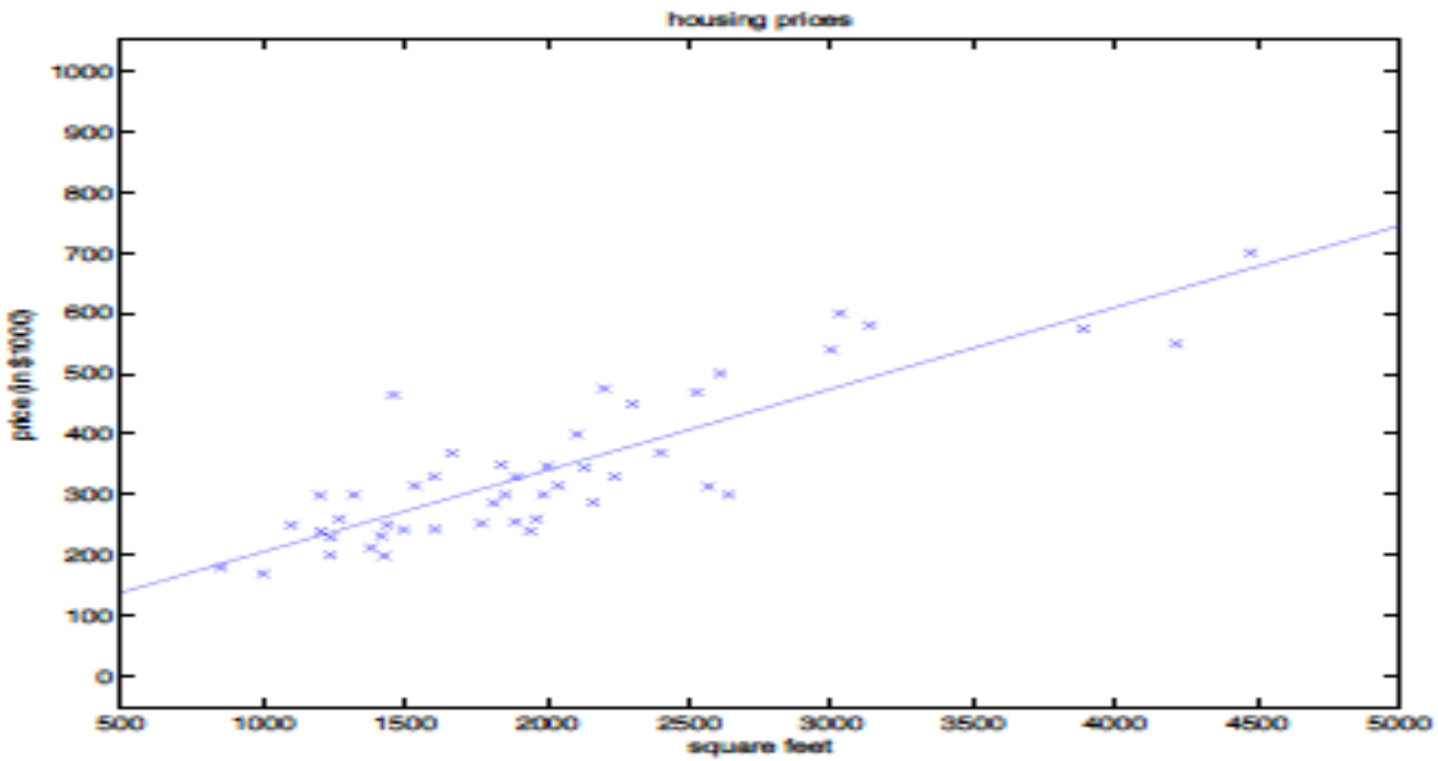
We assume y is a linear function of x
Univariate linear regression
How to learn the values of the parameters θ_i?

Intuition of hypothesis function



- We are attempting to fit a straight line to the data in the training set
- Values of the parameters decide the equation of the straight line
- Which is the best straight line to fit the data?

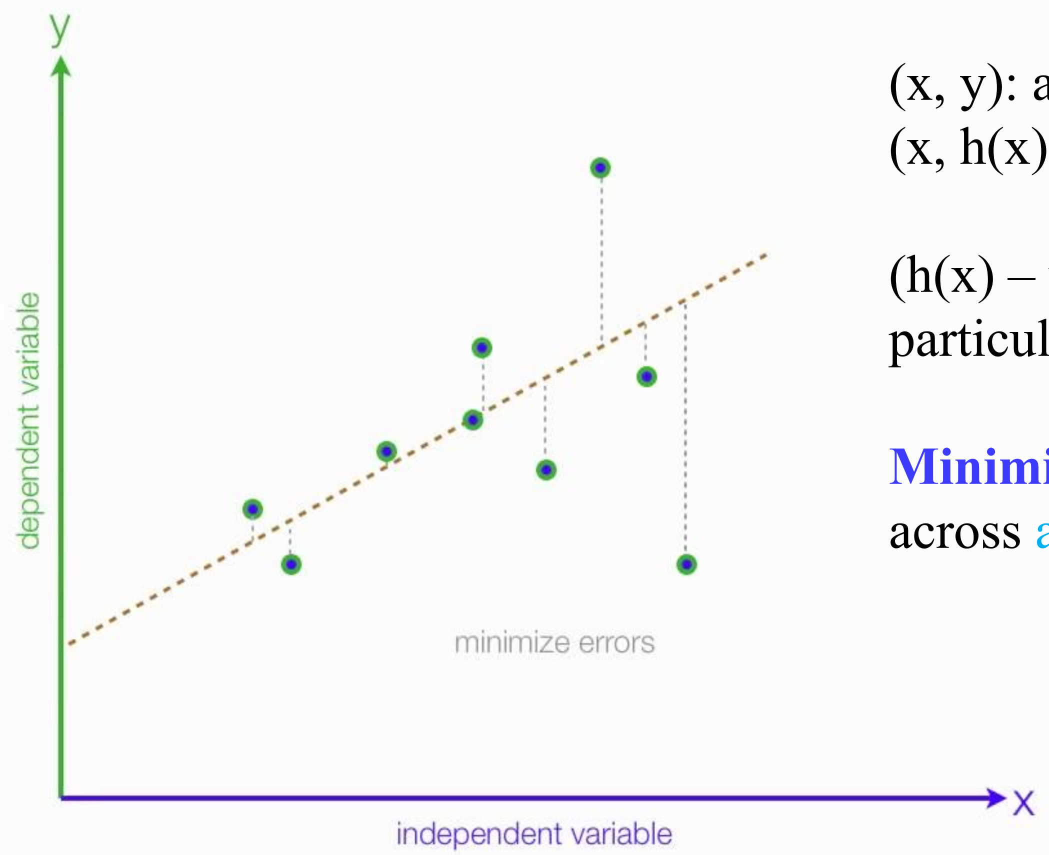
Intuition of hypothesis function



- Which is the best straight line to fit the data?
- How to learn the values of the parameters θ_i?

- Choose the parameters such that the prediction is close to the actual y-value for the training examples

How good is the prediction given by the straight line?



(x, y): a training example
(x, h(x)): prediction of the model

(h(x) – y): prediction error for this particular training example

Minimize the prediction error across all training examples

Cost function

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

- Measure of how close the predictions are to the actual y-values
- Average over all the m training instances

- Squared error cost function $J(\theta)$
- Choose parameters θ so that $J(\theta)$ is minimized

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

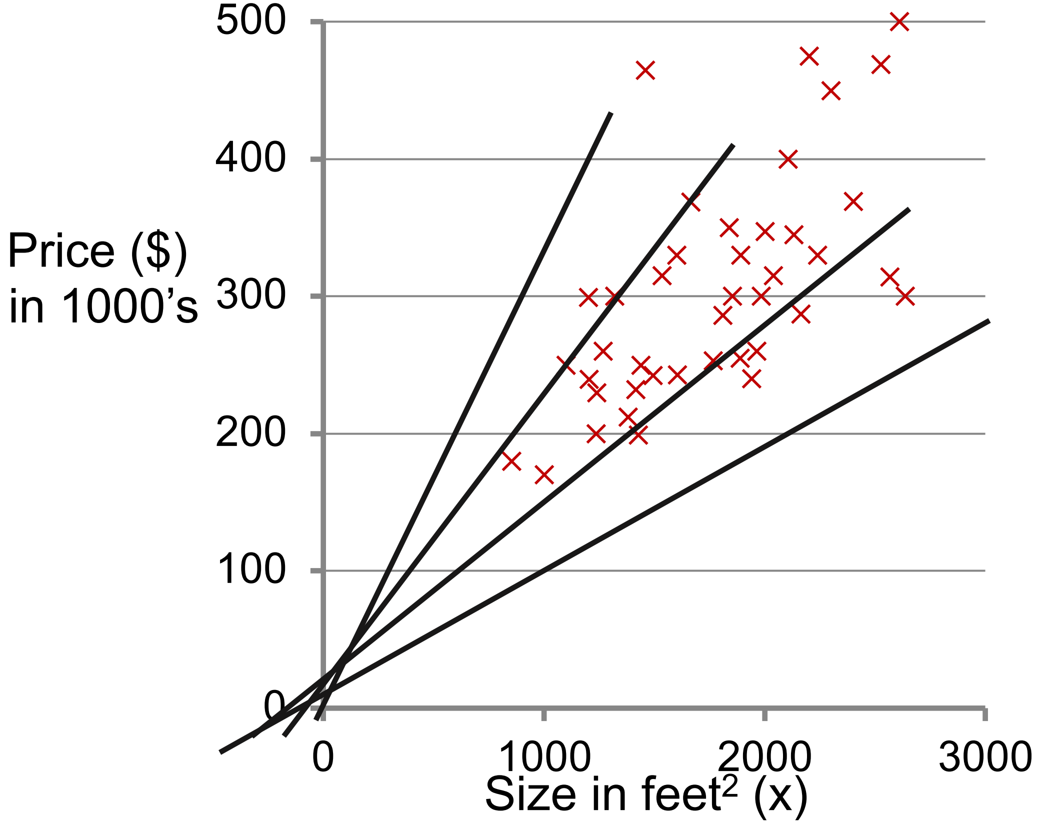
Parameters: θ_0, θ_1

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

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

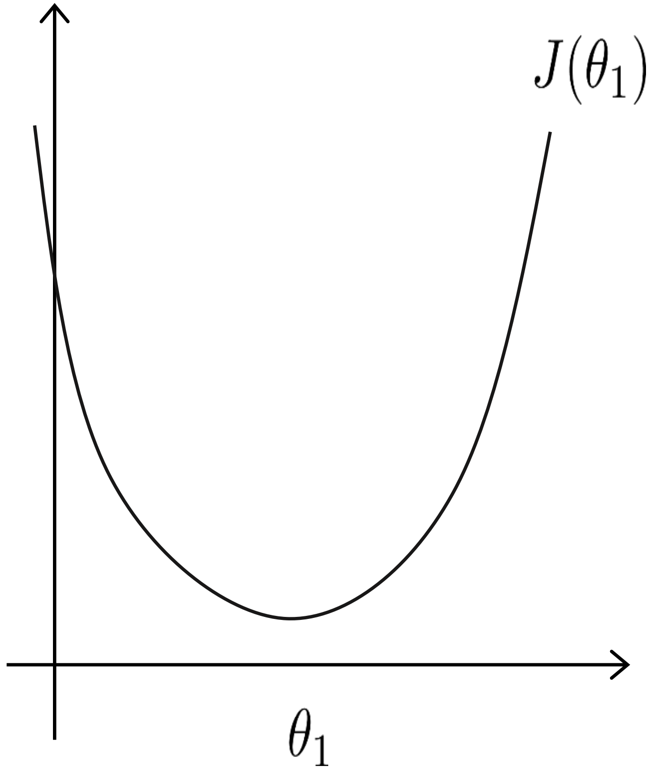
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

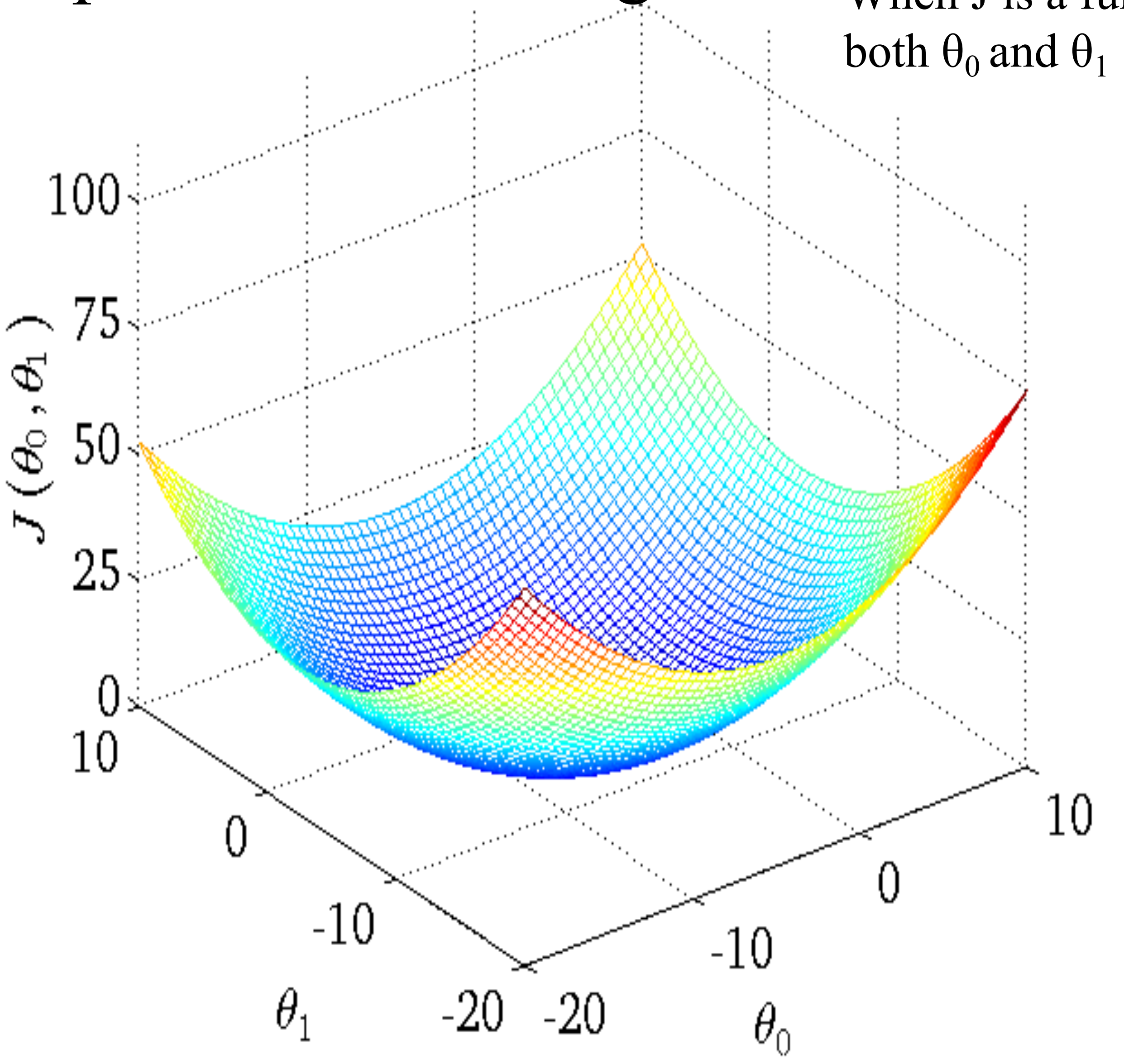
(function of the parameters θ_0, θ_1)



For simplicity, assume Θ_0 is a constant

Contour plot or Contour figure

When J is a function of both θ_0 and θ_1



Minimizing a function

- For now, let us consider some arbitrary function (not necessarily a cost function)

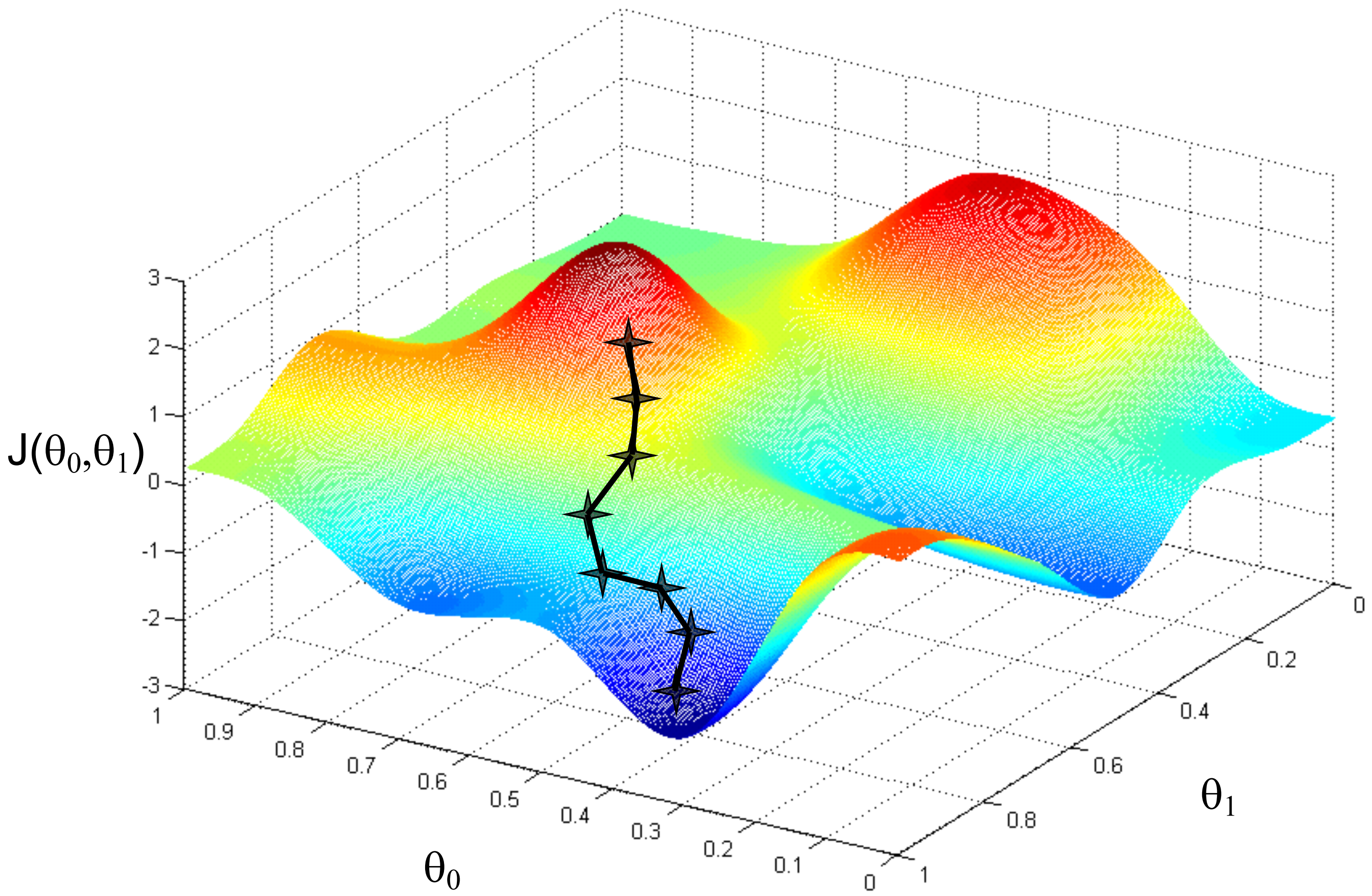
- Analytical minimization not scalable to complex functions of hundreds of parameters

- Algorithm called gradient descent
  - Efficient and scalable to thousands of parameters
  - Used in many applications of minimizing functions

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

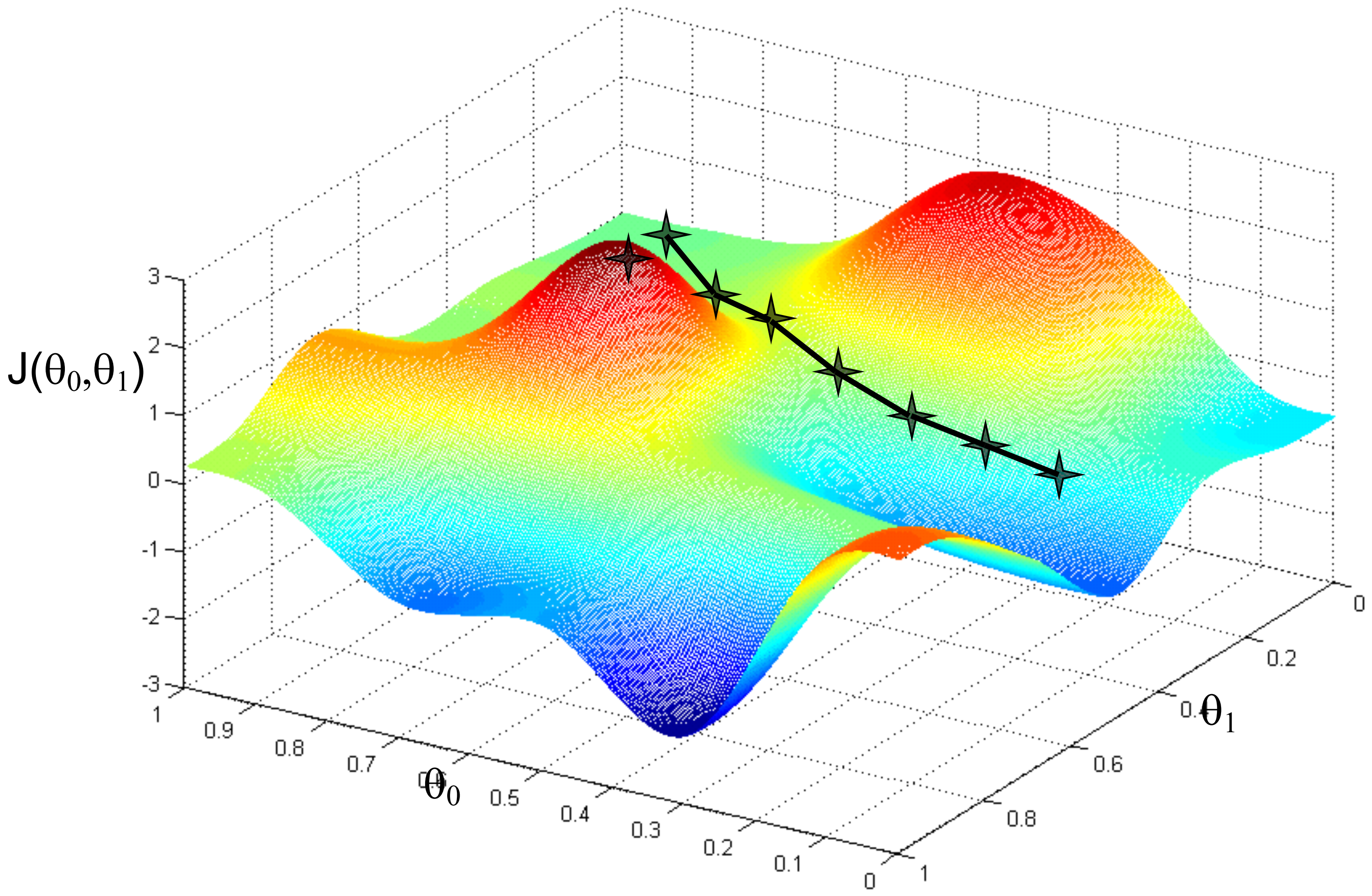
Want $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$

- **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
- Iterative method, similar to Newton-Raphson method for solving equations



J(θ0,θ1)
θ0
θ1
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

If the function has multiple local minima, where one starts can decide which minimum is reached



Gradient descent algorithm

$$\text{repeat until convergence } \{$$
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \quad \text{(simultaneously update } j = 0 \text{ and } j = 1)$$
$$\}$$

α is the learning rate – more on this later

Gradient descent algorithm

$$\text{repeat until convergence } \{$$
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \quad (\text{for } j = 0 \text{ and } j = 1)$$
$$\}$$

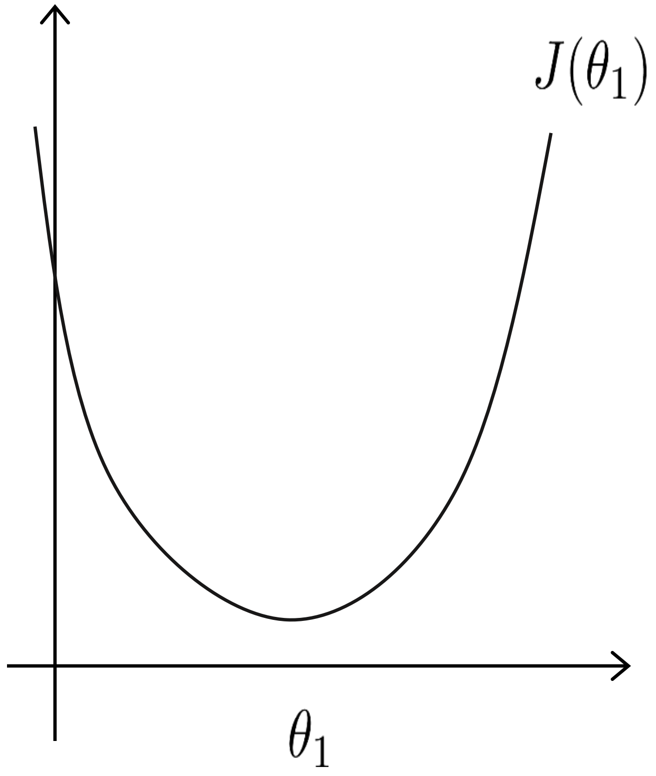
Correct: Simultaneous update

$$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$
$$\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$
$$\theta_0 := \text{temp0}$$
$$\theta_1 := \text{temp1}$$

Incorrect:

$$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$
$$\theta_0 := \text{temp0}$$
$$\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$
$$\theta_1 := \text{temp1}$$

For simplicity, let us first consider a function of a single variable



$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

If the derivative is positive, reduce value of θ_1

If the derivative is negative, increase value of θ_1

The learning rate

- Do we need to change learning rate over time?
  - No, Gradient descent can converge to a local minimum, even with the learning rate α fixed
  - Step size adjusted automatically

- But, value needs to be chosen judiciously
  - If α is too small, gradient descent can be slow to converge
  - If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.

Gradient descent for univariate linear regression

Gradient descent algorithm

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

$$(\text{for } j = 1 \text{ and } j = 0)$$

}

Linear Regression Model

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

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

Gradient descent for univariate linear regression

repeat until convergence {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)}$$

}

update θ_0 and θ_1 simultaneously

“Batch” Gradient Descent

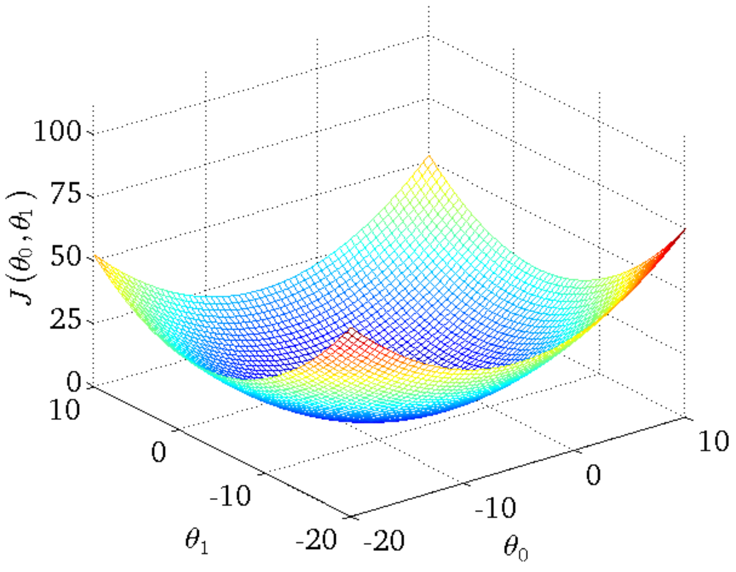
- **“Batch”:** Each step of gradient descent uses all the training examples
  - At each estimate gradient on a batch of m samples

- There are other variations like “stochastic gradient descent” (used in learning over huge datasets)

What about multiple local minima?

- The cost function in linear regression is always a **convex function** – always has a single global minimum

- So, gradient descent will always converge



Linear Regression for multiple variables

Multiple features (variables).

Size (feet^2)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
…	…	…	…	…

Multiple features (variables).

Size (feet^2)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
…	…	…	…	…

Notation:

n = number of features. m = number of training examples

$x^{(i)}$ = input (features) of i^{th} training example.

$x_j^{(i)}$ = value of feature j in i^{th} training example.

Hypothesis:

For univariate linear regression:

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

For multi-variate linear regression:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

For convenience of notation, define $x_0 = 1$

Hypothesis: $h_\theta(x) = \theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$

Parameters: $\theta_0, \theta_1, \ldots, \theta_n$

Cost function:

$$J(\theta_0, \theta_1, \ldots, \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Gradient descent:

Repeat $\{$

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \ldots, \theta_n)$$

$\}$ (simultaneously update for every $j = 0, \ldots, n$)

Gradient Descent

Previously (n=1):

Repeat $\{$

$$\theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})$$

$$\theta_1 := \theta_1 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x^{(i)}$$

(simultaneously update θ_0, θ_1)

$\}$

New algorithm $(n \geq 1)$:

Repeat $\{$

$$\theta_j := \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

simultaneously update θ_j for $j = 0, \ldots, n$

$\}$

...

Gradient Descent

Previously (n=1):

Repeat $\{$

$$\theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})$$

$$\theta_1 := \theta_1 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x^{(i)}$$

(simultaneously update θ_0, θ_1)

$\}$

New algorithm $(n \geq 1)$:

Repeat $\{$

$$\theta_j := \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

simultaneously update θ_j for $j = 0, \ldots, n$

$\}$

$$\theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_2^{(i)}$$

$\ldots$

Practical aspects of applying gradient descent

Feature Scaling

Idea: Make sure features are on a similar scale.

E.g. x_1 = size (0-2000 feet^2)

x_2 = number of bedrooms (1-5)

Normalization wrt the maximum value:

$$x_1 = \frac{\text{size (feet}^2\text{)}}{2000}$$

$$x_2 = \frac{\text{number of bedrooms}}{5}$$

Feature Scaling

Idea: Make sure features are on a similar scale.

E.g. x_1 = size (0-2000 feet^2)

x_2 = number of bedrooms (1-5)

Mean normalization:

Replace x_i with $x_i - \mu_i$ to make features have approximately zero mean (Do not apply to $x_0 = 1$).

Other types of normalization:

$$x_1 = \frac{size - 1000}{2000}$$

$$x_2 = \frac{\#bedrooms - 2}{5}$$

$$-0.5 \leq x_1 \leq 0.5, -0.5 \leq x_2 \leq 0.5$$

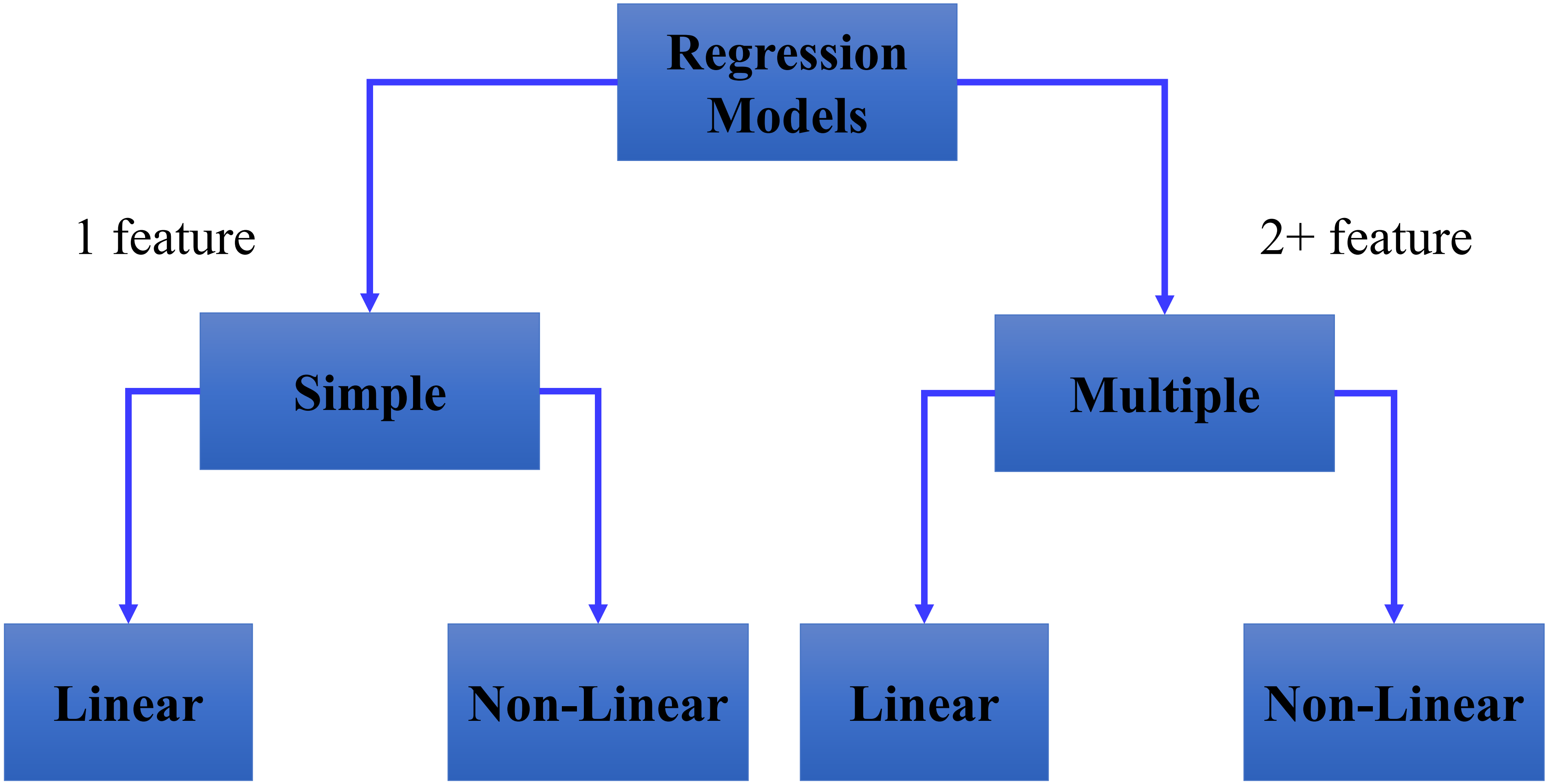
Is gradient descent working properly?

- Plot how $J(\theta)$ changes with every iteration of gradient descent
- For sufficiently small learning rate, $J(\theta)$ should decrease with every iteration
- If not, learning rate needs to be reduced
- However, too small learning rate means slow convergence

When to end gradient descent?

- Example convergence test:
- Declare convergence if $J(\theta)$ decreases by less than 0.001 in an iteration (assuming $J(\theta)$ is decreasing in every iteration)

Types of Regression Models



Thank You