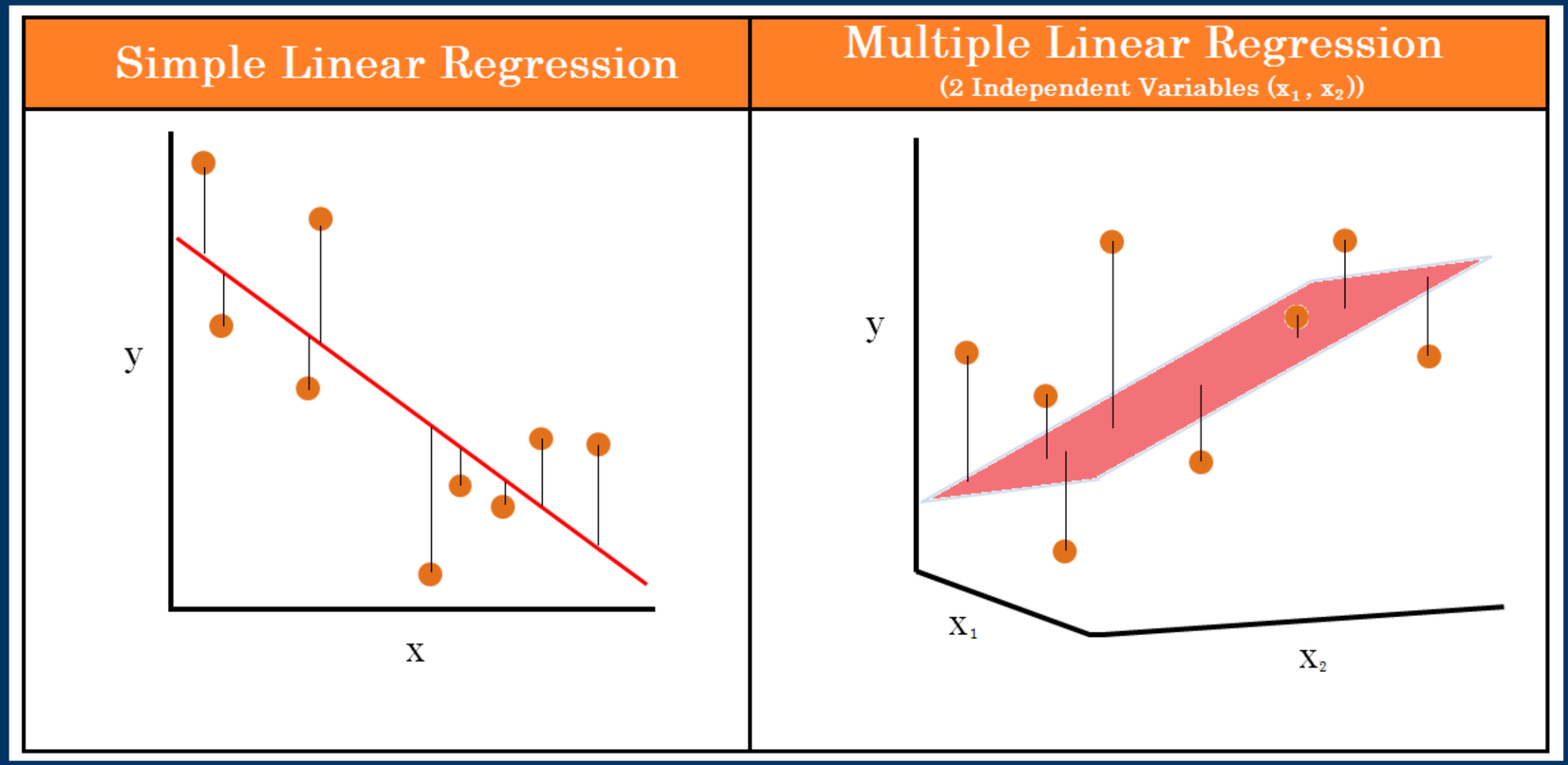


Linear Regression

CSCI-P556 Applied Machine Learning
Lecture 11

D.S. Williamson

Agenda and Learning Outcomes

Today's Topics

- **Topics**:
  - Basics of Linear Regression
  - Regression coefficient estimation
    - Normal Equation
    - Gradient Descent
- **Announcements**
  - Project information will be posted to Canvas later this week
  - Homework #2 will come out next week

Linear Regression

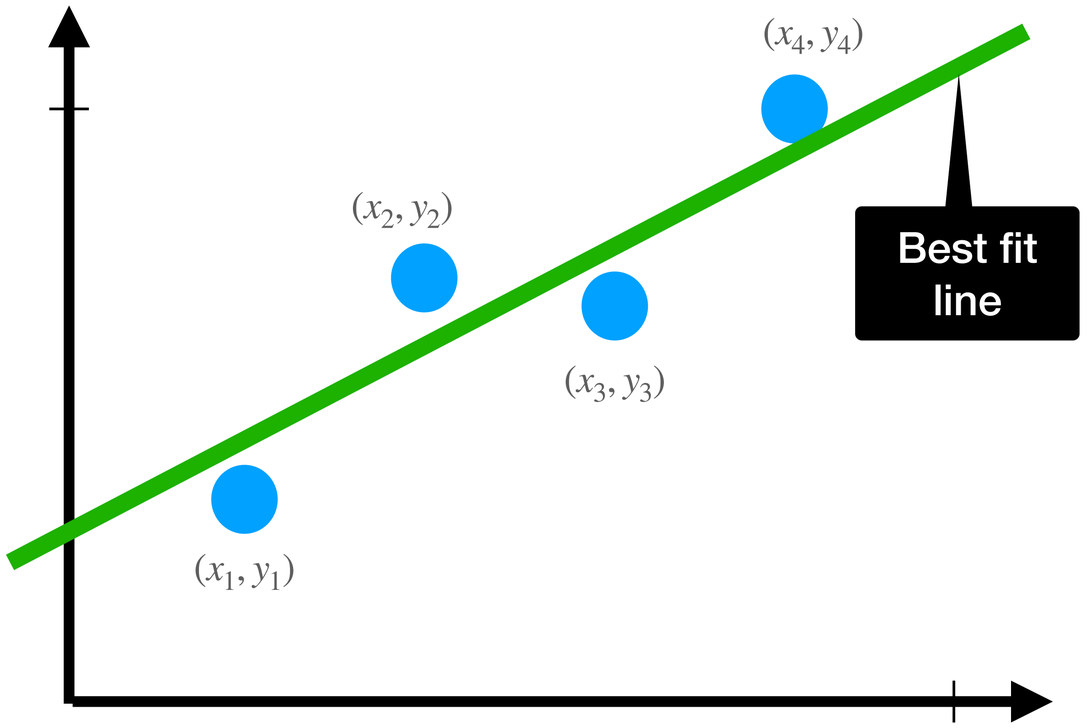
A relationship exists

Feature vector

Label

$$\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_n, y_n)\}$$

- ***Supervised learning assumes a relationship exists between the feature(s) and target***. Though the exact relationship may not be known.
- We're going to discuss a form of regression where a specified relationship (linear) exists between *x* and *y*
- The relationship, however, is not ***perfectly*** linear, so errors occur

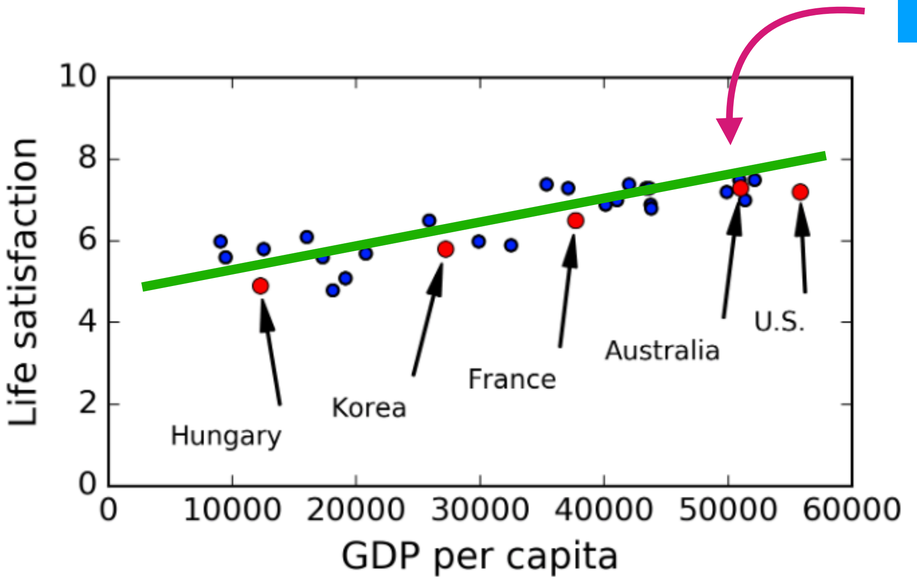


Recall: Life Satisfaction Example

- Suppose you want to know if money makes people happy. You have data that shows the GDP per capita for multiple countries and user rated Life Satisfaction

Table 1-1. Does money make people happier?

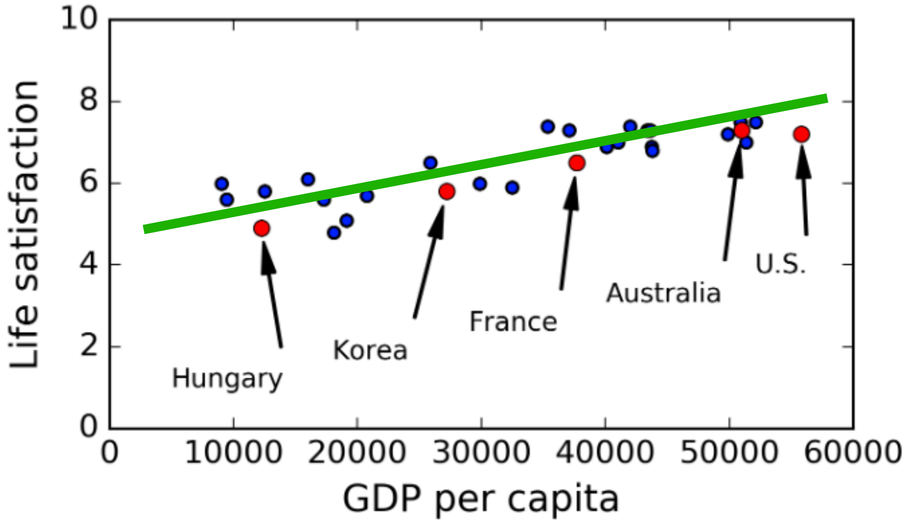
Country	GDP per capita (USD)	Life satisfaction
Hungary	12,240	4.9
Korea	27,195	5.8
France	37,675	6.5
Australia	50,962	7.3
United States	55,805	7.2



The True Relationship is not Perfect!

Noise and uncertainty are always present

- Note that the true relationship between *GDP per capita* and *Life satisfaction* is not perfectly linear
- The ***true relationship*** is modeled as a ***linear relationship with added noise***. Hence, uncertainty and variance exists



$$y = W_0 + W_1 x_1 + W_2 x_2 + \cdots + W_n x_n + \epsilon$$

- y: true value
- W_j: is the j-th weight. $j \in [0,n]$
- ϵ: random noise (e.g. a RV)

A ***statistical*** relationship exists!
Observations do not fall directly on the line

Other Examples

Linear Regression can be used elsewhere

1. The sales of a product can be predicted based on the amount of advertising expenditures
2. The performance of an employee can be predicted from the amount of hours that they work
3. A child's vocabulary size can be predicted based on their age, how many hours they are read to, and their parent's education level
4. The length of a patient's hospital stay can be estimated from the severity of their operation
5. …

Linear Regression

How are estimates generated?

- Linear regression implies that a (quasi-) linear relationship exists between the feature(s) and label. The ***goal*** is then to ***use this relationship to estimate*** values for the label, given the features.

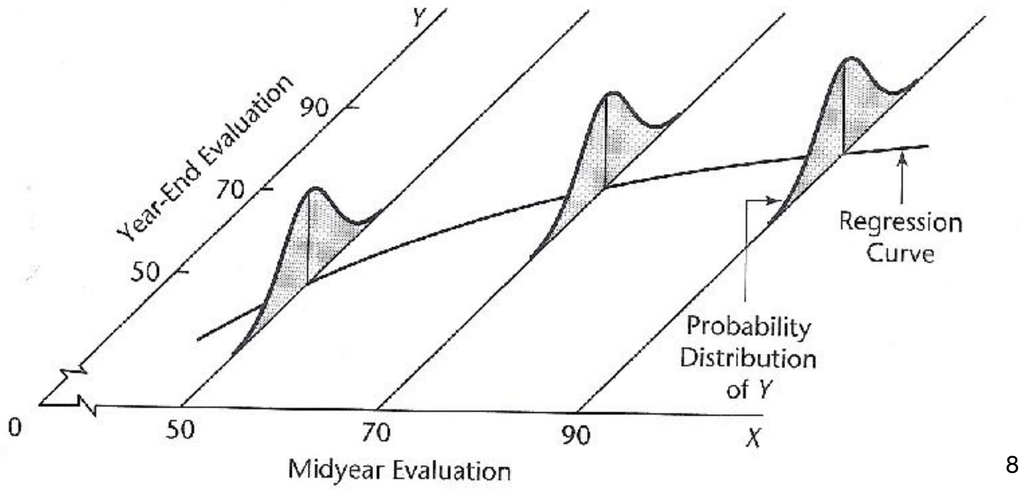
$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

- $\hat{y}$: predicted value
- n: is the number of features (or feature dimension)
- x_i: is the i-th feature value
- θ_j: is the j-th model parameter. $j \in [0,n]$

Example: Performance Evaluation

Characteristics of (Linear) Regression Models

- Two characteristics are embodied in a regression model:
  - There is a probability distribution for y at each value of **x** (e.g. y is a random variable)
  - The means of these probability distributions vary in some systematic fashion with **x.**
- **Example**: Predicting an employees year-end evaluation from their mid-year evaluation
  - For each level of mid-year evaluation, **x**, there is a probability distribution for y
  - The actual year-end evaluation is viewed as a random selection from this probability distribution



Note:

- The means of the distributions have a relation to x. **This relation is called the *regression function or curve***
- The distribution of y is based on the distribution of ϵ (the noise)

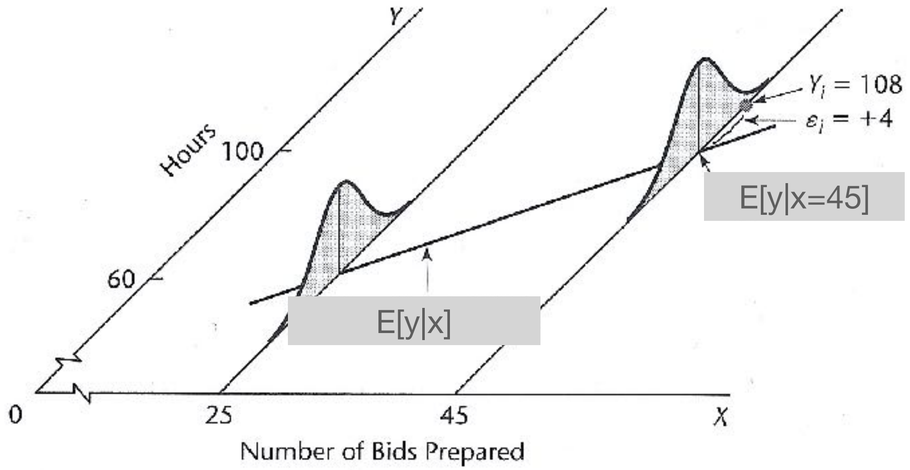
Example

- A consultant for an electrical distributor is studying the relationship between the *number of bids requested (x)* by construction contractors for basic lighting equipment during a week and the *time required to prepare the bids (y)*

- Suppose that the linear ***regression model*** is as follows:

$$y = 9.5 + 2.1x + \epsilon$$

- Assuming that the noise has a mean of 0, then the linear ***regression function*** is as such:



$$E[y \mid x] = 9.5 + 2.1x$$

Meaning of Regression Parameters

- W_i are called ***regression coefficients***.
  - W_0 is the bias term (or y-intercept).
  - Can think of $W_1, \cdots, W_n$ as "***slopes***" or ***weights*** for each feature (or dimension)

$$y = W_0 + W_1 x_1 + W_2 x_2 + \cdots + W_n x_n + \epsilon$$

- **Vector Notation** for Linear Regression

$$y = \mathbf{W}^T \mathbf{x} + \epsilon$$

Dot product between vectors

$\mathbf{W} = [W_0, \cdots, W_n]^T$: parameter vector

$\mathbf{x} = [x_0, \cdots, x_n]^T$: feature vector

$$x_0 = 1$$

Unknown parameters

Regression coefficients need to be estimated

- The ***regression coefficients***, $\mathbf{W}$, of the true linear regression model ***are unknown***. Hence, they ***need to be estimated***.

$$y = \mathbf{W}^T \mathbf{x} + \epsilon$$

- The model's parameter vector, θ, serves as our estimate of the regression coefficients (e.g. $\theta \approx \mathbf{W}$).

$$\hat{y} = \theta^T \mathbf{x}$$

- Our **goal** is then to ***find the model parameters***, θ, that ***minimize the mean-square error*** between y and $\hat{y}$

The Normal Equation

A Closed Form Solution for Estimating Regression Coefficients

- The mean-square error (MSE) for a linear regression model is defined as:

$$MSE(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N} \sum_{i=1}^{N} (\theta^T \mathbf{x}_i - y_i)^2$$

- Our goal is to find the θ that makes this error as small as possible.
  - Using calculus, this can be found by taking the derivative of the MSE with respect to θ, setting this derivative to zero, and then solving for θ. (You should try this on your own).
  - This results in the following ***closed-form solution*** (e.g. gives the result directly)

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

- $\hat{\theta}$ is the value of θ that minimizes the cost function.
- $\mathbf{y}$ is the vector of target values containing y_1 to y_N
- $\mathbf{X}$ is the matrix of feature vectors containing $\mathbf{x}_1$ to $\mathbf{x}_N$ (e.g. along the rows)

The Normal Equation

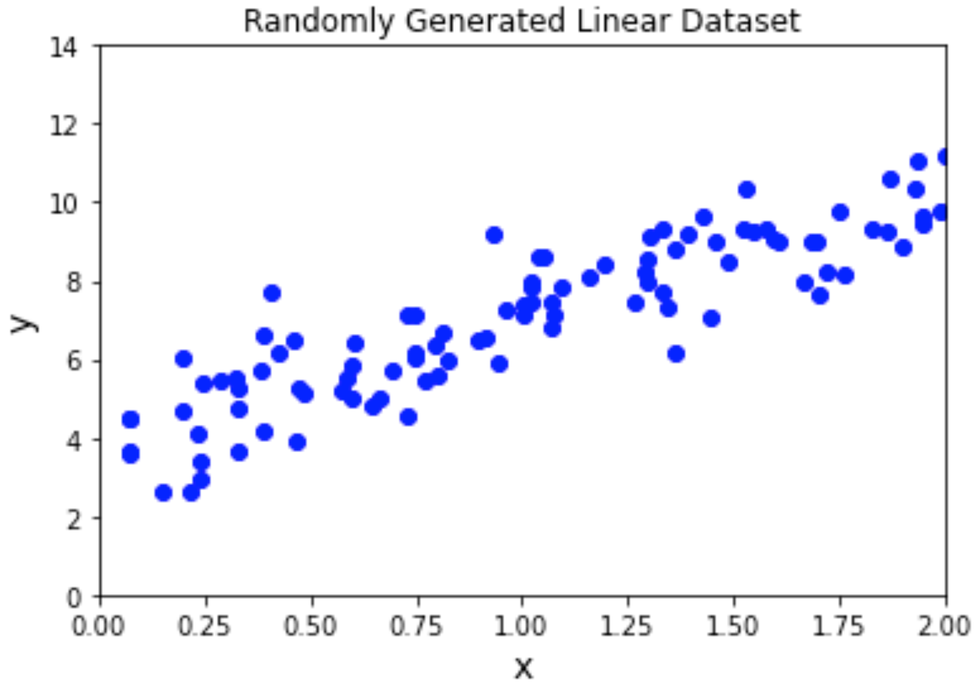
A Python Example

- Randomly generate x values. Model linear equation, y = 4 + 3x, with random Gaussian noise added

```
import numpy as np
import matplotlib.pyplot as plt

X = 2*np.random.rand(100,1)
y = 4 + 3*X + np.random.randn(100,1)

plt.plot(X,y,'bo')
plt.ylabel('y', fontsize=14)
plt.xlabel('x', fontsize=14)
plt.xlim(0,2)
plt.ylim(0,14)
plt.title("Randomly Generated Linear Dataset")
```



The Normal Equation

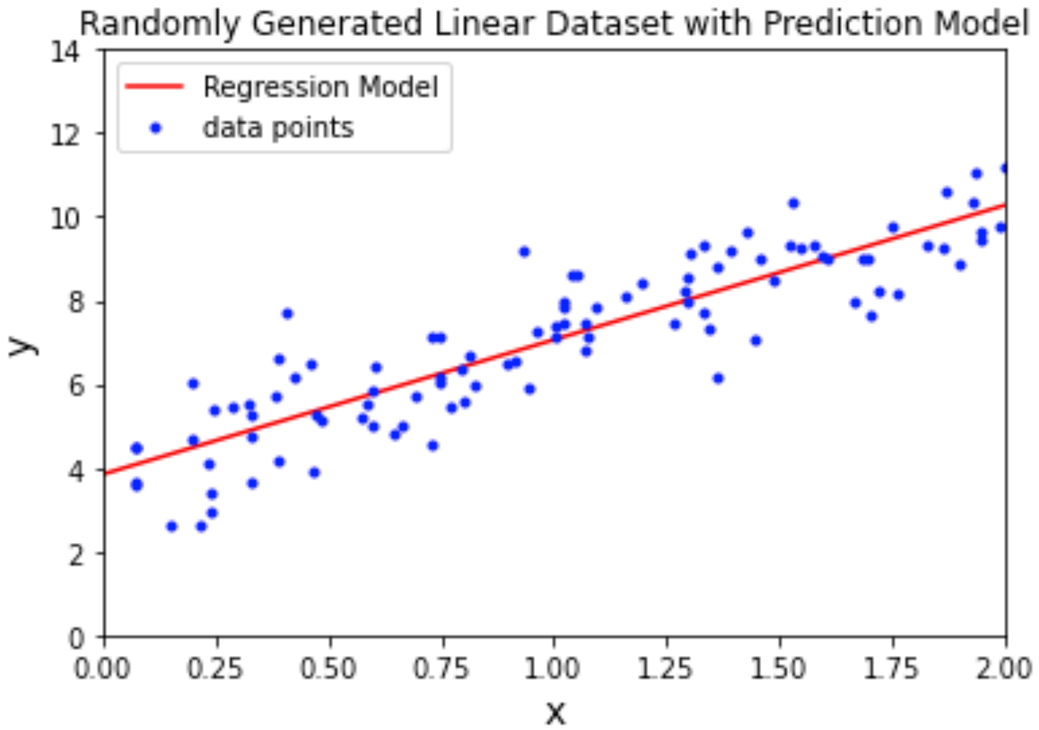
A Python Example

- Use the Normal Equation to estimate the regression coefficients. First need to generate the feature matrix, **X**

```
X_mat     = np.c_[np.ones((100,1)), X] # add x0 = 1 to each instar
theta_hat = np.linalg.inv(X_mat.T.dot(X_mat)).dot(X_mat.T).dot(y)
print(theta_hat)
```

```
[[3.84404191]
 [3.21469577]]
```

- Ideally, $\hat{\theta}$ = [4,3], but this returns [3.84,3.21]
- Now we can plot our ***regression function***



The Normal Equation

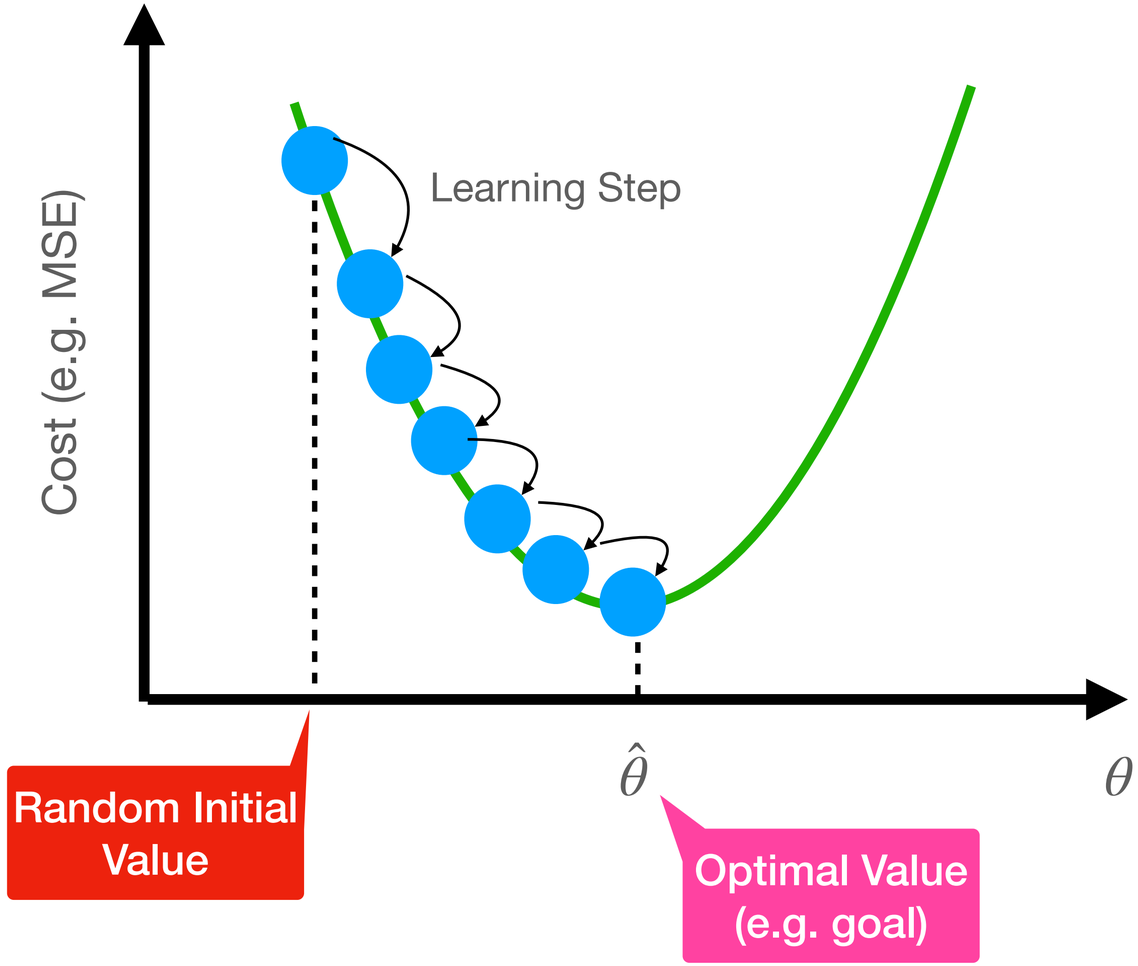
Pros and Cons of this approach

- **Pros**:
  - Easy to implement
  - Predictions are fast (linear computational complexity)
- **Cons**:
  - Inverse of matrix must be computed (not invertible — does not exist)!
    - Computationally complex
    - May use pseudoinverse instead (e.g. estimate of inverse)
  - Batch approach: Uses all training data at once.
    - Computing resources (e.g. memory) may not allow this
    - Computationally slow for large training sets

Gradient Descent

An iterative approach to find optimal regression coefficients

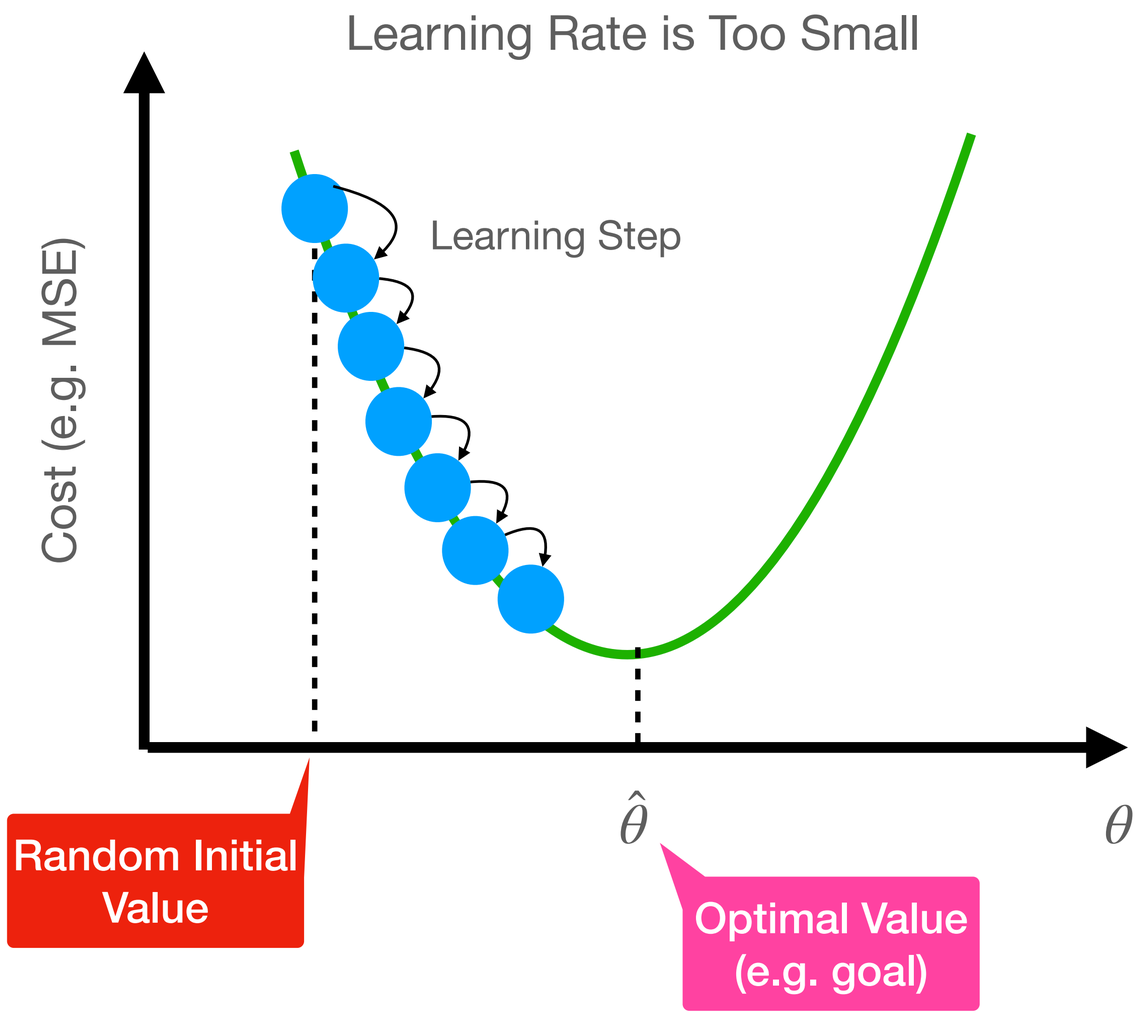
- **Idea**: update parameters (e.g. estimated regression coefficients) iteratively in order to minimize a cost function
- Used by many learning algorithms, include ***neural networks***
- **Approach**:
  - Look at how cost varies with different values of θ (or any parameter)
  - Randomly initialize estimate $\hat{\theta}$
  - Gradually update estimate until the minimum cost value is reached for the estimate (e.g. converges)



Gradient Descent

An iterative approach to find optimal regression coefficients

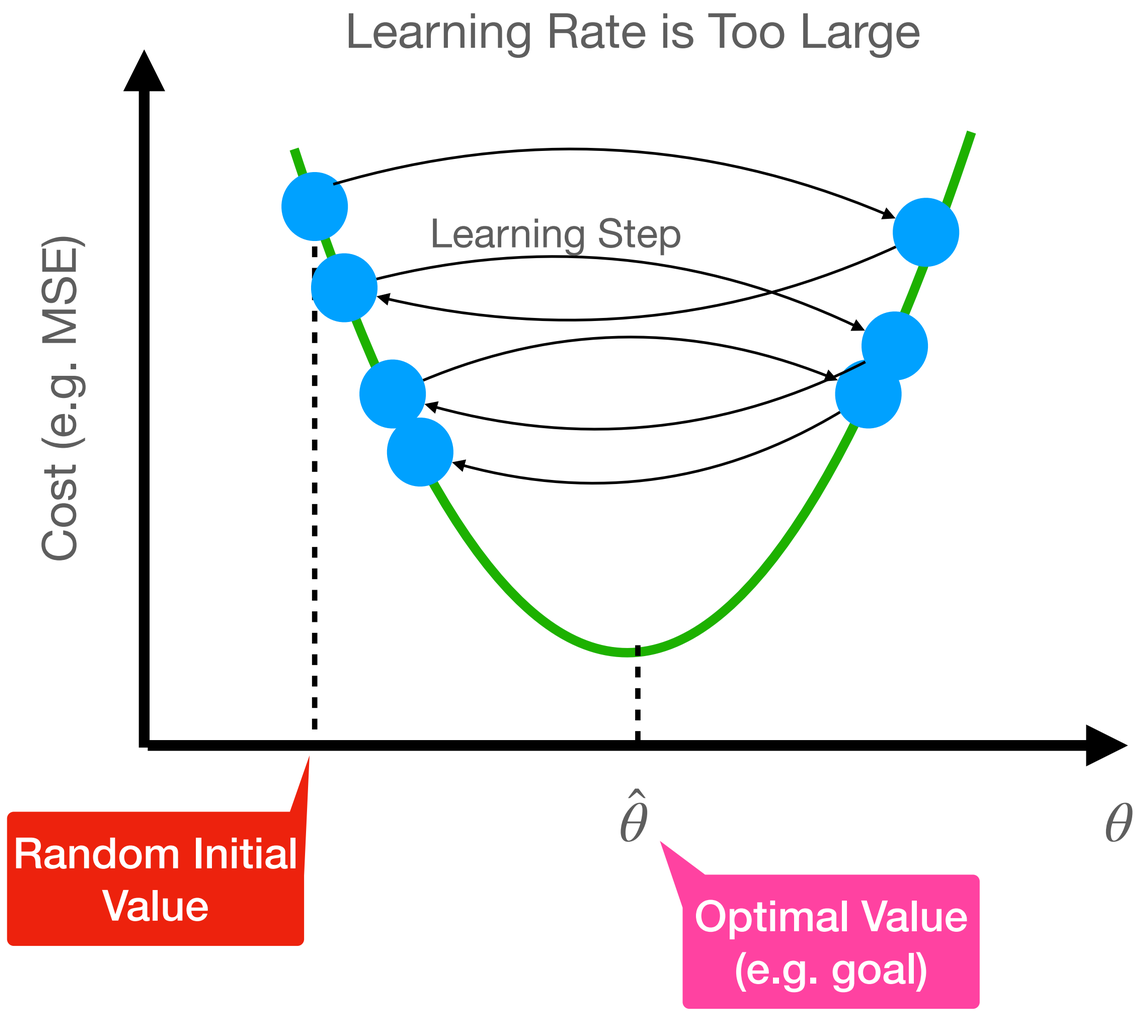
- The ***size of the learning step***, which is known as the learning rate, is an important hyperparameter that ***determines how long*** it will take ***to converge*** to a solution
  - Too small of a learning rate means convergence will take a long time (e.g. many iterations)
  - Too high of a learning rate, then the algorithm may never converge (e.g. jumps from side to side)



Gradient Descent

An iterative approach to find optimal regression coefficients

- The ***size of the learning step***, which is known as the learning rate, is an important hyperparameter that ***determines how long*** it will take ***to converge*** to a solution
  - Too small of a learning rate means convergence will take a long time (e.g. many iterations)
  - Too high of a learning rate, then the algorithm may never converge (e.g. jumps from side to side)



Batch Gradient Descent

Update estimate using entire dataset

- ***Update model parameters*** (θ) based on ***gradient of the cost function*** with respect to each model parameter (θ_j)

Update Equation

$$\theta^t = \theta^{t-1} - \eta \nabla_\theta \text{MSE}$$

- Current iteration
- Learning Rate or step
- Current estimate
- Previous estimate
- Gradient Vector

$$\nabla_\theta \text{MSE} = \begin{bmatrix} \frac{\partial}{\partial \theta_0} \text{MSE} \\ \frac{\partial}{\partial \theta_1} \text{MSE} \\ \vdots \\ \frac{\partial}{\partial \theta_n} \text{MSE} \end{bmatrix} = \frac{2}{N} \mathbf{X}^T (\mathbf{X}\theta - \mathbf{y})$$

$$= \frac{2}{N} \sum_{i=1}^{N} (\theta^T \mathbf{x}_i - y_i) x_{1,i}$$

- For this Linear Regression Problem
- Notice that the entire training dataset is used (e.g. Batch learning)

- Gradient indicates how much the cost function will change if the model parameter is slightly changed

Batch Gradient Descent

Python Implementation

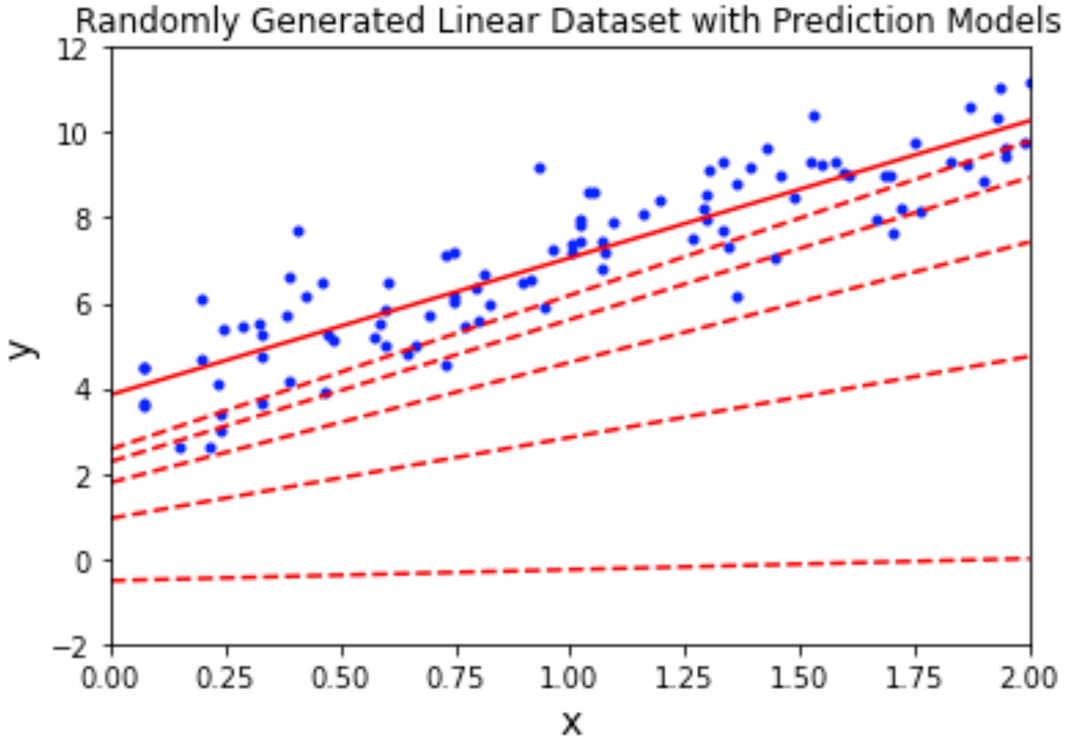
```
eta = 0.1
n_iterations = 1000
N = 100

theta = np.random.randn(2,1)

for iteration in range(n_iterations):
    y_hat = X_new_mat.dot(theta)
    if iteration < 5:
        plt.plot(X_new,y_hat,'r--')
    elif iteration == 999:
        plt.plot(X_new,y_hat,'r-')

    gradients = 2/N * X_mat.T.dot(X_mat.dot(theta) - y)
    theta = theta - eta *gradients

print(theta)
```

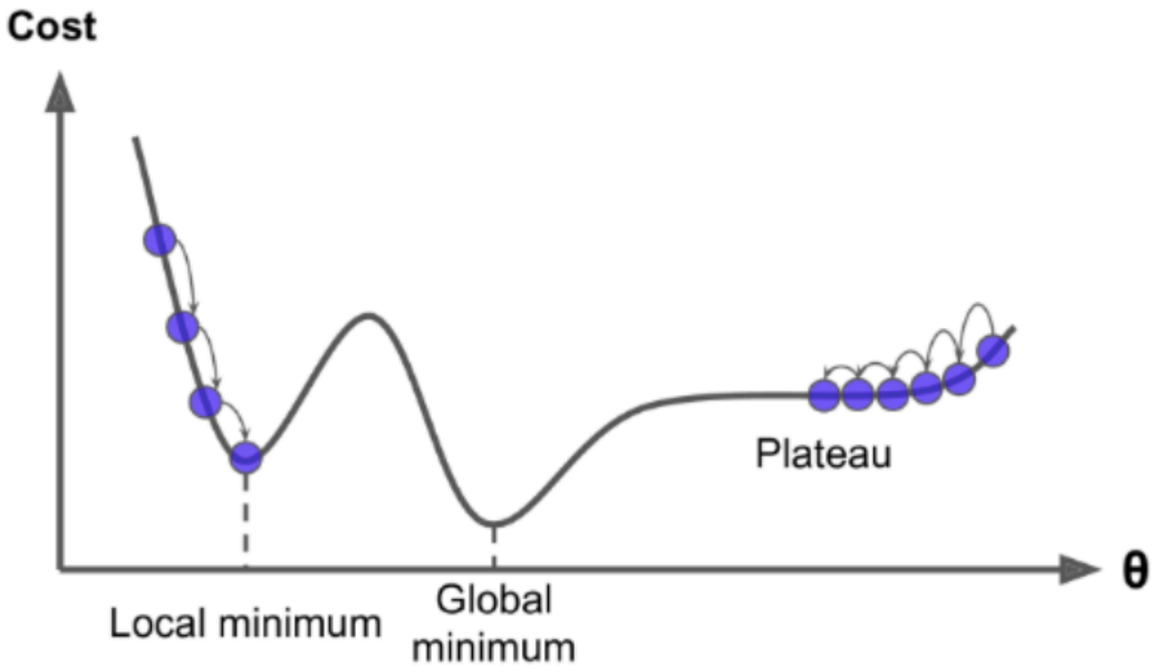


- The final estimated value for θ is [3.84, 3.21]. The same from the Normal Equation
- In the right plot, we can see how the estimated regression curves change over the iterations

General Note about Gradient Descent

All cost functions are NOT convex!

- Some cost functions for certain algorithms are not convex
  - Have local and global minimums
  - Have plateaus
- Optimal solution may be hard to reach, or may never be reached.
- Requires adjustments to hyper parameters (more on this later)
- This is not the case for Linear Regression with MSE cost function. The convex shape, however, may differ (depends on features).

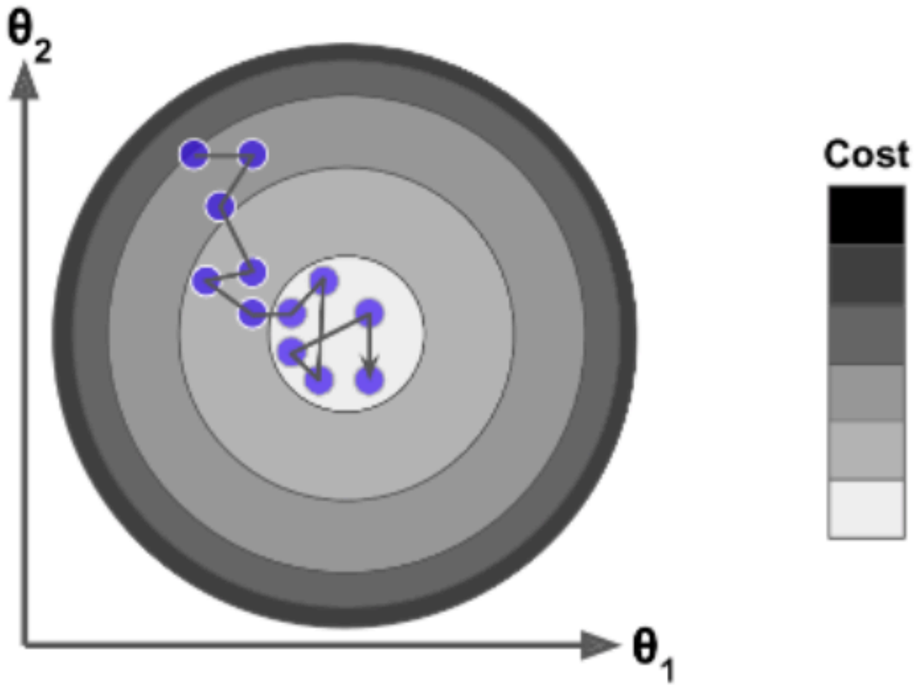


Stochastic Gradient Descent

Online learning approach

$$\frac{\partial}{\partial\theta}\text{MSE} = 2\mathbf{x}_i^T(\mathbf{x}_i\theta - y_i)$$

- Batch Gradient Descent can be slow (depending on the data size), since it uses the entire dataset during each iteration.
- To combat this, ***Stochastic Gradient Descent*** uses only a single feature instance to compute the gradient and update the estimated parameter. Hence, it performs online learning.
- During each iteration, however, an update is applied separately for each training instance. Iterating over all examples during a single iteration, where each example separately updates the estimate, is known as an ***epoch***.
  - This improves computational complexity (less memory, faster computations)
  - The updates, however, are less stable (e.g. gradual), the error may bounce around and not settle to the global minimum. Hence, final estimates may be good, but not optimal
  - May help escape local minimums though and find global minimum, which is good!
- Instability can be addressed by changing the learning rate during each iteration, known as ***annealing*** (more on this later).



Stochastic Gradient Descent

Python Implemenation

- May use SGDRegressor class in Python
- The following code runs 50 epochs, starting with a learning rate of 0.1

```
from sklearn.linear_model import SGDRegressor
sgd_reg = SGDRegressor(max_iter=50,penalty=None, eta0=0.1)
sgd_reg.fit(X,y.ravel())
sgd_reg.intercept_, sgd_reg.coef_
```

- The estimated regression coefficients are: [3.798, 3.253]

Mini-batch Gradient Descent

Combination of Batch and Stochastic Gradient Descent

- During each iteration, ***mini-batch gradient descent*** computes the gradients on “small” random sets of data examples (e.g. a mini-batch).
- Mini-batch gradient descent is less erratic than Stochastic Gradient Descent, but it may get stuck in local minimums. It also may not converge to the optimal solution.
- It, however, is more computationally efficient than Batch Gradient Descent
- The choice of which approach to use, including the Normal Equation, depends on the problem and dataset. Differences are only observed during training.

Next Class:

Non-Linear Regression (e.g. polynomial, logistic, etc.)