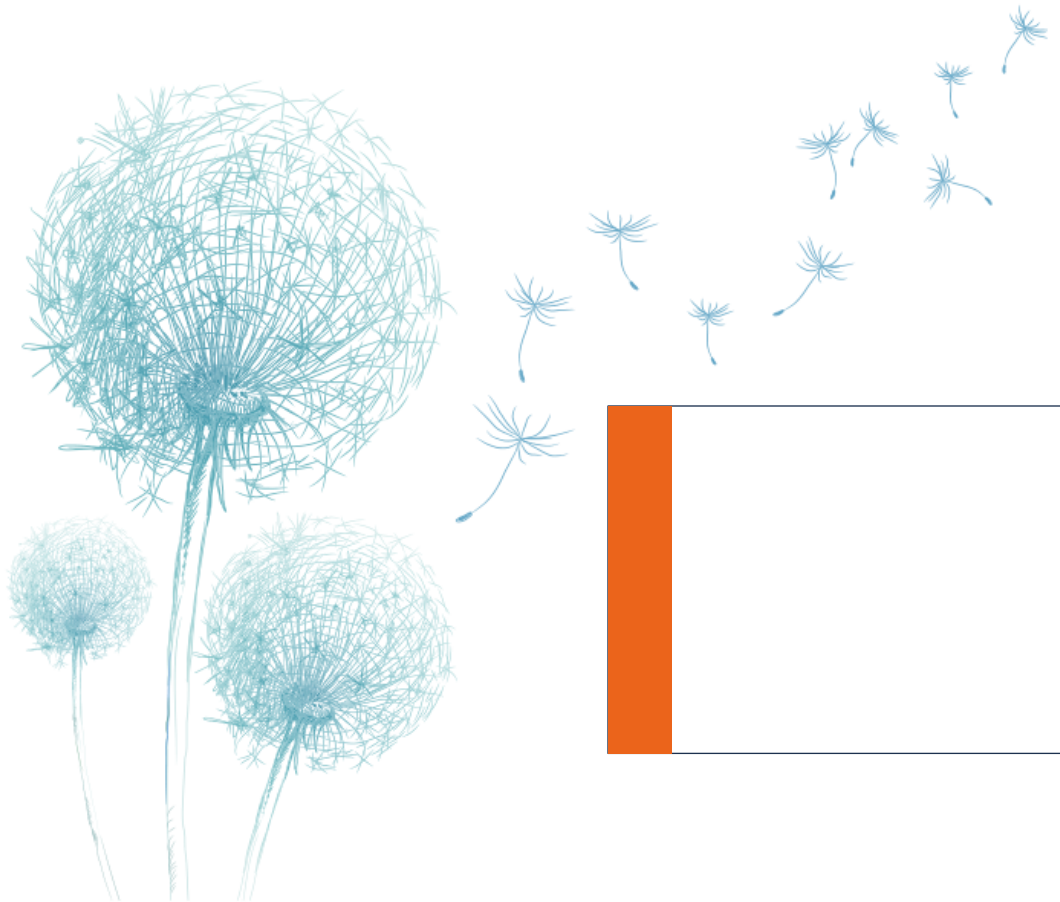
Neural networks and deep learning



Linear model

Kun Suo

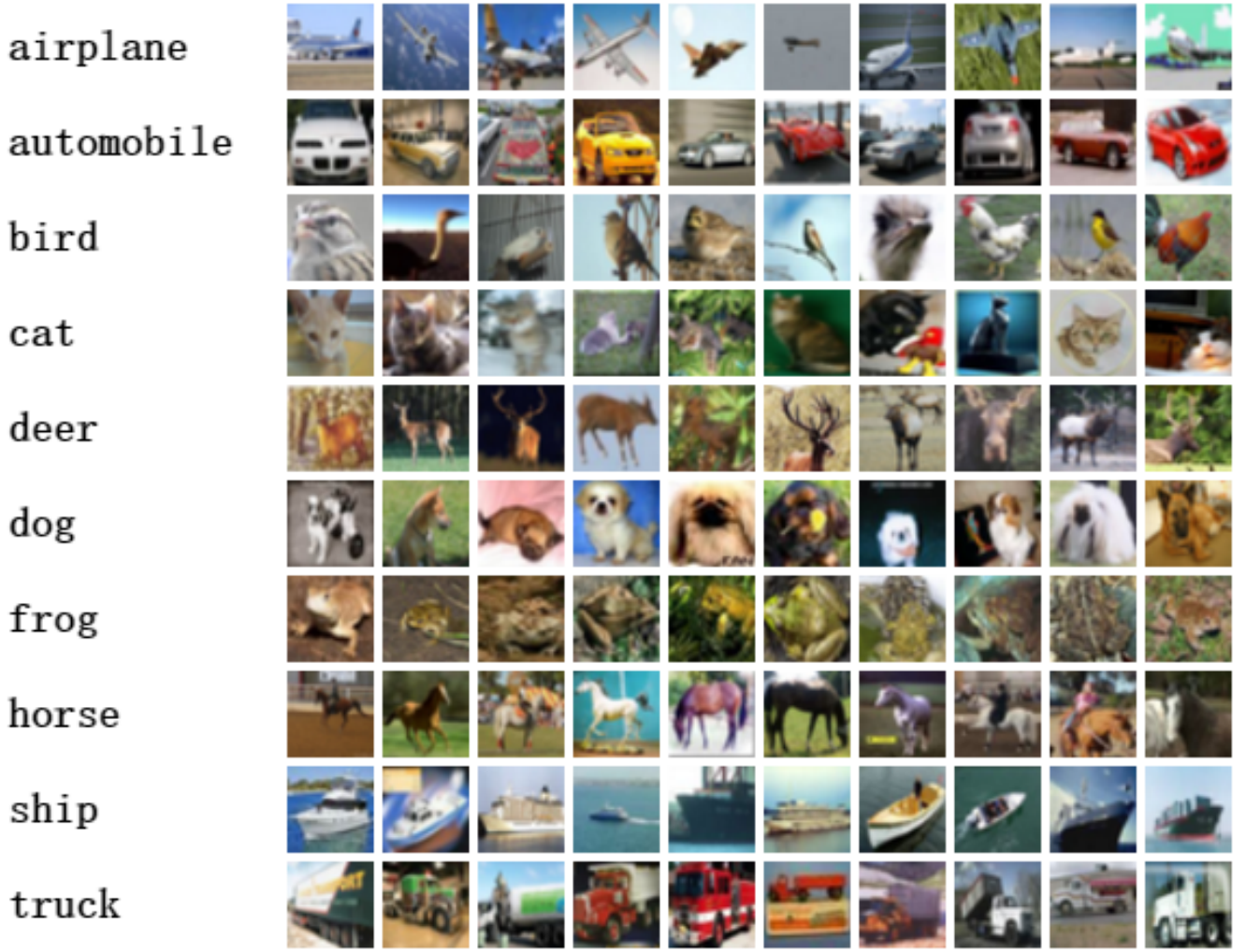
Computer Science, Kennesaw State University

https://kevinsuo.github.io/

Classification example

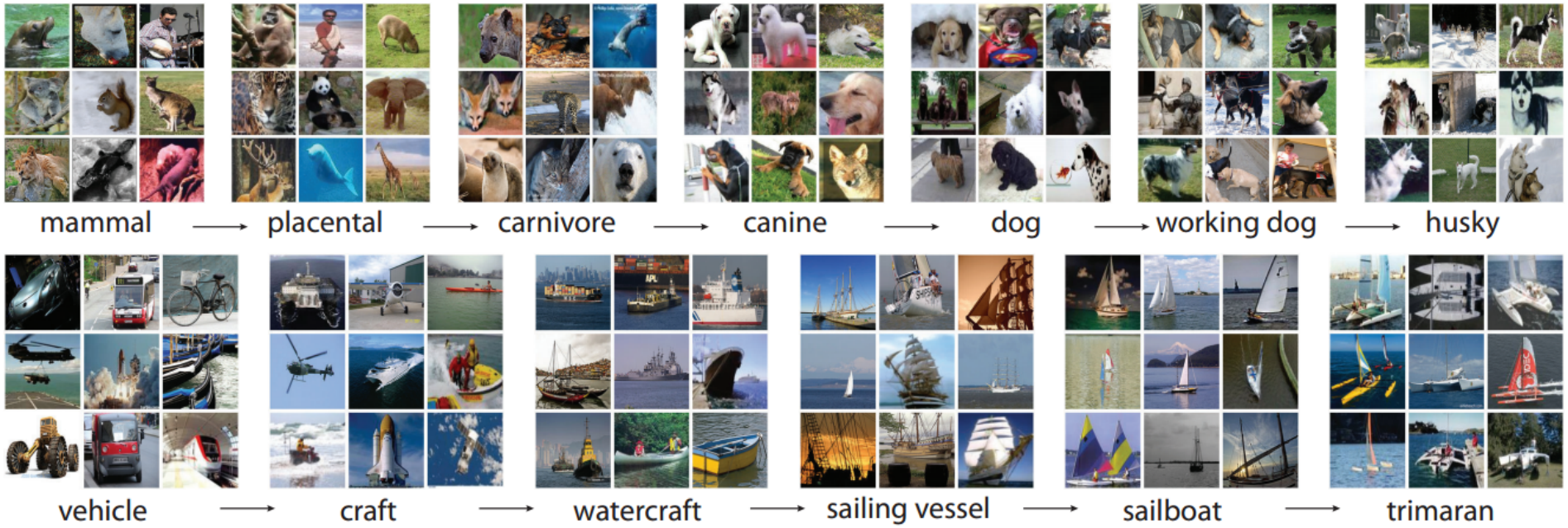
Example: image classification

- Dataset： CIFAR-10
  - 60000 32x32 color images in 10 categories
  - 6000 images per category

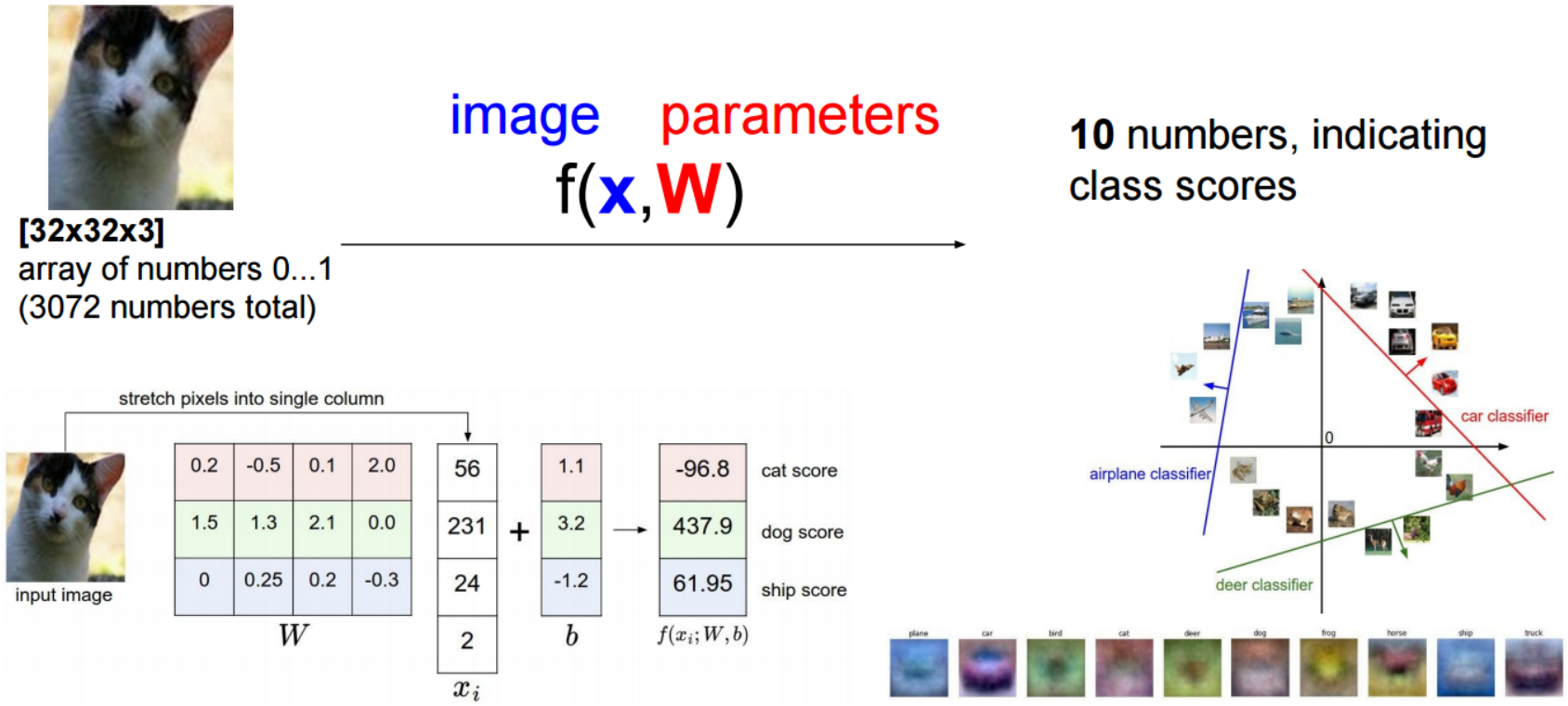


Example: image classification

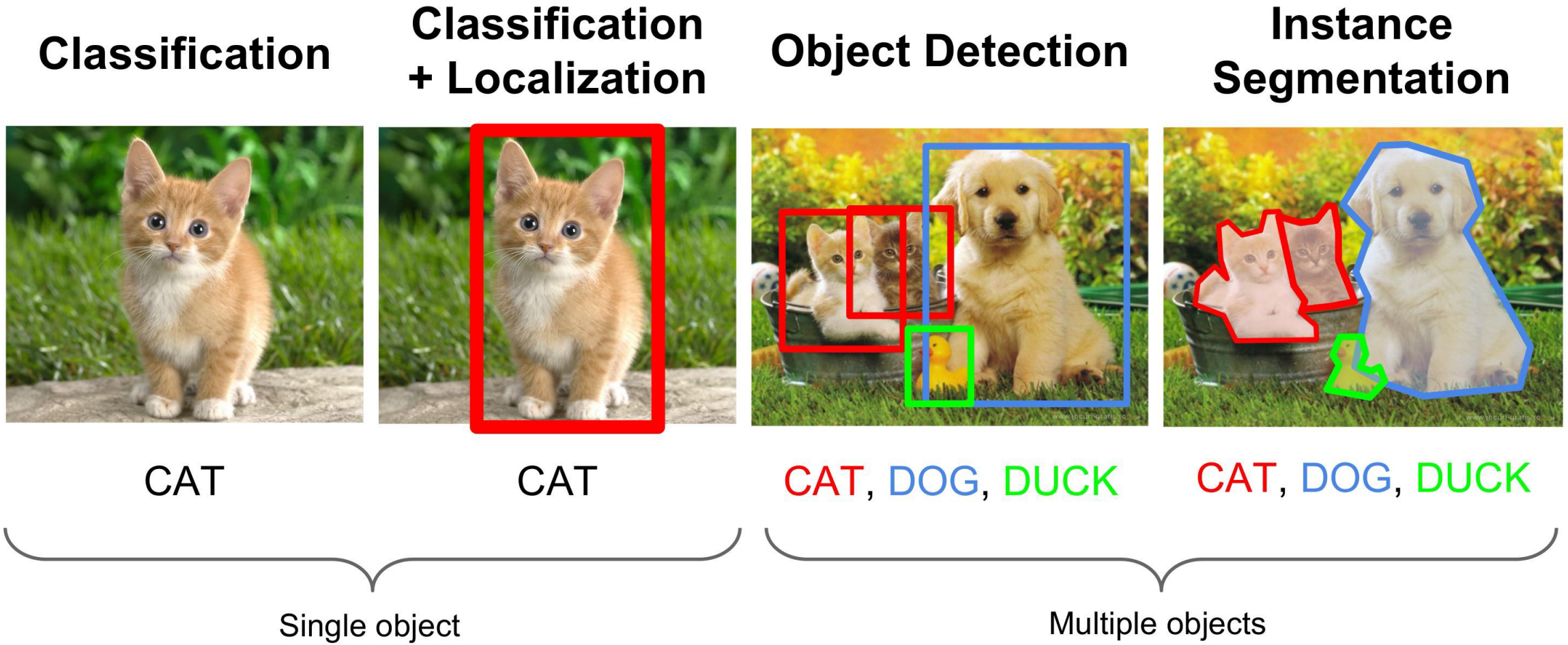
- Dataset： ImageNet
  - 14,197,122 images, 21841 synsets



Example: image classification

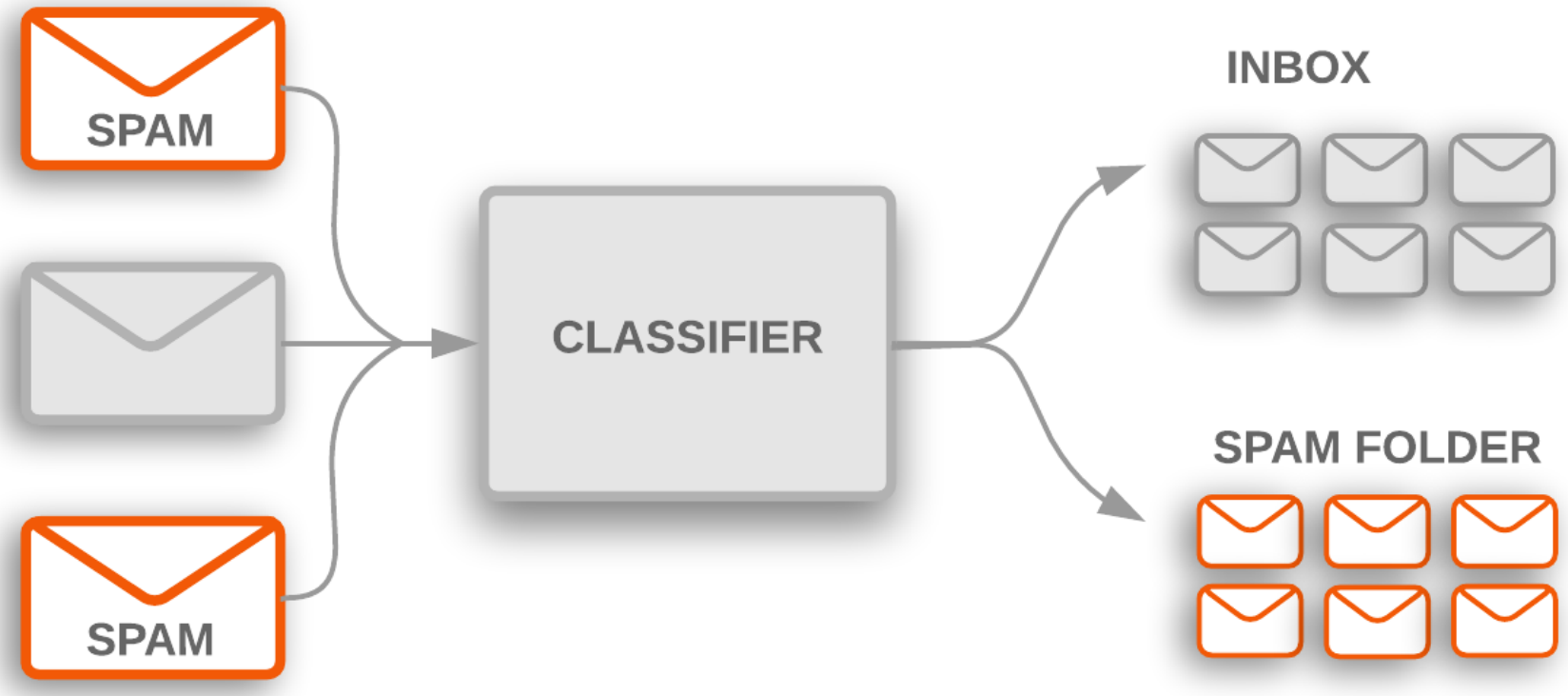


Examples: image classification, target detection, instance segmentation



https://medium.com/zylapp/review-of-deep-learning-algorithms-for-object-detection-c1f3d437b852

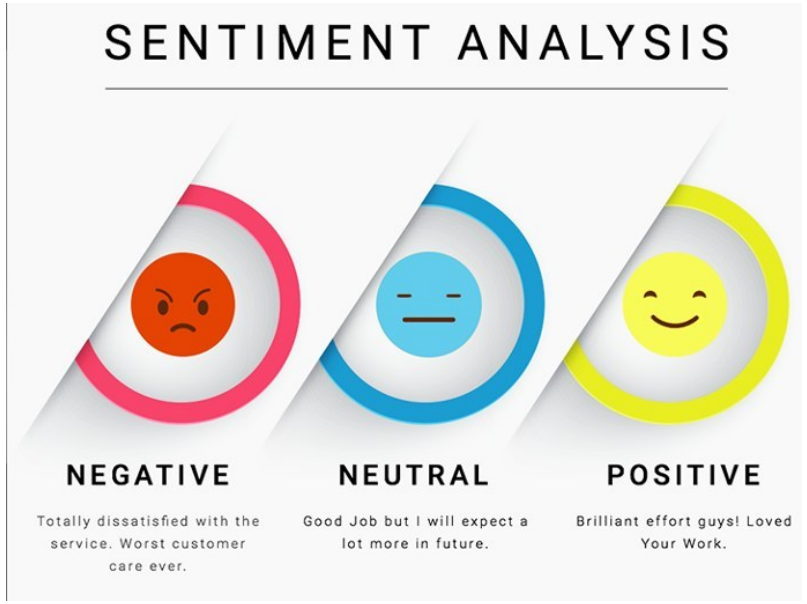
Example: Spam filtering



Example: document classification

https://towardsdatascience.com/automated-text-classification-using-machine-learning-3df4f4f9570b

Example: sentiment classification



Review (X)	Rating (Y)
"This movie is fantastic! I really like it because it is so good!"	★★★★☆
"Not to my taste, will skip and watch another movie"	★★☆☆☆
"This movie really sucks! Can I get my money back please?"	★☆☆☆☆

Example: text classification

- Convert sample x from text form to vector
  - Bag-of-Words (BoW) model

the dog is on the table

0	0	1	1	0	1	1	1
are	cat	dog	is	now	on	table	the

I like reading

I hate reading

I, like, hate, reading

4 words

$$\mathbf{v}_1 = [1\ 1\ 0\ 1\ ]^{\mathrm{T}},$$
$$\mathbf{v}_2 = [1\ 0\ 1\ 1\ ]^{\mathrm{T}}.$$

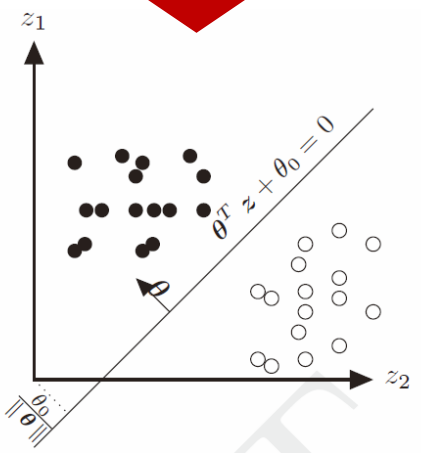
Example: Text sentiment classification

Determine the corresponding category of the text based on the text content

D1: I like reading

D2: I hate reading

	I	like	hate	reading
D1	1	1	0	1
D2	1	0	1	1

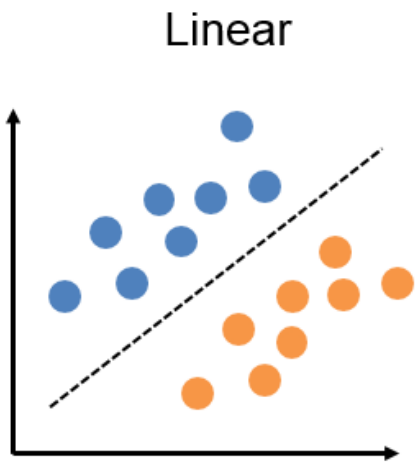


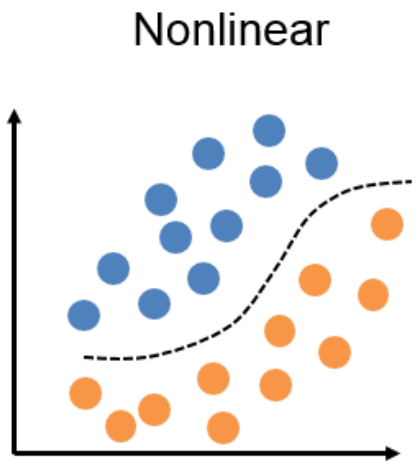
\+

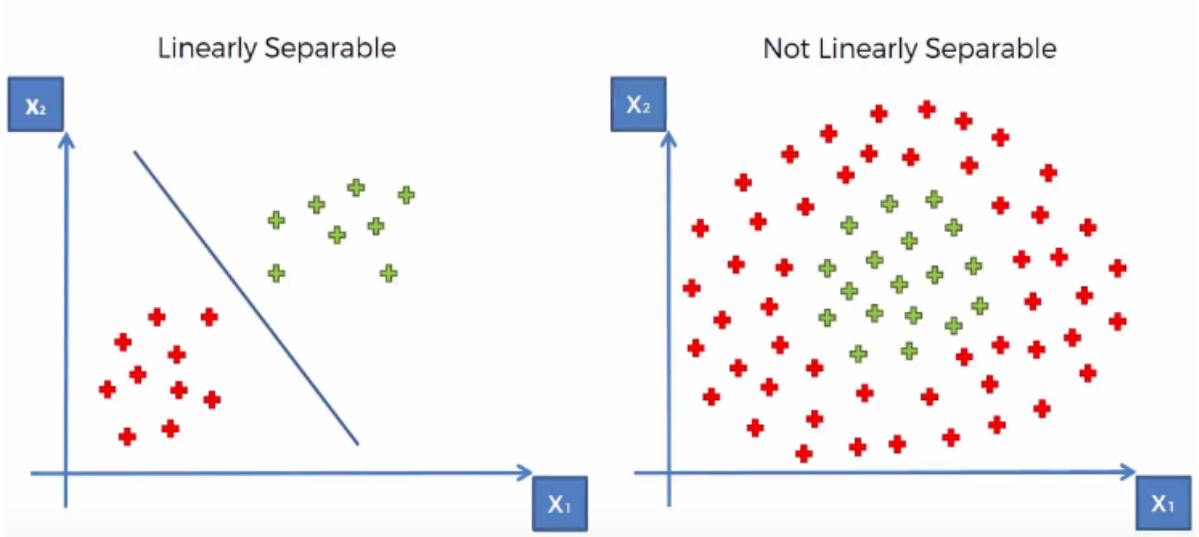
\-

Linear model

Linear and non-linear classifiers

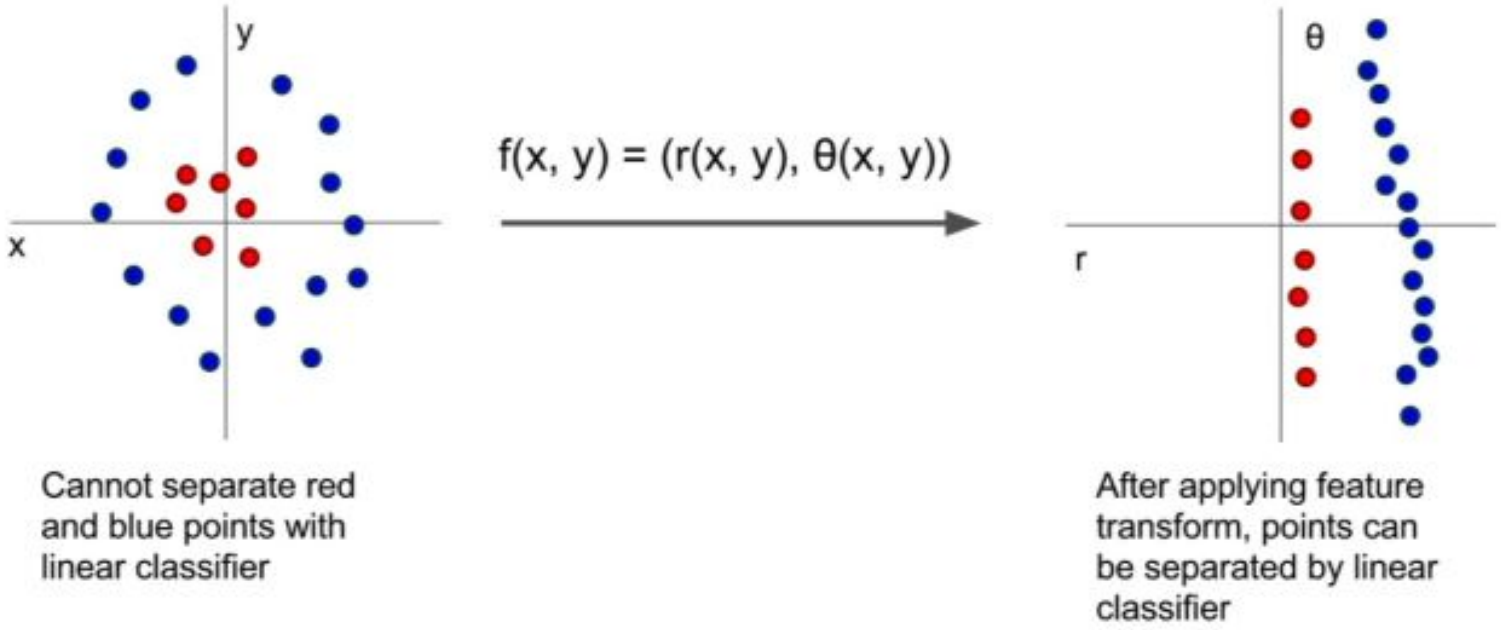






How to use a linear classifier to classify a set of linearly inseparable data?

- First grab some features of the group of data (higher dimensional space)
- Then classify these features linearly

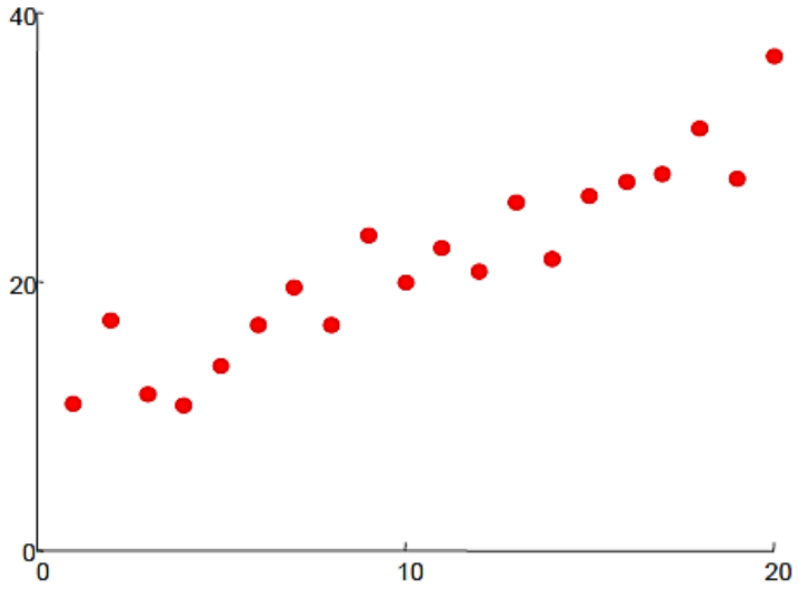


Linear Regression

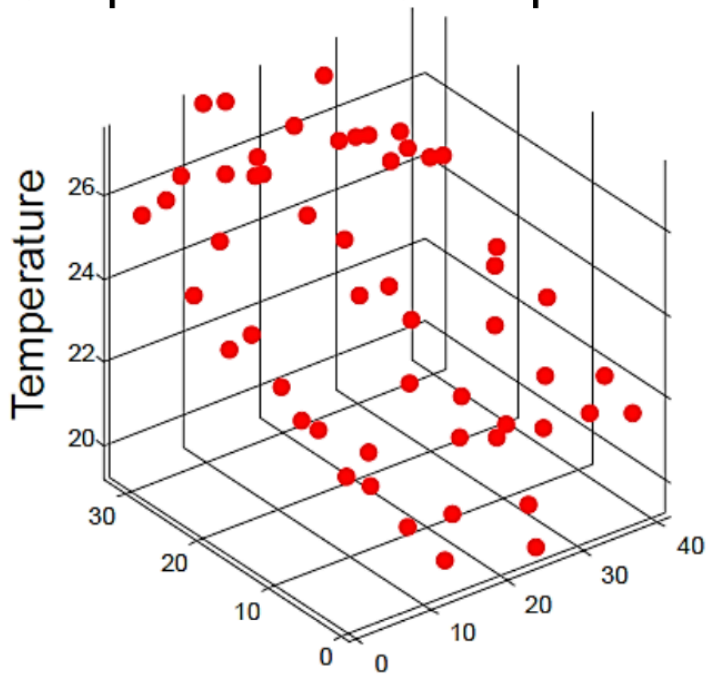
- First look at examples that help us understand intuitively:
  - Give you the following X and Y values, (1,1), (2,2), (4,4), (100,100), (20, 20), when X=5, what is Y?
  - Y = 5
- First, establish the relationship between X and Y through the given data, that is Y=X
- Then, using the established relationship, predict the unknown Y value by the given X value

Linear Regression

Samples with ONE independent variable



Samples with TWO independent variables

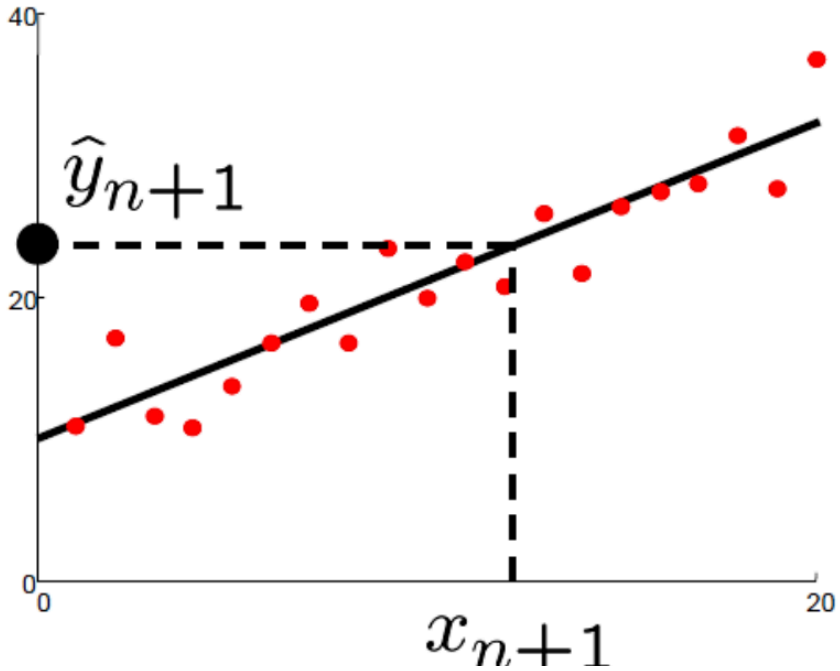


Given examples $(x_i, y_i)_{i=1...n}$

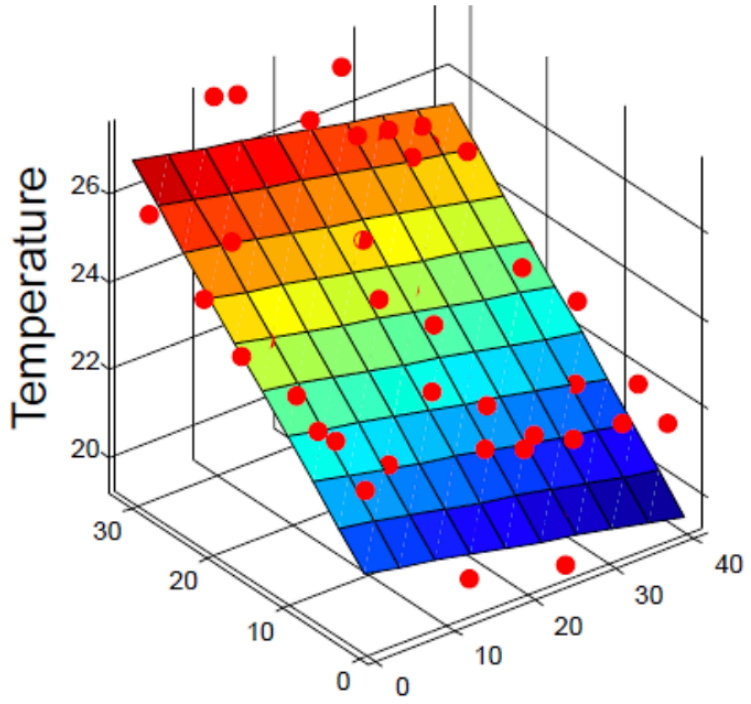
Predict y_{n+1} given a new point x_{n+1}

Linear Regression

Samples with ONE independent variable



Samples with TWO independent variables



Given examples $(x_i, y_i)_{i=1...n}$

Predict y_{n+1} given a new point x_{n+1}

Linear Regression

- Simple linear regression
  - A single independent variable is used to predict

- Multiple linear regression
  - Two or more independent variables are used to predict

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip} + \epsilon$$

where, for $i = n$ observations:

y_i = dependent variable

x_i = expanatory variables

β_0 = y-intercept (constant term)

β_p = slope coefficients for each explanatory variable

ϵ = the model's error term (also known as the residuals)

Define the basic elements in simple linear regression

- $x^{(i)}$ represents the input variable (independent variable), X in the first part example

- $y^{(i)}$ represents the output variable (dependent variable), Y in the first part of the example

- A pair of $(x^{(i)}, y^{(i)})$ represents a training sample

Define the basic elements in simple linear regression

- m training samples $(x^{(i)}, y^{(i)})$, i=1,2,…,m, are called training set
- i represents the i^{-th} sample
- The X represents the space composed of all input values.
- The Y represents the space composed of all output values

Linear regression

- Given a training set, our goal is to "learn" to get a function h: X->Y, so that h(x) is a "good" predicted value of the true value y

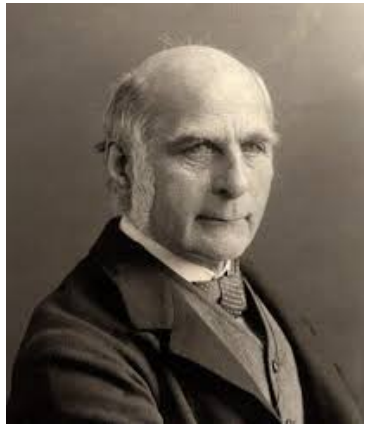
- $h_\theta(x^{(i)}) = \theta_0 + \theta_1 x^{(i)}$

  θ_0 and θ_1 represent the parameters of the model

- The goal of linear regression is to find the most suitable θ_0 and θ_1 to make the model effect the best

Linear regression

- Galton is a physiologist. In 1995, he studied the heights of 1,078 fathers and sons and found that they roughly satisfy a formula, that is
  - Y=0.8567+0.516*x
    - The x in this formula refers to the height of the father, and Y refers to the height of the son



Linear regression

- Galton is a physiologist. In 1995, he studied the heights of 1,078 fathers and sons and found that they roughly satisfy a formula, that is
  - Y=0.8567+0.516*x
    - The x in this formula refers to the height of the father, and Y refers to the height of the son

Linear regression

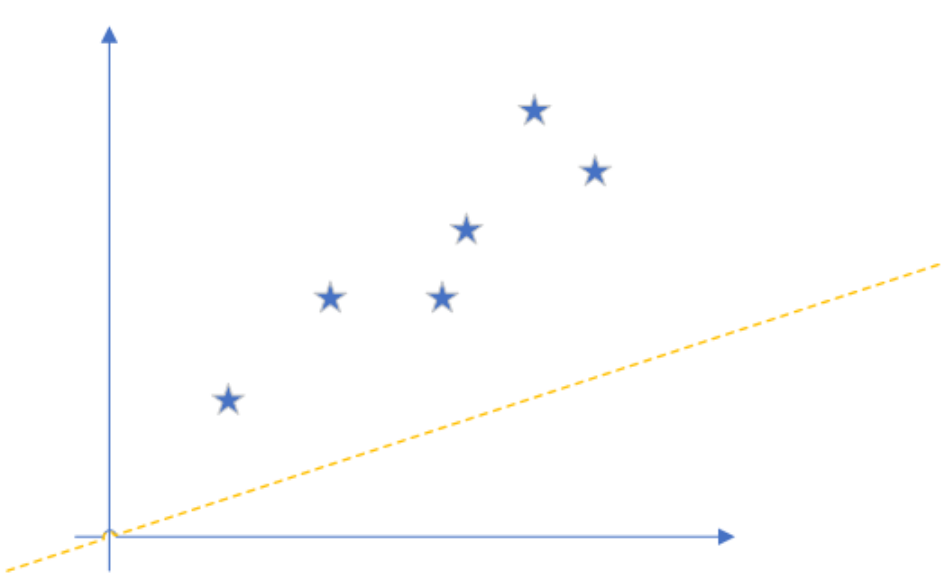
- Galton is a physiologist. In 1995, he studied the heights of 1,078 fathers and sons and found that they roughly satisfy a formula, that is
  - Y=0.8567+0.516*x
    - The x in this formula refers to the height of the father, and Y refers to the height of the son

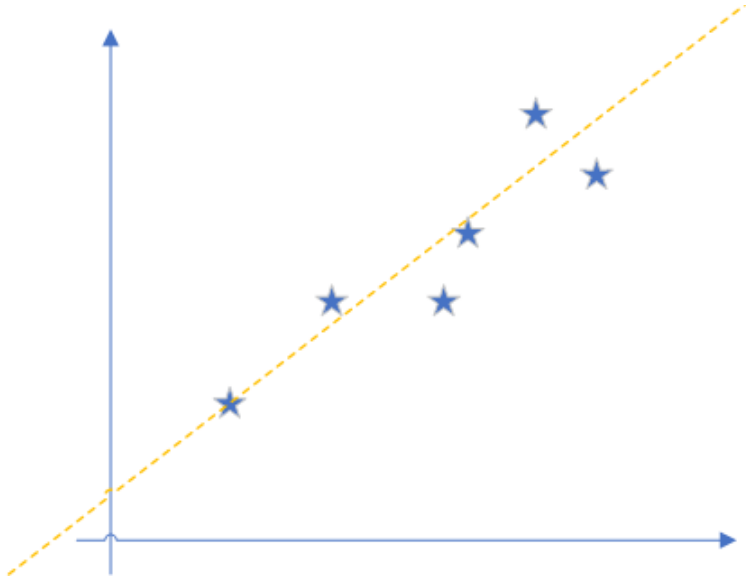
Linear regression

- Assuming there are some points like this, we want to find a line that fits these points, and then predict the next points
  - $y(x)=a_0 + a_1 * x$
- How to find a_1 and a_0?

Cost Function

- Figure 2 is more fitting, which means that the line in Figure 2 is closer to ideal line

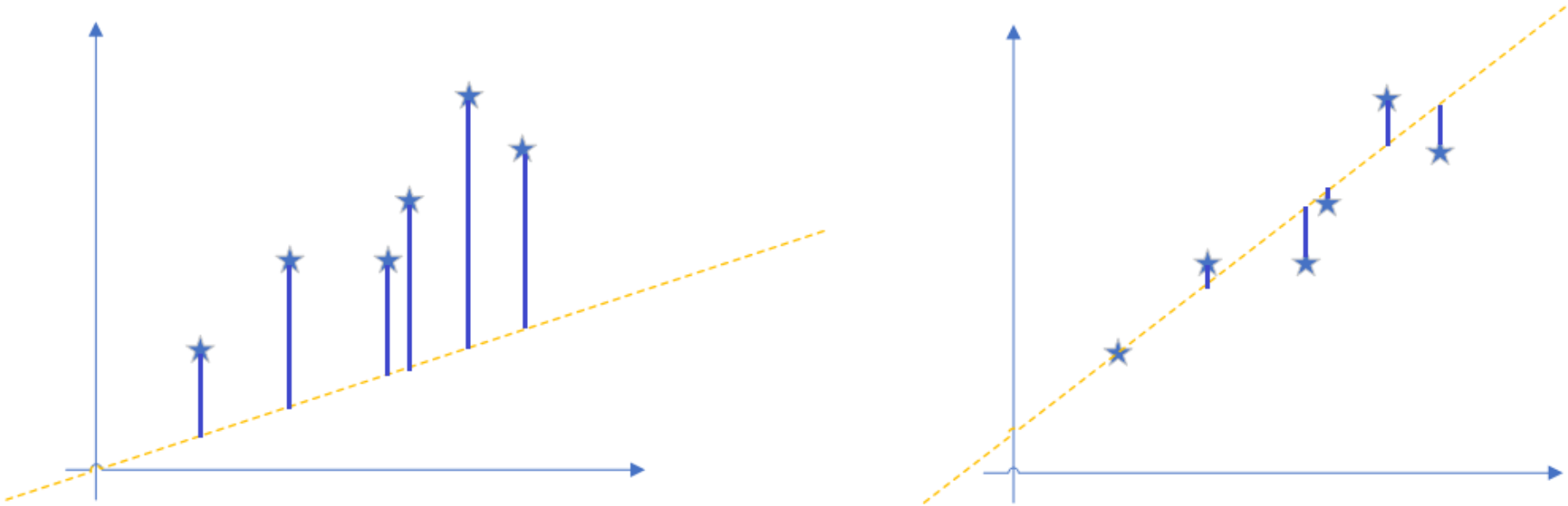




Cost Function

- Cost function: the error of all points along the y axis to a straight line

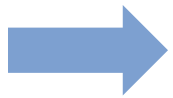
$$J = \frac{1}{n}\sum_{i=1}^{n}(pred_i - y_i)^2$$



Cost Function

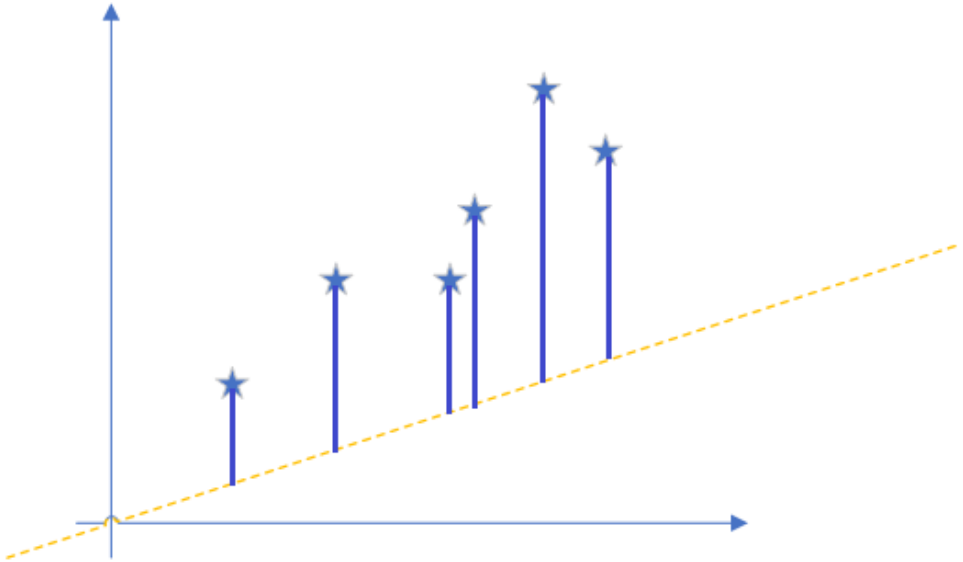
- Cost function: the error of all points along the y axis to a straight line

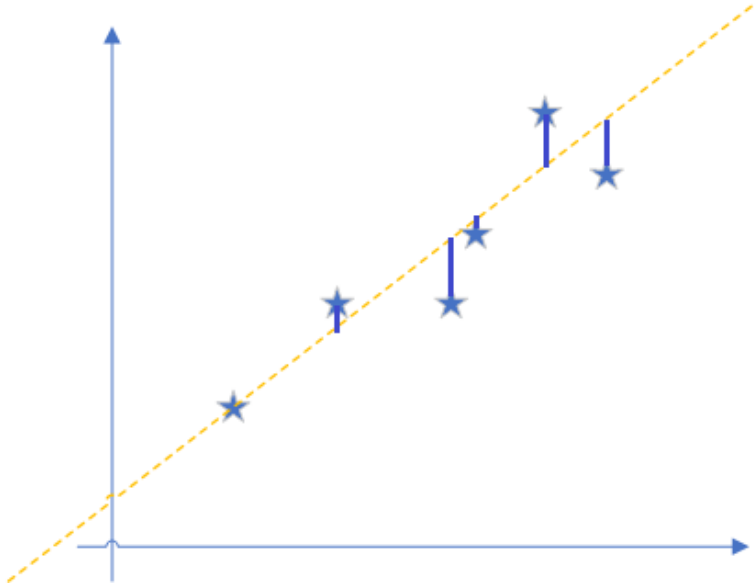
$$minimize \frac{1}{n} \sum_{i=1}^{n} (pred_i - y_i)^2$$



pred(x)=a_0 + a_1 * x

find a_1 and a_0





Cost Function

$$minimize \frac{1}{n} \sum_{i=1}^{n} (pred_i - y_i)^2$$ → pred(x)=a_0 + a_1 * x → find a_1 and a_0

$$J = \frac{1}{n} \sum_{i=1}^{n} (pred_i - y_i)^2$$

$$J = \frac{1}{n} \sum_{i=1}^{n} (a_0 + a_1 \cdot x_i - y_i)^2$$

$$\frac{\partial J}{\partial a_0} = \frac{2}{n} \sum_{i=1}^{n} (a_0 + a_1 \cdot x_i - y_i) \implies \frac{\partial J}{\partial a_0} = \frac{2}{n} \sum_{i=1}^{n} (pred_i - y_i)$$

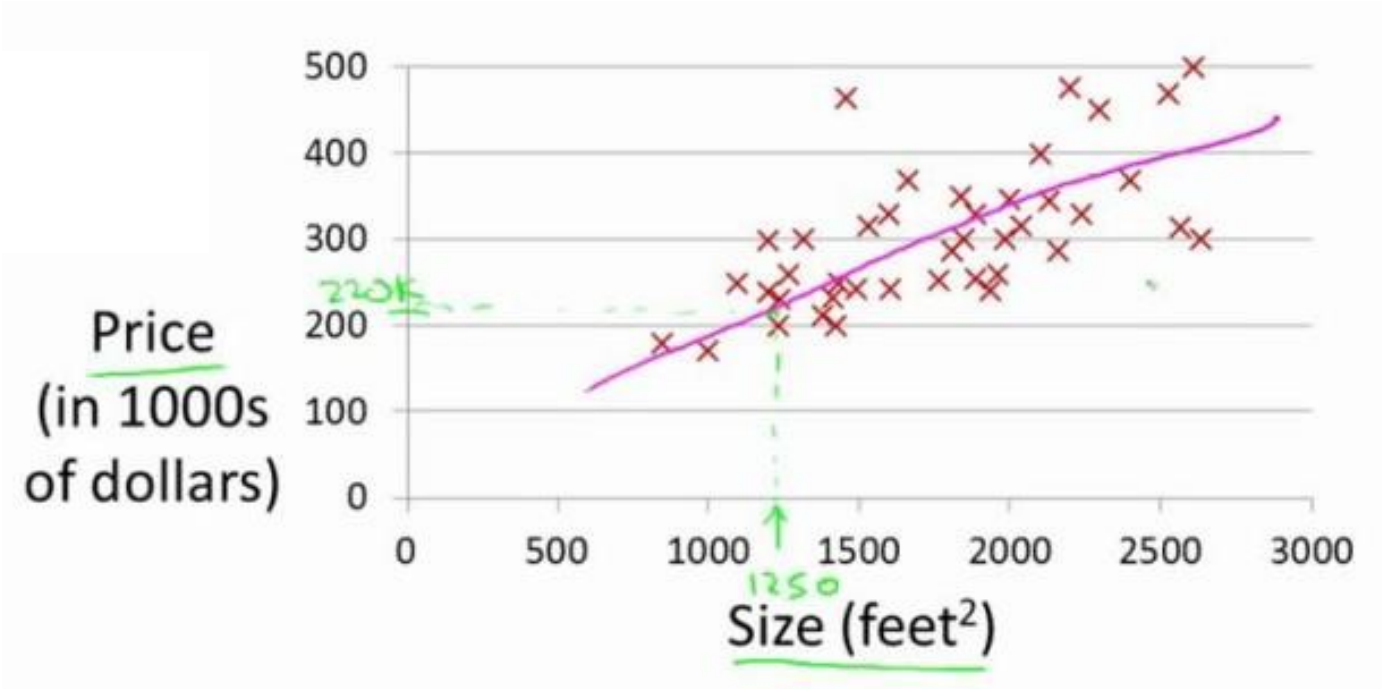
$$\frac{\partial J}{\partial a_1} = \frac{2}{n} \sum_{i=1}^{n} (a_0 + a_1 \cdot x_i - y_i) \cdot x_i \implies \frac{\partial J}{\partial a_1} = \frac{2}{n} \sum_{i=1}^{n} (pred_i - y_i) \cdot x_i$$

Derivation of the cost function to find a_1 and a_0

Linear regression example: predict house price

- If your friend's house is 1,250 square feet in size, tell them how much the house can sell

Size in feet² (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
…	…



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

Linear regression example: predict house price

Hypothesis:

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

Parameters:

$$\theta_0, \theta_1$$

Cost Function:

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

Goal:

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

Normal Equation

- Normal Equation is a method to find the most suitable parameter θ to minimize the cost function

- Suppose training set is $(x^{(i)}, y^{(i)})$, i=1,2,…,m

$\hat{\theta}^* = (X^T X)^{-1} X^T y$, where

$$X = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & \dots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & x_3^{(2)} & \dots & x_n^{(2)} \\ & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_1^{(m)} & x_2^{(m)} & x_3^{(m)} & \dots & x_n^{(m)} \end{bmatrix}$$, and $y = [y^{(1)}, \dots, y^{(m)}]$

Normal Equation

$$\mathbf{y} = \mathbf{Xb} + \mathbf{e}$$

$$\begin{aligned} J &= \mathbf{e}^{\mathrm{T}}\mathbf{e} = (\mathbf{y} - \mathbf{Xb})'(\mathbf{y} - \mathbf{Xb}) \\ &= (\mathbf{y}' - \mathbf{b}'\mathbf{X}')(\mathbf{y} - \mathbf{Xb}) \\ &= \mathbf{y}'\mathbf{y} - \mathbf{y}'\mathbf{Xb} - \mathbf{b}'\mathbf{X}'\mathbf{y} + \mathbf{b}'\mathbf{X}'\mathbf{Xb} \\ &= \mathbf{y}'\mathbf{y} - 2\mathbf{b}'\mathbf{X}'\mathbf{y} + \mathbf{b}'\mathbf{X}'\mathbf{Xb} \end{aligned}$$

https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf

$$\frac{\partial \mathbf{e}'\mathbf{e}}{\partial \mathbf{b}} = -2\mathbf{X}'\mathbf{y} + 2\mathbf{X}'\mathbf{Xb} = 0$$

$$(\mathbf{X}'\mathbf{X})\mathbf{b} = \mathbf{X}'\mathbf{y}$$

$$\mathbf{b} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$$

Example of Normal Equation

- Find a straight line to fit the points (1,1), (2,2), (3,2)

suppose $h(\theta) = \theta_0 + \theta_1 x_1$

X = $A = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix}$ $A^T = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{bmatrix}$ $A^T A = \begin{bmatrix} 3 & 6 \\ 6 & 14 \end{bmatrix}$

y = $\overrightarrow{h(\theta)} = (1, 2, 2)^T$

$\overrightarrow{\theta^*} = (A^T A)^{-1} A^T \overrightarrow{h(\theta^*)}$

Example of Normal Equation

- Find a straight line to fit the points (1,1), (2,2), (3,2)

suppose $h(\theta) = \theta_0 + \theta_1 x_1$

$$\overrightarrow{\theta^*} = \begin{pmatrix} \frac{2}{3} \\ \\ \frac{1}{2} \end{pmatrix}$$

That is, the straight line $h(x) = \frac{2}{3} + \frac{1}{2}x$ is the fitting result of the above three points.

Advantage of Normal Equation

- Can be solved in one time

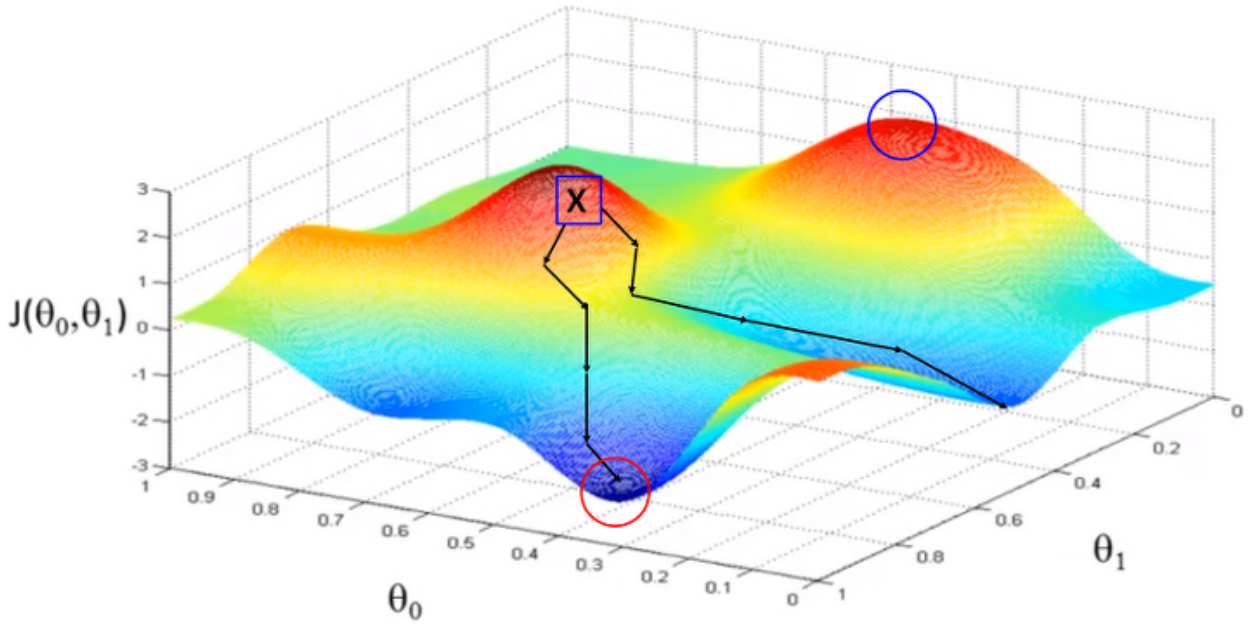
Disadvantage of Normal Equation

- Large amount of calculation.
  - Especially when the dimension of the input variable is large (n is relatively large), the computational complexity of the algorithm increases exponentially.

- Take up resources.
  - Since the results need to be calculated in the memory at one time, when the amount of data is relatively large, the algorithm requires a lot of computer resources.

Gradient Descent

- Gradient descent is another method of solving parameters θ



- The blue area in the figure is the point with the smallest cost function. How to find the corresponding lowest point θ ?

Gradient Descent

- How to find the corresponding lowest point θ ?

- The answer is to take the partial derivative of the cost function:

$$\frac{\partial}{\partial\theta_j}J(\theta) = \frac{\partial}{\partial\theta_j}\frac{1}{2}(h_\theta(x)-y)^2 = \frac{1}{2}\times 2\times(h_\theta(x)-y)\frac{\partial}{\partial\theta_j}(\sum_{i=0}^{n}\theta_i x_i - y) = (h_\theta(x)-y)x_j$$

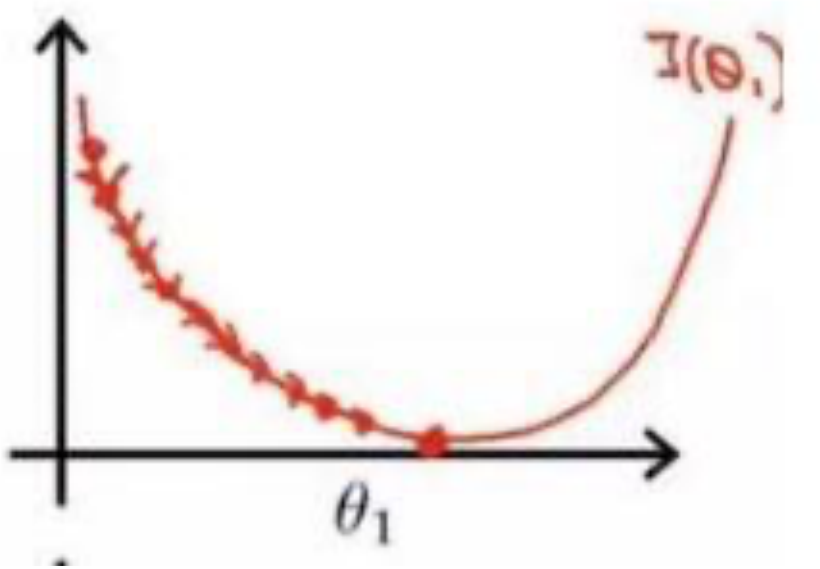
- The meaning of finding the partial derivative is to get the slope of the tangent at this point, which will give us a direction to move towards the minimum. Keep moving until convergence.

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

Learning rate

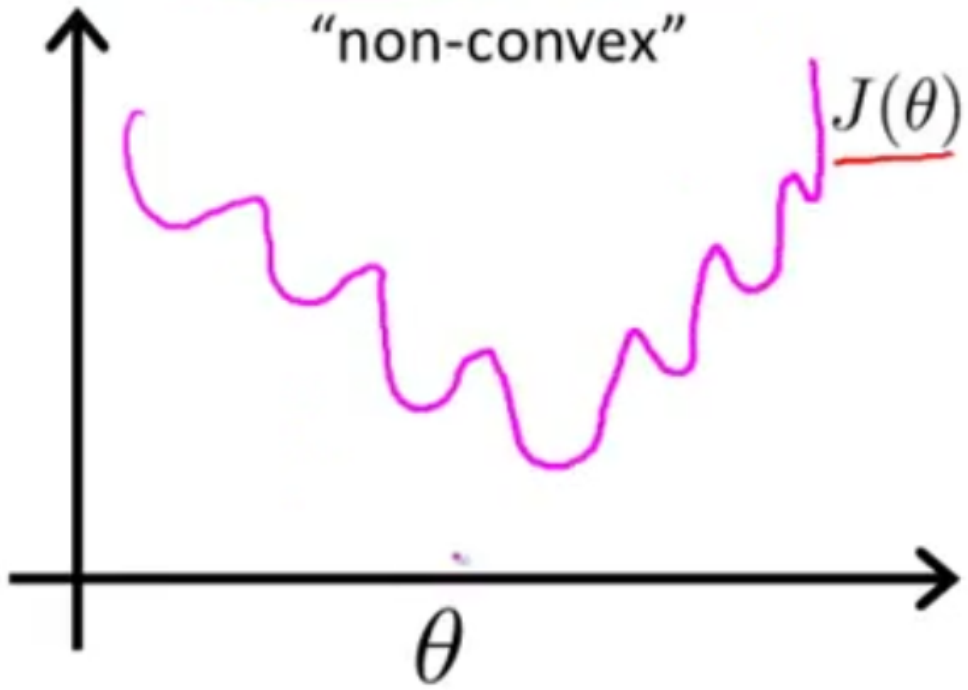
- The move step is called learning rate

- When the learning rate is relatively small, the speed at which we get the optimal solution will be very slow.



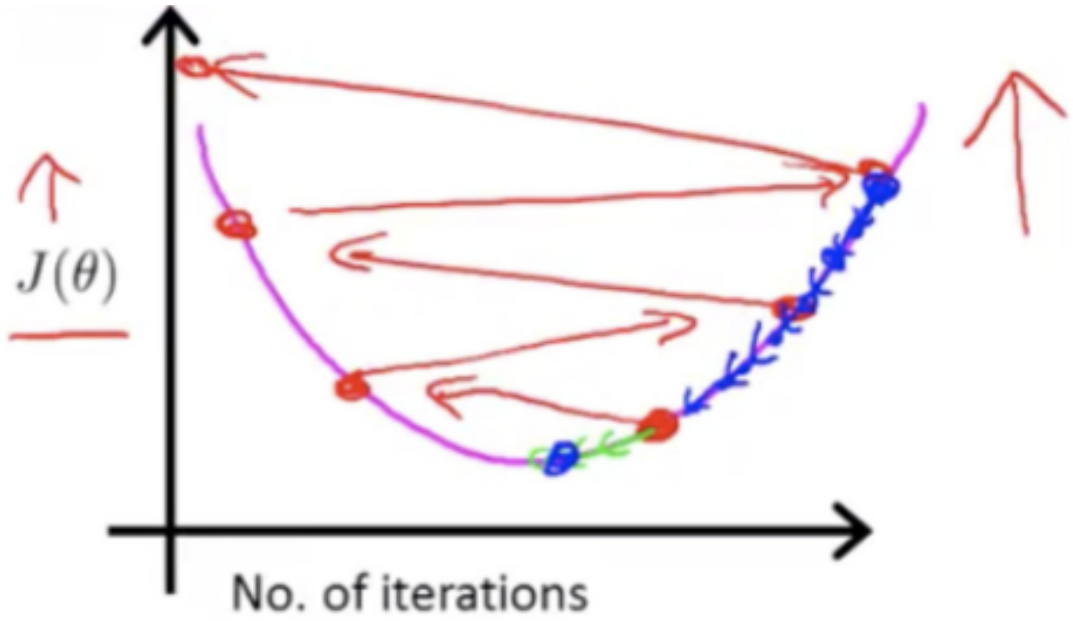
Learning rate

- If our learning rate setting is relatively small, and we get a local optimal solution instead of the global optimal solution.

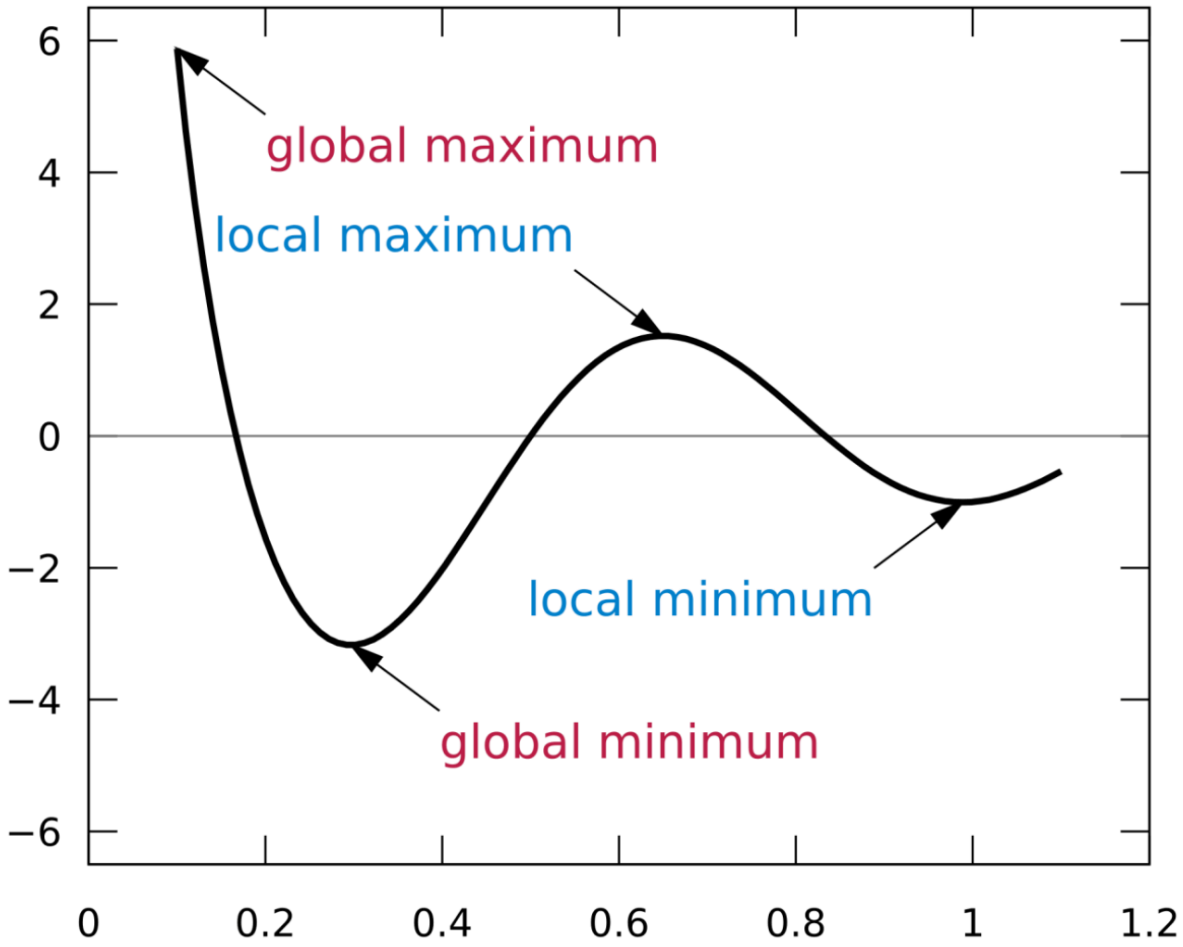


Learning rate

- When the learning rate is relatively large, we can easily not get the global optimal solution.



Global/Local Optimization



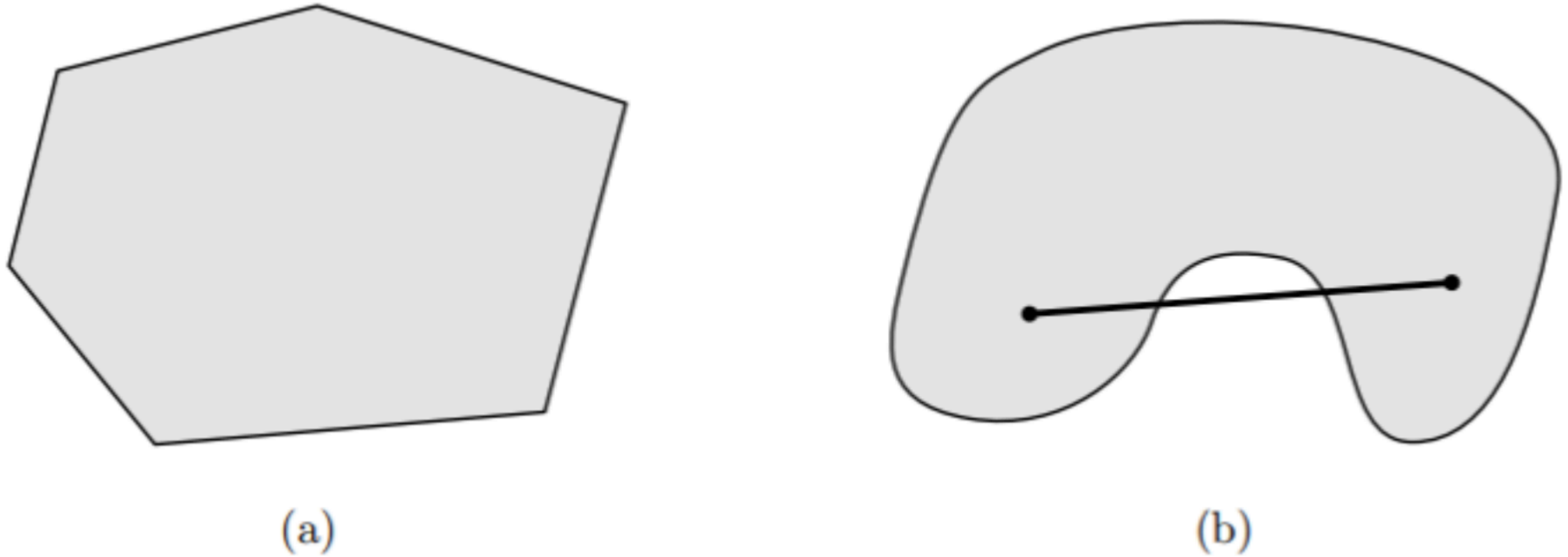
Convex set

- Definition
  - A set C is convex if, for any $x, y \in C$ and $\theta \in \Re$

    with $0 \leq \theta \leq 1$, $\theta x + 1 - \theta\ y \in C$

  - If we take any two elements in C, and draw a line segment between these two elements, then every point on that line segment also belongs to C

Convex set



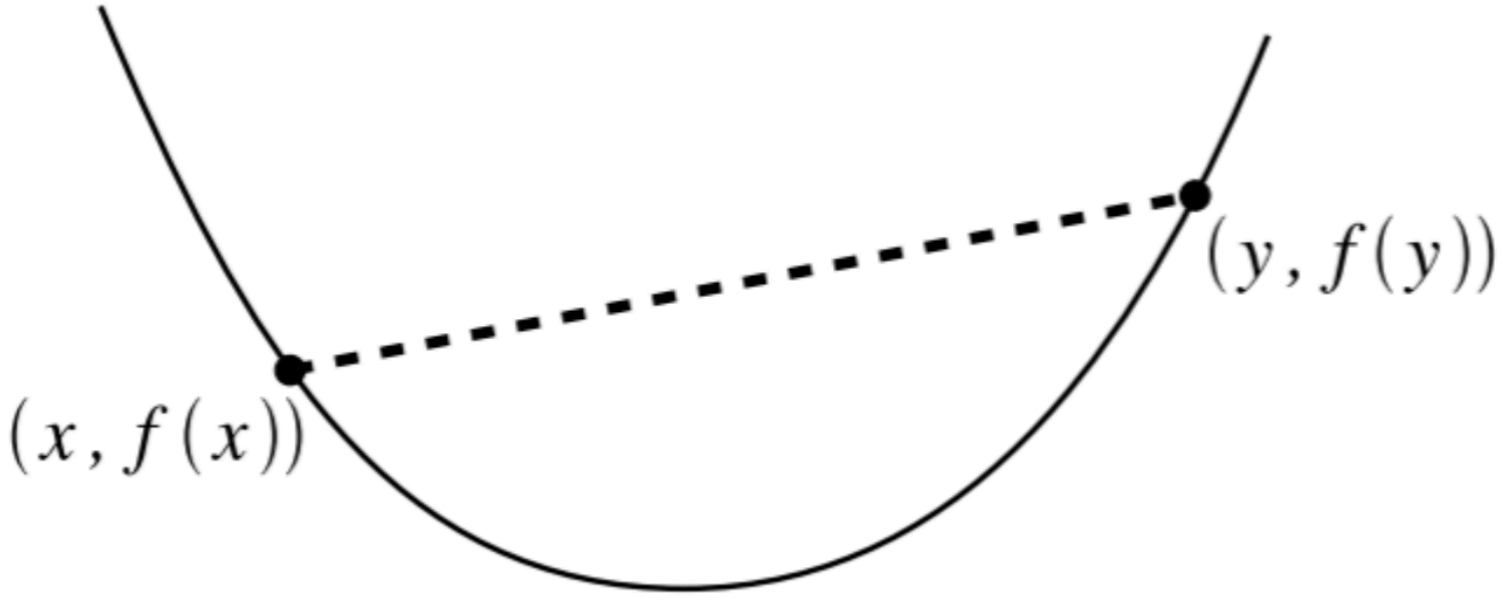
Examples of a convex set (a) and a non-convex set

Convex functions

- Definition
  - A function $f: \Re_n \rightarrow \Re$ is convex if its domain (denoted $D(f)$) is a convex set, and if, for all $x, y \in D(f)$ and $\theta \in R$, $0 \leq \theta \leq 1$, $f\ (\theta x + (1 - \theta)\ y) \leq \theta f\ (x) + (1 - \theta)\ f(y)$.

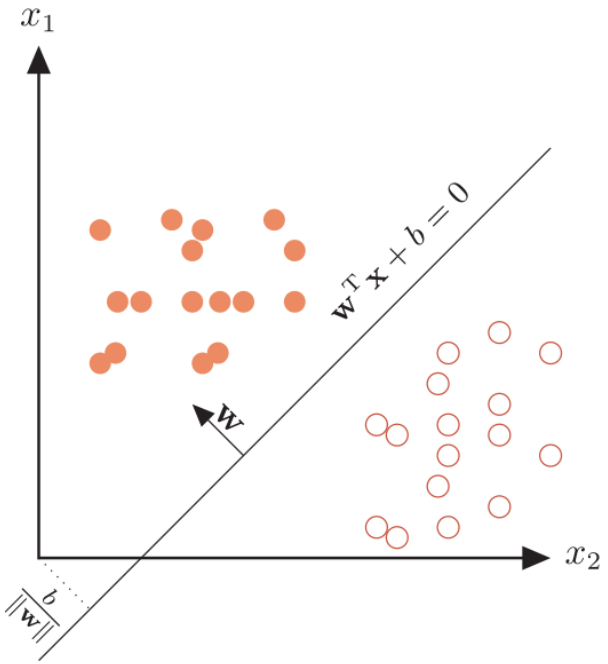
  - If we pick any two points on the graph of a convex function and draw a straight line between them, the n the portion of the function between these two points will lie below this straight line

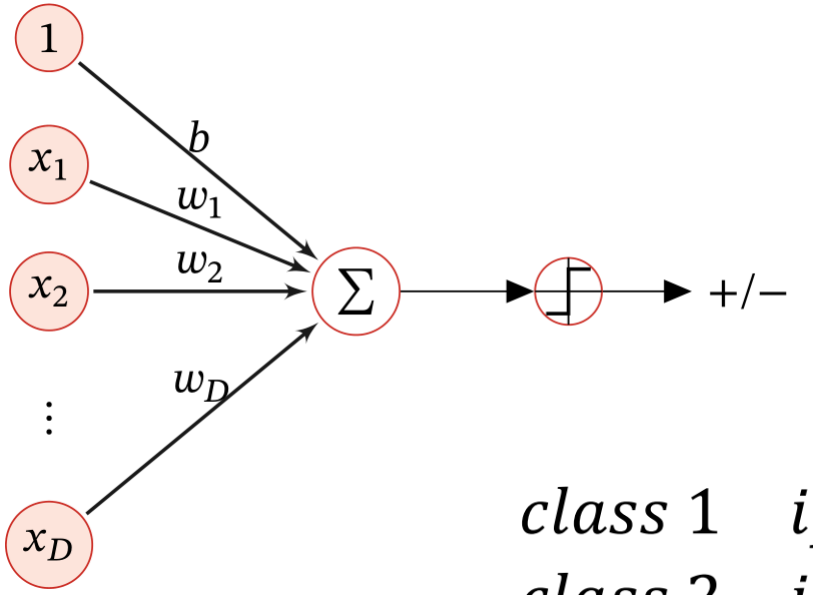
Convex function



The line connecting two points on the graph must lie above the function

Linear model for classification





$$class\ 1 \quad if\ f(x) \geq 0$$
$$class\ 2 \quad if\ f(x) < 0$$

Multivariate linear regression

Multivariate linear regression

- Simple linear regression
  - A single independent variable is used to predict

- Multiple linear regression
  - Two or more independent variables are used to predict

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip} + \epsilon$$

where, for $i = n$ observations:

y_i = dependent variable

x_i = expanatory variables

β_0 = y-intercept (constant term)

β_p = slope coefficients for each explanatory variable

ϵ = the model's error term (also known as the residuals)

Multivariate linear regression definition

- $x_j^{(i)}$ represents the i-th training sample value of the j-th input variable
- $x^{(i)}$ represents all the input variable values of the i-th training sample.
- m represents the number of samples
- n represents the number of input variables
- Multiple linear regression model is

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \cdots + \theta_n x_n$$

Multivariate linear regression definition

- Multiple linear regression model is

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \cdots + \theta_n x_n$$

- If written in the form of a matrix

$$h_\theta(x) = \begin{bmatrix} \theta_0 & \theta_1 & \dots & \theta_n \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} = \theta^T x$$

Multivariate linear regression solution → θ

- Solution 1: Extend the gradient descent formula to each parameter:

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial \theta_j} J(\theta) = (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \qquad for\ j := 0 \ldots n$$

- Reference: single linear regression

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

$$\frac{\partial}{\partial \theta_0} J(\theta) = (h_\theta(x) - y) x_0 = (h_\theta(x) - y)$$

$$\frac{\partial}{\partial \theta_1} J(\theta) = (h_\theta(x) - y) x_1$$

Multivariate linear regression solution → θ

- Solution 2: based on the least square method:

$$\hat{\theta} = [\theta_0 \; \theta_1 \; \ldots \; \theta_n]$$

$$X = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & \ldots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & x_3^{(2)} & \ldots & x_n^{(2)} \\ & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_1^{(m)} & x_2^{(m)} & x_3^{(m)} & \ldots & x_n^{(m)} \end{bmatrix}$$

$$\hat{y} = X\hat{\theta}$$

- Then

$$\hat{\theta}^* = (X^T X)^{-1} X^T y$$

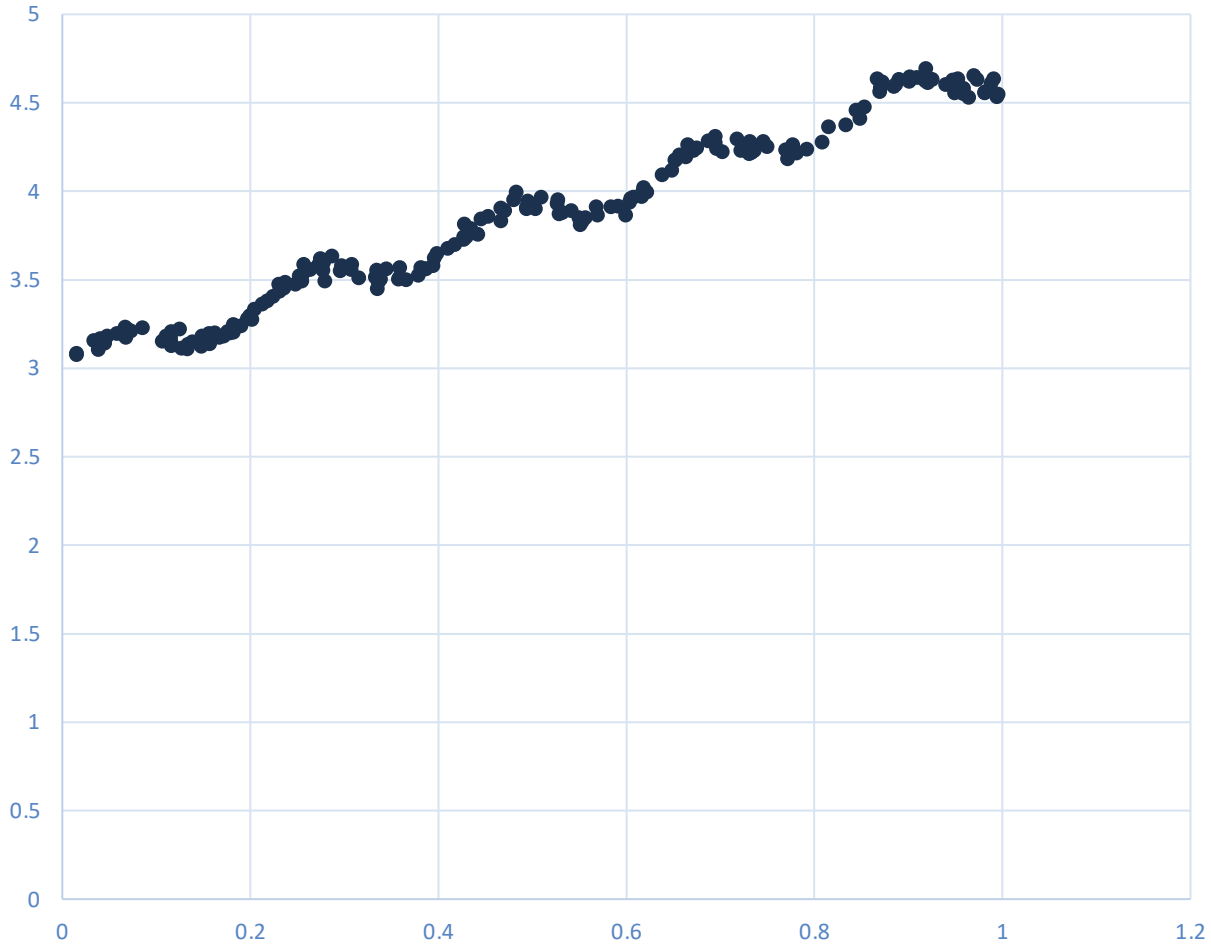
Linear regression example

https://github.com/kevinsuo/CS7357/blob/master/linear-regression/data.txt

1	1.000000	0.067732	3.176513
2	1.000000	0.427810	3.816464
3	1.000000	0.995731	4.550095
4	1.000000	0.738336	4.256571
5	1.000000	0.981083	4.560815
6	1.000000	0.526171	3.929515
7	1.000000	0.378887	3.526170
8	1.000000	0.033859	3.156393
9	1.000000	0.132791	3.110301
10	1.000000	0.138306	3.149813
11	1.000000	0.247809	3.476346
12	1.000000	0.648270	4.119688
13	1.000000	0.731209	4.282233
14	1.000000	0.236833	3.486582
15	1.000000	0.969788	4.655492
16	1.000000	0.607492	3.965162
17	1.000000	0.358622	3.514900
18	1.000000	0.147846	3.125947
19	1.000000	0.637820	4.094115

Linear regression example

https://github.com/kevinsuo/CS7357/blob/master/linear-regression/data.txt



Linear regression example

- The loadDataSet() function opens a text file separated by the tab key, where the last value of each line of the default file is the target value

```
#The loadDataSet() function opens a text file separated by the tab key,
#where the last value of each line of the default file is the target value.
def loadDataSet(fileName):
    numFeat = len(open(fileName).readline().split('\t')) - 1
    dataMat = []; labelMat = []
    fr = open(fileName)
    for line in fr.readlines():
        lineArr =[]
        curLine = line.strip().split('\t')
        for i in range(numFeat):
            lineArr.append(float(curLine[i]))
        dataMat.append(lineArr)
        labelMat.append(float(curLine[-1]))
    return dataMat,labelMat
```

Linear regression example

```
xArr,yArr = loadDataSet('data.txt')
```

```
>>> xArr[0:2]
[[1.0, 0.067732], [1.0, 0.42781]]
```

The first value is always equal to 1.0

The second value is the x-axis value in the figure below.

yArr

xArr

1	1.000000	0.067732	3.176513
2	1.000000	0.427810	3.816464
3	1.000000	0.995731	4.550095
4	1.000000	0.738336	4.256571
5	1.000000	0.981083	4.560815
6	1.000000	0.526171	3.929515
7	1.000000	0.378887	3.526170
8	1.000000	0.033859	3.156393
9	1.000000	0.132791	3.110301
10	1.000000	0.138306	3.149813
11	1.000000	0.247809	3.476346
12	1.000000	0.648270	4.119688
13	1.000000	0.731209	4.282233
14	1.000000	0.236833	3.486582
15	1.000000	0.969788	4.655492
16	1.000000	0.607492	3.965162
17	1.000000	0.358622	3.514900
18	1.000000	0.147846	3.125947
19	1.000000	0.637820	4.094115

Linear regression example

- The standRegres() function is used to calculate the best fit straight line.
- This function first reads x and y and saves them in the matrix, then calculates xTx, and then judges whether its determinant is 0. If the determinant is 0, an error will occur when calculating the inverse matrix.

```
19  #The standRegres() function is used to calculate the best fit straight line.
20  #This function first reads x and y and saves them in the matrix, then calculates xTx, and then jud
21  #If the determinant is 0, an error will occur when calculating the inverse matrix.
22  def standRegres(xArr,yArr):
23      xMat = np.mat(xArr); yMat = np.mat(yArr).T
24      xTx = xMat.T*xMat
25      if np.linalg.det(xTx) == 0.0:
26          print ("This matrix is singular, cannot do inverse")
27          return
28      ws = xTx.I * (xMat.T*yMat)
29      return ws
```

Linear regression example

- Take a look at the execution effect of the standRegres() function:

```
>>> ws = standRegres(xArr,yArr)
>>> ws
matrix([[ 3.00774324],
        [ 1.69532264]])
```

The variable ws stores the regression coefficient k and b

Input variable x_1, get y=ws[0]+ws[1]*x_1, where y is actually predicted

Linear regression example

- The following uses the new ws value to calculate the predicted value yHat

```
>>> xMat = np.mat(xArr)
>>> yMat = np.mat(yArr)
>>> yHat = xMat*ws
```

- Plot the data set scatter plot and best fit straight line plot

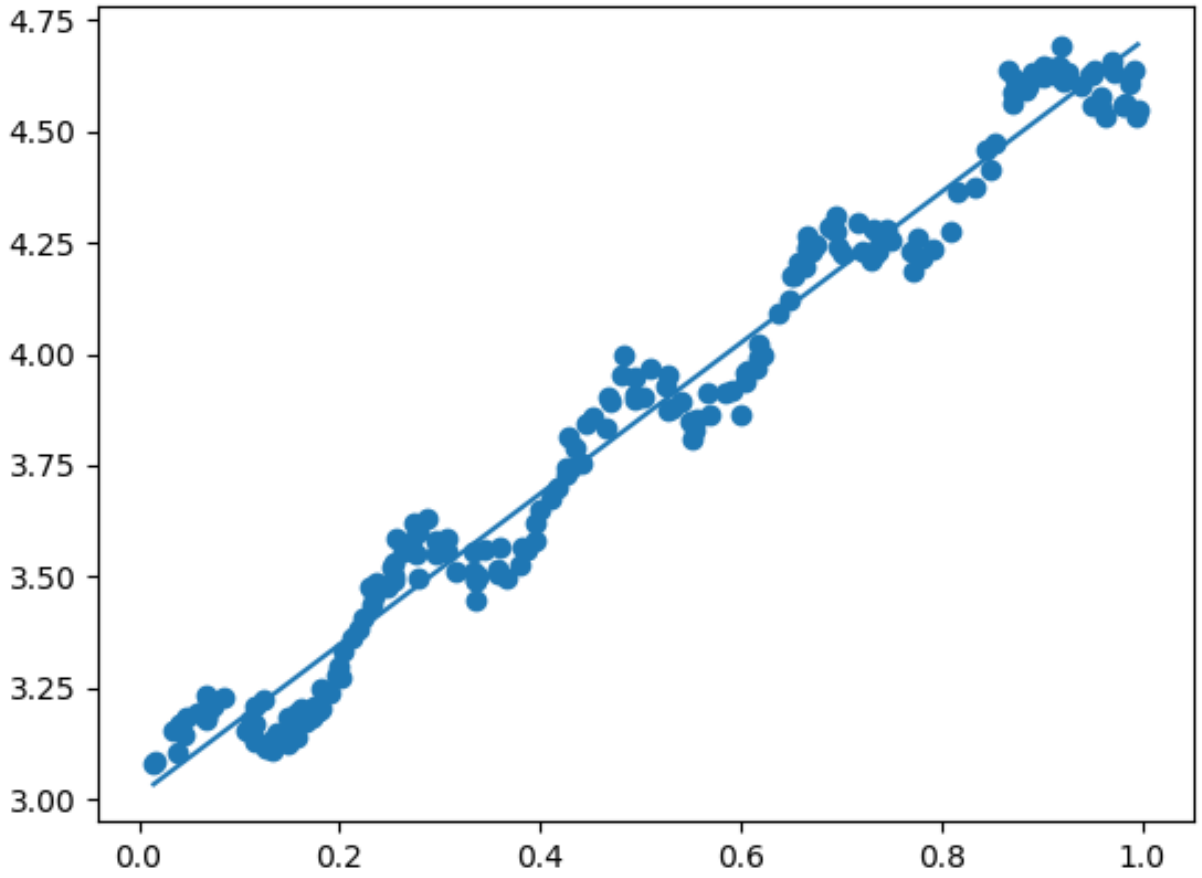
```
>>> fig = plt.figure()
>>> ax = fig.add_subplot(111)
>>> ax.scatter(xMat[:,1].flatten().A[0],yMat.T[:,0].flatten().A[0])
```

Linear regression example

- If the order of the data points on the straight line is out of order, there will be problems when drawing, so the points should be arranged in ascending order

```
>>> xCopy = xMat.copy()
>>> xCopy.sort(0)
>>> yHat = xCopy*ws
>>> ax.plot(xCopy[:,1],yHat)
[<matplotlib.lines.Line2D object at 0x000002041E0AAF98>]
>>> plt.show()
```

Linear regression example



Linear regression example 2

- Example: There are 20 students. To study the correlation between the review time and the scores obtained by these students before the exam

```
from collections import OrderedDict
import pandas as pd
examDict={
    'Study time':[0.50,0.75,1.00,1.25,1.50,1.75,1.75,2.00,2.25,
            2.50,2.75,3.00,3.25,3.50,4.00,4.25,4.50,4.75,5.00,5.50],
    'Score':    [10,  22,  13,  43,  20,  22,  33,  50,  62,
             48,  55,  75,  62,  73,  81,  76,  64,  82,  90,  93]
}
examOrderDict=OrderedDict(examDict)
exam=pd.DataFrame(examOrderDict)
```

Output result:

```
>>> exam.head()
0  0.50  10
1  0.75  22
2  1.00  13
3  1.25  43
4  1.50  20
```

Linear regression example 2

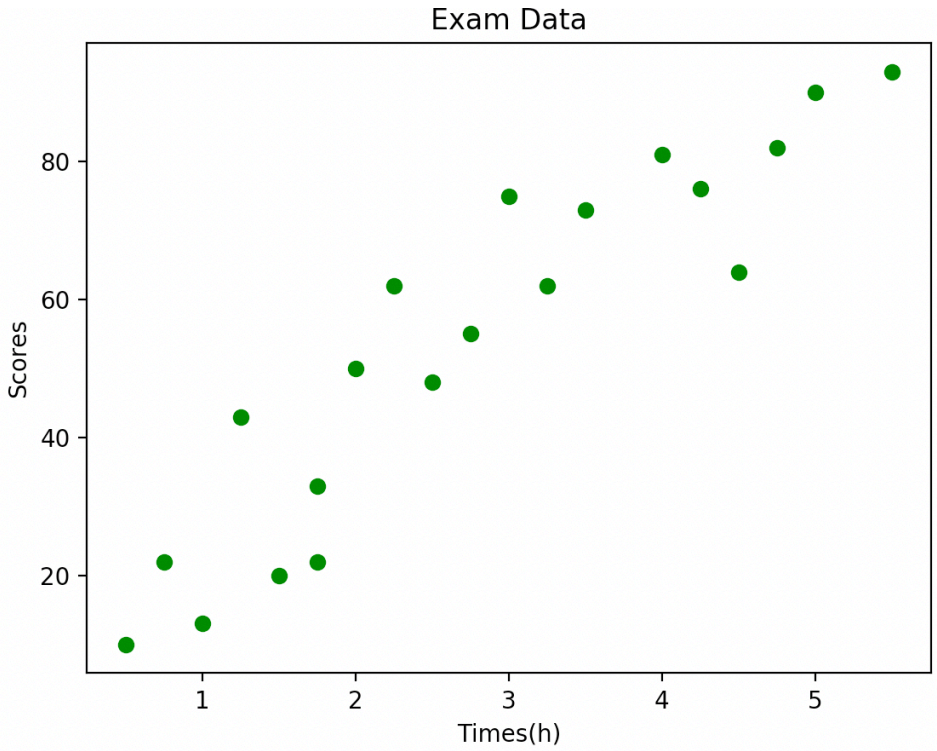
- Judge whether it is suitable for linear regression model by drawing

```
from collections import OrderedDict
import matplotlib.pyplot as plt
import pandas as pd
examDict={
    'Study time':[0.50,0.75,1.00,1.25,1.50,1.75,1.75,2.00,2.25,
                  2.50,2.75,3.00,3.25,3.50,4.00,4.25,4.50,4.75,5.00,5.50],
    'Score':    [10,  22,  13,  43,  20,  22,  33,  50,  62,
                 48,  55,  75,  62,  73,  81,  76,  64,  82,  90,  93]
}
examOrderDict=OrderedDict(examDict)
exam=pd.DataFrame(examOrderDict)


exam_X = exam['Study time']
exam_Y = exam['Score']


plt.scatter(exam_X, exam_Y, color = 'green')
plt.ylabel('Scores')
plt.xlabel('Times(h)')
plt.title('Exam Data')

plt.show()
```



This data set is suitable for linear regression models

Linear regression example 2

https://github.com/kevinsuo/CS7357/blob/master/linear-regression/lr.py

- Divide training set and test set

```
from sklearn.model_selection import train_test_split
# The first parameter is the feature, the second parameter is the label

X_train, X_test, Y_train, Y_test = train_test_split(exam_X,exam_Y, train_size = 0.7)
# Ratio is 7:3

# Create a model
model = LinearRegression()

# Train the model
model.fit(X_train, Y_train)
```

Linear regression example 2

- Model parameters

```
#Because the general equation of linear regression is y = a+bx
#b is the slope, a is the intercept

#Intercept with intercept_
#Slope usemodel.coef_
a = model.intercept_
b = model.coef_
a = float(a)
b = float(b)
```

Linear regression example 2

https://github.com/kevinsuo/CS7357/blob/master/linear-regression/lr.py

- Model parameters

```
print('The linear regression equation of the model is y = {} + {} * x'.format(a, b))
```

```
ksuo@ltksup50143mac ~/G/g/C/linear-regression> python3 lr.py
The linear regression equation of the model is y = 8.838671411625121 + 15.909845788849356 * x
```

Linear regression example 2

https://github.com/kevinsuo/CS7357/blob/master/linear-regression/lr.py

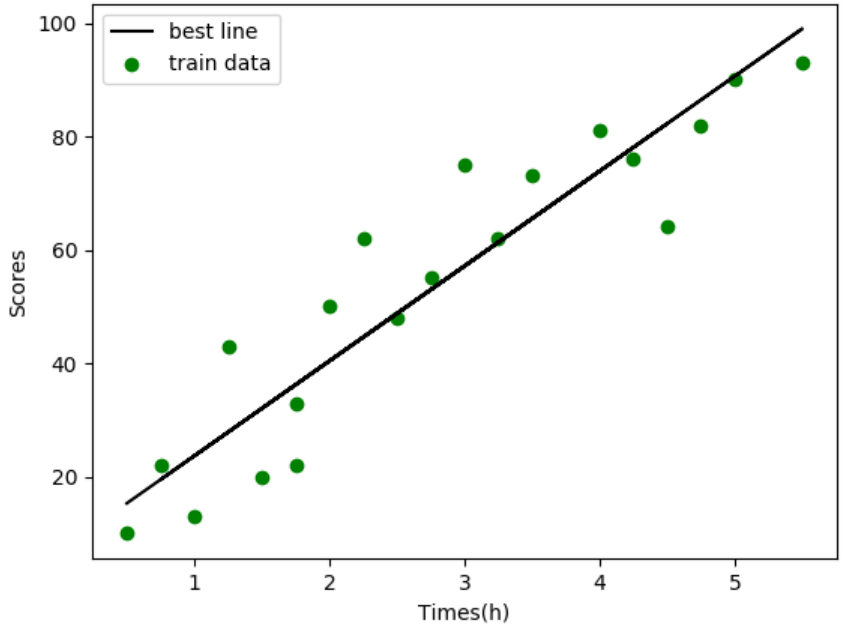
- Draw the figure

```
#Draw a scatter chart
plt.scatter(exam_X, exam_Y, color = 'green', label = 'train data')

#Set X, Y axis label and title
plt.ylabel('Scores')
plt.xlabel('Times(h)')

#Draw the best fit curve
Y_train_pred = model.predict(X_train)
plt.plot(X_train, Y_train_pred, color = 'black', label = 'best line')

#Output
plt.legend(loc = 2)
plt.show()
```



Linear regression example 2

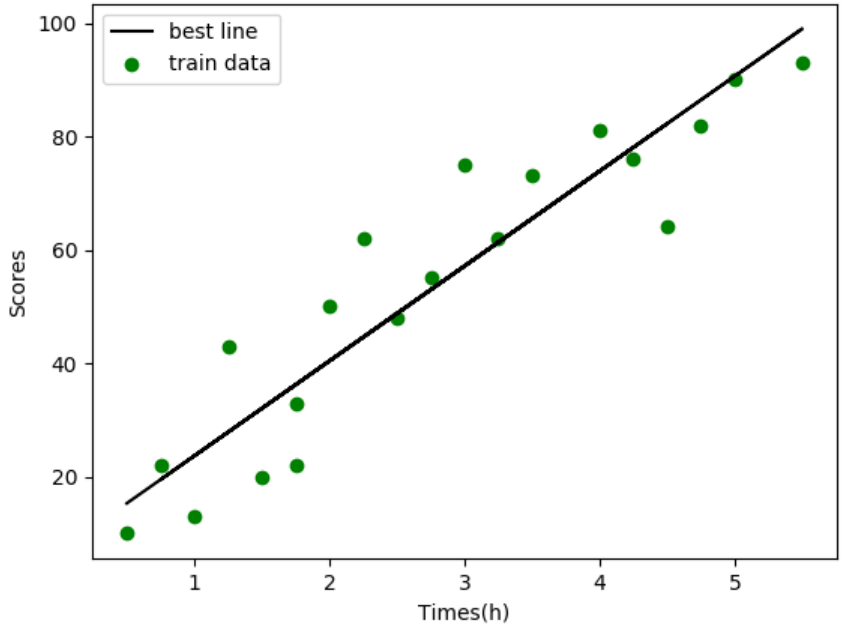
https://github.com/kevinsuo/CS7357/blob/master/linear-regression/lr.py

- Evaluate model accuracy

```
accuracy = model.score(X_test, Y_test)
print(accuracy)
```

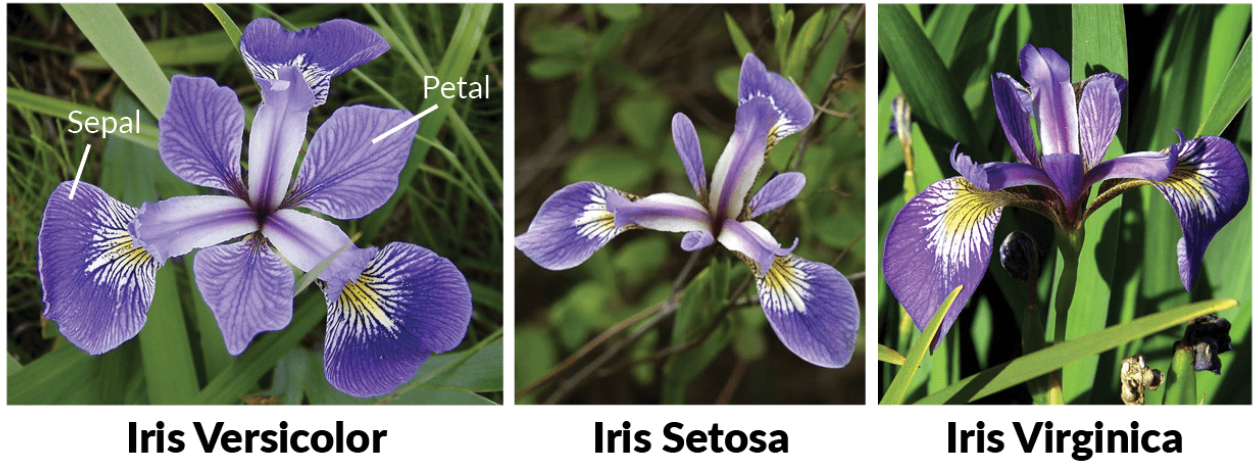
When the ratio of training set to test set is 7:3
>> accurate = 0.6398676613383307 3

When the ratio of training set to test set is 8:2
>> accurate = 0.8367023142103885 4

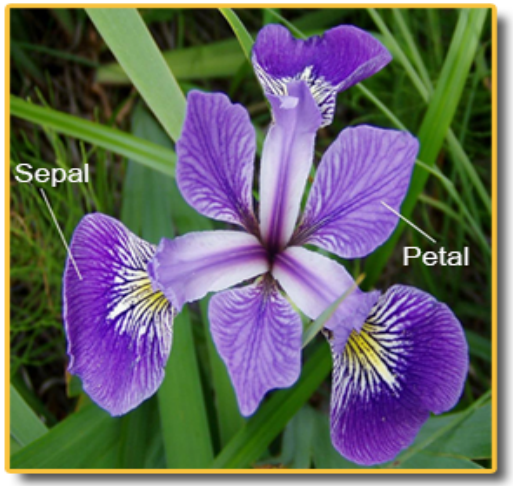


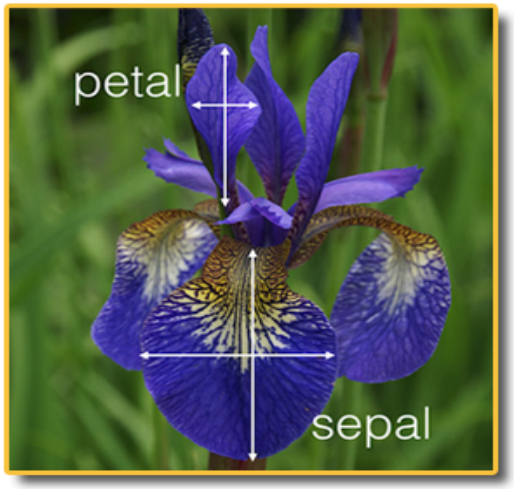
Linear regression example 3

- x='PetalLengthCm’
- y='PetalWidthCm'



Iris Versicolor **Iris Setosa** **Iris Virginica**





Linear regression example 3

- Realize regression analysis through python's scikit-learn lib

- https://github.com/kevinsuo/CS7357/blob/master/linear-regression/lr2.ipynb