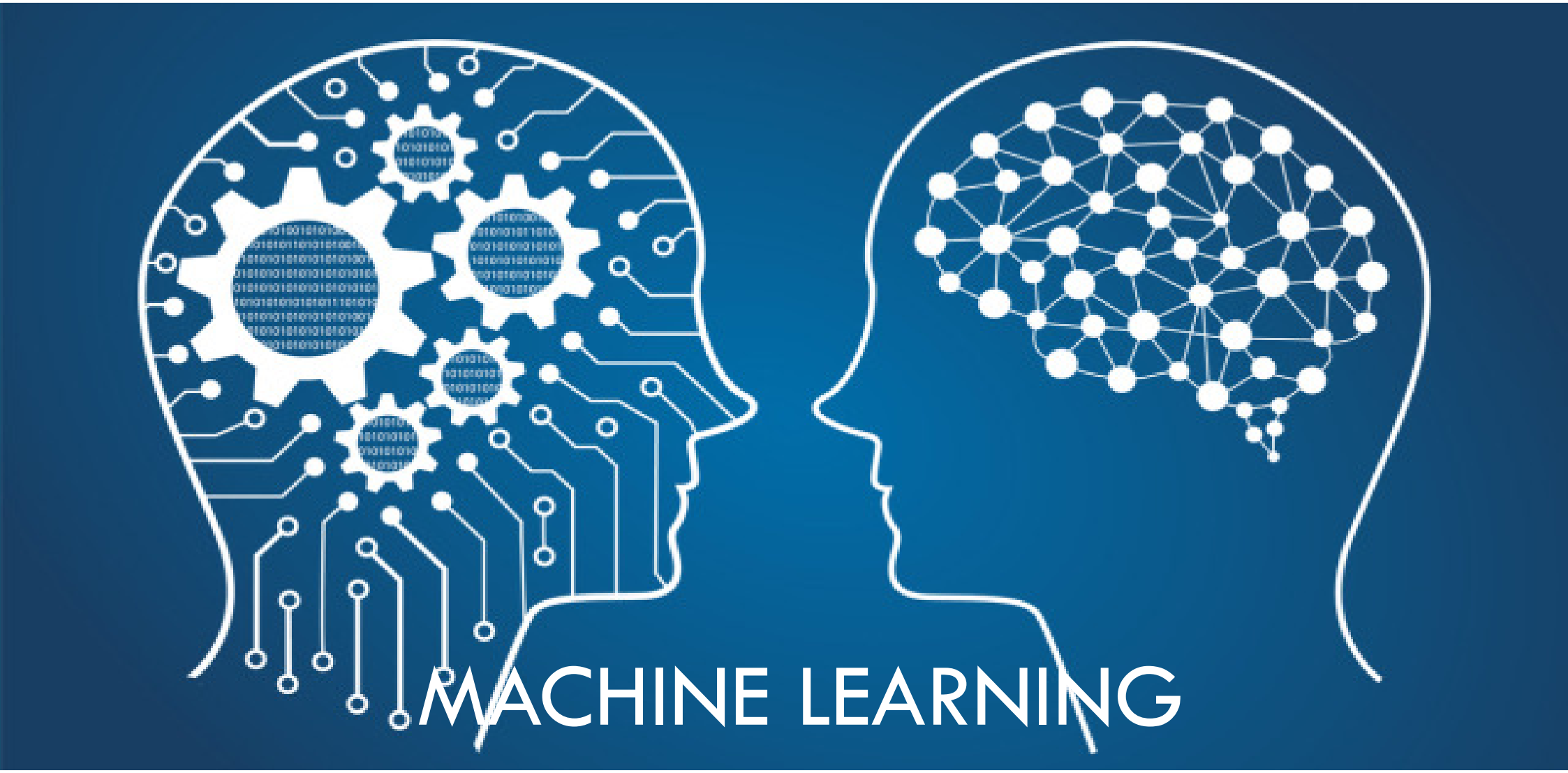
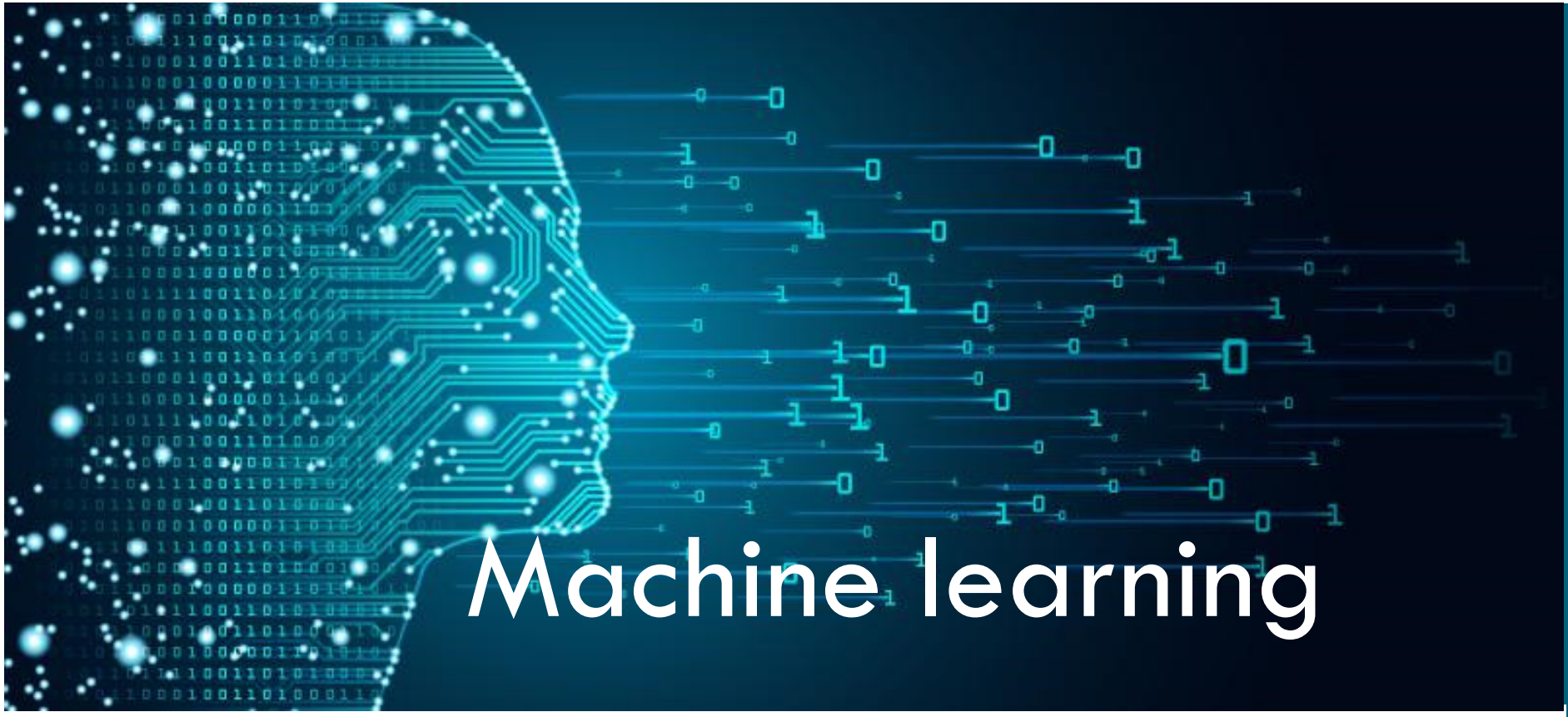


MACHINE LEARNING

Pr. BEN LAHMAR EL Habib



Machine learning

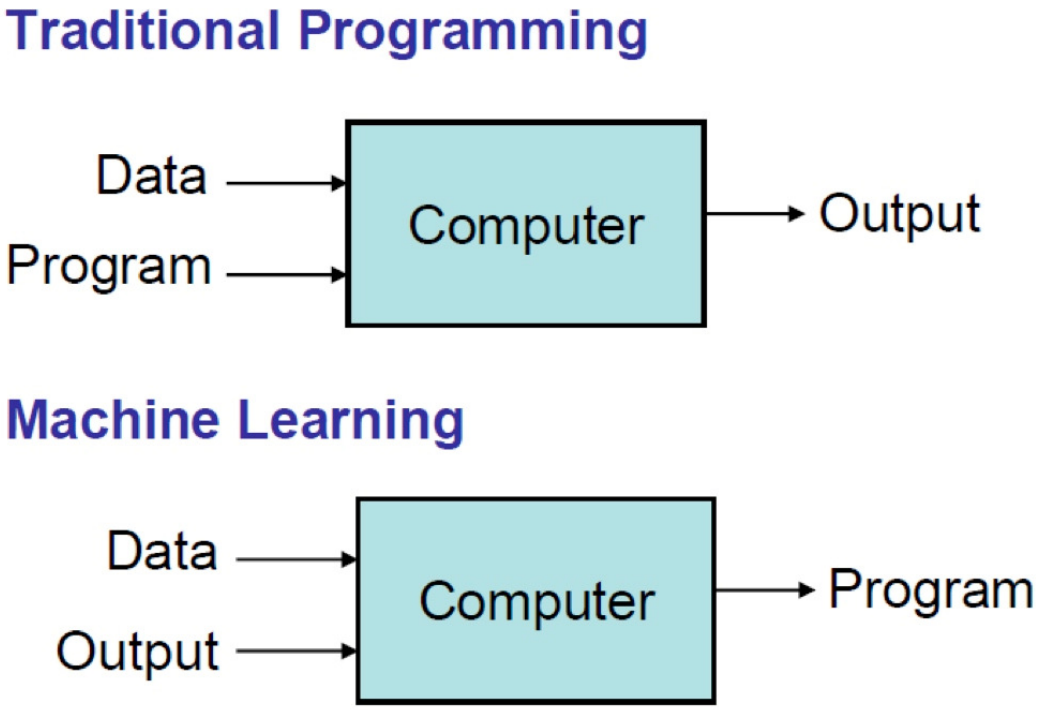
Algorithms

Pr. BEN LAHMAR EL Habib

General programming vs Machine learning-model

Basic Paradigm

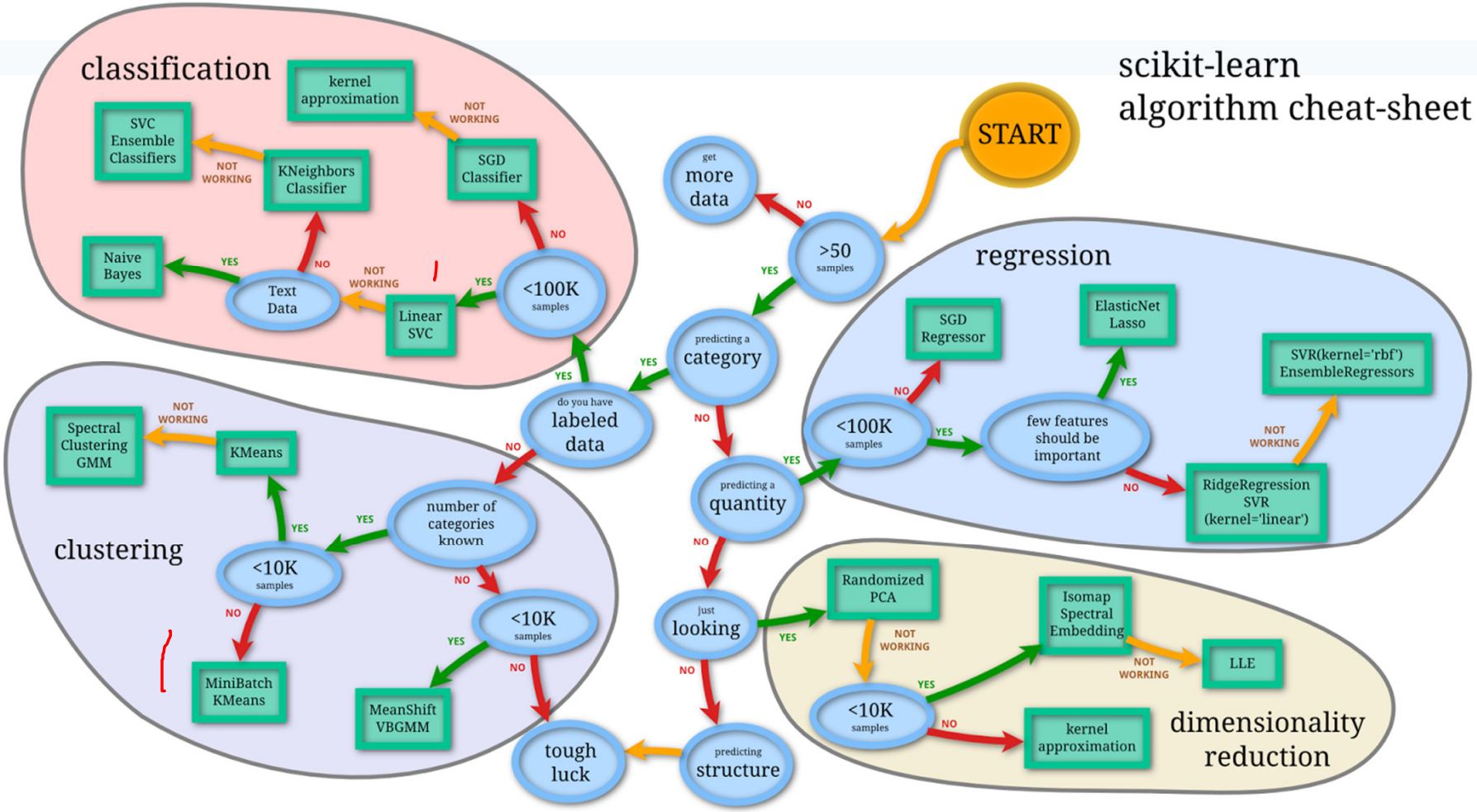
- ❑ Observe set of examples: **training data**
- ❑ Infer something about process that generated that data
- ❑ Use inference to make predictions about previously unseen data: **test data**



Procedures

1. Representation of the features
2. Distance metric for feature vectors
3. Objective function and constraints
4. Optimization method for learning the model
5. Evaluation method

Modèles


scikit-learn
algorithm cheat-sheet
START
>50 samples
NO
get more data
YES
predicting a category
YES
do you have labeled data
YES
<100K samples
NO
Naive Bayes
YES
Text Data
NO
NOT WORKING
Linear SVC
YES
NOT WORKING
SVC Ensemble Classifiers
NOT WORKING
KNeighbors Classifier
NOT WORKING
kernel approximation
NO
SGD Classifier
classification
NO
predicting a quantity
YES
<100K samples
NO
SGD Regressor
YES
ElasticNet Lasso
YES
few features should be important
NO
RidgeRegression SVR (kernel='linear')
NOT WORKING
SVR(kernel='rbf') EnsembleRegressors
regression
NO
just looking
YES
Randomized PCA
NOT WORKING
<10K samples
YES
Isomap Spectral Embedding
NOT WORKING
LLE
NO
kernel approximation
dimensionality reduction
NO
predicting structure
NO
tough luck
NO
number of categories known
YES
<10K samples
NO
MeanShift VBGMM
YES
<10K samples
NO
MiniBatch KMeans
YES
KMeans
NOT WORKING
Spectral Clustering GMM
clustering

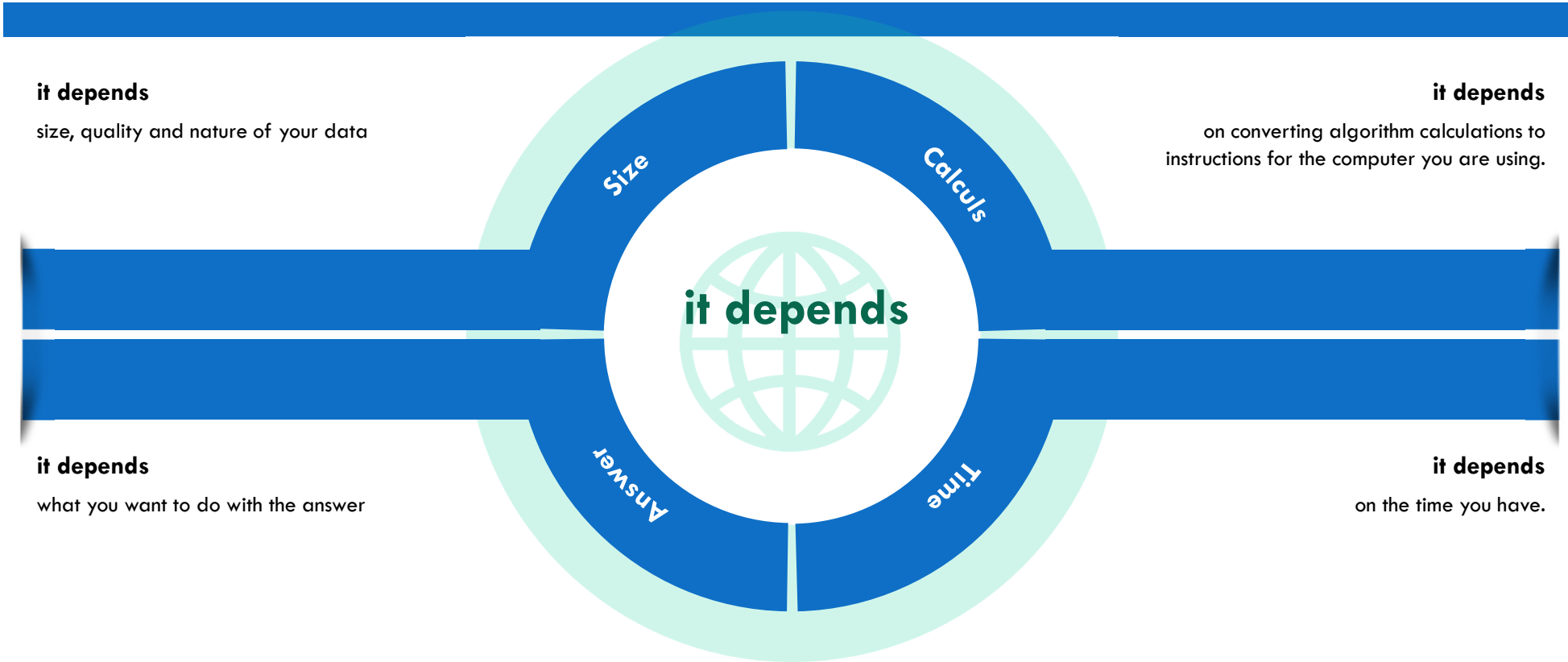
"What's the best machine learning algorithm to use? "

it depends

size, quality and nature of your data

it depends

on converting algorithm calculations to instructions for the computer you are using.



it depends

what you want to do with the answer

it depends

on the time you have.

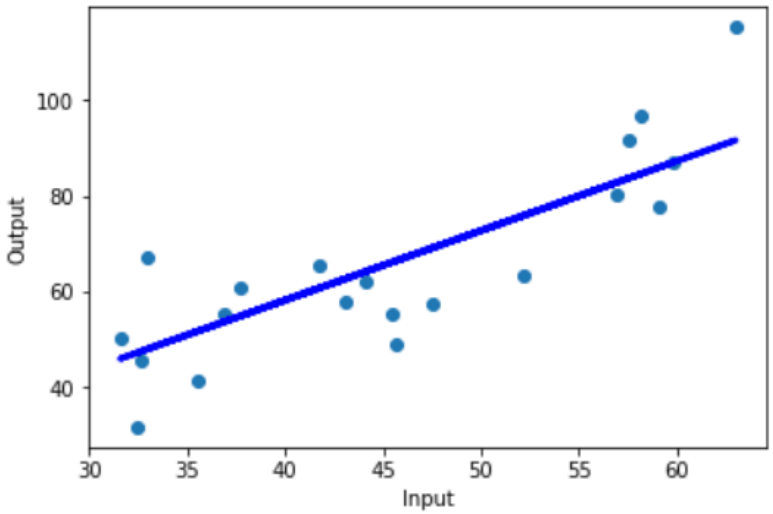
Self drive

Precision

Time & cost

Linear Regression

- It is used to estimate real values (cost of houses, number of calls, total sales etc.) based on continuous variable(s)..

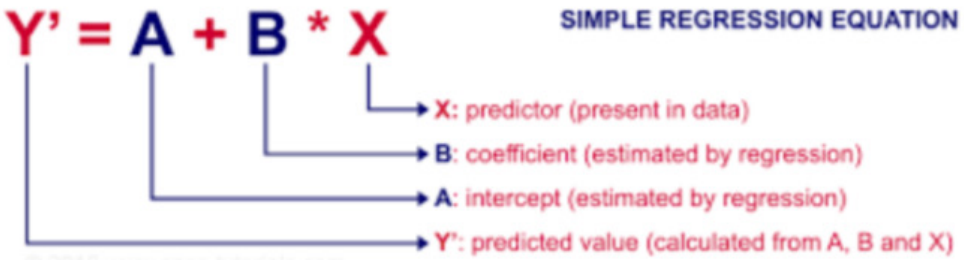


Linear Regression

- Linear regression is used for finding linear relationship between target and one or more predictors.
- The core idea is to obtain a **line** that best fits the data.
- The best fit line is the one for which total prediction error (all data points) are as small as possible. Error is the distance between the point to the regression line.

How

- Establish relationship between independent and dependent variables by fitting a best line. This best fit line is known as regression line and represented by a linear equation Y= a *X + b.



$$y_i = \beta_0 1 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i = \mathbf{x}_i^{\mathsf{T}} \beta + \varepsilon_i, \qquad i = 1, \ldots, n,$$

Types

- Linear Regression is of mainly two types:
  - Simple Linear Regression;
  - Multiple Linear Regression.

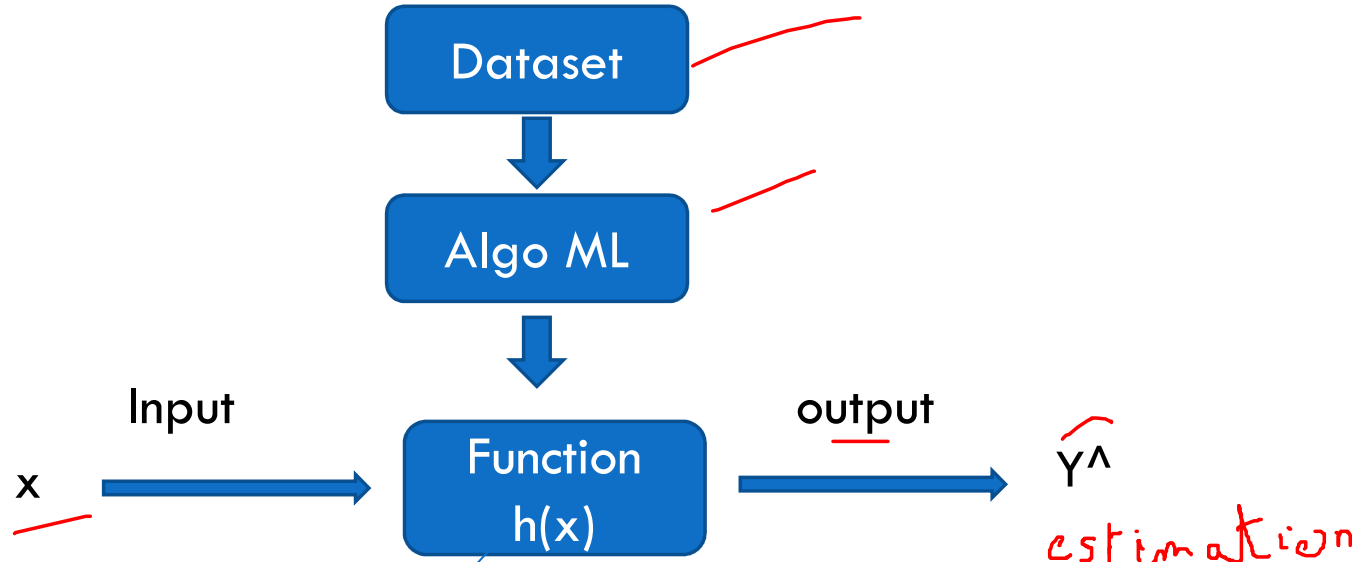
Simple Linear Regression;

- Single dimension linear regression has pairs of x and y values as input training samples.
- It uses these training sample to derive a line that predicts values of y.
- The training samples are used to derive the values of a and b that minimise the error between actual and predicated values of y

$$\hat{y} = a_1 x + b_1$$

- a is the **slope** and "b" is the y-intercept

Principe



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

Exemple

X	Y
4	12
2	8
4,5	13
5	14
6	14,5
6,5	16
7	16

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

$\theta_0 = ?$

$\theta_1 = ?$

$12 = \theta_0 + \theta_1 * 4$

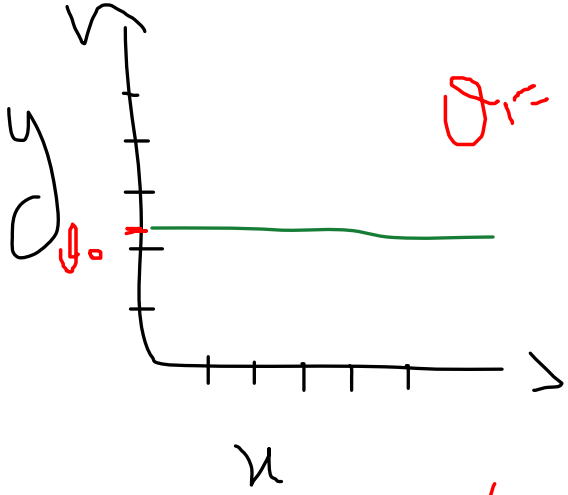
$8 = \theta_0 + \theta_1 * 2$

$\Rightarrow 4 = 2\theta_1 \Rightarrow \theta_1 = 2$

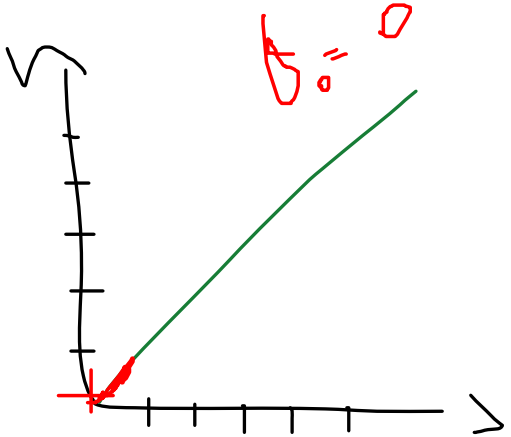
$\theta_0 = 4$

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

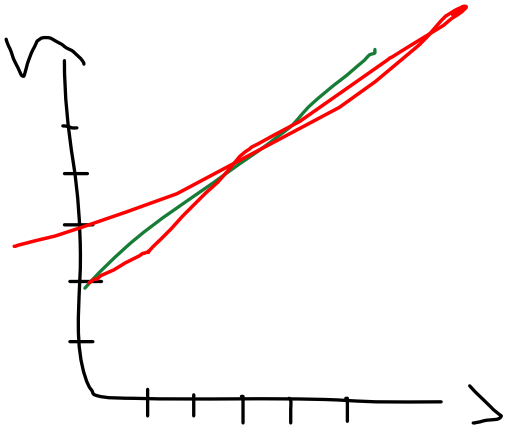
θ_0? θ_1?

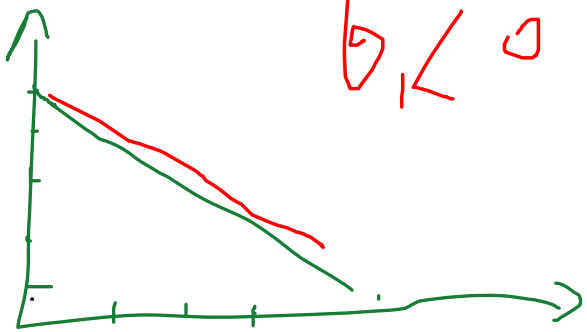


$\theta_1 = 0$



$\theta_0 = 0$





$\theta_1 < 0$

$h(x)$

Chose θ_0? θ_1? so that

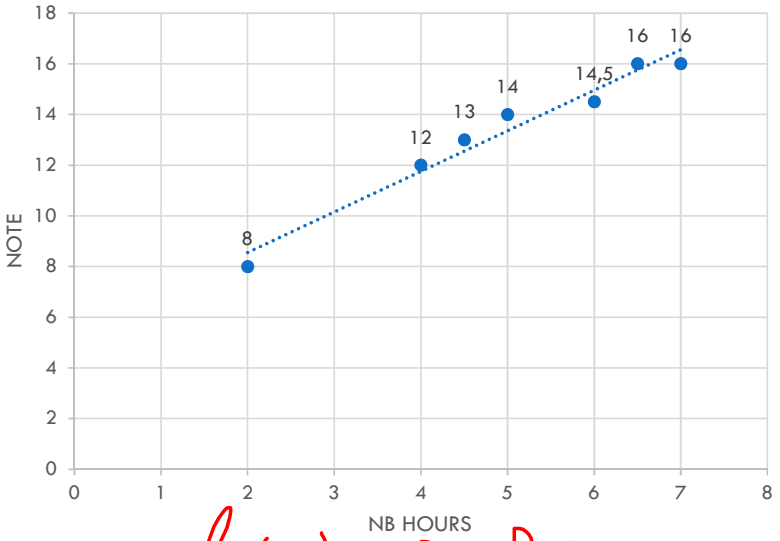
is close to y for training dataset

$\theta_1 \to 0$

$\theta_1 x + \theta_0$

Exemple

X (nb hours)	Y (note)
4	12
2	8
4,5	13
5	14
6	14,5
6,5	16
7	16



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

Chose θ_0 θ_1 so that $h(x)$ is close to y for training dataset

We want a line that minimises the **error** between the Y values in training samples and the $h(x)$ values that the line passes through.

So we define the error function for our algorithm so we can minimise that error.

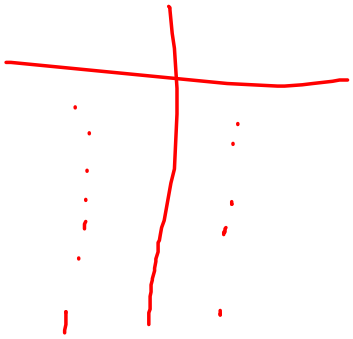
$$E = \frac{1}{n}\sum_{i=1} \left(y_i - h(x_i)\right)^2$$

Cost function

- Minimise $E = \frac{1}{n}\sum_{i=0}^{n}(y_i - h(x_i))^2$

$$= \frac{1}{n}\sum_{i=0}^{n}\left(y_i - (\theta_0 + \theta_1 x_i)\right)^2$$

$J(\theta_0, \theta_1)$ ← cost function



Cost Function

- The cost function helps us to figure out the best possible values for a and b which would provide the best fit line for the data points.
- Since we want the best values for a and b, we convert this search problem into a minimization problem where we would like to minimize the error between the predicted value and the actual value.

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

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

Minimize the error

- The values a and b must be chosen so that they minimize the error. If sum of squared error is taken as a metric to evaluate the model, then goal to obtain a line that best reduces the error.
  - Mean Absolute Error
  - Mean Square Error
  - Mean Absolute Percentage Error
  - Mean Percentage Error
  - ….
- Mean Absolute Error (MAE) is the mean of the absolute value of the errors
- Mean Squared Error (MSE) is the mean of the squared errors
- Root Mean Squared Error (RMSE) is the square root of the mean of the squared errors

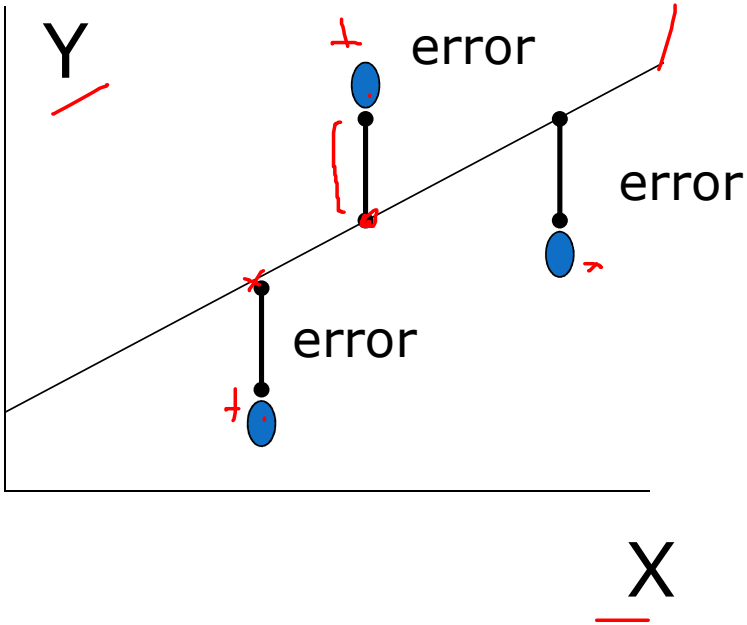
Error

$$BCE = -\frac{1}{N}\sum_{i}^{N} y_i \cdot log(\hat{y}_i) + (1 - y_i) \cdot log(1 - \hat{y}_i)$$

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(f_i - y_i)^2$$

where N is the number of data points,
f_i the value returned by the model and
y_i the actual value for data point i.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$



MSE

- We square the error difference and sum over all data points and divide that value by the total number of data points. This provides the average squared error over all the data points.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(f_i - y_i)^2$$

where N is the number of data points, f_i the value returned by the model and y_i the actual value for data point i.

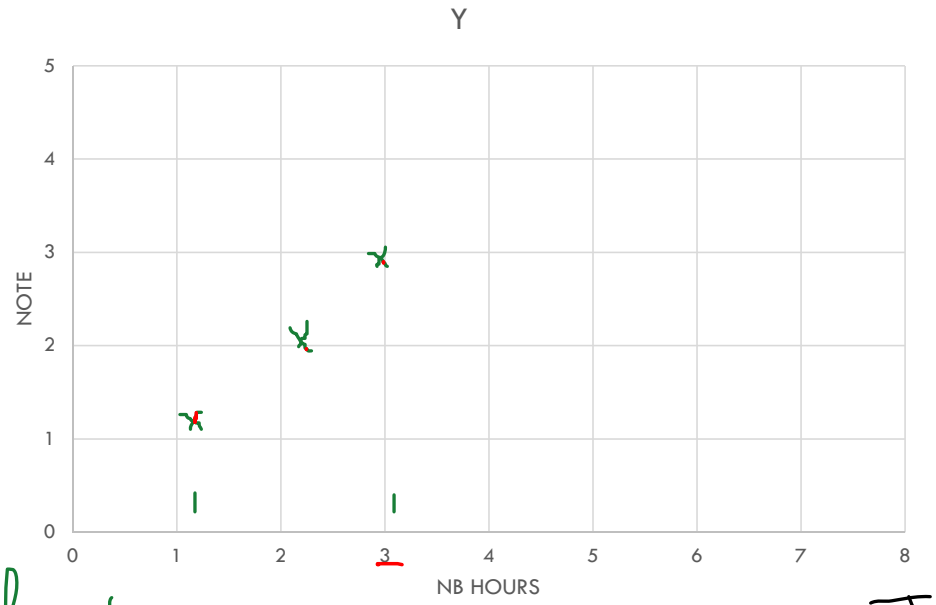
Cost function

- Goal :
- Minimize $J(\theta_0, \theta_1)$

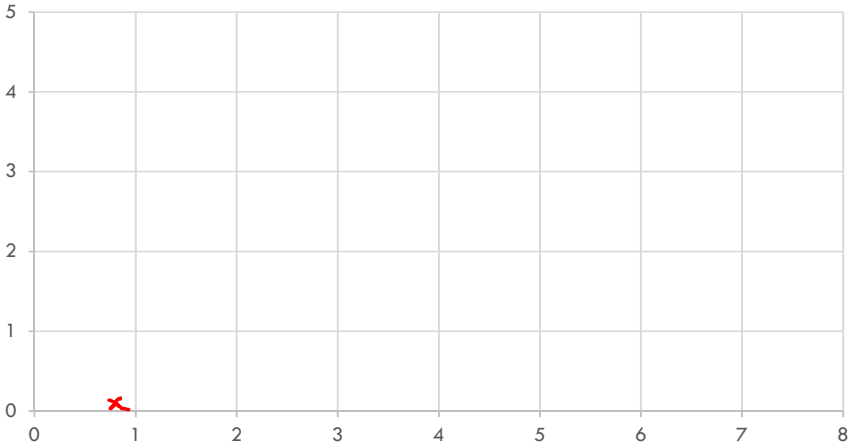
$$\frac{1}{2n} \sum_{i=0}^{n} \left(y_i - (\theta_0 + \theta_1 x_i) \right)^2$$

Simple example [(1,1),(2,2),(3,3)]

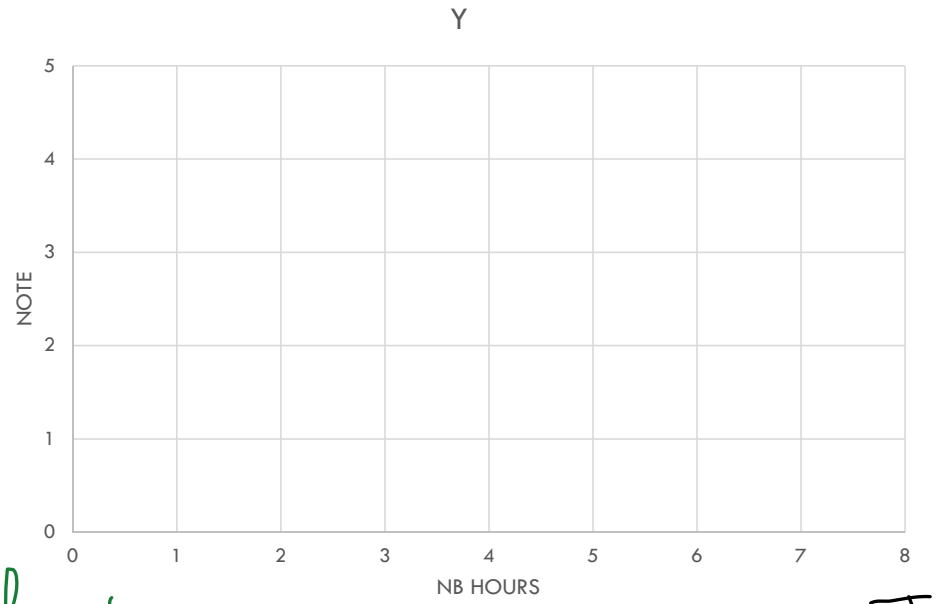
Function h



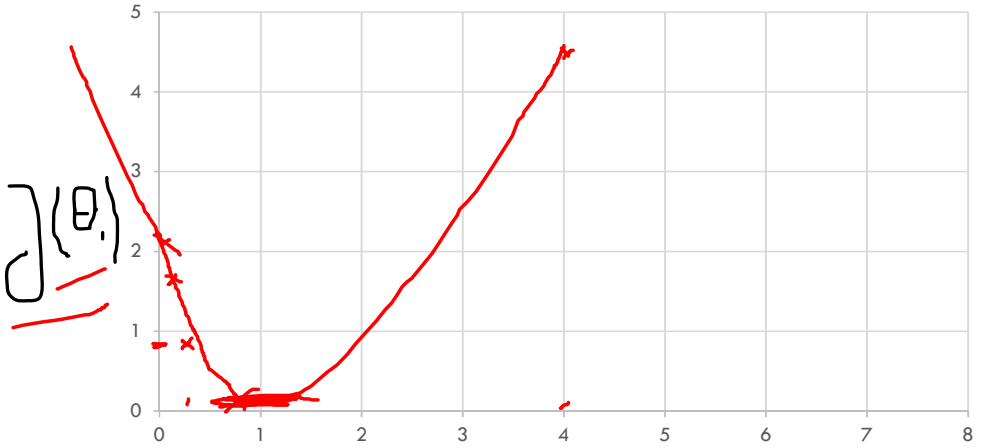
Function $J(\theta_1)$



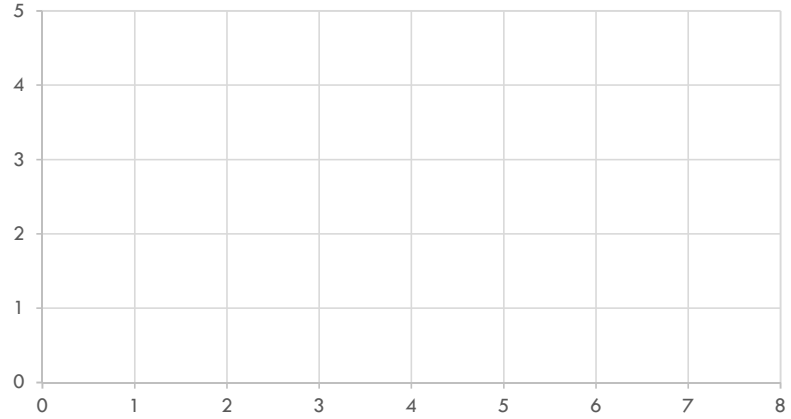
Function h



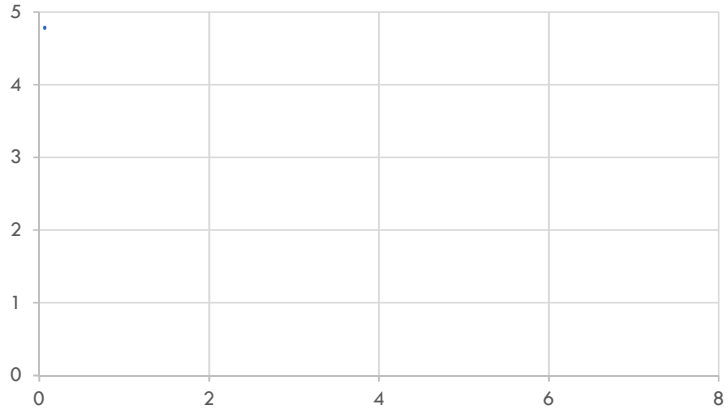
Function $J(\theta_1)$



Function h

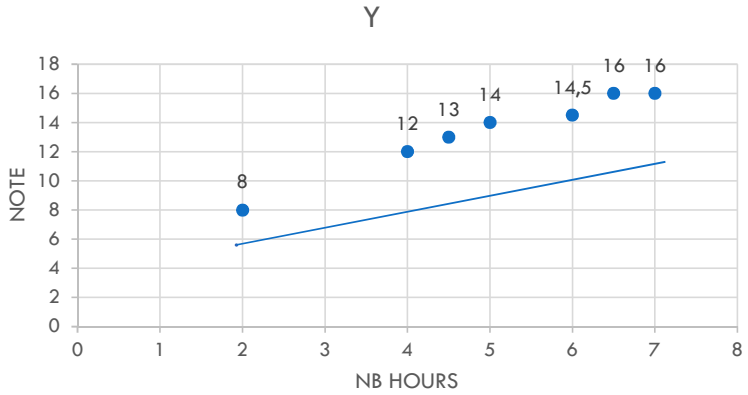


Function $J(\theta_1)$



if $\theta_0 = 0$ $\theta_1 = 0$ $J(0) = ?$ $\frac{1}{3}(1^2 + 2^2 + 3^2) = \frac{14}{3} = 4,67$

Function h

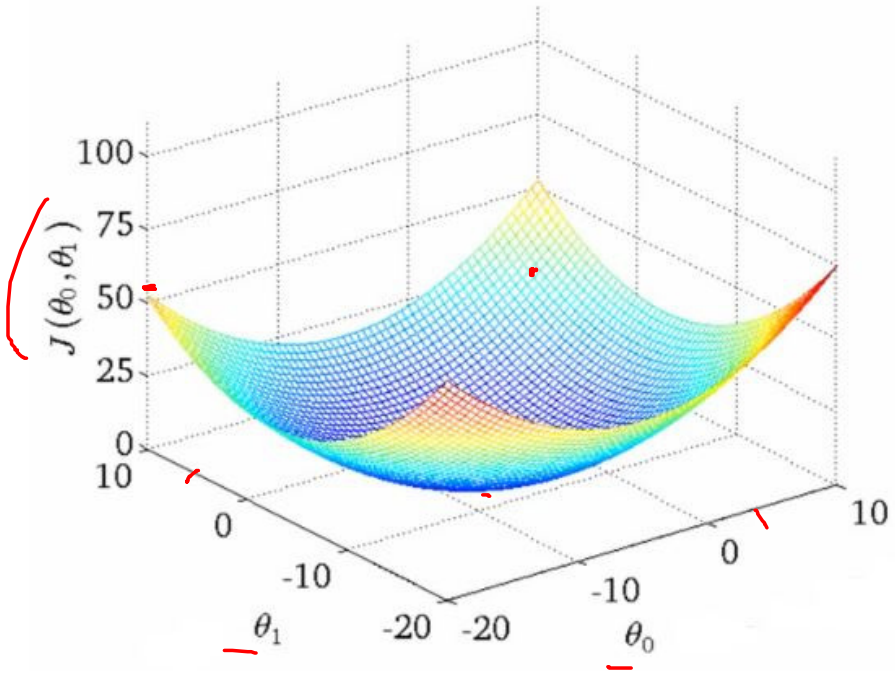


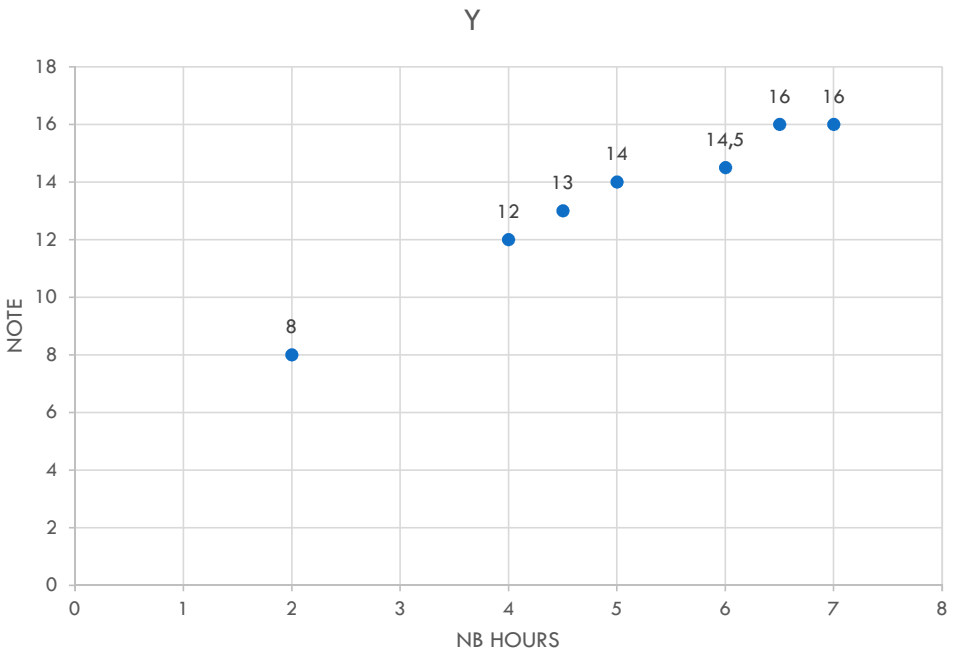
$\theta_0 = 3$

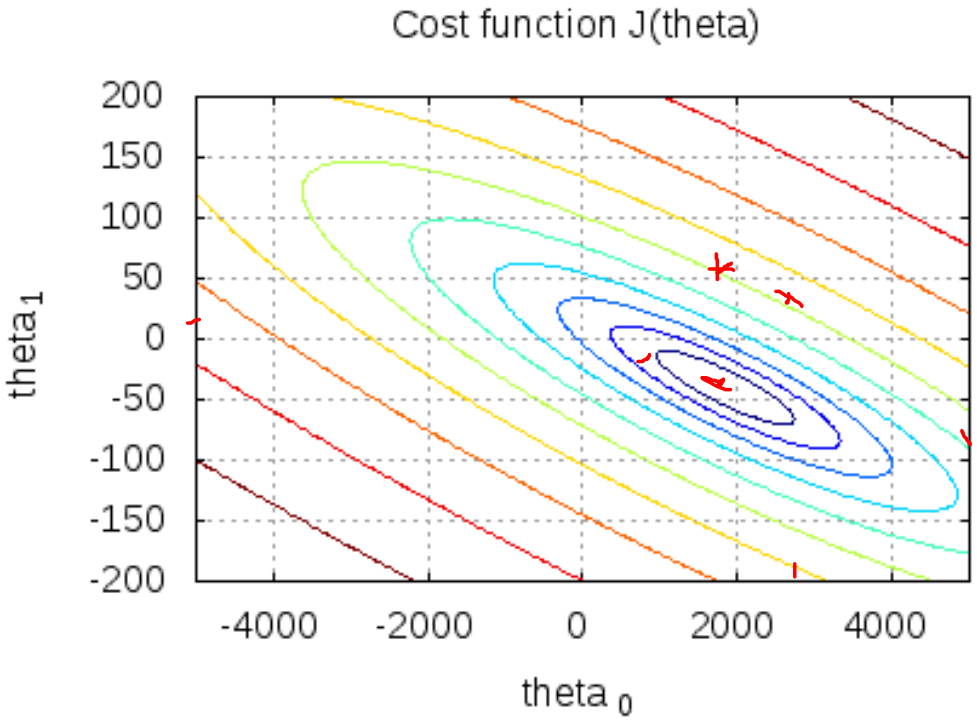
$\theta_1 = 0.5$

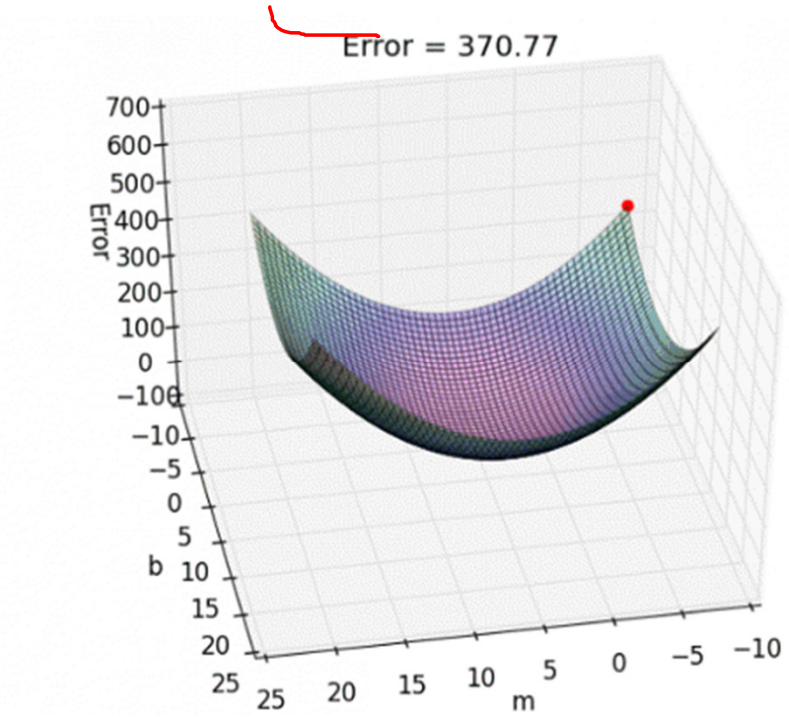
$h(x) = 3 + 0.5\,y$

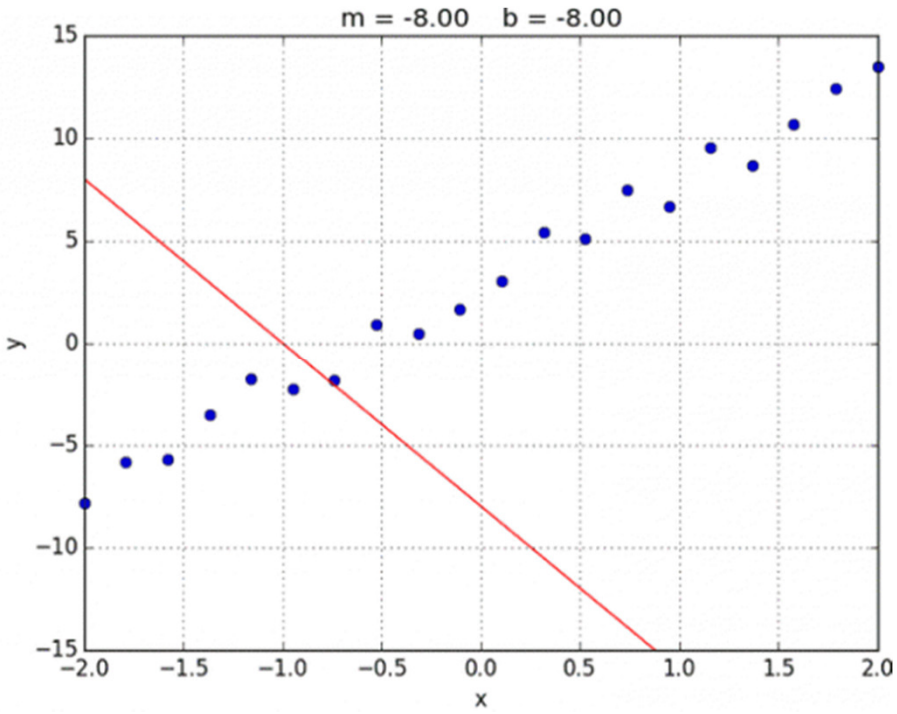
Function $J(\theta_0, \theta_1)$




Y
NOTE
NB HOURS
0
2
4
6
8
10
12
14
16
18
1
3
5
7
8
12
13
14
14,5
16
16


Cost function J(theta)
theta 1
theta 0
200
150
100
50
0
-50
-100
-150
-200
-4000
-2000
0
2000
4000

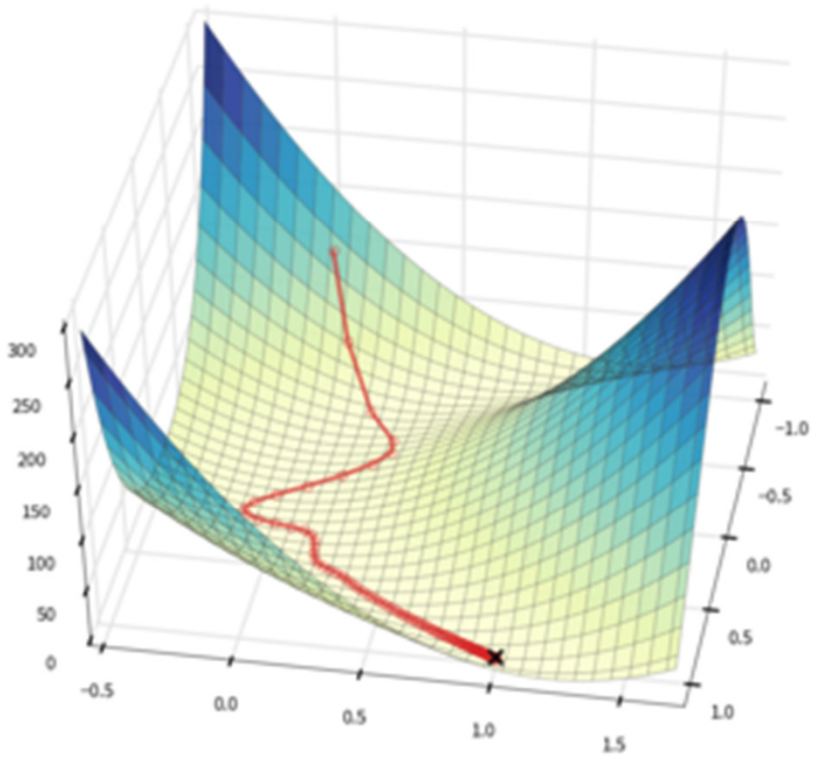

Error = 370.77
Error
700
600
500
400
300
200
100
0
−100
b
−10
−5
0
5
10
15
20
25
m
25
20
15
10
5
0
−5
−10


m = -8.00 b = -8.00
y
15
10
5
0
−5
−10
−15
x
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

How

- To determine how to best fit our model with given set of points, we want to minimize the distance between each of these point to our linear model.
- Calculating the totalError helps us determine how bad our model is so we can update it every step.
- but …

Minimizing the cost function: Gradient descent



Repeat until convergence:

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

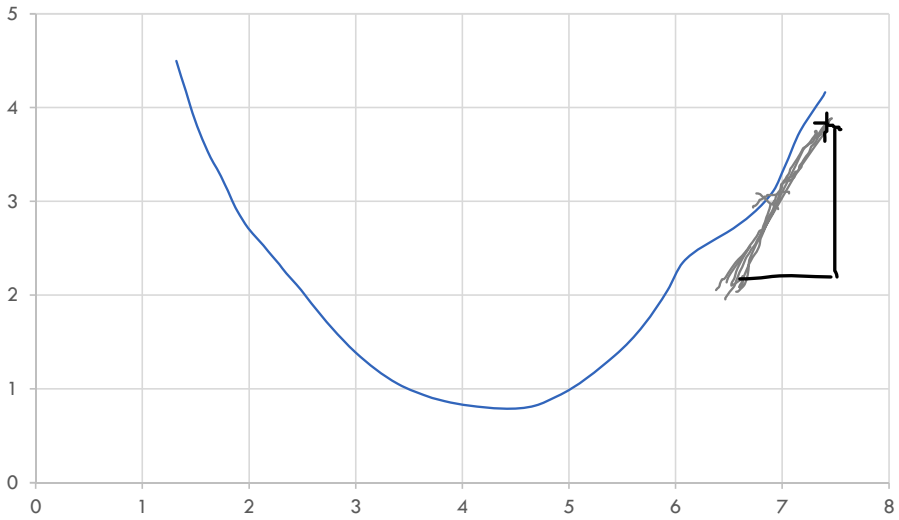
Learning Rate

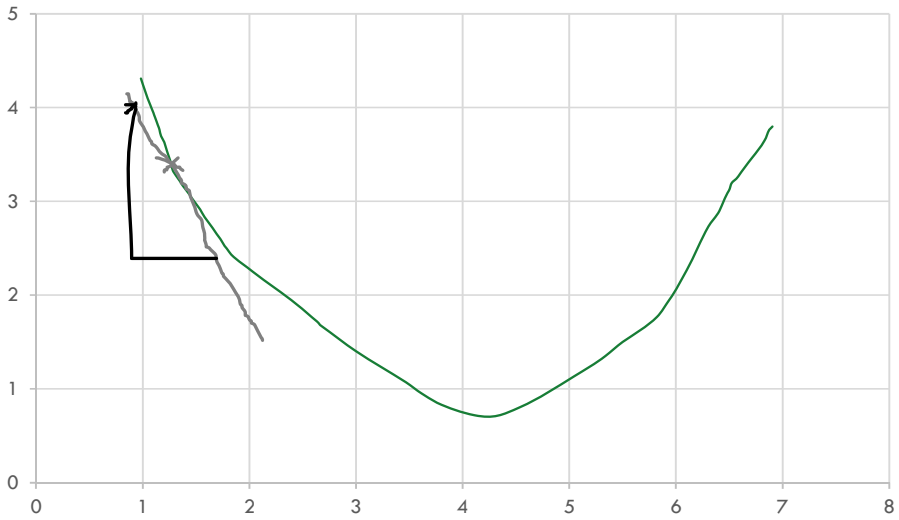
Partial derivative

The derivative of a function of a real variable measures the sensitivity to change of the function value (output value) with respect to a change in its argument (input value).

Slope

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

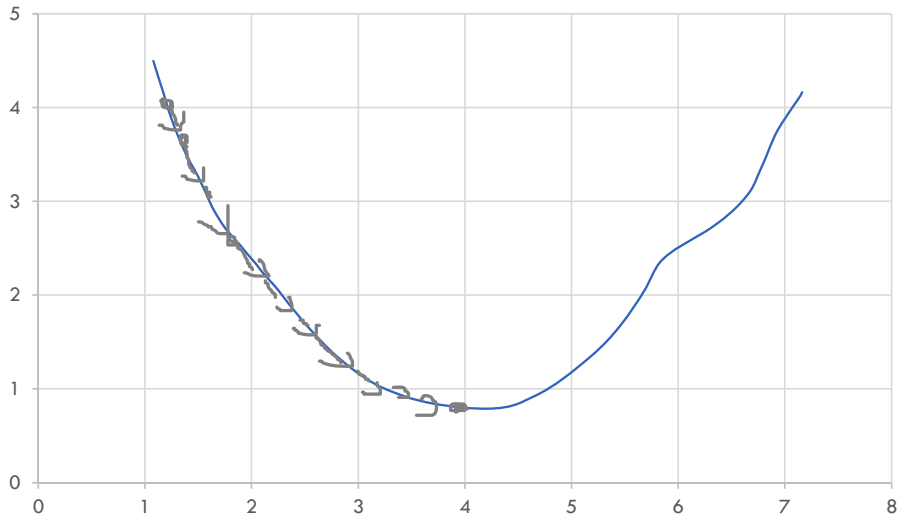




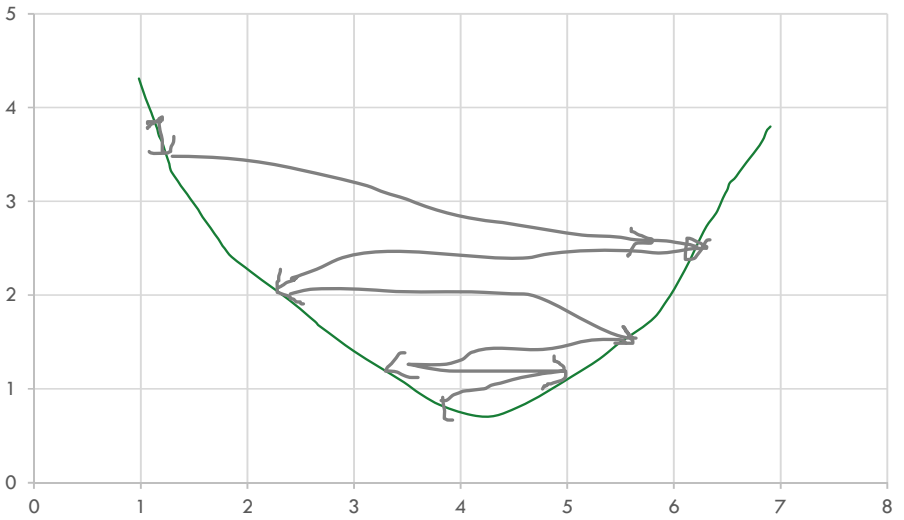
Learning rate

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

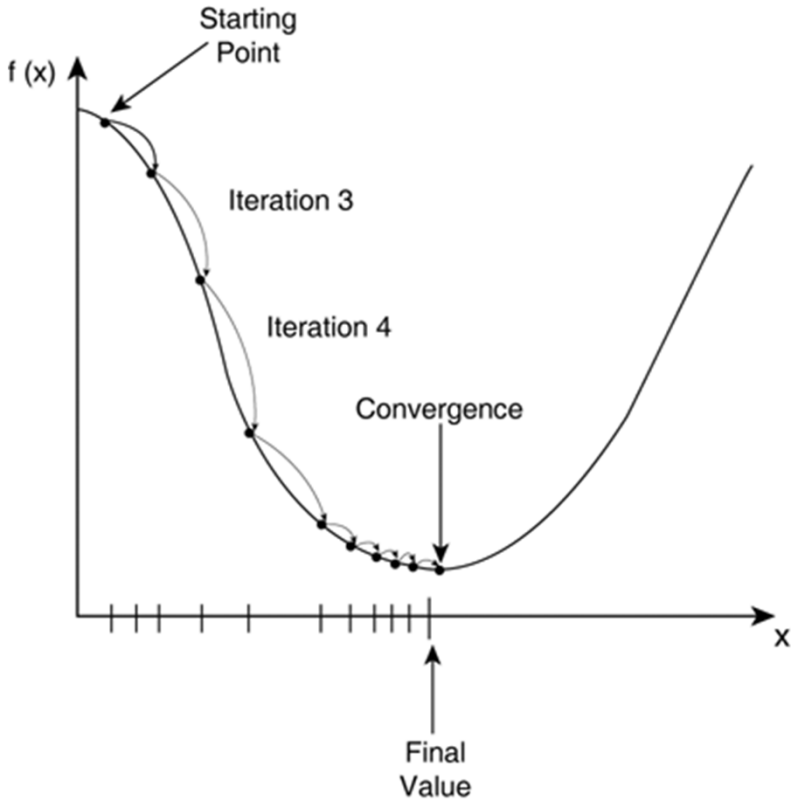
If α is too small



If α is too larger



Minimizing the cost function: Gradient descent



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

$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = ?$$

$$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = ?$$

$$J = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x_i) - y_i \right)^2$$

$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) =$$

$$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=0}^{m} 2 \left(h_\theta(x_i) - y_i \right) \left(\frac{\partial}{\partial \theta_1} \theta_0 + \frac{\partial \theta_1 x_i}{\partial \theta_1} - \frac{\partial}{\partial \theta_1} y_i \right)$$

$$= \frac{1}{m} \sum_{i=0}^{m} \left(h_\theta(x_i) - y_i \right) \times x_i$$

Update Θ_0 and Θ_1

$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=0}^{m} (h(x_i) - y_i)$$
$$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=0}^{m} (h(x_i) - y_i) x_i$$

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=0}^{m} (h(x_i) - y_i)$$
$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=0}^{m} (h(x_i) - y_i) x_i$$

Example

- Tuto

Multi Dimension Linear Regression

Multi Dimension Linear Regression

Multi Dimension Linear Regression

- Each training sample has an x made up of multiple input values and a corresponding y with a single value.
- The inputs can be represented as an X matrix in which each row is sample and each column is a dimension.
- The outputs can be represented as y matrix in which each row is a sample.

$$X = \begin{bmatrix} x_{1,1} & x_{1,\dots} & x_{1,d} \\ x_{\dots,1} & x_{\dots,\dots} & x_{\dots,d} \\ x_{n,1} & x_{n,\dots} & x_{n,d} \end{bmatrix}$$

$$X_{ij} \rightarrow sample_i, dimension_j$$

$$y = \begin{bmatrix} y_1 \\ y_{\dots} \\ y_n \end{bmatrix}$$

LR Multiple

- Our predicated y values are calculated by multiple the X matrix by a matrix of param, Θ.
- If there are 2 dimension, then this equation defines plane. If there are more dimensions then it defines a hyper-plane.

$$y\hat{} = X^* \Theta$$

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ . \\ \theta_n \end{bmatrix}$$

LR Multiple

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

LR simple

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

LR Multiple

$x_0 = 1$

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

Example

X1	X2	X3	X4	X5
8	78	284	9,10000038	109
9,30000019	68	433	8,69999981	144
7,5	70	739	7,19999981	113
8,89999962	96	1792	8,89999962	97
10,1999998	74	477	8,30000019	206
8,30000019	111	362	10,8999996	124
8,80000019	77	671	10	152
8,80000019	168	636	9,10000038	162
10,6999998	82	329	8,69999981	150
11,6999998	89	634	7,59999991	134

The data (X1, X2, X3, X4, X5) are by city.
X1 = death rate per 1000 residents
X2 = doctor availability per 100,000 residents
X3 = hospital availability per 100,000 residents
X4 = annual per capita income in thousands of dollars
X5 = population density people per square mile
Reference: *Life In America's Small Cities*, by G.S. Thomas

LR

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

$$X = \begin{bmatrix} x_1 \\ x_2 \\ . \\ \\ \\ x_n \end{bmatrix} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ . \\ \theta_n \end{bmatrix} \qquad \theta^T = [\theta_0 \theta_1 \cdots \theta_n]$$

$$h_\theta(x) = \theta^T * X$$

Gradient Descent

$$J(\Theta_0, \Theta_1, \ \ldots \ \Theta_n) = \frac{1}{2n} \sum_i^n (h_\Theta(x_i) - y_i)^2$$

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ . \\ \theta_n \end{bmatrix}$$ Dimension = n+1

$$J(\Theta) = \frac{1}{2n} \sum_i^n (h_\Theta(x_i) - y_i)^2$$

Gradient Descent

Repeat

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

LR

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

$$\frac{\partial}{\partial \theta} J_\theta = \frac{1}{m} \sum_{i=1}^{m} \left[\left(h_\theta(x_i) - y \right) x_i \right]$$

$$\frac{\partial}{\partial \theta} J_\theta = \frac{\partial}{\partial \theta} \frac{1}{2m} \sum_{i=1}^{m} \left[h_\theta(x_i) - y_i \right]^2$$

$$\theta_j := \theta_j - \frac{\alpha}{m} \sum_{i=1}^{m} \left[\left(h_\theta(x_i) - y_i \right) x_i \right]$$

$$\frac{\partial}{\partial \theta} J_\theta = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x_i) - y_i \right) \cdot \frac{\partial}{\partial \theta_j} \left(\theta x_i - y_i \right)$$

θ_i : Weights of the hypothesis. –
$h_\theta(x_i)$: predicted y value for i^{th} input.
j : Feature index number (can be 0, 1, 2, ..., n).
α : Learning Rate of Gradient Descent.

Algo

repeat until convergence: {

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

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

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

...

}

Example

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$$

	Death rate per 1000 residents **(y)**	Doctor availability per 100,000 residents (x1)	Hospital availability per 100,000 (x2)	Annual per capita income in thousands of dollars (x3)	Population density people per square mile (x4)
0	8	78	284	9,100000381	109
1	9,300000191	68	433	8,699999809	144
2	7,5	70	739	7,199999809	113
3	8,899999619	96	1792	8,899999619	97
4	10,19999981	74	477	8,300000191	206

$$x^{(1)} = \begin{bmatrix} 78 \\ 284 \\ 9.100000381 \\ 109 \end{bmatrix}$$

$$h_\theta(x) = \theta^T X$$

where

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \end{bmatrix}$$

and

$$X = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

Vectorized Cost function

$$J(\theta) = \frac{1}{2m}(X\theta - Y)^T(X\theta - Y)$$

Vectorized gradient algorithm

$$\theta = \theta - \alpha\frac{1}{m}X^T(X\theta^T - Y)$$

Feature normalization

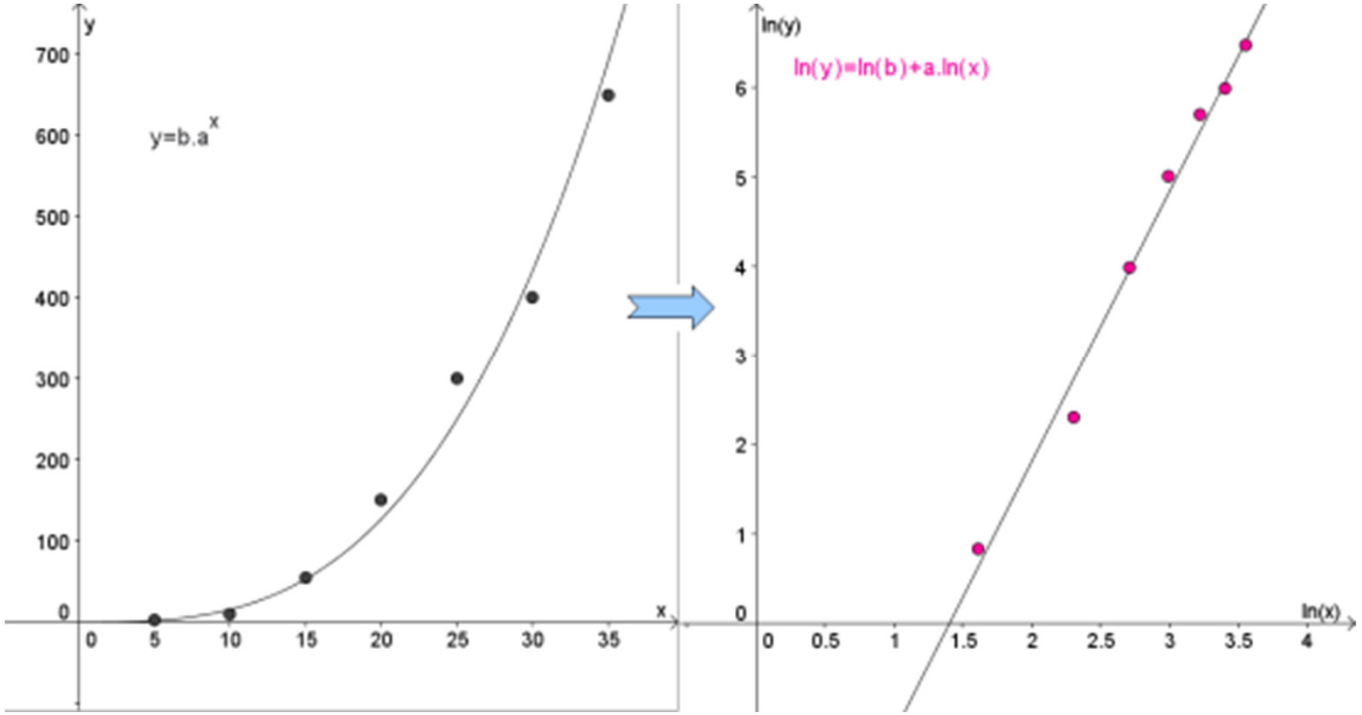
- When building a linear regression model with multiple features, we face another problem.
- The values of features may differ by orders of magnitude. For instance *Annual per capita income in thousands of dollars (x3)* value is about 10 while *Hospital availability per 100,000 (x2)* may reach several hundred.
- This may influence the gradient descent calculation effectiveness. So we need to scale features to make algorithm converge much faster.
- The procedure is straightforward:
  - Subtract the mean value of each feature from the training set.
  - Then additionally scale (divide) the feature values by their respective standard deviations.

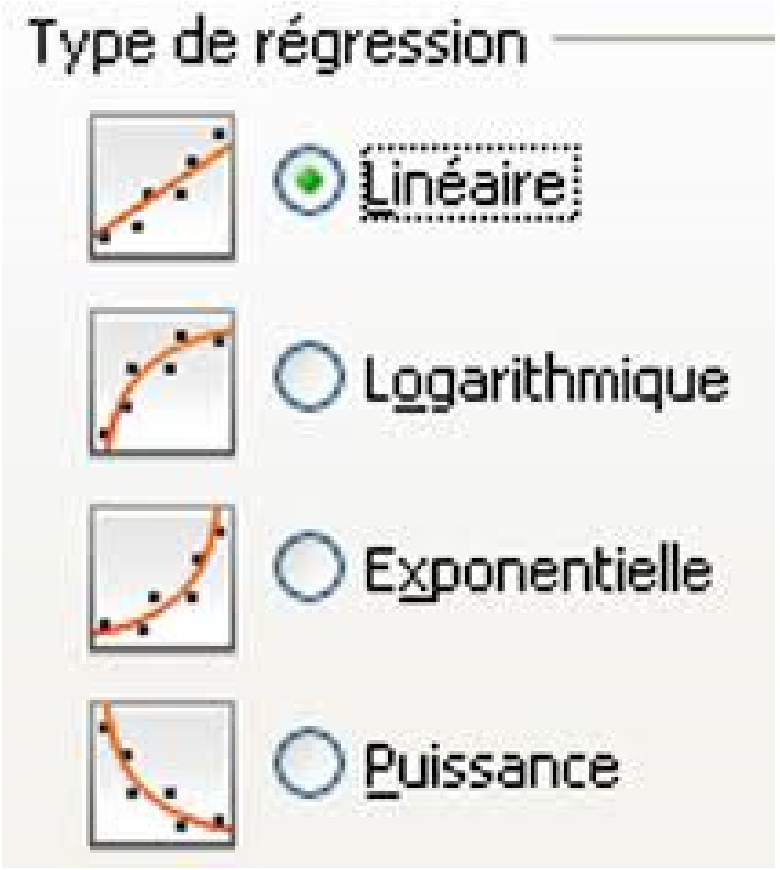
Feature Scaling

	Death rate per 1000 residents **(y)**	Doctor availability per 100,000 residents (x1)	Hospital availability per 100,000 (x2)	Annual per capita income in thousands of dollars (x3)	Population density people per square mile (x4)
0	8	78	284	9,100000381	109
1	9,300000191	68	433	8,699999809	144
2	7,5	70	739	7,199999809	113

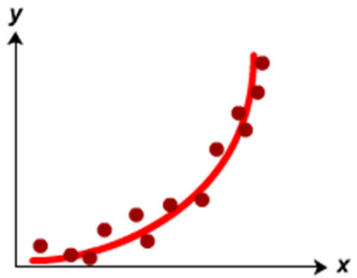
Generalisation, Over-fitting & Regularisation

Types

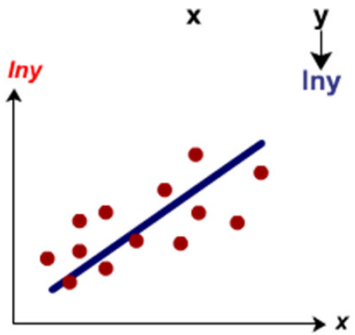

y
y=b.a^x
x
ln(y)
ln(y)=ln(b)+a.ln(x)
ln(x)


Type de régression
Linéaire
Logarithmique
Exponentielle
Puissance

x: temps (mois)
y: nombre de campagnols par km²

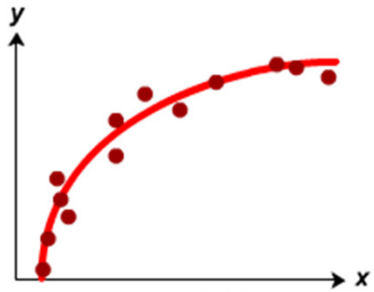


$y = a.e^{b.x}$

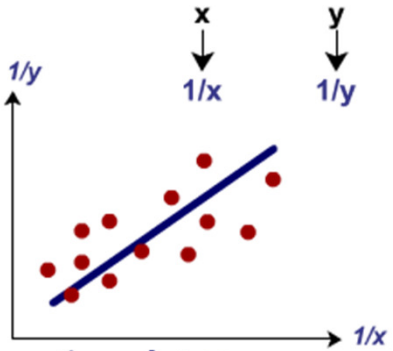


$lny = lna + b.x$

x: [substrat]
y: vitesse de réaction

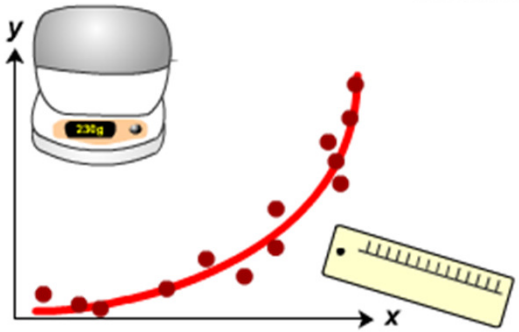


$$y = \frac{a.x}{b + x}$$

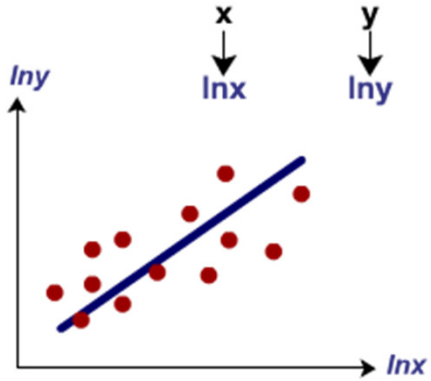


$$\frac{1}{y} = \frac{b + x}{a.x}$$

x: taille (cm)
y: masse (g)



$y = a.x^b$



$lny = lna + b.lnx$

Impléme

Pyhton code

```
#Import Library
#Import other necessary libraries like pandas, numpy...
from sklearn import linear_model
#Load Train and Test datasets
#Identify feature and response variable(s) and values must be numeric and numpy arrays
x_train=input_variables_values_training_datasets
y_train=target_variables_values_training_datasets
x_test=input_variables_values_test_datasets
# Create linear regression object
linear = linear_model.LinearRegression()
# Train the model using the training sets and check score
linear.fit(x_train, y_train)
linear.score(x_train, y_train)
#Equation coefficient and Intercept
print('Coefficient: \n', linear.coef_)
print('Intercept: \n', linear.intercept_)
#Predict Output
predicted= linear.predict(x_test)
```

Implémentation

R Code

```
#Load Train and Test datasets
#Identify feature and response variable(s) and values must be numeric and numpy arrays
x_train <- input_variables_values_training_datasets
y_train <- target_variables_values_training_datasets
x_test <- input_variables_values_test_datasets
x <- cbind(x_train,y_train)
# Train the model using the training sets and check score
linear <- lm(y_train ~ ., data = x)
summary(linear)
#Predict Output
predicted= predict(linear,x_test)
```

Tuto