Scikit learn

ABOUT THE TUTORIAL 1

HOME PAGE: https://scikit-learn.org/

Website learn python basic

1. Learn Python in Y Minutes
2. https://www.geeksforgeeks.org/python-numpy/
3. Scikit-learn tutorial

4. https://colab.research.google.com/
5. https://www.kaggle.com/

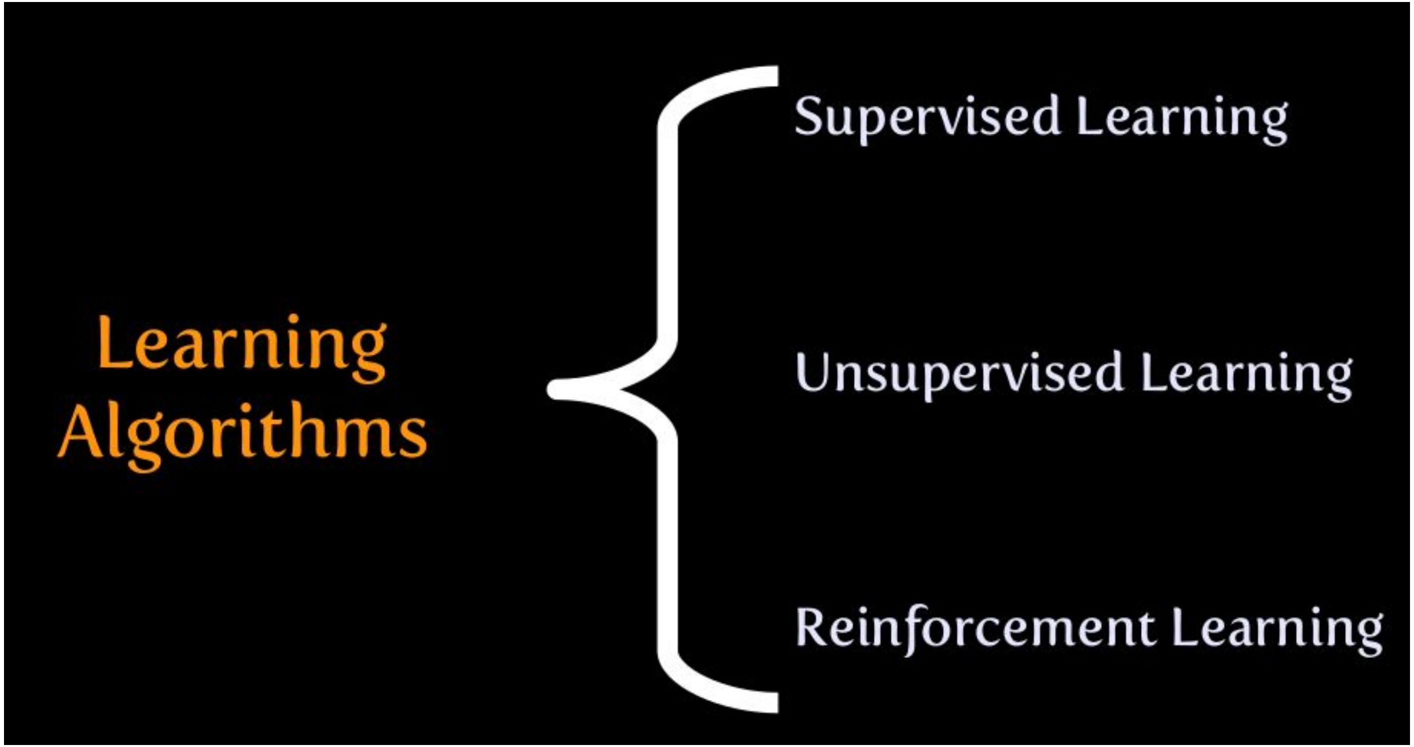
Introduction to Machine Learning Using Python

Contents:

Definition:

A computer program is said to 'learn' from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E

Major Classes of Learning Algorithms



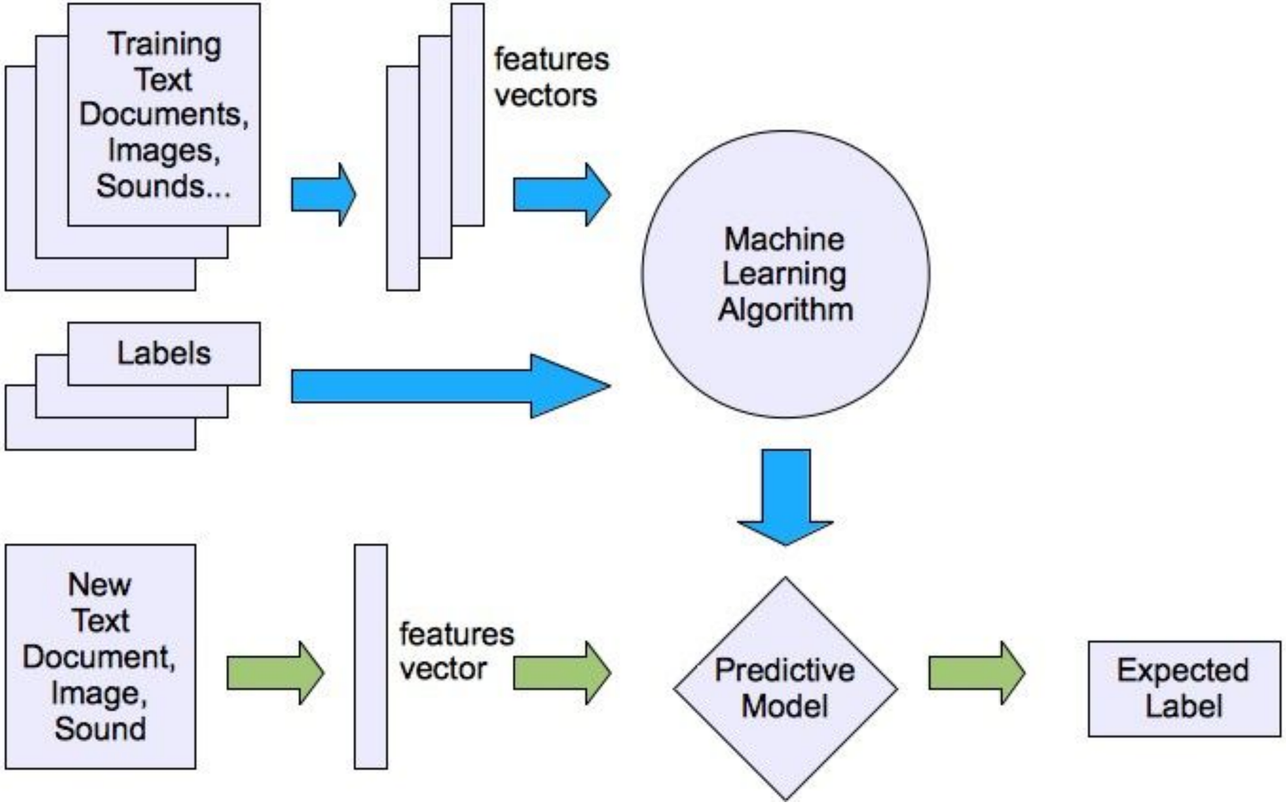
Supervised Learning

- The set of data (training data) consists of a set of input data and correct responses corresponding to every piece of data.
- Based on this training data, the algorithm has to generalize such that it is able to correctly (or with a low margin of error) respond to all possible inputs..

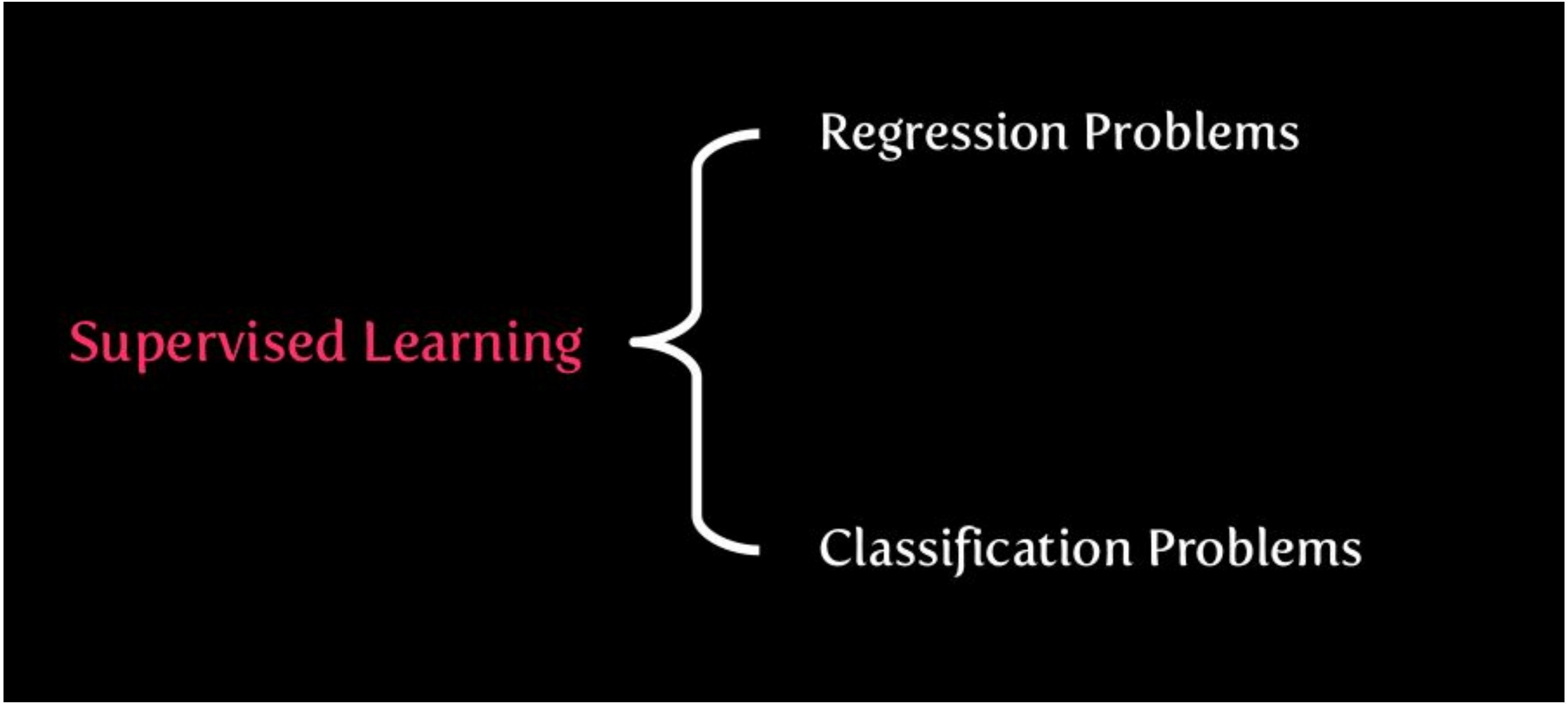
Supervised Learning

- In essence: The algorithm should produce sensible outputs for inputs that weren't encountered during training.
- Also called learning from exemplars

Supervised Learning



Supervised Learning

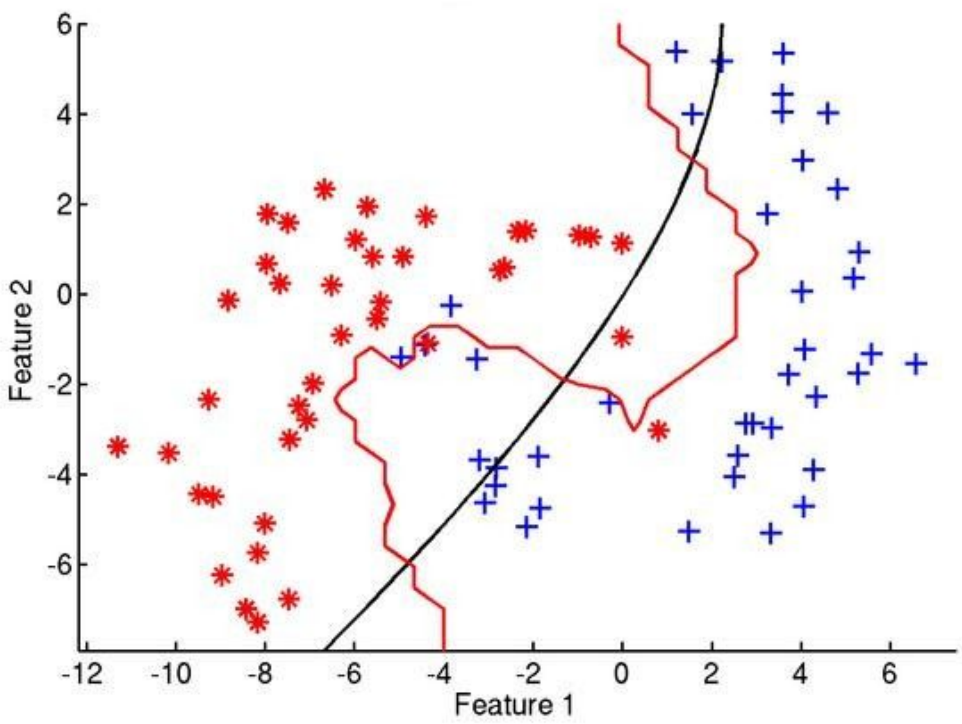


Supervised Learning: Classification Problems

“ Consists of taking input vectors and deciding which of the N classes they belong to, based on training from exemplars of each class.“

- Is discrete (most of the time). i.e. an example belongs to precisely one class, and the set of classes covers the whole possible output space.
- How it's done: Find 'decision boundaries' that can be used to separate out the different classes.
- Given the features that are used as inputs to the classifier, we need to identify some values of those features that will enable us to decide which class the current input belongs to

Supervised Learning: Classification Problems



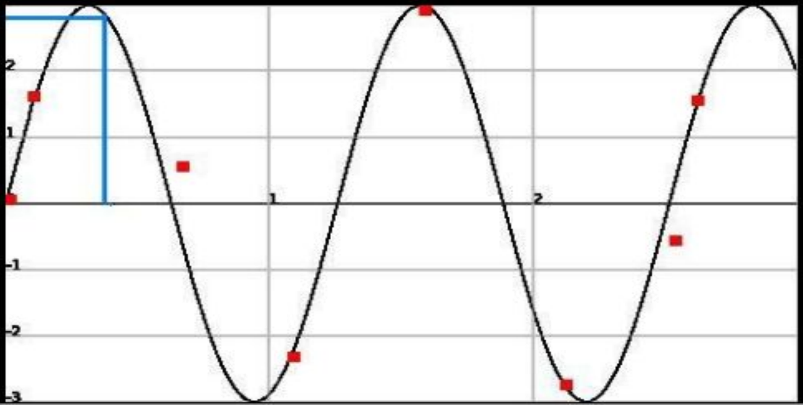
Supervised Learning: Regression Problems

- Given some data, you assume that those values come from some sort of function and try to find out what the function is.
- In essence: You try to fit a mathematical function that describes a curve, such that the curve passes as close as possible to all the data points.
- So, regression is essentially a problem of function approximation or interpolation

Supervised Learning: Regression Problems

x	y
0	0
0.5236	1.5
1.5708	3.0
2.0944	-2.5981
2.6180	1.5
2.6180	1.5
3.1416	0

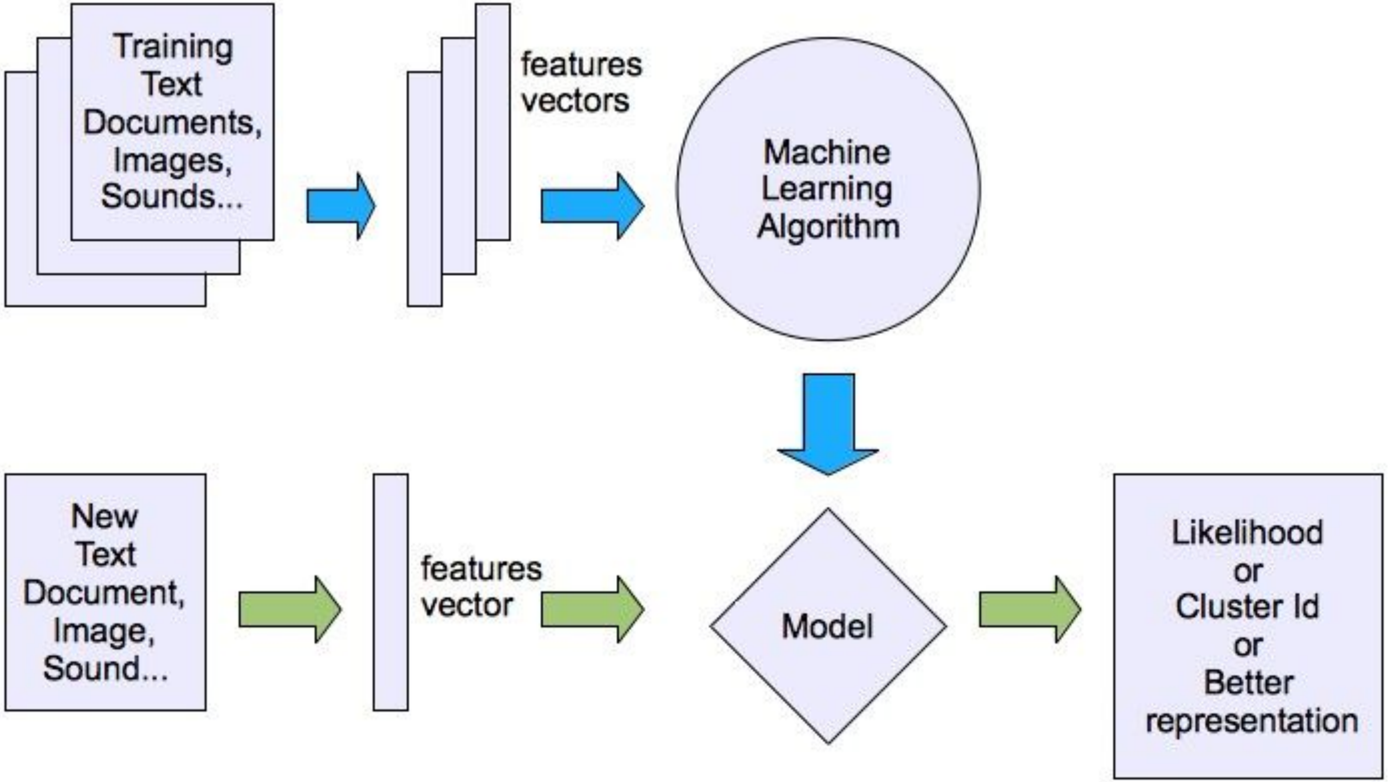
To Find: y at x =0.44



Unsupervised Learning

- Conceptually Different Problem.
- No information about correct outputs are available.
- No Regression => No guesses about the function can be made
- Classification?
  - No information about the correct classes. But if we design our algorithm so that it exploits similarities between inputs so as to cluster inputs that are similar together, this might perform classification automatically
  - In essence: The aim of unsupervised learning is to find clusters of similar inputs in the data without being explicitly told that some datapoints belong to one class and the other in other classes. The algorithm has to discover this similarity by itself

Unsupervised Learning



Supervised Learning: Linear Regression & Gradient Descent

Notation:

m : Number of training examples

x : Input variables (Features)

y: Output variables (Targets)

(x,y): Training Example (Represents 1 row on the table)

(x (i) , y (i)) : ith training example (Represent's ith row on the table)

n : Number of features (Dimensionality of the input)

Representation of the Hypothesis (Function):

- In this case, we represent 't' as a linear combination of the inputs (x)
- Which leads to:

$$h(\Theta) = \Theta_0 + \Theta_1 x_1 + \Theta_2 x_2$$

where (Θ_i's) are the parameters (also called weights).

Representation of the Hypothesis (Function):

- For convenience and ease of representation: x_0 =1

  So that, the above equation becomes:

$$h(x) = \sum_{i=0}^{n} (\Theta^T x)$$

- The objective now, is to 'learn' the parameters Θ

  So that h(x) becomes as close to 'y' at least for the training set.

Linear Regression

Define a function that measures for each value of the theta's, how close the $h(x^{(i)})'s$

are to the corresponding $y^{(i)}'s$

We define the 'cost function' as:

$$J(\Theta)=\frac{1}{2}\sum_{i=1}^{m}(h_{(\Theta)}(x^{(i)})-y^{(i)})^2$$

We want to choose the parameters so as to minimize $J(\Theta)$

The LMS (Least Mean Squares) algorithm begins with some initial value of Θ and repeatedly changes Θ so as to make $J(\Theta)$ smaller

Gradient Descent algorithm

We now come to the Gradient Descent algorithm:

Gradient Descent starts off with some initial Θ, and continually performs the following update:

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

(This update is simultaneously performed for all values of j=0,1,....,n)

α *is called the learning rate*

This, in effect assumes the following form:

$$\Theta_j := \Theta_j - \alpha \frac{\partial}{\partial \theta_j} \left(\frac{1}{2} \sum_{i=1}^{m} \left(h_{(\Theta)}(x^{(i)}) - y^{(i)} \right)^2 \right)$$

I'll leave this to you :)

A little mathematical 'hacking' of the above equation yields: (for a single training example)

$$\Theta_j := \Theta_j + \alpha\left(y^{(i)} - h_\Theta(x^{(i)})\right) x_j^{(i)}$$

There are 2 ways of generalizing the above equation for more than one training example:

The first one is:

Repeat until convergence

{

$$\Theta_j := \Theta_j - \alpha \sum_{i=1}^{m} \left(y^{(i)} - h_\Theta(x^{(i)})\right) x_j^{(i)}$$ For every j

}

This above method is called batch gradient descent

The second one is:

```
Loop
{
    for i=1 to m
    {
```

$$\Theta_j := \Theta_j + \alpha\left(y^{(i)} - h_\Theta(x^{(i)})\right) x_j^{(i)} \qquad \text{(for every j)}$$

```
    }
}
```

This is called stochastic gradient descent or incremental gradient descent

Practice: Linear Regression using numpy, pandas, matplotlib and scikit-learn

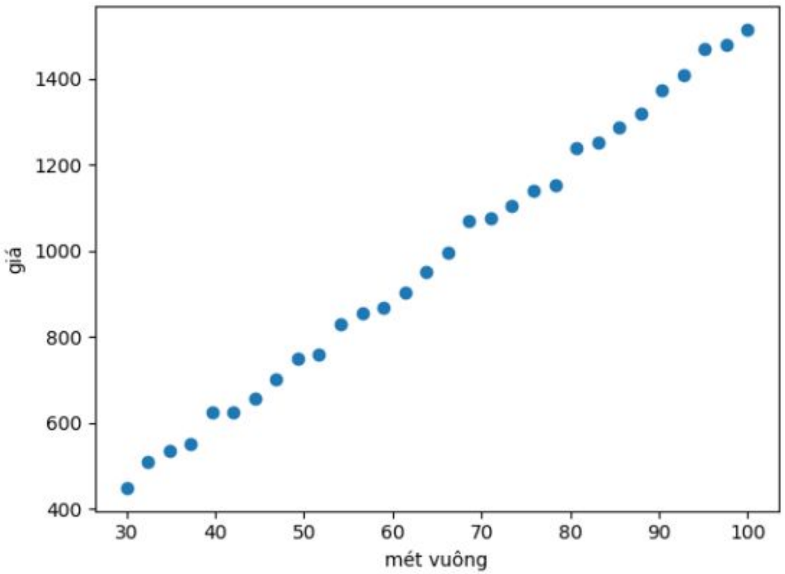
SECSION 2:

Problem

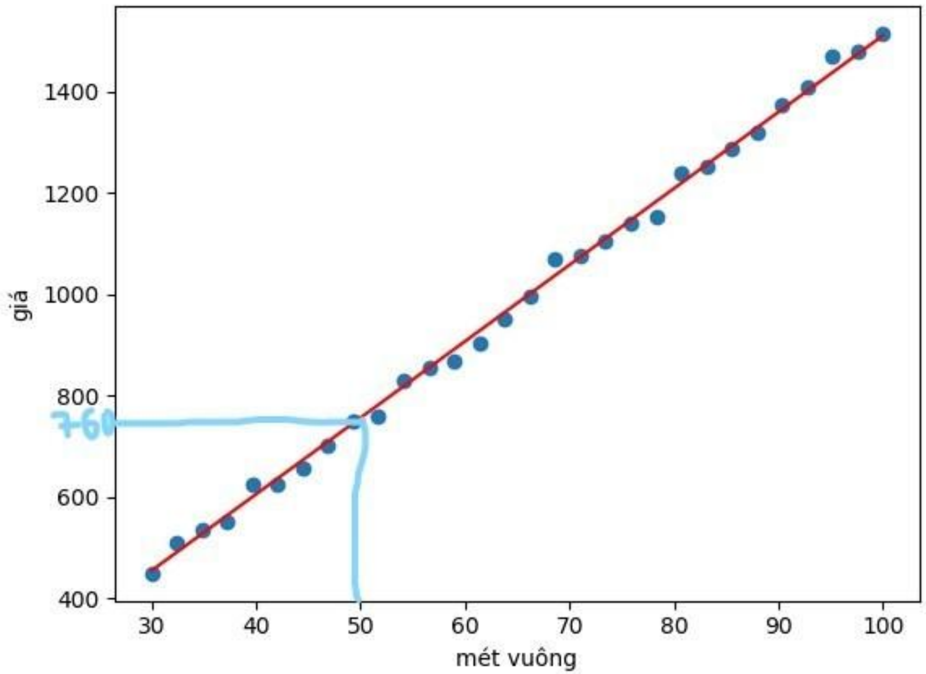
Data

Diện tích(m2)	Giá bán (triệu VNĐ)
30	448.524
32.4138	509.248
34.8276	535.104
37.2414	551.432
39.6552	623.418

EDA Data

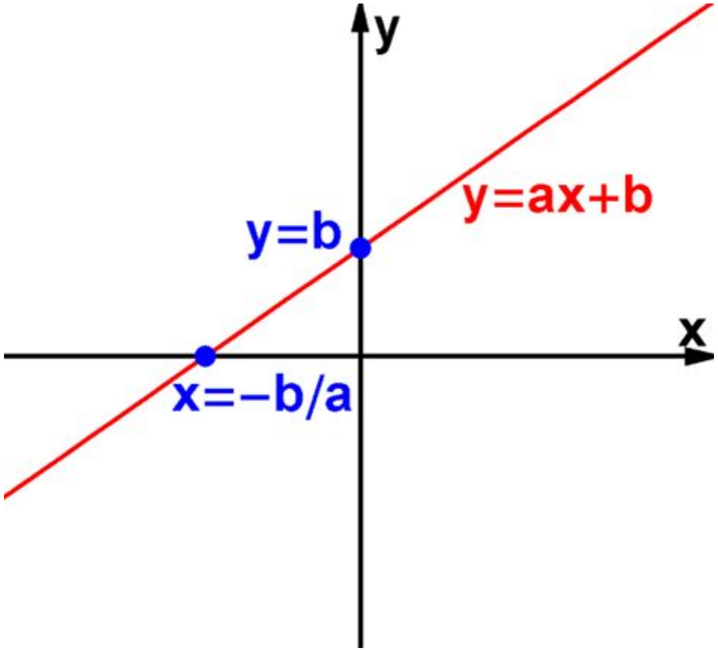


Predict



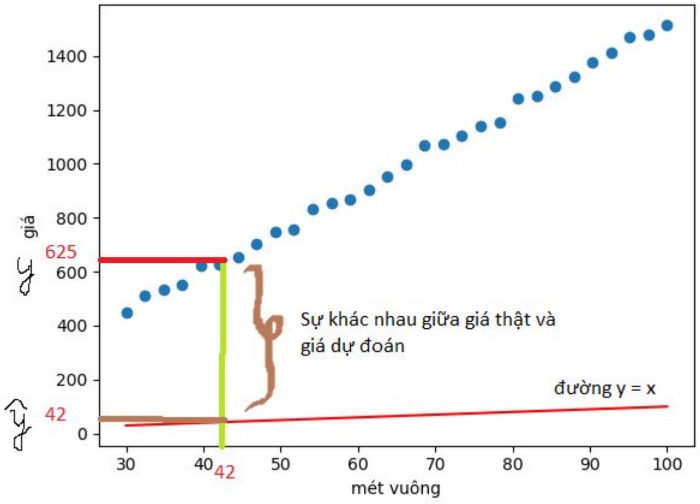
1400
1200
1000
800
760
600
400
giá
30
40
50
60
70
80
90
100
mét vuông

Linear equation



$$y = w_1 * x + w_0$$

Loss function



Loss function

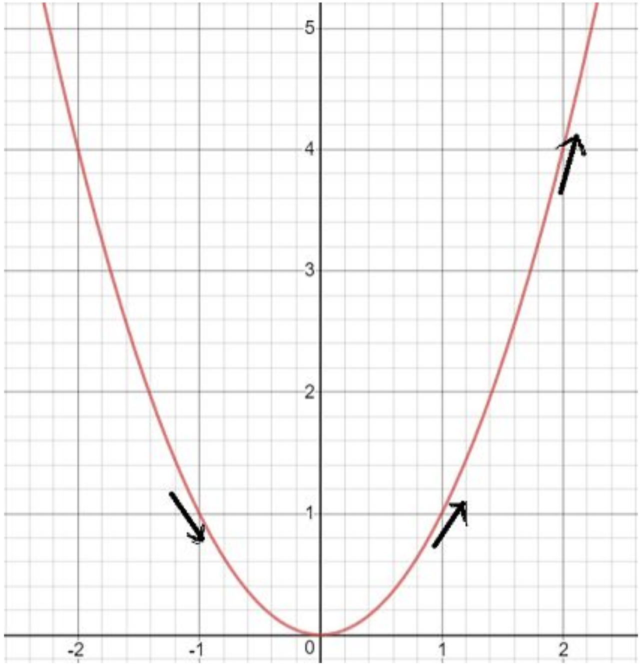
$$J = \frac{1}{2} * \frac{1}{N} * \left(\sum_{i=1}^{N} (\hat{y_i} - y_i)^2\right)$$

$$\hat{y_i} = w_1 * x_i + w_0$$

- J không âm
- J càng nhỏ thì đường thẳng càng gần điểm dữ liệu.
- Nếu J = 0 thì đường thẳng đi qua tất các điểm dữ liệu.

Derivative

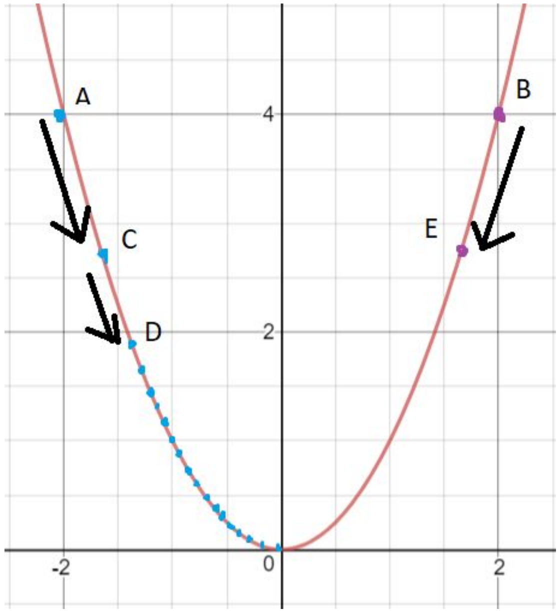
$f(x) = x^2$ là $f'(x) = \frac{df(x)}{dx} = 2 * x$



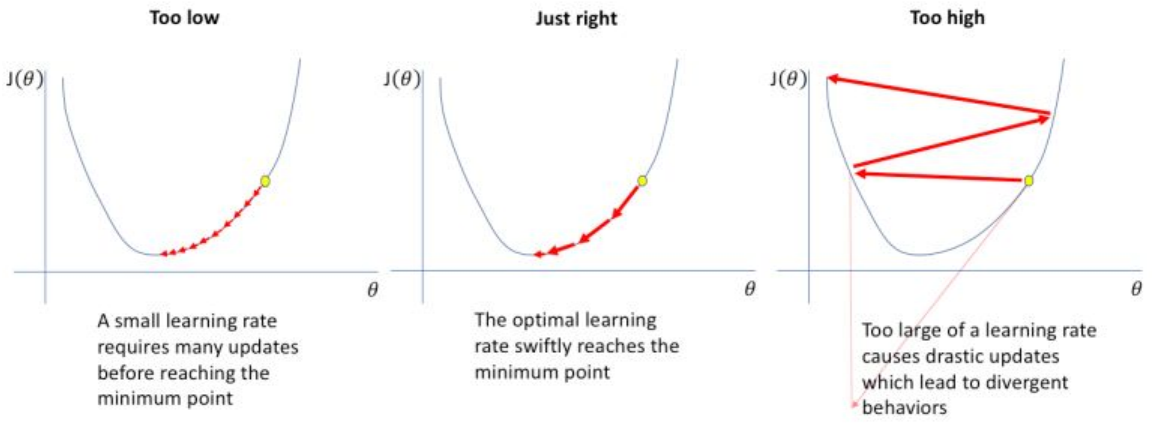
- f'(1) = 2 * 1 < f'(2) = 2 * 2 => đồ thị gần điểm x = 2 dốc hơn đồ thị gần điểm x = 1 => trị tuyệt đối của đạo hàm tại một điểm càng lớn thì gần điểm đấy càng dốc.
- f'(-1) = 2 * (-1) = -2 < 0 => đồ thị đang giảm hay khi tăng x thì y sẽ giảm; ngược lại đạo hàm tại điểm nào đó mà dương thì đồ thị quanh điểm đấy đang tăng.

Gradient descent

- Bước 1: Khởi tạo giá trị x tùy ý
- Bước 2: Gán x = x – learning_rate * f'(x)(learning_rate là hằng số không âm ví dụ learning_rate = 0.001)
- Bước 3: Tính lại f(x):
  - Nếu f(x) đủ nhỏ thì dừng lại.
  - Ngược lại tiếp tục bước 2.

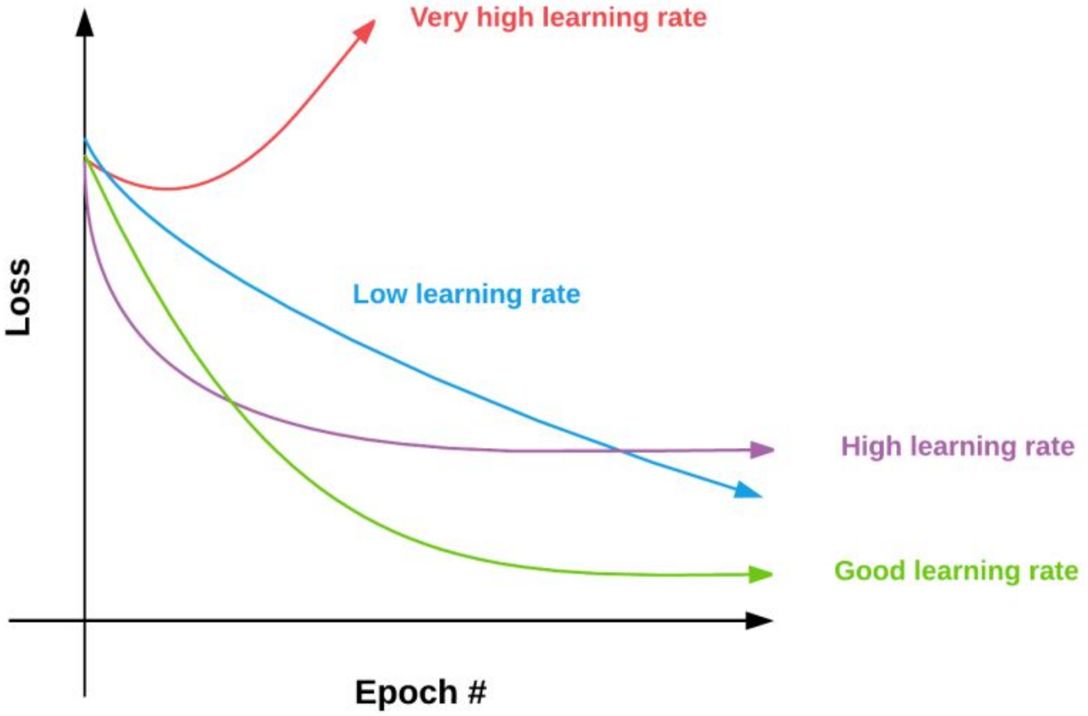


Check learning rate



- Nếu learning_rate nhỏ: mỗi lần hàm số giảm rất ít nên cần rất nhiều lần thực hiện bước 2 để hàm số đạt giá trị nhỏ nhất
- Nếu learning_rate hợp lý: sau một số lần lặp bước 2 vừa phải thì hàm sẽ đạt giá trị đủ nhỏ.
- Nếu learning_rate quá lớn: sẽ gây hiện tượng overshoot và không bao giờ đạt được giá trị nhỏ nhất của hàm.

Check loss function



Derivative calculative

$$J(w_0, w_1) = \frac{1}{2} * (\sum_{i=1}^{N} (\hat{y}_i - y_i)^2) = \frac{1}{2} * (\sum_{i=1}^{N} (w_0 + w_1 * x_i - y_i)^2)$$

$$\frac{dJ}{dw_0} = \sum_{i=1}^{N} (w_0 + w_1 * x_i - y_i)$$

$$\frac{dJ}{dw_1} = \sum_{i=1}^{N} x_i * (w_0 + w_1 * x_i - y_i)$$

Vector representation

$$X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ ... & ... \\ 1 & x_n \end{bmatrix}, Y = \begin{bmatrix} y_1 \\ y_2 \\ ... \\ y_n \end{bmatrix}, W = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$$

$$\hat{Y} = X * W = \begin{bmatrix} w_0 + w_1 * x_1 \\ w_0 + w_1 * x2 \\ ... \\ w_0 + w_1 * x_n \end{bmatrix}$$

Derivative vector

$$X[:,1] = \begin{bmatrix} x_1 \\ x_2 \\ ... \\ x_n \end{bmatrix}, sum(X[:,1]) = x_1 + x_2 + ... + x_n$$

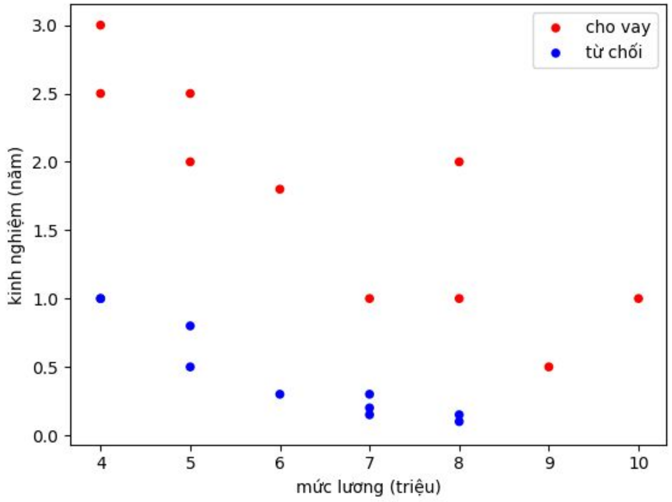
$$\frac{dJ}{dw_0} = sum(\hat{Y} - Y), \frac{dJ}{dw_1} = sum(X[:,1] \otimes (\hat{Y} - Y))$$

Logistic regression

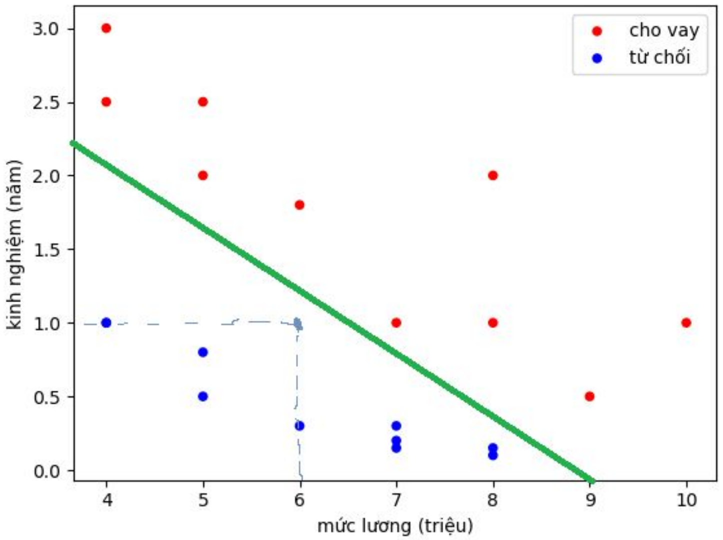
Data

Lương	Thời gian làm việc	Cho vay
10	1	1
5	2	1
...	...	1
8	0.1	0
7	0.15	0
...	...	0

EDA Data


cho vay
từ chối
kinh nghiệm (năm)
mức lương (triệu)
3.0
2.5
2.0
1.5
1.0
0.5
0.0
4
5
6
7
8
9
10

Predict



3.0
2.5
2.0
1.5
1.0
0.5
0.0
kinh nghiệm (năm)
4
5
6
7
8
9
10
mức lương (triệu)
cho vay
từ chối

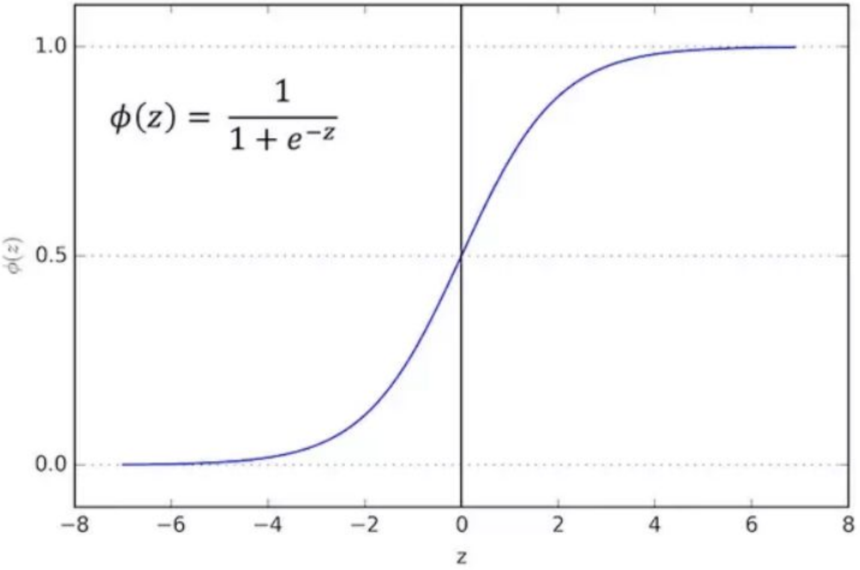
Probability

Theo wiki, “*Các nhà toán học coi xác suất là các số trong khoảng [0,1], được gán tương ứng với một biến cố mà khả năng xảy ra hoặc không xảy ra là ngẫu nhiên*“

- Xác xuất của 1 sự kiện trong khoảng [0,1]
- Sự kiện bạn càng chắc chắc xẩy ra thì xác xuất càng cao. Ví dụ bạn lương cao và còn đi làm lâu lăm thì xác xuất bạn được vay mua chung cư là cao.
- Tổng xác xuất của sự kiện A và và sự kiện phủ định của A là 100% (hay 1). Ví dụ sự kiện A: tung đồng xu mặt ngửa, xác xuất 50%; phủ định của sự kiện A: tung đồng xu mặt sấp, xác xuất 50% => tổng 100%

Sigmoid function

- Hàm số liên tục, nhận giá trị thực trong khoảng (0,1).
- Hàm có đạo hàm tại mọi điểm (để áp dụng gradient descent)



Data Presentation

Với dòng thứ i trong bảng dữ liệu, gọi $x_1^{(i)}$ là lương và $x_2^{(i)}$ là thời gian làm việc của hồ sơ thứ i .

p($x^{(i)} = 1) = \hat{y}_i$ là xác xuất mà model dự đoán hồ sơ thứ i được cho vay.

p($x^{(i)} = 0) = 1 - \hat{y}_i$ là xác xuất mà model dự đoán hồ sơ thứ i không được cho vay.

=> p($x^{(i)} = 1)$ + p($x^{(i)} = 0)$ = 1

Hàm sigmoid: $\sigma(x) = \dfrac{1}{1+e^{-x}}$.

Như bài trước công thức của linear regression là: $\hat{y}_i = w_0 + w_1 * x_i$ thì giờ công thức của logistic regression là:

$$\hat{y}_i = \sigma(w_0 + w_1 * x_1^{(i)} + w_2 * x_2^{(i)}) = \frac{1}{1+e^{-(w_0 + w_1 * x_1^{(i)} + w_2 * x_2^{(i)})}}$$

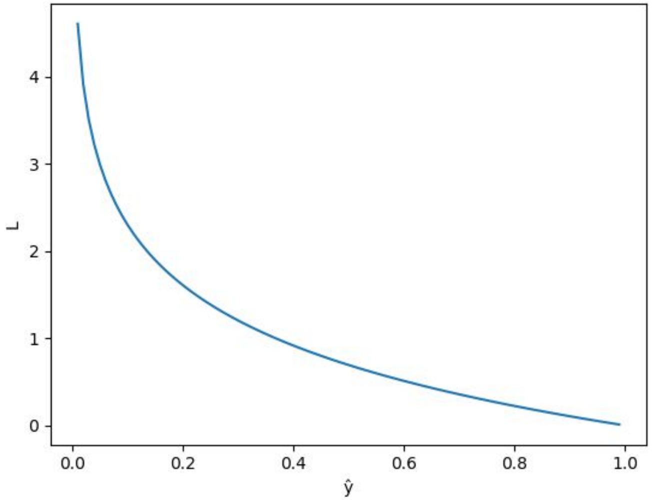
Loss function

$$L = -(y_i * log(\hat{y_i}) + (1-y_i) * log(1-\hat{y_i}))$$

- Nếu hồ sơ thứ i là cho vay (y=1) thì ta cũng mong muốn giá trị dự đoán càng gần 1 càng tốt hay model dự đoán xác xuất người thứ i được vay vốn càng cao càng tốt.
- Nếu hồ sơ thứ i không được vay (y=0) thì ta cũng mong muốn giá trị dự đoán càng gần 0 càng tốt hay model dự đoán xác xuất người thứ i được vay vốn càng thấp càng tốt

Loss function

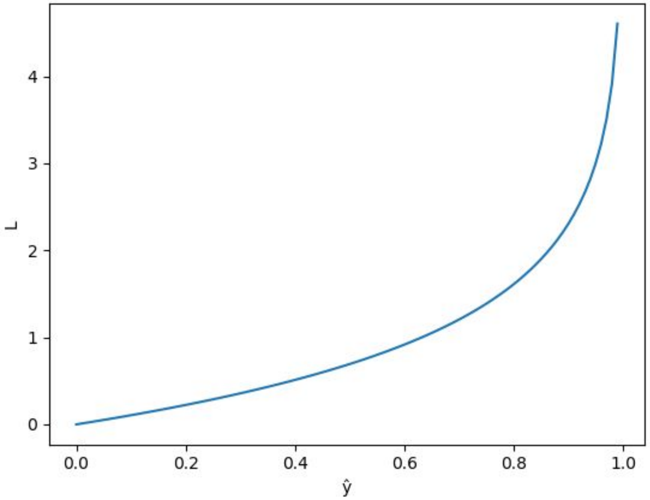
y=1



- Hàm L giảm dần từ 0 đến 1
- Khi model dự đoán $\hat{y_i}$ gần 1, tức giá trị dự đoán gần với giá trị thật y_i thì L nhỏ, xấp xỉ 0
- Khi model dự đoán $\hat{y_i}$ gần 0, tức giá trị dự đoán ngược lại giá trị thật y_i thì L rất lớn

Loss function

y=0



- Hàm L tăng dần từ 0 đến 1
- Khi model dự đoán $\hat{y_i}$ gần 0, tức giá trị dự đoán gần với giá trị thật y_i thì L nhỏ, xấp xỉ 0
- Khi model dự đoán $\hat{y_i}$ gần 1, tức giá trị dự đoán ngược lại giá trị thật y_i thì L rất lớn

Loss function

$z = f(y)$ và $y = g(x)$ hay $z = f(g(x))$ thì $\dfrac{dz}{dx} = \dfrac{dz}{dy} * \dfrac{dy}{dx}$

Ví dụ:

$z(x) = (2x+1)^2$, có thể thấy $z = f(g(x))$ trong đó $f(x) = x^2, g(x) = 2x + 1$

$$\frac{dz}{dx} = \frac{dz}{dy} * \frac{dy}{dx} = \frac{d(2x+1)^2}{d(2x+1)} * \frac{d(2x+1)}{dx} = 2 * (2x+1) * 2 = 4 * (2x+1)$$

$\dfrac{d(\sigma(x))}{dx}$ = ???

Chain rule

L = $-(y_i * log(\hat{y_i}) + (1-y_i) * log(1-\hat{y_i}))$ trong đó $\hat{y_i} = \sigma(w_0 + w_1 * x_1^{(i)} + w_2 * x_2^{(i)})$

$$\frac{dL}{dw_0} = \frac{dL}{d\hat{y_i}} * \frac{d\hat{y_i}}{dw_0}$$

$$\frac{dL}{d\hat{y_i}} = -\frac{d(y_i * log(\hat{y_i}) + (1-y_i) * log(1-\hat{y_i}))}{d\hat{y_i}} = -(\frac{y_i}{\hat{y_i}} - \frac{1-y_i}{(1-\hat{y})})$$

$$\frac{d\hat{y_i}}{dw_0} = \frac{\sigma(w_0 + w_1 * x_1^{(i)} + w_2 * x_2^{(i)})}{dw_0} = \hat{y_i} * (1 - \hat{y_i})$$

$$\frac{dL}{dw_0} = \frac{dL}{d\hat{y_i}} * \frac{d\hat{y_i}}{dw_0} = -(\frac{y_i}{\hat{y_i}} - \frac{1-y_i}{(1-\hat{y_i})}) * \hat{y_i} * (1 - \hat{y_i}) = -(y_i * (1 - \hat{y_i}) - (1 - y_i) * \hat{y_i})) = \hat{y_i} - y_i$$

Vector representation

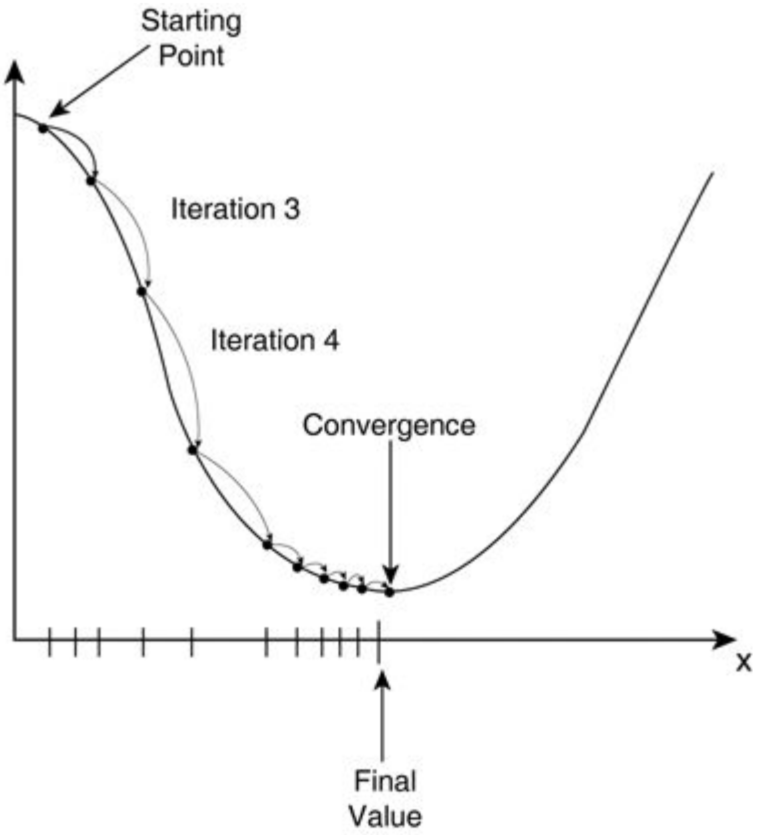
$$X = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)}) \\ 1 & x_1^{(2)} & x_2^{(2)}) \\ 1 & ... & ... \\ 1 & x_1^{(n)} & x_2^{(n)}) \end{bmatrix}, y = \begin{bmatrix} y_1 \\ y_2 \\ ... \\ y_n \end{bmatrix}, w = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}$$

$$\hat{y} = \sigma(Xw)$$

$$J = -sum(y \otimes log(\hat{y}) + (1 - y) \otimes log(1 - \hat{y}))$$

$$\frac{dJ}{dw} = X^T * (\hat{y} - y), X^T = \begin{bmatrix} 1 & 1 & ... & 1 \\ x_1^{(1)} & x_1^{(2)} & ... & x_1^{(n)} \\ x_2^{(1)} & x_2^{(2)} & ... & x_2^{(n)}) \end{bmatrix}$$

stochastic gradient descent



Practice

1. Using package numpy, pandas, matplotlib for EDA Data for linear regression
2. Write Code Linear Regression (submit code to Practice)
3. Using scikit-learn caculative auc metric.
4. Using package numpy, pandas, matplotlib for EDA Data for logistic regression
5. Write Code Linear Regression (submit code to Practice)

Scikit-learin - Introduction

[Scikit-learn tutorial]

Install package using pip:

```
pip install -U scikit-learn
```

Using Scikit-learn:

```
import skelearn
```

Scikit-learn Example

```
import matplotlib.pyplot as plt

import numpy as np

from sklearn import datasets, linear_model

from sklearn.metrics import mean_squared_error, r2_score



# Load the diabetes dataset

diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)



# Use only one feature

diabetes_X = diabetes_X[:, np.newaxis, 2]
```

Scikit-learn Example

```
# Split the data into training/testing sets

diabetes_X_train = diabetes_X[:-20]

diabetes_X_test = diabetes_X[-20:]



# Split the targets into training/testing sets

diabetes_y_train = diabetes_y[:-20]

diabetes_y_test = diabetes_y[-20:]



# Create linear regression object

regr = linear_model.LinearRegression()
```

Scikit-learn Example

```
# Train the model using the training sets

regr.fit(diabetes_X_train, diabetes_y_train)



# Make predictions using the testing set

diabetes_y_pred = regr.predict(diabetes_X_test)



# The coefficients

print('Coefficients: \n', regr.coef_)
```

Scikit-learn Example

```
# The mean squared error

print('Mean squared error: %.2f'

      % mean_squared_error(diabetes_y_test, diabetes_y_pred))

# The coefficient of determination: 1 is perfect prediction

print('Coefficient of determination: %.2f' % r2_score(diabetes_y_test, diabetes_y_pred))

# Plot outputs

plt.scatter(diabetes_X_test, diabetes_y_test,  color='black')

plt.plot(diabetes_X_test, diabetes_y_pred, color='blue', linewidth=3)

plt.xticks(())

plt.yticks(())
```

Machine learning - Example

1. Linear_regression
2. Logistic_regression
3. K_mean
4. CNN Example for cifar10
5. CNN Example for MNIST