🤖Machine Learning-Notebook

Andrew Ng-吴恩达©
Standford ONLINE & DeepLearning.AI
Mungeryang-杨桂森总结

1 📖Introduction

“Field of study that gives computers the ability to learn without being explicitly programmed.”——Arthur Samuel(1995)

Practical advice for applying learning algorithm

2 Basic conception cookbook

Data set:

Training set:

Test set:

Cost function:

Gradient:

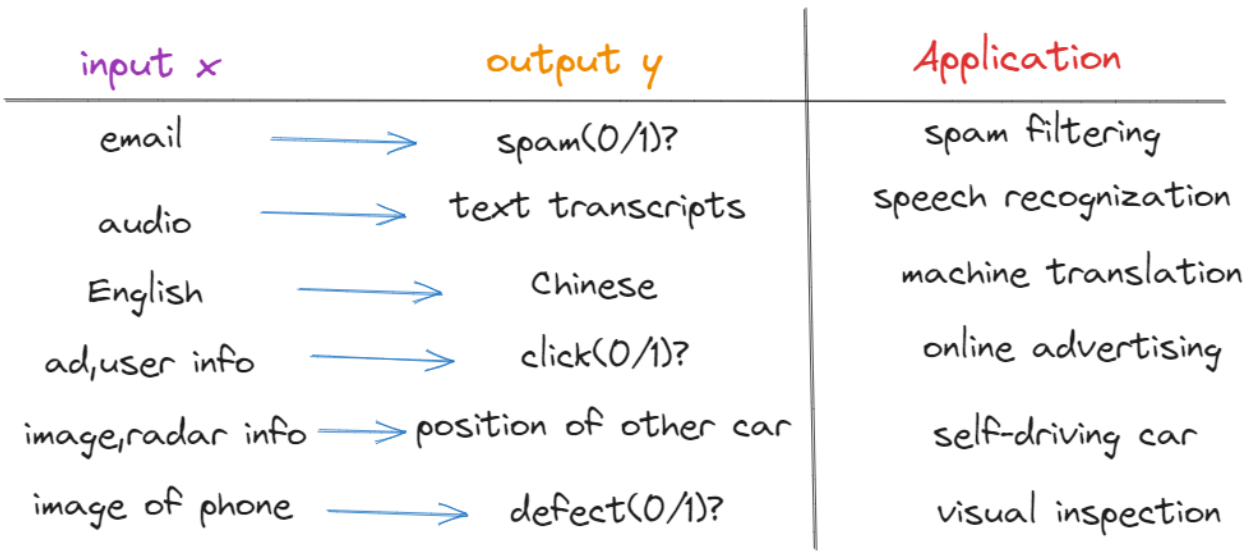
Gradient descent:

Recall:

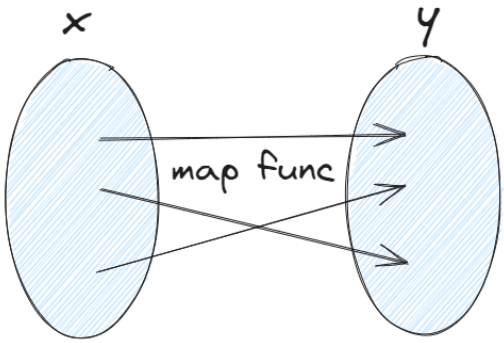
Precision:

3 📒Supervised learning-监督学习

Learns from being given “**right answers**(labels)”

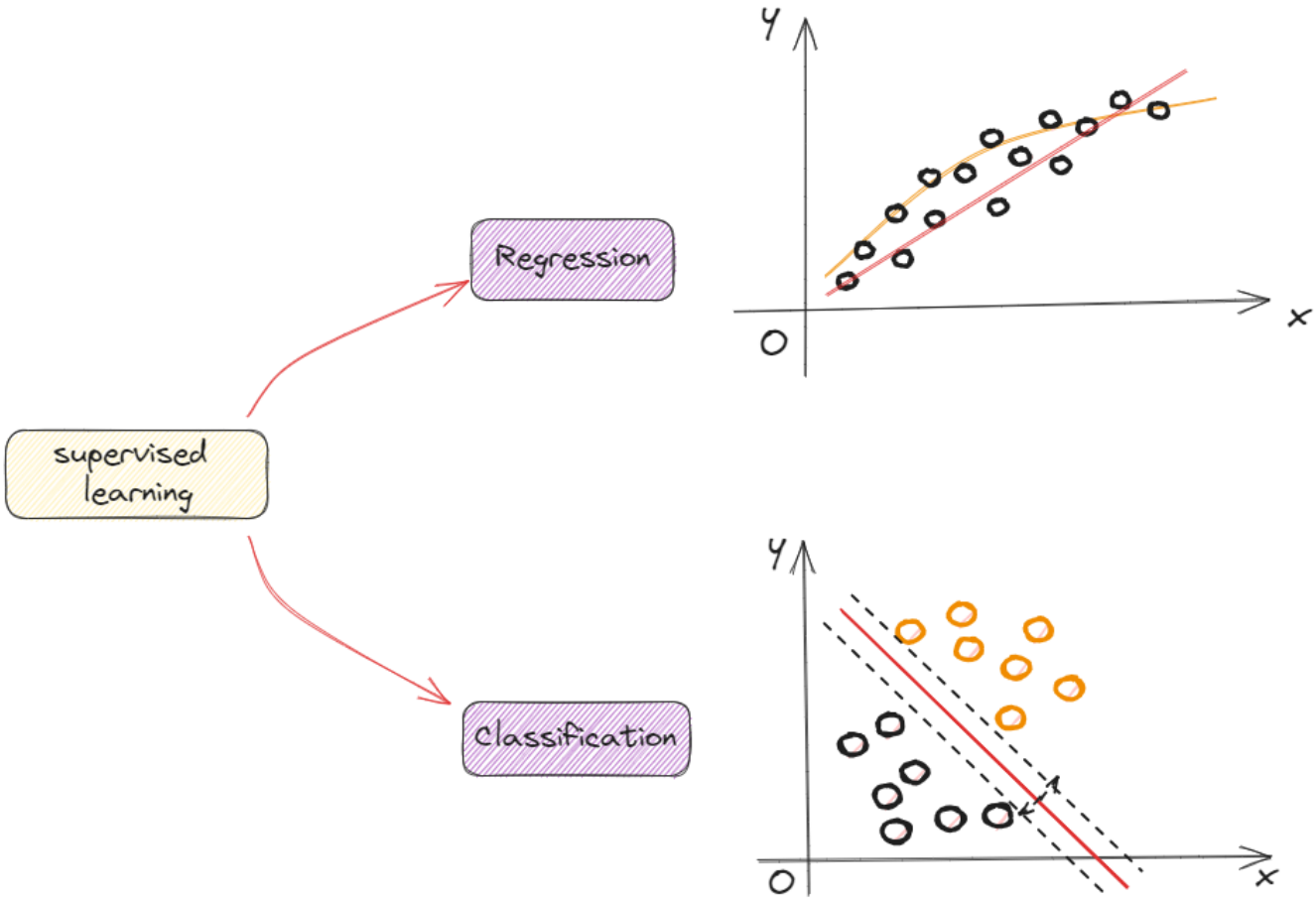


In all of these applications, we will first train our model with examples of inputs **x** and right answers, that is the labels **y**. After the model has learned from these input, output, or x and y pairs, they can take a brand new input x, something it has never seen before, and try to produce the appropriate corresponding output y.



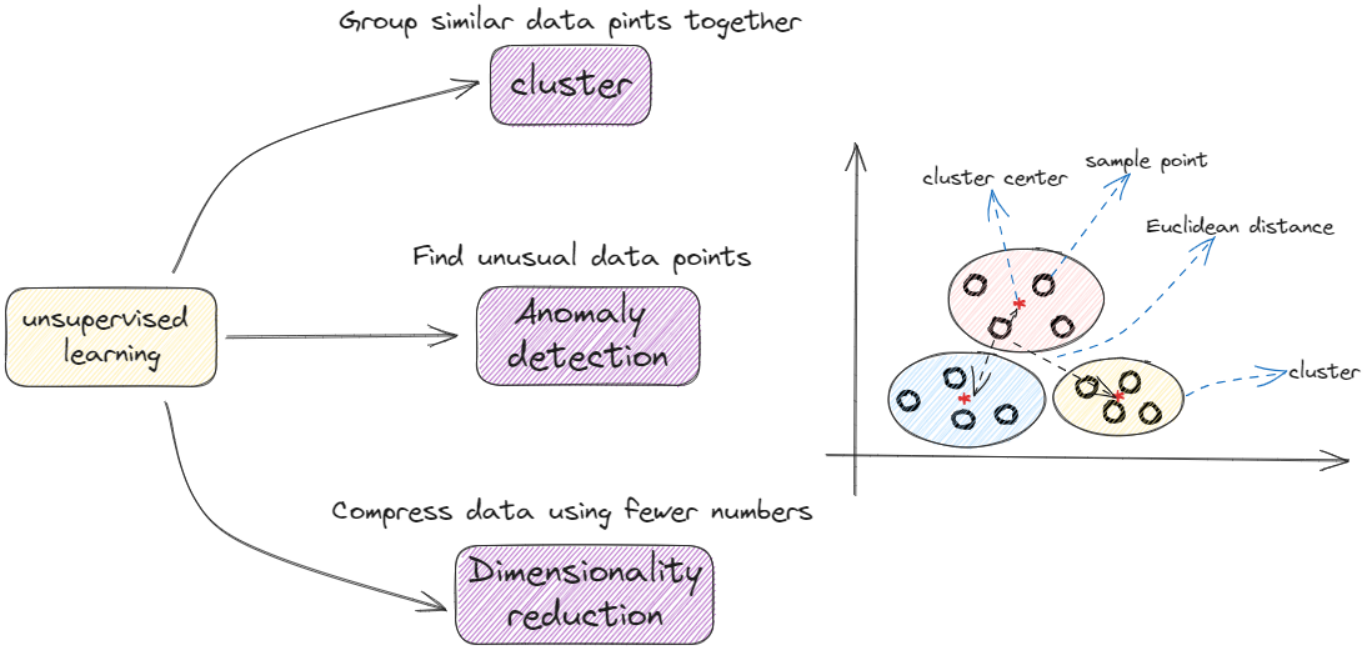
The task of the supervised learning algorithm is to produce more of these right answers based on labels.

Classification is to predict categories,Regression is to predict a number.



4 📝Unsupervised learning-无监督学习

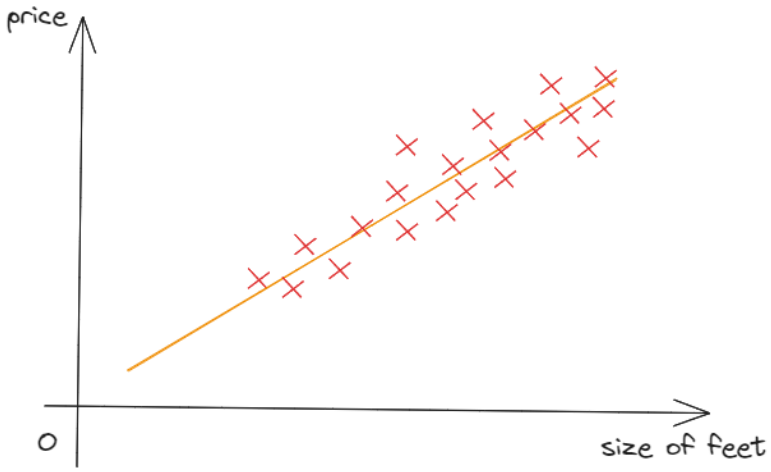
Learns from being given "**no-right answers**(unlabeled data)", data only comes with inputs x, but not output labels y. Algorithm has to find structure automatically in the data and automatically figure out whether the major types of individuals.



5 Linear Regression with one variable

5.1 Definition

Linear regression means fitting a `straight line` to the data. ->linear regression



Regression model is to predict numbers

Classification model is to predicts categories

5.2 🍦 Terminology in ML

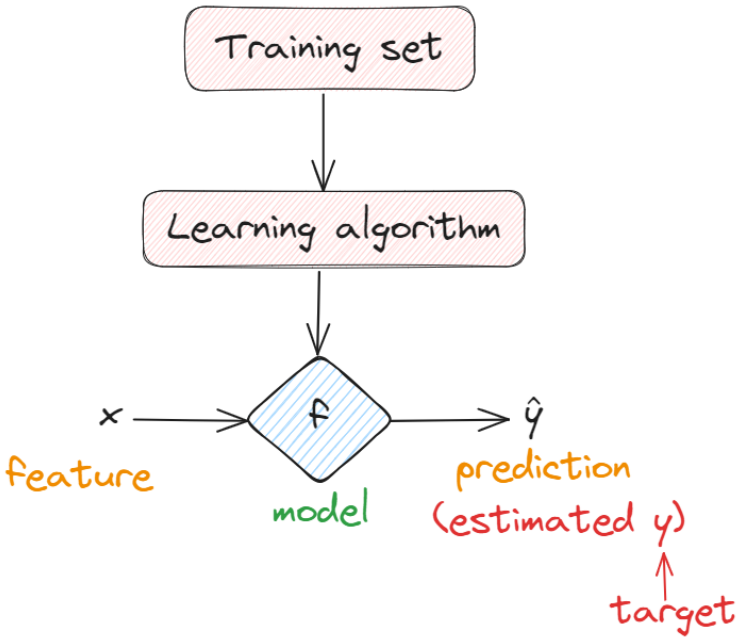
`Training dataset` ->data used to train the model

x =“input”variables feature

y =“output ”variables or “target”variables

(x, y) =single training example

$(x^{(i)}, y^{(i)}) = i^{th}$ training example not exponent



In the linear regression, we instantly believe the function is a linear function as follow:

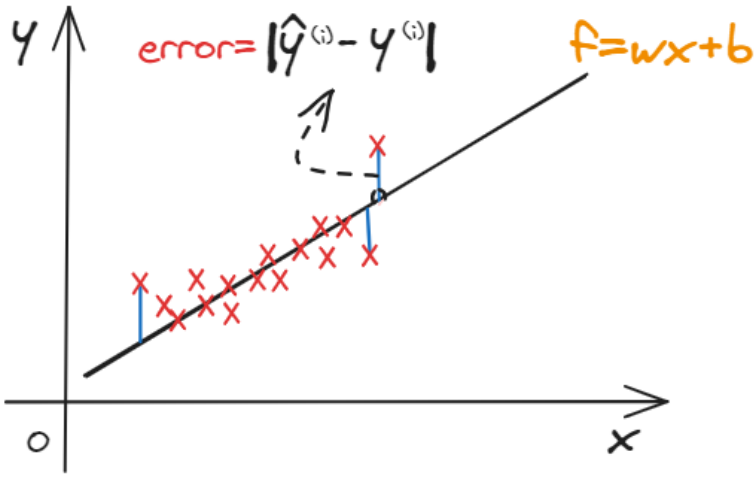
$$\begin{aligned} f_{w,b}(X) &= wX + b \\ f(X) &= wX + b \end{aligned} \tag{1}$$

when we give a “x”as a input variable, we can get a “y-hat”variable as a result.

5.3 Cost function

squared error

$$\sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)}\right)^2 \tag{2}$$



Model: $f_{w,b}(X) = wX + b$. $w(slope), b(intersects)$ are parameters.

$$\begin{aligned} J_{(w,b)} &= \frac{1}{2m} \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)}\right)^2 = \frac{1}{2m} \sum_{i=1}^{m} \left(f_{w.b}\left(x^{(i)}\right) - y^{(i)}\right)^2 \\ &= \frac{1}{2m} \sum_{i=1}^{m} \left(wx^{(i)} + b - y^{(i)}\right)^2 \end{aligned} \tag{3}$$

Our goal is to minimize the parameters to fit the model:

$$\min_{w.b} \ J\left(w,b\right) \tag{4}$$

```
#define cost function
def compute_cost(x,y,w,b):
    m = x.shape[0]
    cost_sum = 0
    for i in range(m):
        f_wb = w * x[i] + b
        cost = (f_wb - y[i]) ** 2
        cost_sum += cost
    total_cost = (1/(2*m))*cost_sum
    return total_cost
```

5.4 Gradient descent-梯度下降

Simultaneous update the parameters w and b until the cost function is `convergence`:

Simultaneous update the parameters is significant, we must focus on the order about the algorithm.

Correct order:

$$\begin{aligned} tmp\_w &= w - \alpha \frac{\partial}{\partial w} J\left(w,b\right) \\ tmp\_b &= b - \alpha \frac{\partial}{\partial b} J\left(w,b\right) \\ w &= tmp\_w \\ b &= tmp\_b \end{aligned} \tag{5}$$

Incorrect order:

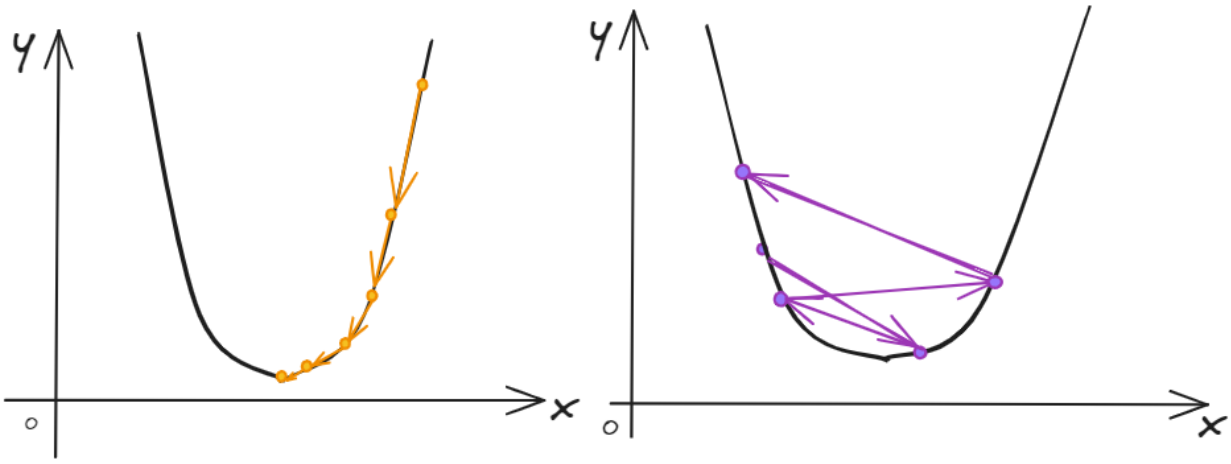
$$\begin{aligned} tmp\_w &= w - \alpha \frac{\partial}{\partial w} J\left(w,b\right) \\ w &= tmp\_w \\ tmp\_b &= b - \alpha \frac{\partial}{\partial b} J\left(w,b\right) \\ b &= tmp\_b \end{aligned} \tag{6}$$

```
#compute gradient
def compute_gradient(x,y,w,b):
    m = x.shape[0]
    dj_dw = 0
    dj_db = 0
    for i in range(m):
        f_wb = w * x[i] + b
        dj_dw_i = (f_wb - y[i]) * x[i]
        dj_db_i = f_wb - y[i]
        dj_dw += dj_dw_i
        dj_db += dj_db_i
    dj_dw = dj_dw / m
    dj_db = dj_db / m
    return dj_dw, dj_db
```

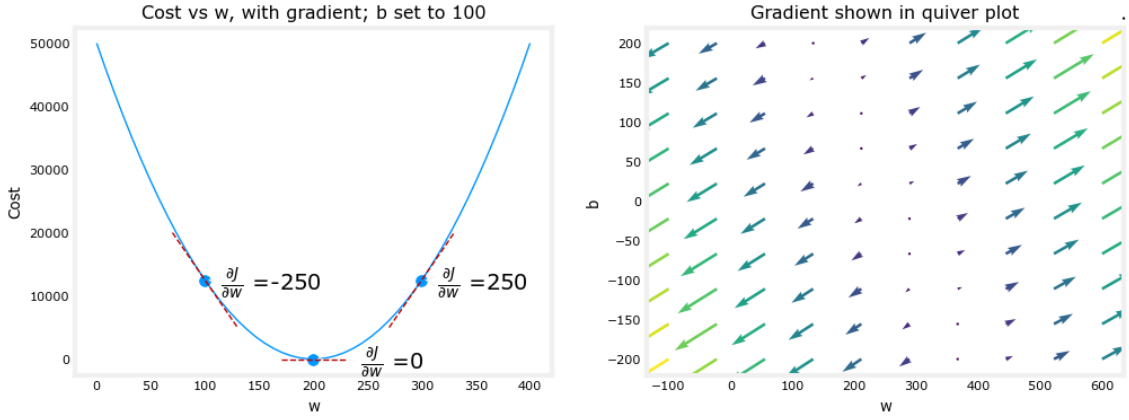
5.5 Learning Rate

The choice of learning rate, alpha(α) will have a huge impact on the efficiency of our implementation of Gradient descent.

$$\begin{aligned} w - \alpha \frac{\partial}{\partial w} J(w,b) \\ b - \alpha \frac{\partial}{\partial b} J(w,b) \end{aligned} \tag{7}$$



If we chose the alpha is too large, the fit efficiency is not very well. We can design the algorithm to decrease the alpha(α) following by the w and b.



6 Multiple linear regression

6.1 Multiple Features

【Model】

$$f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n \tag{8}$$

`Dot product(inner product)` of two vectors about w and b.

$$\begin{aligned} \vec{w} &= [w_1, w_2 \cdots, w_n] \\ \vec{x} &= [x_1, x_2 \cdots, x_n] \end{aligned} \tag{9}$$

$x_j = j^{th} features$

n = numbers of features

$\vec{x}^{(i)}$ = features of i^{th} training example

$\vec{x}_j^{(i)}$ = value of feature j in i^{th} training example

6.2 Vectorization

We contrast the two process between vectorization and without vectorization.

without vectorization

```
for i in range(0,n):
    f = f + w[i]*x[i]
f = f + b
```

If n is infinite, the algorithm is time consuming.

with vectorization

```
import numpy as np
f = np.dot(w,b)
```

Without vectorization, we just only use the for loop to calculate the results one by one. In the contrast, we can use the function numpy.dot() to calculate the result. Using numpy can decrease the time complexity and improve the algorithm efficiency. Numpy can use `parallel process` hardware to carry out the data.

Dot product of two vectors:

$$a \cdot b = \sum_{i=0}^{n-1} a_i b_i = \begin{bmatrix} a_0 \\ a_1 \\ \cdots \\ a_{n-1} \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ \cdots \\ b_{n-1} \end{bmatrix} = [a_0 b_0 + a_1 b_1 + \cdots + a_{n-1} b_{n-1}] \tag{10}$$

6.3 Gradient descent in Multiple regression

Parameters include $w_1, w_2, \cdots w_n$ and b

Model is $f_{\vec{w},b}\left(\vec{x}\right) = \vec{w} \cdot \vec{x} + b = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b$

Cost function is $J\left(w_1, w_2, \cdots w_n, b\right)$

Gradient descent:

repeat{

$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J\left(w_1, w_2, \cdots w_n, b\right) = w_j - \alpha \frac{\partial}{\partial w_j} J\left(\vec{w}, b\right) \tag{11}$$

$$b = b - \alpha \frac{\partial}{\partial b} J\left(w_1, w_2, \cdots w_n, b\right) = b - \alpha \frac{\partial}{\partial b} J\left(\vec{w}, b\right) \tag{12}$$

}

`for i in range(1,n)`:

$$\begin{aligned}\frac{\partial}{\partial w_1}J\left(\vec{w},b\right)&=\frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)-y^{(i)}\right)x_1^{(i)}\\\frac{\partial}{\partial w_n}J\left(\vec{w},b\right)&=\frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)-y^{(i)}\right)x_n^{(i)}\\\frac{\partial}{\partial b}J\left(\vec{w},b\right)&=\frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)-y^{(i)}\right)\end{aligned}\tag{13}$$

Normal equation:

- Only for linear regression
- Solve for w,b without iterations

Disadvantages:

- Does not generalize to other learning algorithm
- Slow when number of features is large(n>10000)

6.4 Feature engineering

The technology of Features scaling will enable Gradient descent to run much faster.

Z-score normalization:

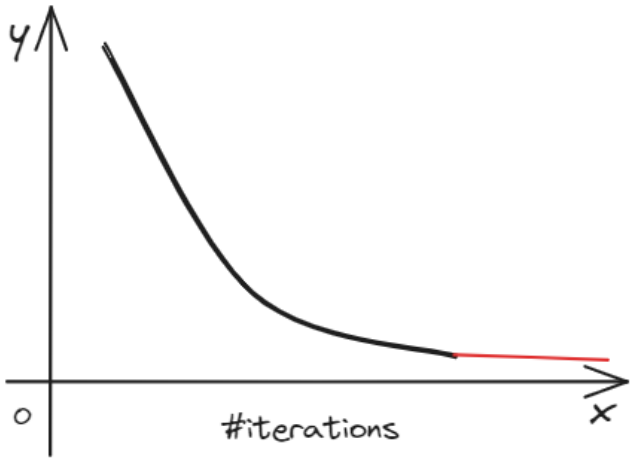
$$x_i=\frac{x_i-\mu}{\sigma}\tag{14}$$

μ is the sample average value, and σ is standard error of sample.

Aim for about $-1\leqslant x_j\leqslant 1$ for x_j.

The objective of Gradient descent in Multiple regression is:

$$\min_{\vec{w}.b}\ J\left(\vec{w},b\right)\tag{15}$$



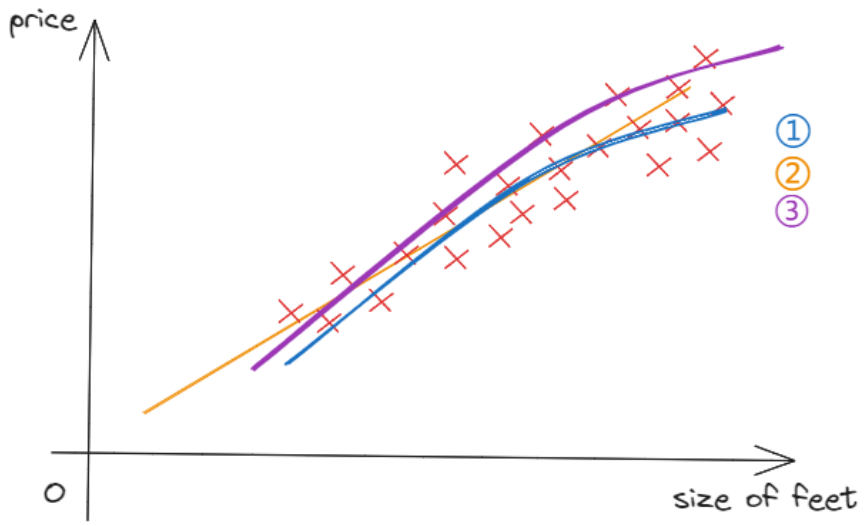
As the calculate process, $y=J\left(\vec{w},b\right)$ is decreased until a low score. If we can get the similar the result in the experiment, the cost function is `convergence` and we can find the min $J\left(\vec{w},b\right)$.

【Skill】

At the beginning, we can set the learning rate(α) in a small value like 0.001. As running the algorithm, we can update the parameter α gradually.(0.001->0.01->0.1)

Feature Engineering : Using `intuition` to design new features, by transforming or combining original features.

> [Wiki]:**Feature engineering** or **feature extraction** or **feature discovery** is the process of extracting features (characteristics, properties, attributes) from raw data.

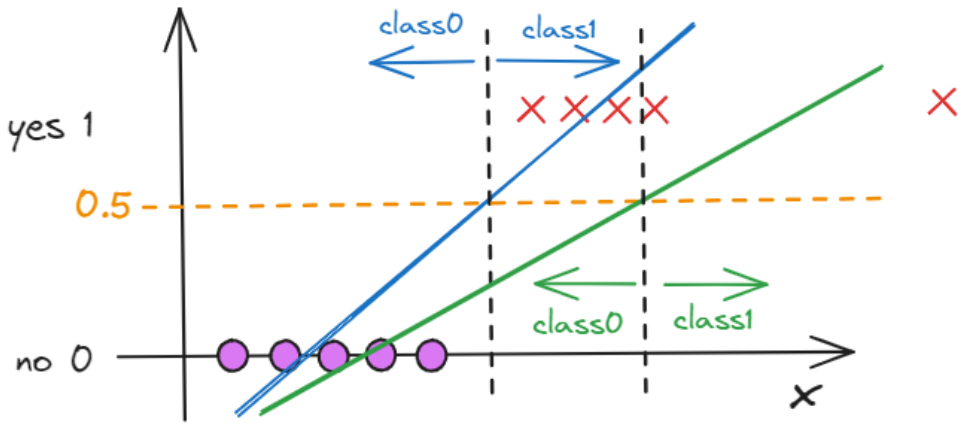


Except for using linear regression, we can also use `polynomial regression` to fit data. Through finding the segment of the data distribution(scatter), we can create special features($\sqrt{x}$ x^3) to fit data successfully.

$$
\begin{aligned}
f_{\vec{w},b}(x) &= w_1 x + b \quad ② \\
f_{\vec{w},b}(x) &= w_1 x + w_2 x^2 + b \quad ① \\
f_{\vec{w},b}(x) &= w_1 x + w_2 \sqrt{x} + b \quad ③
\end{aligned} \tag{16}
$$

7 Classification

It turns out that, linear regression is not the good algorithm for classification. For classification, the output is not a continuous number. In the fact, it is a class variable like **0(false-negative class)** or **1(true-positive class)**.



Sometimes, if we want to classify the output, the method of linear regression maybe not fit. Through the figure, the result of predict can be changed when we add a sample. The original fit result is the blue line, but, the new result is the green line. The standard of classification can be changed as the sample we adding.

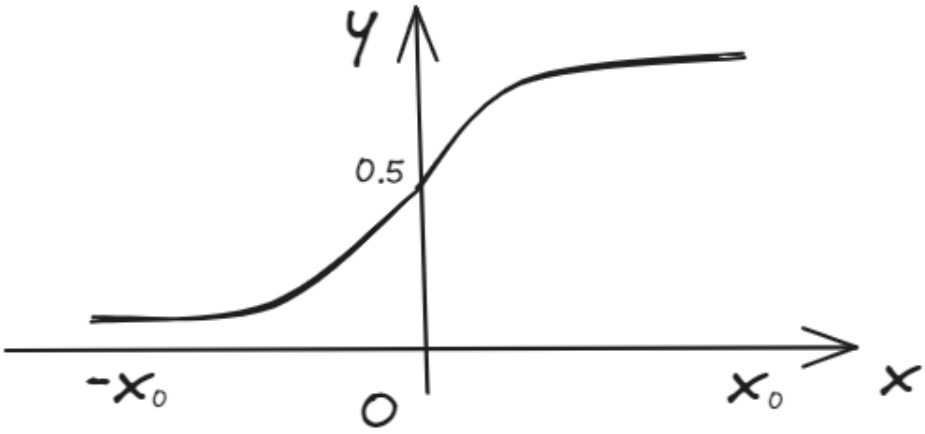
7.1 Logistic Regression

In this chapter, we will learn a useful algorithm model——`Logistic Regression` to solve the classification problem.

With linear regression method, the model is $f_{\vec{w},b}\left(x^i\right)=\vec{w}\cdot x^i+b$, to predict y given x. However, we would like the predictions of our classification model to be between 0 and 1 since our output variable y is either 0 or 1.

This can be accomplished by using a `"sigmoid function"` which maps all input values to values between 0 and 1.

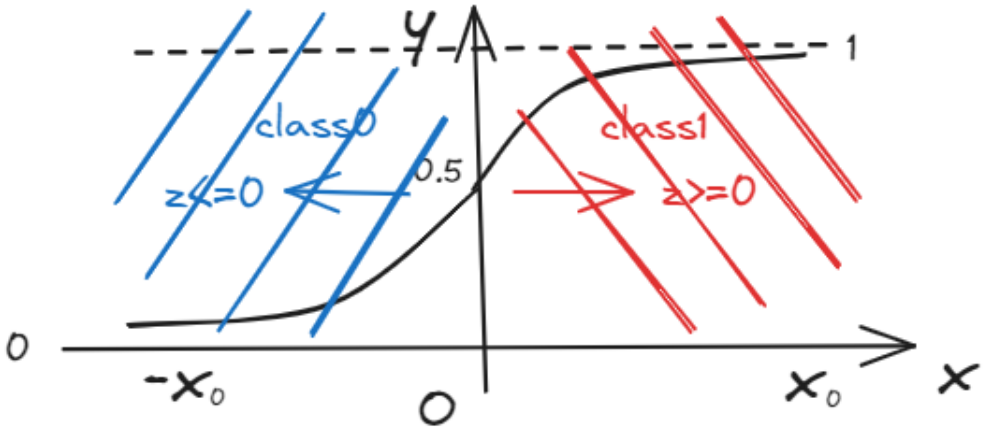
【sigmoid function】



$$
\begin{aligned}
g(Z)&=\frac{1}{1+e^{-Z}}, 0<g(Z)<1\\
f_{w,b}\left(X^{(i)}\right)&=g\left(w\cdot X^{(i)}+b\right)\\
f_{\vec{w},b}\left(\vec{x}\right)&=g\left(\vec{w}\cdot\vec{x}+b\right)=\frac{1}{1+e^{-(\vec{w}\cdot\vec{x}+b)}}
\end{aligned}
\tag{17}
$$

We can calculate the Z-score to classify the predict value by the probability.

$$
P(y=0)+P(y=1)=1 \tag{18}
$$



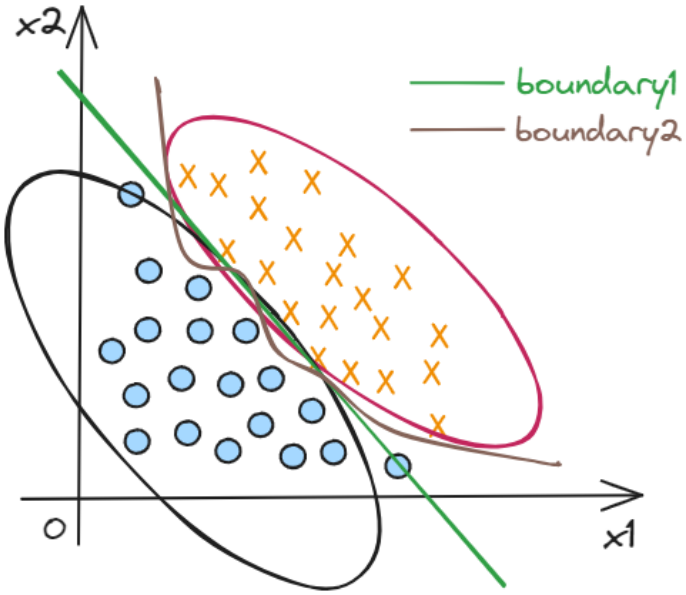
$$
\begin{aligned}
f_{\vec{w},b}\left(\vec{X}\right)&=P\left(y=1|\vec{X};\vec{w},b\right)\\
f_{\vec{w},b}\left(\vec{X}\right)&=P\left(y=0|\vec{X};\vec{w},b\right)
\end{aligned}
\tag{19}
$$

Probability that $y = 1$ or $y = 0$, given input $\vec{x}$, parameters w, b.

If $y = f_{\vec{w},b}\left(\vec{X}\right) > 0.5$ -> $g(Z) > 0.5$ -> $Z > 0$ -> $\vec{w} \cdot \vec{x} + b > 0$ == YES $Y = 1$.

Else, NO $y = 0$.

7.2 Decision boundary



$$g\left(Z\right) = g\left(w_1 x_1 + w_2 x_2 + b\right) \tag{20}$$

$$g\left(Z\right) = g\left(w_1 x_1 + w_2 x_2 + w_3 x_1 x_2 + w_4 x_4^2 + b\right)$$

$$boundary1 = w_1 x_1 + w_2 x_2 + b = 0 \tag{21}$$

$$boundary2 = w_1 x_1 + w_2 x_2 + w_3 x_1 x_2 + w_4 x_4^2 + b = 0$$

7.3 Cost function for regularized logistic regression

Cost function:

$$J_{(w,b)} = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{2} \left(w x^{(i)} + b - y^{(i)}\right)^2 = L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) \tag{22}$$

Our goal is to minimize the parameters to fit the model:

$$\min_{w.b} \; J\left(w, b\right) \tag{23}$$

Simplified loss function:

$$L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) = \begin{cases} -\log\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right), if \; y^{(i)} = 1 \\ -\log\left(1 - f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right), if \; y^{(i)} = 0 \end{cases}$$

$$\Longrightarrow \tag{24}$$

$$L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) = -y^{(i)} \log\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) - \left(1 - y^{(i)}\right) \log\left(1 - f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right)$$

For regularized **logistic** regression, the cost function is of the form

$$J(\mathbf{w},b) = $$
$$\frac{1}{m}\sum_{i=0}^{m-1}\left[-y^{(i)}\log\left(f_{\mathbf{w},b}\left(\mathbf{x}^{(i)}\right)\right) - \left(1-y^{(i)}\right)\log\left(1-f_{\mathbf{w},b}\left(\mathbf{x}^{(i)}\right)\right)\right] + \frac{\lambda}{2m}\sum_{j=0}^{n-1} w_j^2 \tag{25}$$

where:

$$f_{\mathbf{w},b}(\mathbf{x}^{(i)}) = sigmod(\mathbf{w}\cdot\mathbf{x}^{(i)} + b) \tag{26}$$

Compare this to the cost function without regularization (which you implemented in a previous lab):

$$J(\mathbf{w},b) = \frac{1}{m}\sum_{i=0}^{m-1}\left[(-y^{(i)}\log\left(f_{\mathbf{w},b}\left(\mathbf{x}^{(i)}\right)\right) - \left(1-y^{(i)}\right)\log\left(1-f_{\mathbf{w},b}\left(\mathbf{x}^{(i)}\right)\right)\right] \tag{27}$$

As was the case in linear regression above, the difference is the regularization term, which is $\frac{\lambda}{2m}\sum_{j=0}^{n-1} w_j^2$

Gradient descent:

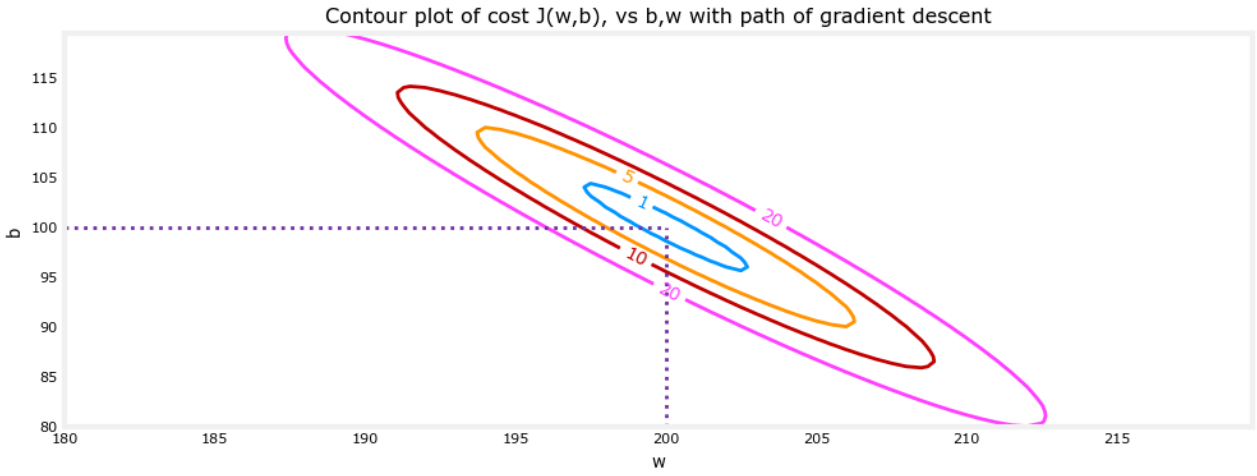
repeat{

$$tw_j = w_j - \alpha\frac{\partial}{\partial w_j}J\left(w_1, w_2, \cdots w_n, b\right) = w_j - \alpha\frac{\partial}{\partial w_j}J\left(\vec{w}, b\right) \tag{28}$$

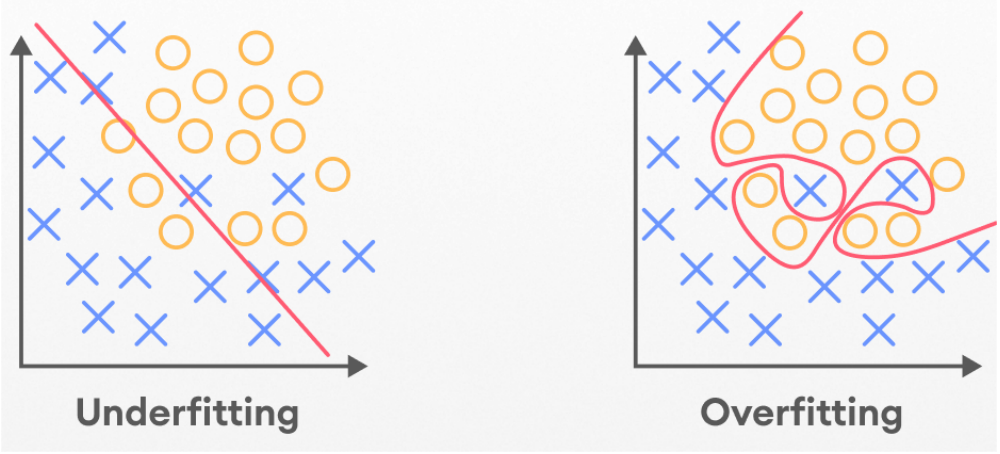
$$tb = b - \alpha\frac{\partial}{\partial b}J\left(w_1, w_2, \cdots w_n, b\right) = b - \alpha\frac{\partial}{\partial b}J\left(\vec{w}, b\right) \tag{29}$$

$$\begin{aligned} w &= tw_j \\ b &= tb \end{aligned} \tag{30}$$

}



To provide the overfitting problem, we apply the regularized method to add the penalty coefficient. That is why the we add the $\frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2$ at the end of the formula.



Regularized:

$$\begin{aligned} J_{(\vec{w},b)} &= \frac{1}{2m}\sum_{i=1}^{m}\left(\vec{w}\cdot\vec{x}^{(i)}+b-y^{(i)}\right)^2+\frac{\lambda}{2m}\sum_{j=1}^{n}w_j^2 \\ J_{(\vec{w},b)} &= \frac{1}{2m}\sum_{i=1}^{m}\left(\vec{w}\cdot\vec{x}^{(i)}+b-y^{(i)}\right)^2+\frac{\lambda}{2m}\sum_{j=1}^{n}w_j^2+\frac{\lambda}{2m}\sum_{j=1}^{n}b_j^2 \end{aligned} \tag{31}$$

The gradient calculation for both linear and logistic regression are nearly identical, differing only in computation of $f_{\mathbf{w}b}$.

$$\frac{\partial J(\mathbf{w},b)}{\partial w_j} = \frac{1}{m}\sum_{i=0}^{m-1}(f_{\mathbf{w},b}(\mathbf{x}^{(i)})-y^{(i)})x_j^{(i)}+\frac{\lambda}{m}w_j \tag{32}$$

$$\frac{\partial J(\mathbf{w},b)}{\partial b} = \frac{1}{m}\sum_{i=0}^{m-1}(f_{\mathbf{w},b}(\mathbf{x}^{(i)})-y^{(i)}) \tag{33}$$

- m is the number of training examples in the data set
- $f_{\mathbf{w},b}(x^{(i)})$ is the model's prediction, while $y^{(i)}$ is the target
- For a **linear** regression model
  $f_{\mathbf{w},b}(x) = \mathbf{w}\cdot\mathbf{x}+b$
- For a **logistic** regression model
  $z = \mathbf{w}\cdot\mathbf{x}+b \setminus f_{\mathbf{w},b}(x) = g(z)$
  where $g(z)$ is the sigmoid function:

$$g(z) = \frac{1}{1+e^{-z}} \tag{34}$$

```
#cpmpute cost function through itertion process
def gradient_descent(x,y,w_in,b_in,alpha,num_iters,cost_function,gradient_function):
    w = copy.deepcopy(w_in)
    J_history = []
    p_history = []
    b = b_in
    w = w_in
    for i in range(num_iters):
        # Calculate the gradient and update the parameters using gradient_function
        dj_dw,dj_db = gradient_function(x,y,w,b)
        # update the parameters
        b = b - alpha * dj_db
```

```
        w = w - alpha * dj_dw
        # Save cost J at each iteration
        if i < 100000:
            J_history.append(cost_function(x,y,w,b))
            p_history.append([w,b])
        # Print cost every at intervals 10 times or as many iterations if < 10
        if i % math.ceil(num_iters/10) == 0:
            print(f"Iteraton {i:4}: Cost {J_history[-1]:0.2e}",f"dj_dw: {dj_dw:
0.3e}, dj_db: {dj_db: 0.3e}",f"w: {w: 0.3e}, b:{b: 0.5e}")
    return w,b,J_history,p_history
```

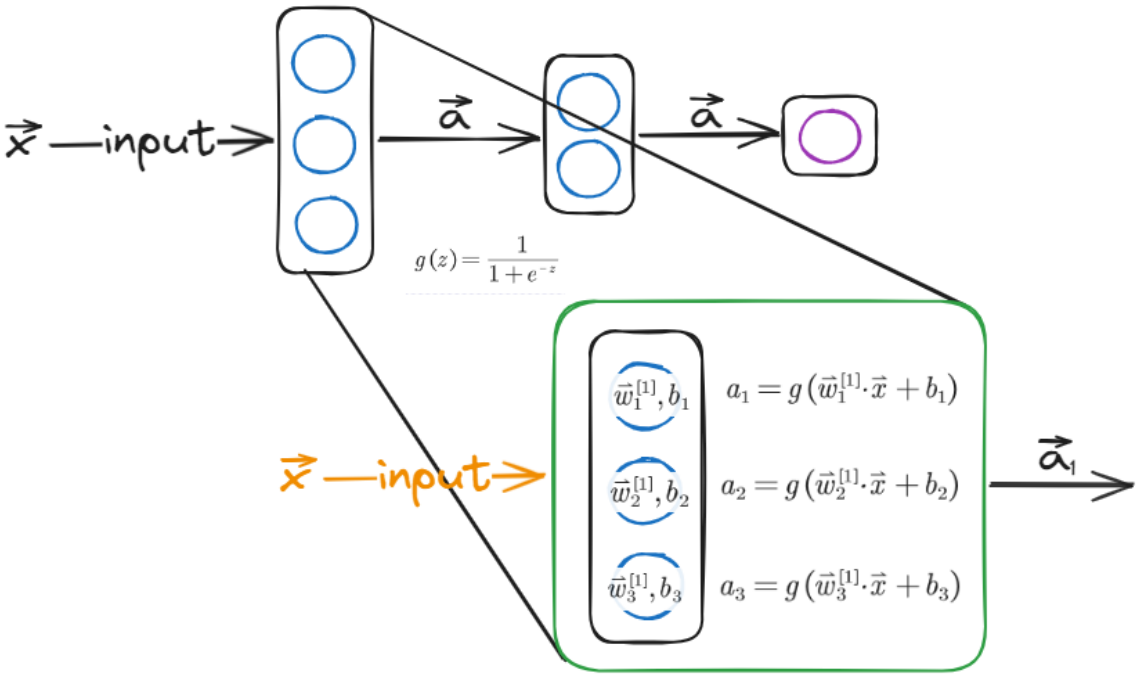
8 Neural Network🏹

8.1 How to install TensorFlow

- conda install tensorflow
- pip install tensorflow

8.2 How the brain works

The following picture shows the basic structure about neutral network. Like human`s neutral cell, it passes information by layers, which every cell includes a logistic function.



8.3 Data format in Tensorflow

`matrix and tensor`

Numpy, a standard library created in 1970s, is used to calculate linear algebra in python(data analysis). Tensorflow is a machine learning framework created by Google.

In Tensorflow, the input format must **a matrix**. We should focus on the special characteristic in our work.

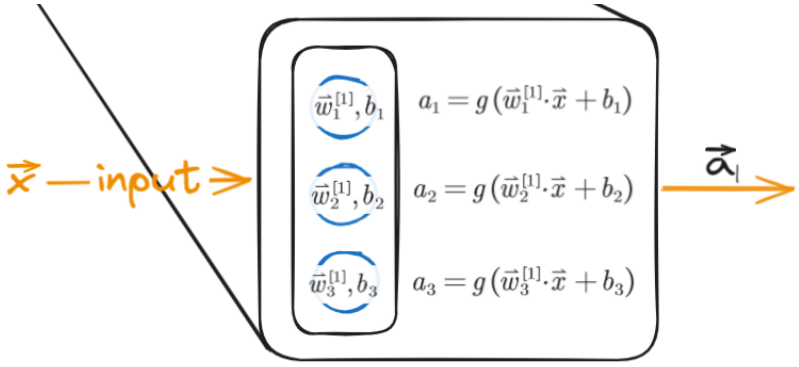
```
a = np.array([1,2,3])#一维数组
b = np.array([[1,2,3]])#一维矩阵
x = np.array([
    [1,2,3],
    [4,5,6],
    [11,12,14],
])#3X3矩阵
Z = np.matmul(A_in,W) + B #input matrix to simplify for loop
```

8.4 Build a Tensorflow

1. build the structure of the model
2. compile the model
3. input training data
4. fit the model
5. predict the model

```
1.
layer_1 = Dense(units=25,activation='sigmoid')
layer_2 = Dense(units=15,activation='sigmoid')
layer_3 = Dense(units=1,activation='sigmoid')
model = Sequential([layer_1,layer_2,...layer_n])
----------------------------
model = Sequential([
    Dense(units=25,activation='relu'),
    Dense(units=15,activation='relu'),
    Dense(units=1,activation='sigmoid')
])
2.
model.compile()
3.
x = np.array([[0...,245,...,17],[0...,200,...,284]])
y = np.array([1,0])
4.
model.fit
5.
model.predict(x_new)
```

8.5 Implementation of the preceding communication



$$\vec{w}_1^{[1]} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \vec{w}_2^{[1]} = \begin{bmatrix} -3 \\ 4 \end{bmatrix}, \vec{w}_3^{[1]} = \begin{bmatrix} 5 \\ -6 \end{bmatrix} \tag{35}$$

```
W = np.array([
    [1,-3,5],
    [2,4,-6]
])
b = np.array([-1,1,2])
a_in = np.array([-2,4])
```

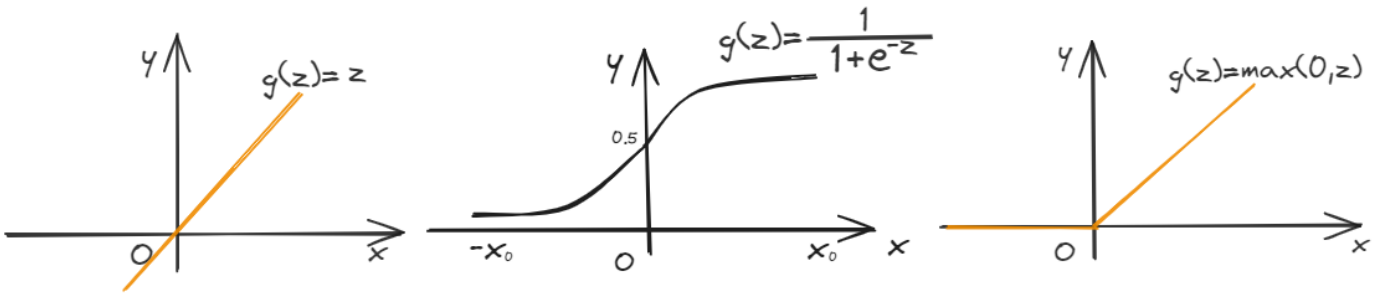
Implement the coding of dense function with python:

```
def dense(a_in,W,b,g):
    #units equals to the numbers of cols of W
    units = W.shape[1]
    #Create a matrix with the same size of units--[0,0,0]
    a_out = np.zeros(units)
    #compete the g(z)
    for j in range(units):
        w = W[:,j]
        z = np.dot(w,a_in) + b[j]
        a_out[j] = g(z)
    return a_out

def sequential(x):
    a1 = dense(x,W1,b1)
    a2 = dense(a1,W2,b2)
    a3 = desne(a2,W3,b3)
    a4 = dense(a3,W4,b4)
    f_x = a4
    return f_x
```

8.6 Choose activation function

Three types activation function:




```
activation="linear"
activation="sigmoid"
activation="relu"
```

type y	0/1	+/-	0/+
sigmoid	◆ binary classfication		
linear		◆ regression	
relu			◆ regression

[why we need activation function?]

If we use the linear activation function in hidden layers all the time, the predict results of neutral network equal to linear regression. that`s why we should use sigmoid or relu function as the activation function to build the network.

8.7 Soft-max regression

Softmax function has been used to solve the multi-class problem. Essentially, it is still a **classification problem** `based on probability`. The expression of Softmax function as follow:

$$
\begin{aligned}
z_j &= \vec{w}_j \cdot \vec{x} + b_j \\
a_j &= \frac{e^{z_j}}{\sum_{k=1}^{n} e^{z_k}} = P\left(y = j|\vec{x}\right)
\end{aligned} \tag{36}
$$

a_j is the possibility of predict result.

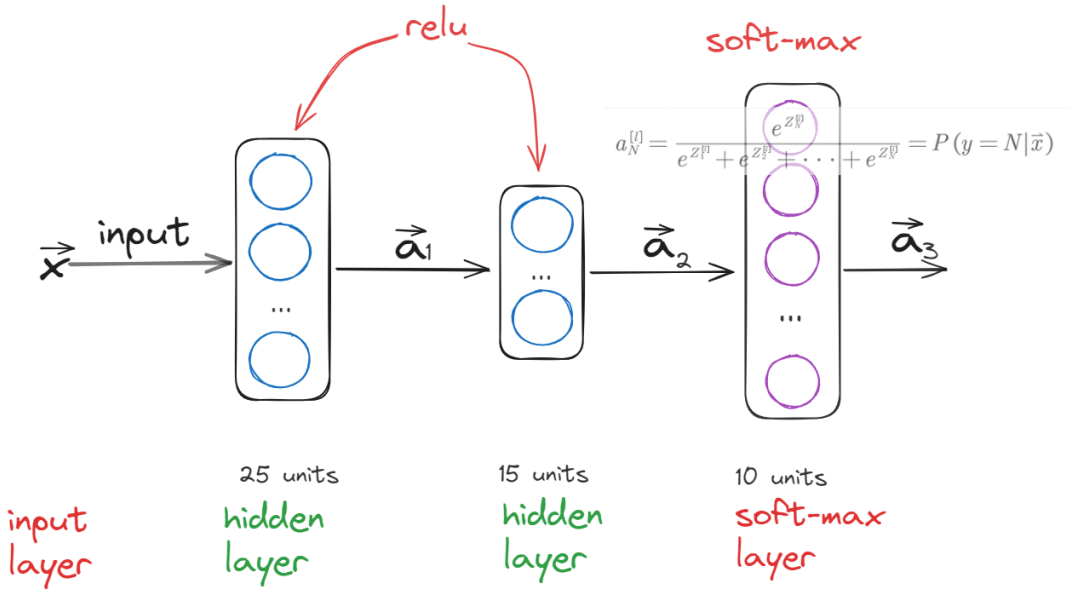
The cost function of Soft-max function is:

$$
a_N = \frac{e^{Z_N}}{e^{Z_1} + e^{Z_2} + \cdots + e^{Z_N}} = P\left(y = N|\vec{x}\right) \tag{37}
$$

$$
loss\left(a_1, \ldots, a_N, y\right) = \begin{cases} -\log a_1 \ if \ y = 1 \\ -\log a_2 \ if \ y = 2 \\ \vdots \\ -\log a_n \ if \ y = n \end{cases} \tag{38}
$$

The result is `one of` the loss functions.

8.8 Output of the soft-max



The outcome of soft-max classification is multiply. Every outcome will be competed a score to predict the right answer.

Carefully, the loss function we must choose the `SparseCategoricalCrossentropy`.

$$
\begin{aligned}
a_N^{[l]} &= \frac{e^{Z_N^{[l]}}}{e^{Z_1^{[l]}} + e^{Z_2^{[l]}} + \cdots + e^{Z_N^{[l]}}} \\
-\log a_N^{[l]} &\neq -\log \frac{e^{Z_N^{[l]}}}{e^{Z_1^{[l]}} + e^{Z_2^{[l]}} + \cdots + e^{Z_N^{[l]}}}
\end{aligned}
\tag{39}
$$

The difference of competing order can lead to the different outcome, which has different model accurancy.

8.9 Improve the soft-max model

In order to improve the accuracy of calculations, we have made the following improvements to the model algorithm:

- In soft-max layer, we adopt the `linear` function as the activation.
- In the process of compiling model, we add a parameter to improve the accurancy.

```
#linear function as the activation function in the soft-max layer
model = sequential([
    Dense(units = 25, activation="relu")
    Dense(units = 15, activation="relu")
    Dense(units = 10, activation="linear")
])
#add parameter:from_logits=True
model.compile(loss=SparseCategoricalCrossentropy(from_logits=True))
model.fit(X,Y,epochs=100)
logit = model(X)
f_x = tf.nn.sigmoid(logit)
```

8.10 MNIST Adam

9 Evaluating a model

9.1 Data set

Model fits the training data well(over-fit) but will fail to generalize to new examples not in the training set.

Hence, we need to **partition the dataset** to test the accurancy of the model.

The data set was divided into `test set` and `train set`.

【regression】

Fit parameters by minimizing cost function $J(\vec{w},b)$:

$$J\left(\overrightarrow{w},b\right)=\min_{\overrightarrow{w},b}\left[\frac{1}{2m_{train}}\sum_{i=1}^{m_{train}}\left(f_{\overrightarrow{w},b}\left(\vec{x}^{(i)}\right)-y^{(i)}\right)^2+\frac{\lambda}{2m_{train}}\sum_{j=1}^{n}\omega_j^2\right] \tag{40}$$

compute test error:

$$J_{test}\left(\overrightarrow{w},b\right)=\frac{1}{2m_{test}}\left[\sum_{i=1}^{m_{test}}\left(f_{\overrightarrow{w},b}\left(\vec{x}_{test}^{(i)}\right)-y_{test}^{(i)}\right)^2\right] \tag{41}$$

compute train error:

$$J_{train}\left(\overrightarrow{w},b\right)=\frac{1}{2m_{train}}\left[\sum_{i=1}^{m_{train}}\left(f_{\overrightarrow{w},b}\left(\vec{x}_{train}^{(i)}\right)-y_{train}^{(i)}\right)^2\right] \tag{42}$$

【train】

Fit parameters by minimizing cost function $J(\vec{w},b)$:

$$\begin{aligned}J\left(\overrightarrow{w},b\right)=-\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(f_{\overrightarrow{w},b}\left(\vec{x}^{(i)}\right)\right)+\left(1-y^{(i)}\right)\log\left(1-f_{\overrightarrow{w},b}\left(\vec{x}^{(i)}\right)\right)\right]\\+\frac{\lambda}{2m}\sum_{j=1}^{n}\omega_j^2\end{aligned} \tag{43}$$

compute test error:

$$J_{test}=-\frac{1}{m_{test}}\sum_{i=1}^{m_{test}}\left[y_{test}^{(i)}\log\left(f_{\overrightarrow{w},b}\left(\vec{x}_{test}^{(i)}\right)\right)+\left(1-y_{test}^{(i)}\right)\log\left(1-f_{\overrightarrow{w},b}\left(\vec{x}_{test}^{(i)}\right)\right)\right] \tag{44}$$

compute train error:

$$-\frac{1}{m_{train}}\sum_{i=1}^{m_{train}}\left[y_{train}^{(i)}\log\left(f_{\overrightarrow{w},b}\left(\vec{x}_{train}^{(i)}\right)\right)+\left(1-y_{train}^{(i)}\right)\log\left(1-f_{\overrightarrow{w},b}\left(\vec{x}_{train}^{(i)}\right)\right)\right] \tag{45}$$

9.2 Model selection and cross validation

If we want to fit a function to predict a problem or classification, we often use test error J_{test} to judge the accurancy of the model. But, the J test is likely to be an `optimistic estimate` of generalization error. Because, when we choose the degree of parameter d in polynomial fit.,This fit of J test may lower than the actual estimate. The optimistic estimate can lead to a low score of J_test.

So, we need to partition the dataset as three parts to avoid optimistic estimate:

- training set
- cross validation set
- test set

compute cross validation error:

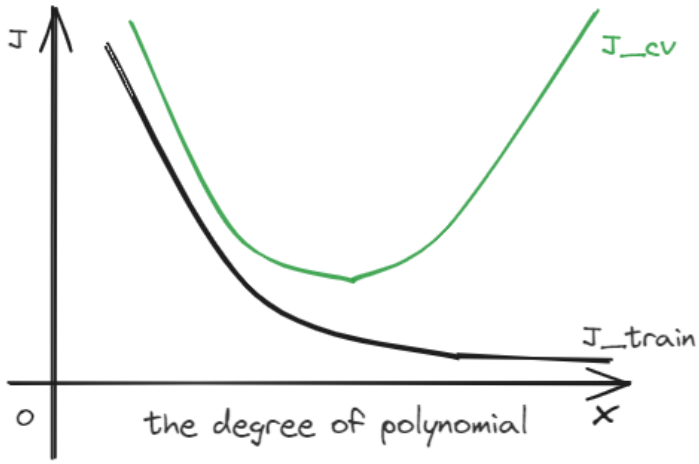
$$J_{cv}\left(\vec{w},b\right)=\frac{1}{2m_{cv}}\left[\sum_{i=1}^{m_{cv}}\left(f_{\vec{w},b}\left(\vec{x}_{cv}^{(i)}\right)-y_{cv}^{(i)}\right)^2\right] \tag{46}$$

9.3 Diagnosing bias and variance

Have idea-Train model-Diagnose bias and variance

J_{train} reflects bias, and J_{cv} reflects variance. A perfect model has low J_{train} and low J_{cv}.

As the increasing of degree of d, J_{train} will typically go down. Meanwhile, J_{cv} will also go down and then it will increase.J_{cv} will have a min value for different degree of d.



How do you tell if our algorithm has a bias or variance problem?

- High bias(under fit): J_{train} will be high($J_{train}\approx J_{cv}$)
- High variance(over fit): J_{cv}>>J_{train}(J_{train} may be low)
- High bias and High variance J_{train} will be high and J_{cv}>>J_{train}

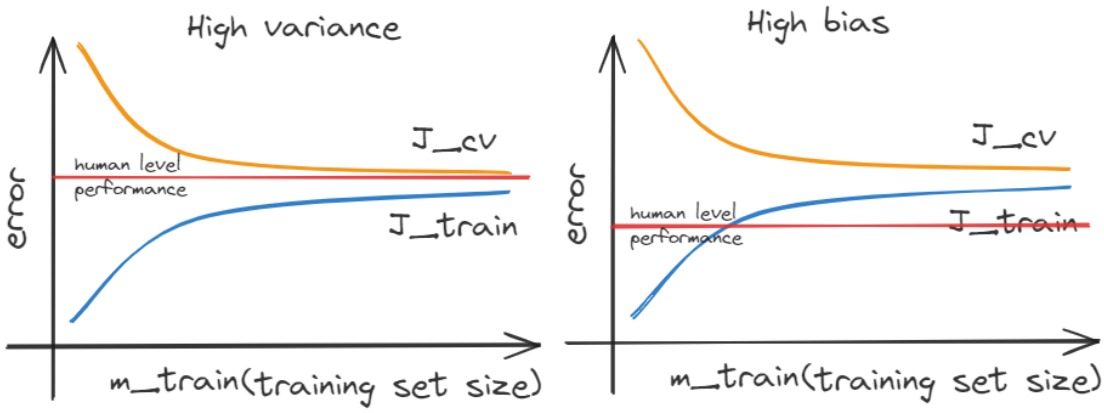
9.4 Regularization and bias/variance

A large neutral network will usually do as well or better than a smaller one so long as regularization is chosen appropriately.

```
layer_1 = Dense(units=25,activation="relu",kernal_regularizer=L2(0.01))
layer_2 = Dense(units=15,activation="relu",kernal_regularizer=L2(0.01))
layer_1 = Dense(units=1,activation="sigmoid",kernal_regularizer=L2(0.01))
model = Sequential([layer_1,layer_2,layer_3])
```

9.5 Learning curves

When we increase the size of training set, the train error will increase. But, the error of cross validation will decrease. As the increasing of sample points. a regression function(a line or a curve) cannot fit all the point.



If a algorithm suffers from high variance, getting more training data is **likely** to help.

If a algorithm suffers from high bias, getting more training data **will not** help much.

【Debugging algorithm】

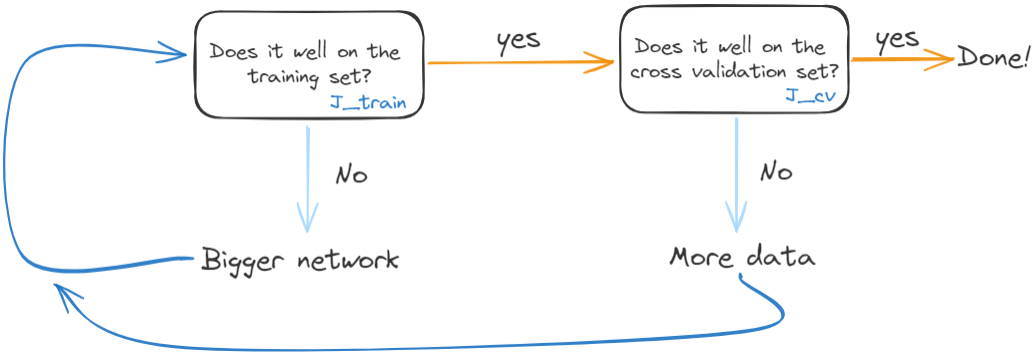
we have implemented the regularized linear regression:

$$J\left(\vec{w}, b\right) = \min_{\vec{w},b}\left[\frac{1}{2m}\sum_{i=1}^{m}\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{2m}\sum_{j=1}^{n}\omega_j^2\right] \tag{47}$$

operation	what should do
get more training examples	fixes high variance
try smaller sets of features	fixes high variance
try getting additional features	fixes high bias
try adding polynomial features	fixes high bias
try decreasing λ	fixes high bias
try increasing λ	fixes high variance

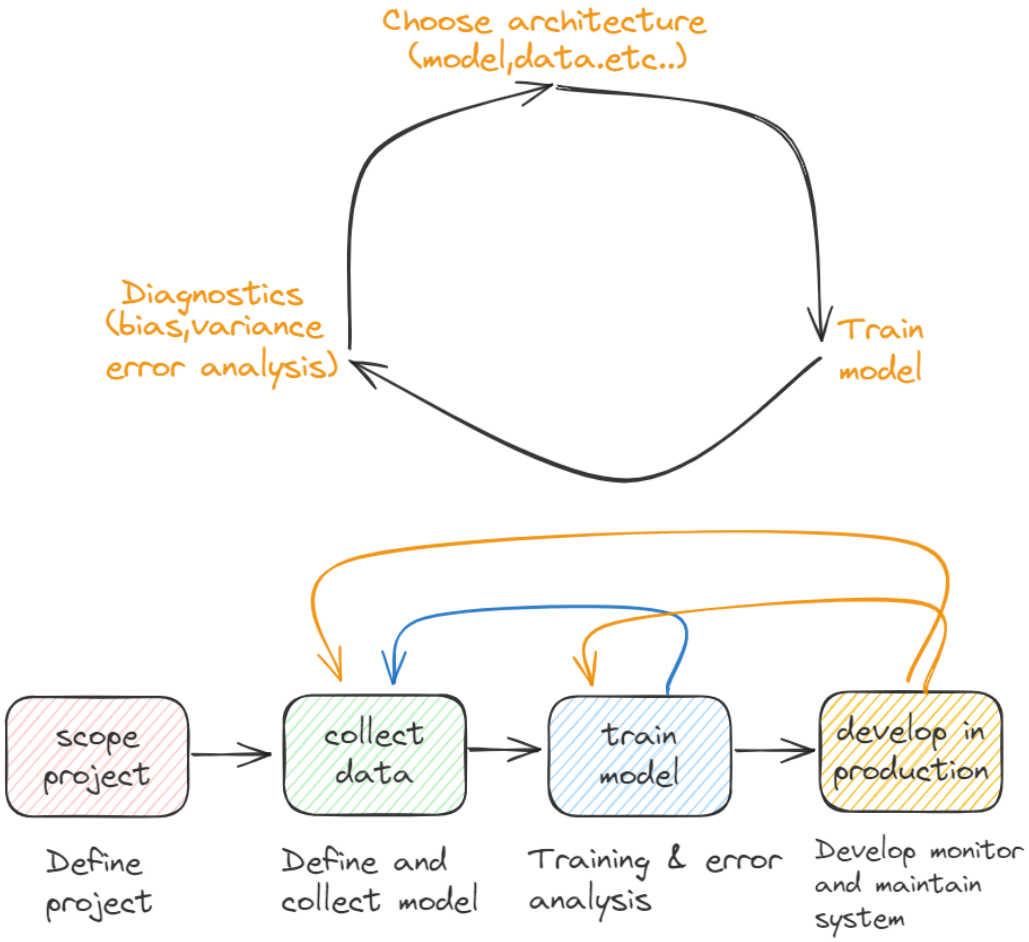
`Trade-off`

Simple model($f_{\vec{w},b}\left(x\right) = w_1x + b$) will get high bias VS complex model($f_{\vec{w},b}\left(x\right) = w_1x + w_2x^2 + w_3x^3 + w_4x^4 + b$) will get high variance.

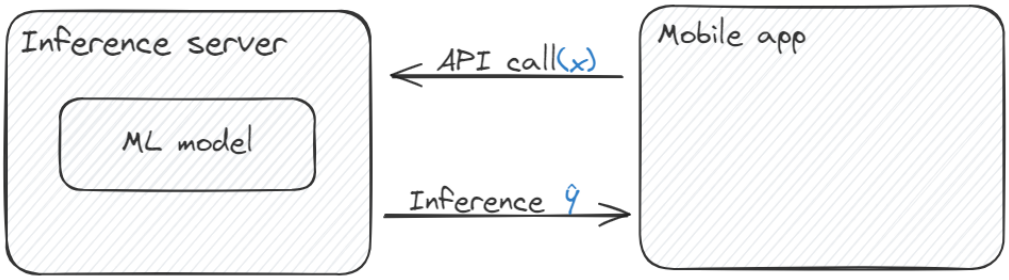


9.6 Cycle of machine learning

The cycle of ML process:



How to apply the ML model to solve the actual problem in software engineering design?



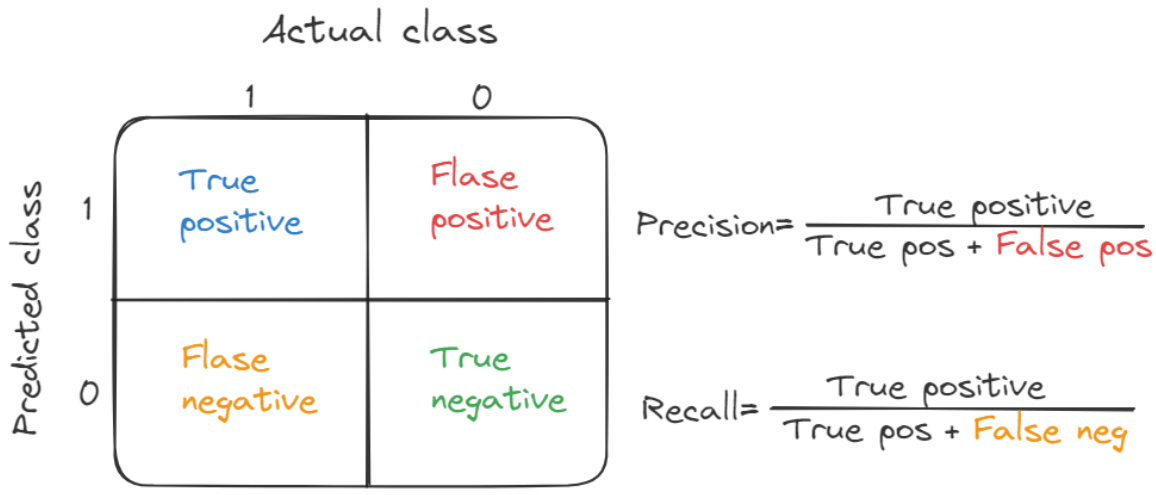
ML model is collected in the inference server. we use mobile app through API call to achieve these function.

9.7 Precision and Recall

Precision (also called positive predictive value) is the fraction of relevant instances among the retrieved instances.

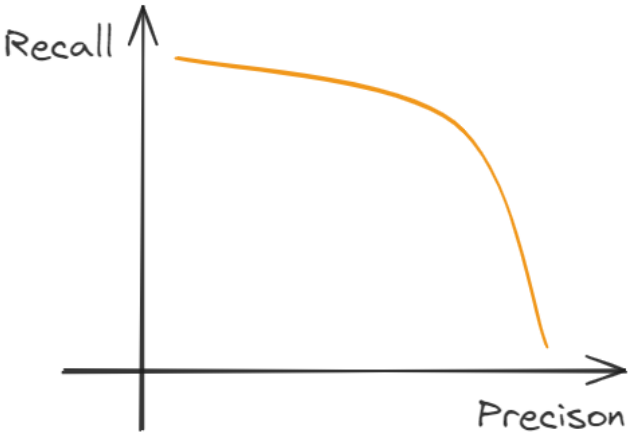
Recall (also known as sensitivity) is the fraction of relevant instances that were retrieved.

We design a `confusion matrix` to show it:



In the trade-off between Precision(P) and Recall(R), we use F1 score to evaluate the efficiency about the model.

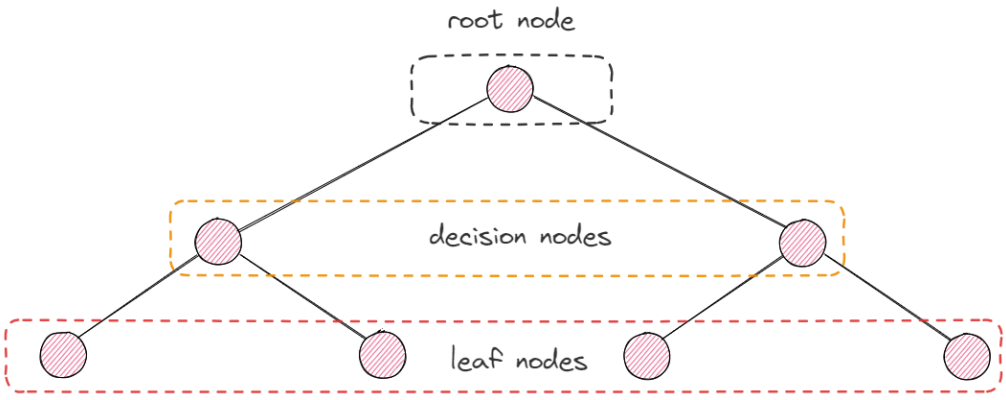
the trade-off between Precision(P) and Recall(R) has shown in the figure:



$$F1 = \frac{1}{\frac{1}{2}\left(\frac{1}{P} + \frac{1}{R}\right)} = \frac{2PR}{P+R} \tag{48}$$

10 Decision tree

The structure of a decision tree:



10.1 Methods chosen

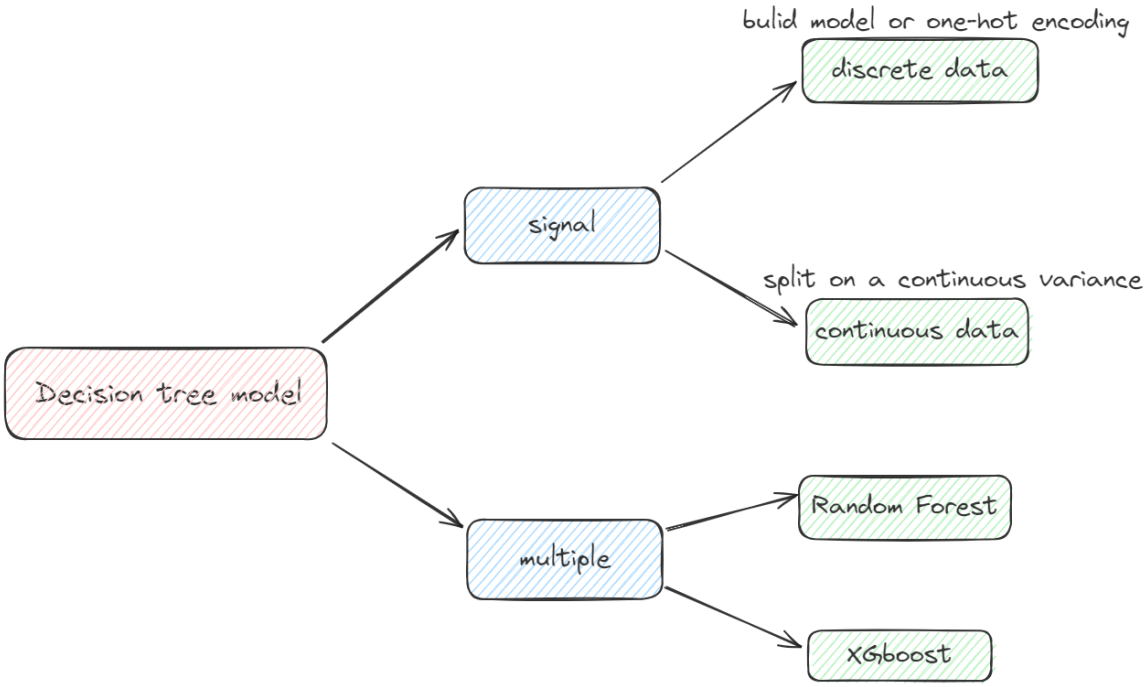
For Signal Decision tree, we should focus on the problem is that the data features.

If the data is discrete(just like 0 or 1), we can build the Signal Decision tree model. But,a row data may includes more than two classes, in this situation we should use `one-hot encoding`.

> one-hot encoding only fit for the decision tree model.

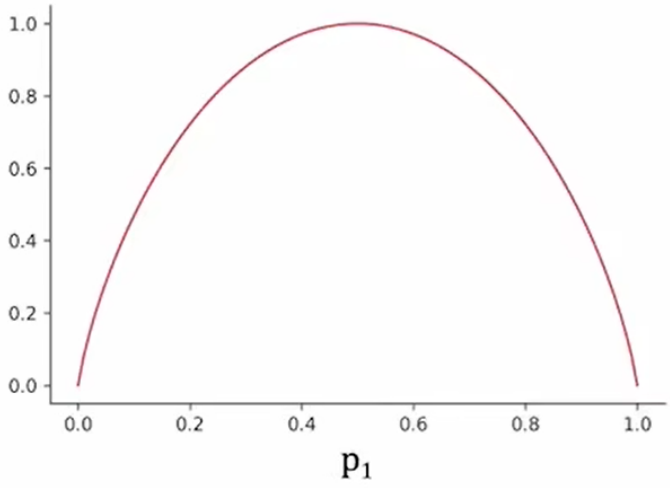
If the data has continuous data(not only just like 0 or 1), we should split on a continuous variance.

For Multiple trees, we can use **Random Forest** and **XGboost** algorithm to solve.

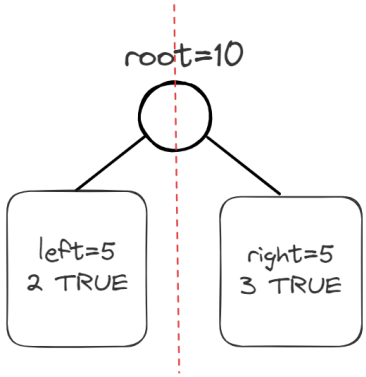


10.2 Purity(entropy)

p_1 = fraction of examples that are True.



$$
\begin{aligned}
H\left(p_1\right) &= -p_1 \log\left(p_1\right) - p_0 \log\left(p_0\right) \\
&= -p_1 \log\left(p_1\right) - \left(1-p_1\right) \log\left(1-p_1\right)
\end{aligned} \tag{49}
$$



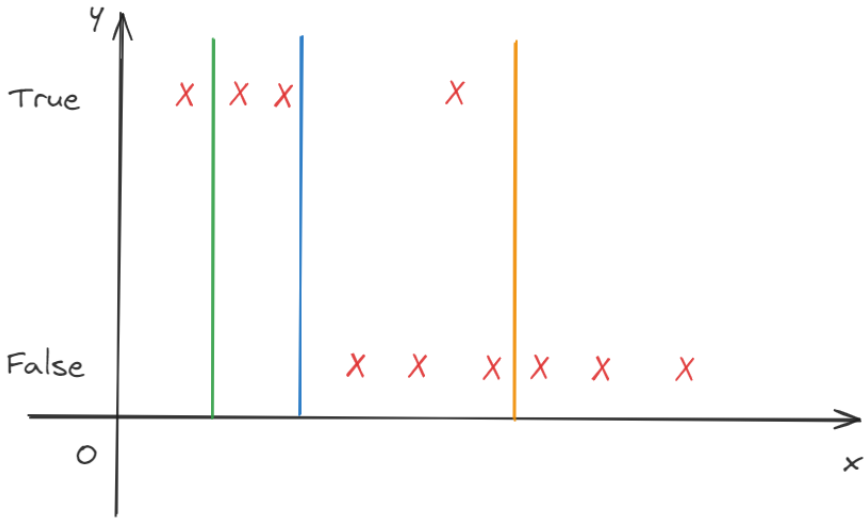
In this figure, w^{left}=2/5、 w^{right}=3/5、p_1^{left}=5/10、p_2^{left}=5/10.

Information Purity

$$
InformationPurity = H\left(p_1^{root}\right) - \left(w^{left} H\left(p_1^{left}\right) + w^{right} H\left(p_1^{right}\right)\right) \tag{50}
$$

❓ We should choose the `max` value of the Information Purity to **recursive** the decision tree model, which is called `Information Gain`.

In the process of split on a continuous variance(**Regression tree**), we also choose the max decreasing variance result as a good fit model.



The purity of regression tree(equal to information gain):

$$
D = V^{root} - \left(w^{left} V^{left} + w^{right} V^{right}\right) \tag{51}
$$

V instead of `variance`.

10.3 Decision tree learning

- Start with all examples at the root node
- Calculate information gain for all features, and pick the one with the highest information gain
- Split dataset according to selected features, and create left and right branches of the tree

- Keep repeating splitting process until stopping criteria is met:

    when a node is 100% one class

    when splitting a node will result in the tree exceeding a maximum depth

    Information gain from additional splits is less than threshold

    when number of examples in a node is below a threshold

11 Decision tree VS Neutral network

11.1 Decision tree

- Works well on tabular(structured) data
- Not recommended for unstructured data(images,audios,text)
- Small decision tree may be human interpretable

11.2 Neutral network

- Works well on all types of data,including tabular(structured) data and unstructured data(images,audios,text)
- May be slower than decision tree
- Works with transfer learning
- When building a system of multiple models working together, it might be easier to string together multiple neutral network

12 K-MEANS