01-Machine-Learning

March 1, 2021

0.1 Overview

Machine Learning refers to a vast set of tools for *understanding data*. What is the main task of this tools? **Making inference and prediction**. We make an inference from the data, and we predict the unknown using this inference. For example; making inference from home features and after that, predicting new house price from a given new home feature. We must use the some fileds of mathematics like: - Linear Algebra - Probability and Statistics - Analysis/Calculus

Making inference and prediction refers to term Machine Learning. Machine Learning is a subfield of Artifical Intelligence and Computer Science. Image Recognition, Speech Analysis, Language Translation, Classification, Regression, Autoregressive Prediction and so on... All of them is a practical and theorical results of Machine Learning and Artifical Intelligence.

For being more formally, making inference is estimating a function $f : \mathbb{R} \rightarrow \mathbb{R}$. This is a function that will gives us new house prices using house features as a parameter. For example lets say x_1 is the area of the house and x_2 is the year of the house. We can write our function f as $f(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$. For example given a optimal parameters $\theta = [\theta_1 = 25,\ \ \theta_2 = 120,\ \ \theta_3 = -20]$ our function becomes $f(x) = 25 + 120x_1 - 20x_2$. After locating this optimal parameters, we received an house features; area of 150 m^2, year of 15. Then f outputs $25 + 120 \times 150 - 20 \times 15 = 17725$. That means our new home's price is 17725 dollar. But how to choose those parameters θ? From now on, we will see how to estimate function f.

0.2 The Data

Data is important for estimating f. Without a data, we cannot estimate f. This means we learn the data, and after we predict. We represent our samples as a feature matrix X, and for related rows we represent our labels as y,

$$X = \begin{bmatrix} x_1^1 & x_2^1 & \cdots & x_n^1 \\ x_1^2 & x_2^2 & \cdots & x_n^2 \\ \vdots & \vdots & \cdots & \vdots \\ x_1^m & x_2^m & \cdots & x_n^m \end{bmatrix} \quad Y = \begin{bmatrix} y^1 \\ y^2 \\ \vdots \\ y^m \end{bmatrix}$$

Superscripts are representation of a sample, subscripts are representation of a feature. For example x_9^7 refers to 9th feature (may be location of our home) of our 7th home and y^7 refers to price of our 7th home. This implies that for a data of n features and m samples, our function became $y = f(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$ where

$$x_k = \begin{bmatrix} x_k^1 \\ x_k^2 \\ \vdots \\ x_k^m \end{bmatrix} \quad k \in 0, 1, ..., n$$

As you can clearly see, this corresponds to matrix multiplication of our parameter vector and matrix X:

$$y = X\theta$$

$$\begin{bmatrix} y^1 \\ y^2 \\ \vdots \\ y^m \end{bmatrix} = \begin{bmatrix} 1 & x_1^1 & x_2^1 & \cdots & x_n^1 \\ 1 & x_1^2 & x_2^2 & \cdots & x_n^2 \\ \vdots & \vdots & \cdots & \vdots & \\ 1 & x_1^m & x_2^m & \cdots & x_n^m \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$$

0.3 Training With Data

The term *training* means learning from data. We train our *models* with the data. There are 2 forms of learning:

- Supervised Learning: for each of the observation X, the correct class label y is available.
- Unsupervised Learning: only the observation X is available.

We can list our Machine Learning Algorithms in this two class

- Supervised Learning Algorithms: Linear Regression, Logistic Regression, KNN, Decision Trees, Support Vector Machines etc..
- Unsupervised Learning: K-Means Clustering, Hierarchical Clustering, Principal Component Analysis etc.

0.4 Types of Data

Let's see what this data looks like!

```
[2]: import pandas as pd
     data = pd.read_csv('./data/house-prices.csv')
     data['Output: Price'] = data['Price']
     del data['Price']
     data.head()
```

```
[2]:    Home  SqFt  Bedrooms  Bathrooms  Offers Brick Neighborhood  Output: Price
     0     1  1790         2          2       2    No         East         114300
     1     2  2030         4          2       3    No         East         114200
     2     3  1740         3          2       1    No         East         114800
     3     4  1980         3          2       3    No         East          94700
     4     5  2130         3          3       3    No         East         119800
```

You can easily see that the target, in other words the price, is numerical value. The task on this data is obviously predicting house prices depend on other attributes of houses like bathrooms, bedrooms as shown above.

Let's see another type of data.

```
[3]: import pandas as pd
     data_2 = pd.read_csv('./data/diabetes.csv')
     data_2 = data_2.rename(columns={'Outcome': 'Is Diabetes?'})
     data_2.head()
```

```
[3]:    Pregnancies  Glucose  …  Age  Is Diabetes?
     0            2      138  …   47             1
     1            0       84  …   23             0
     2            0      145  …   31             1
     3            0      135  …   24             1
     4            1      139  …   21             0

     [5 rows x 9 columns]
```

You can easily see that the target, in other words the outcome (whether diabetes or not) , is categorical value. If diabetes outcome is 1, else 0. The task on this data obviously predicting if diabetes or not depend on other attributes like blood pressure, insulin as shown above.

0.5 Regression Models and Linear Regression

Linear regression is a very simple approach for supervised learning. It serves as a good jumping-off point for newer approaches: as we will see in later lectures, many fancy machine learning and deep learning can be seen as generalizations or extensions of linear regression. Before evaluate the linear regression problem, let's see a simple data that we will use.

```
[24]: import pandas as pd
      df_3 = pd.read_csv('./data/Advertising_simple.csv')
      df_3.drop(columns = ['Unnamed: 0'], inplace = True)
      df_3.head()
```

```
[24]:       TV  sales
      0  230.1   22.1
      1   44.5   10.4
      2   17.2    9.3
      3  151.5   18.5
      4  180.8   12.9
```

Simple linear regression lives up its name: it is a very straightforward approach for predicting a quantitative response on Y on the basis of a single predictor variable X. It assumes that there is a approximately a linear relationship between X and Y. Mathematically, we can write this linear relationship as,

$$Y \approx \theta_0 + \theta_1 X$$

For our data, X may represent TV advertising and Y may represent sales, then the equation becomes more clear,

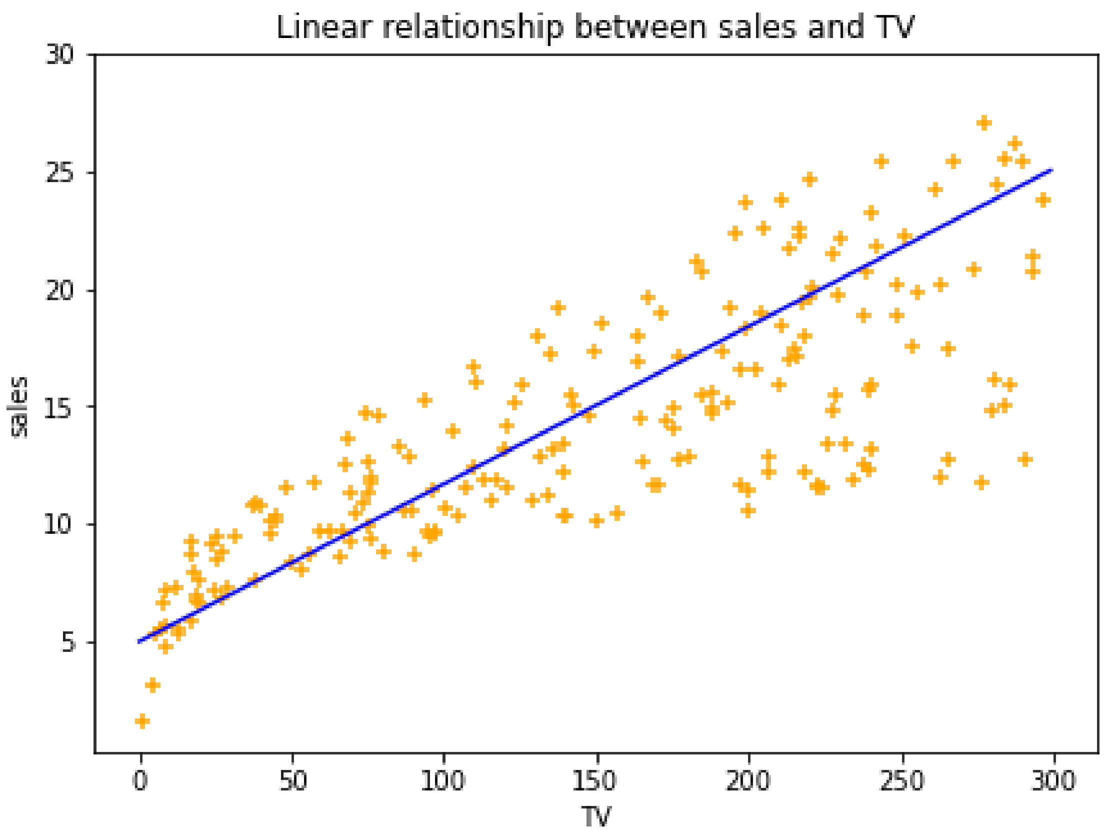
$$\text{sales} \approx \theta_0 + \theta_1 \times \text{TV}$$

Together $[\theta_0, \theta_1]$ known as parameter vector, as we discussed firstly. Our X matrix is made from only TV data. So we can represent X as $X = x_1$. Once we estimate the parameters, we can predict future sales on the basis of a particular value of tv advertising by computing

$$\hat{y} = \theta_0 + \theta_1 x_1$$

Lets see this linear relationship:

[60]:
```python
import matplotlib.pyplot as plt
import numpy as np
x_axis = np.array([point for point in range(0,300)])
x = np.array([point for point in range(0,300)])
y = 0.067*x +5
fig = plt.figure(figsize = (7, 5))
plt.scatter(df_3['TV'], df_3['sales'],color='orange',marker='+')
plt.xlabel('TV')
plt.ylabel('sales')
plt.title('Linear relationship between sales and TV')
plt.ylim(top = 30)
plt.plot(x,y,color='blue')
plt.show()
```



So, we see how a line fit into linear data. But in technical way, we need to estimate the parameters θ_0, θ_1 of this line. Let's estime this parameters with maximizing the likelihood.

Suppose that our data is randomly choosen from a Gaussian Distribution. The probability density funtion of our dataset becomes

$$p(y|x) \sim \mathcal{N}(g(x|\theta_x), \sigma^2), \quad \text{since } \ g(x|\theta_x) \ \text{ is our estimator which is } \ \theta_0 + \theta_1 x_1$$

For the dataset of ours (y^i, x^i). We can define our likelihood distribution as $p(x, y) = p(y|x)\ p(x)$ From this point, let's find our loss function and estimate parameters.

$$log\mathcal{L}(\theta|x) = \log\prod_i p(x^i, y^i) = \log\prod_i p(y^i|x^i) + \log\prod_i p(x^i)$$

$$= \log\prod_i \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{[y^i - g(x^i|\theta)]^2}{2\sigma^2}\right)$$

$$= \log\left(\frac{1}{\sqrt{2\pi}}\right)^m \left[-\frac{1}{\sqrt{2\pi}\sigma}\sum_i [y^i - g(x^i|\theta)]^2\right]$$

$$= -m\log\sqrt{2\pi}\sigma - \frac{1}{2\sigma^2}\sum_i [y^i - g(x^t|\theta)]^2$$

You can easily see that the first term independent from our parameters. So, what does the equation

$$\frac{1}{2\sigma^2}\sum_i [y^i - g(x^t|\theta)]^2$$

tell us?

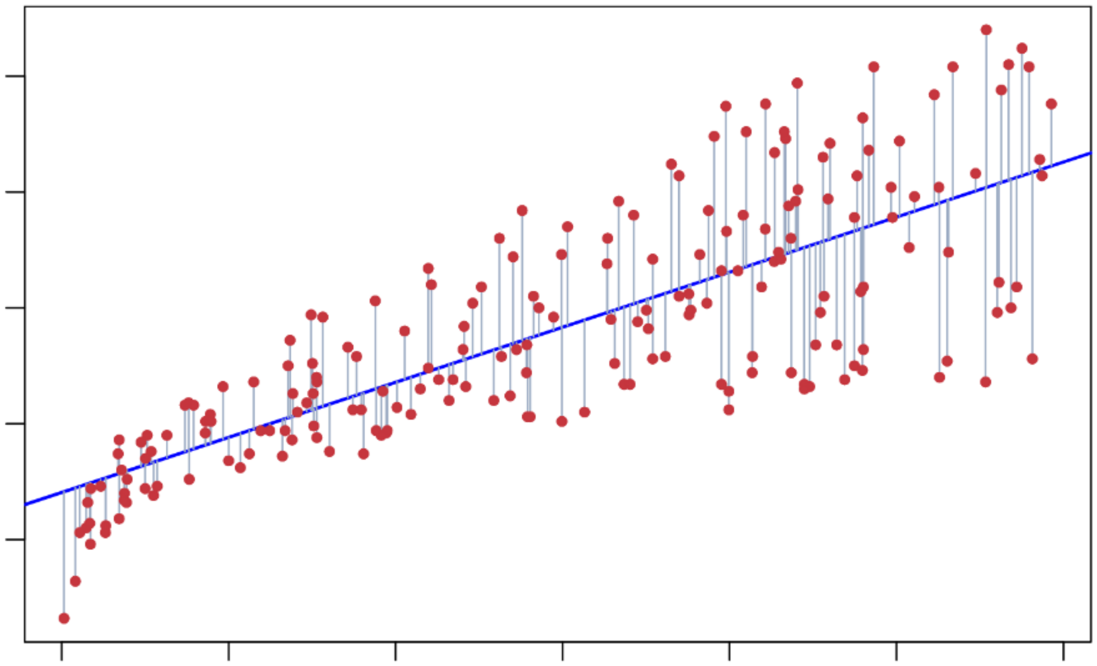
We can modify this equation, the constant factor is just a normalization term, we can change σ^2 with m which is our size of data. And the estimator $g(x^i|\theta)$ just the same as $\theta_0 + \theta_1 x_1$:

$$L(y, \hat{y}) = \frac{1}{2m}\sum_i [y^i - (\theta_0 + \theta_1 x_1^i)]^2$$

This is a something like a error term between actual sales between predicted sales! The term $y^i - (\theta_0 + \theta_1 x_1)$ is a single error between sales i and predicted sales i. If we add square term, then our error will be positive defined which we want. If we sum all positive error, we will get total error. After that, just have a mean error, we multiply by $\frac{1}{m}$. This error looks like this

[64]:
```
from IPython.display import Image
from IPython.core.display import HTML
Image(filename= "./img/loss1.png",width=500, height=500)
```

[64]:

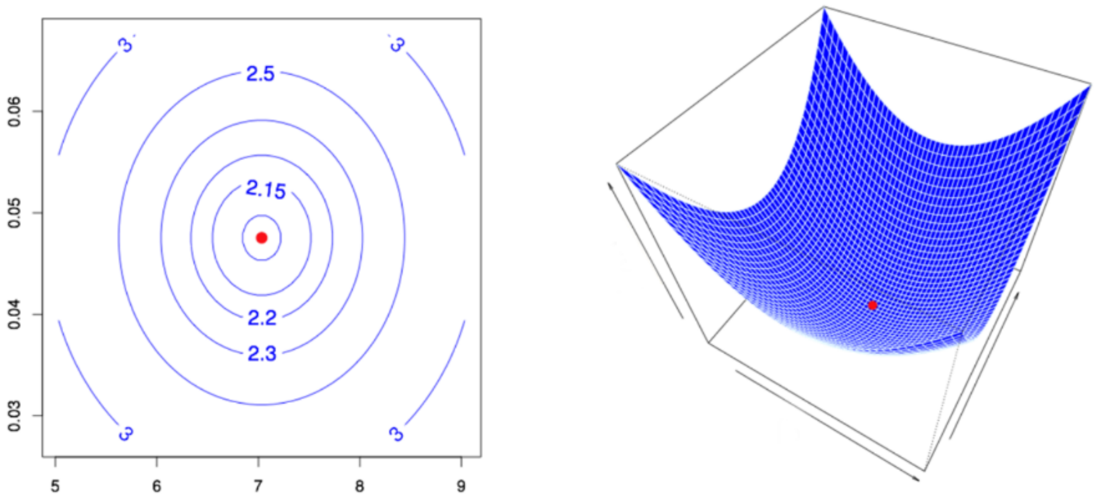


So our loss is mean of summation of errors from dots to line.

What we want to have is minimum error, in other words minimum loss, that means we want to minimize the loss function $\frac{1}{2m}\sum_i[y^i - (\theta_0 + \theta_1 x_1^i)]^2$. Our loss function is a convex function:

[68]:
```
from IPython.display import Image
from IPython.core.display import HTML
Image(filename= "./img/loss2.png",width=700, height=500)
```

[68]:



When we minimize the loss, we will get the optimal values of parameters. Befor we minimize, we need find the derivatives respect to parameters.

$$\frac{\partial L(y,\hat{y})}{\partial \theta_0} = \frac{\partial}{\partial \theta_0}\frac{1}{2m}\sum_i [y^i - (\theta_0 + \theta_1 x_1^i)]^2 = \frac{1}{m}\sum_i [y^i - (\theta_0 + \theta_1 x_1^i)]$$
$$\frac{\partial L(y,\hat{y})}{\partial \theta_1} = \frac{\partial}{\partial \theta_1}\frac{1}{2m}\sum_i [y^i - (\theta_0 + \theta_1 x_1^i)]^2 = \frac{1}{m}\sum_i [y^i - (\theta_0 + \theta_1 x_1^i)]x_1^i$$

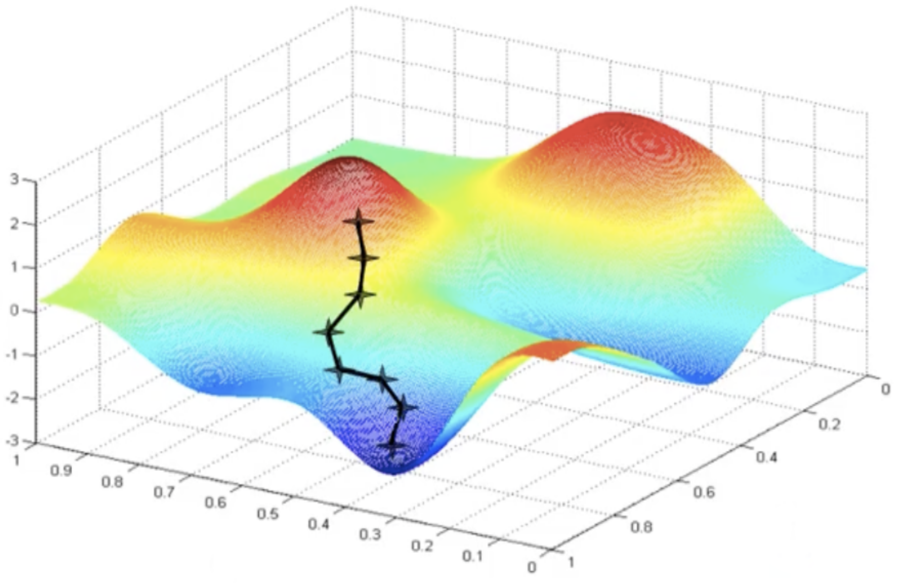
Now our goal is tune the parameters. We need optimal values of parameters. We will use an algorithm named Gradient Descent. Gradient descent is defined as:

```
for i = 0 to k:
    $\theta_0 = \theta_1 - \eta\frac{\partial}{\partial\theta_1}L(\hat{y},y)$
    $\theta_1 = \theta_1 - \eta\frac{\partial}{\partial\theta_1}L(\hat{y},y)$
end
```

Where η is our learning rate and k is the number of iterations. What η does is basically how big a step we take downhill with creating descent, so if η is very large then that corresponds to a very agressive gradient descent procedure when we trying to take huge steps downhill. If η is very small then we are taking little baby steps downhill.

[69]:
```
from IPython.display import Image
from IPython.core.display import HTML
Image(filename= "./img/loss3.png",width=500, height=500)
```

[69]:



0.5.1 Higher Dimensions

What if we have dimensions more than two? Let's see this type of data

```
[70]: import pandas as pd
      df_4 = pd.read_csv('./data/Advertising.csv')
      df_4.head()
```

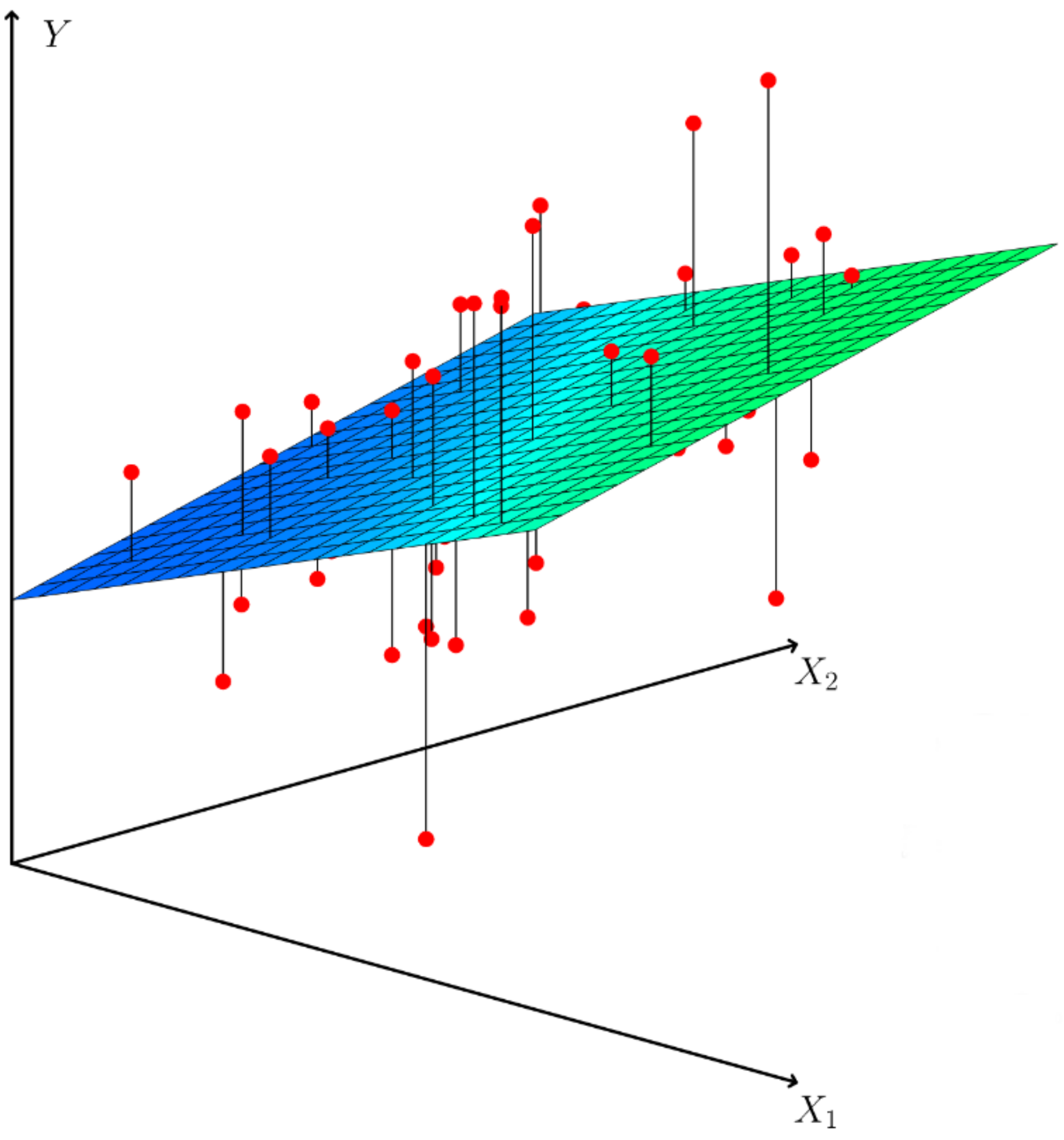
```
[70]:    id     TV  radio  newspaper  sales
      0   1  230.1   37.8       69.2   22.1
      1   2   44.5   39.3       45.1   10.4
      2   3   17.2   45.9       69.3    9.3
      3   4  151.5   41.3       58.5   18.5
      4   5  180.8   10.8       58.4   12.9
```

We cannot visualize this data but there is a some linear plane in higher dimension that fits to data. Assume that we have n dimensons. From our perspective, n attributes of sales. Our estimator will be

$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

```
[71]: from IPython.display import Image
      from IPython.core.display import HTML
      Image(filename= "./img/loss4.png",width=500, height=500)
```

```
[71]:
```



What we will do is the same as before but in higher dimension. We need the derivatives of parameters.

$$\frac{\partial L(y,\hat{y})}{\partial \theta_0} = \frac{\partial}{\partial \theta_0}\frac{1}{2m}\sum_i [y^i - (\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n]^2 = \frac{1}{m}\sum_i [y^i - (\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n)]$$

$$\frac{\partial L(y,\hat{y})}{\partial \theta_1} = \frac{\partial}{\partial \theta_1}\frac{1}{2m}\sum_i [y^i - (\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n)]^2 = \frac{1}{m}\sum_i [y^i - (\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n)]x_1^i$$

$$\frac{\partial L(y,\hat{y})}{\partial \theta_2} = \frac{\partial}{\partial \theta_2}\frac{1}{2m}\sum_i [y^i - (\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n)]^2 = \frac{1}{m}\sum_i [y^i - (\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n)] x_2^i$$

$$\frac{\partial L(y,\hat{y})}{\partial \theta_3} = \frac{\partial}{\partial \theta_3}\frac{1}{2m}\sum_i [y^i - (\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n)]^2 = \frac{1}{m}\sum_i [y^i - (\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n)] x_3^i$$

$$\vdots$$

$$\frac{\partial L(y,\hat{y})}{\partial \theta_n} = \frac{\partial}{\partial \theta_n}\frac{1}{2m}\sum_i [y^i - (\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n)]^2 = \frac{1}{m}\sum_i [y^i - (\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n)] x_n^i$$

But it will be worst practice to code every derivatives! We need an general formula that is:

$$\frac{\partial L(y,\hat{y})}{\partial \theta_j} = \frac{\partial}{\partial \theta_j}\frac{1}{2m}\sum_i [y^i - (\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n)]^2 = \frac{1}{m}\sum_i [y^i - (\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n)] x_j^i$$

From now, our gradient descent algorithm in higher dimension will be

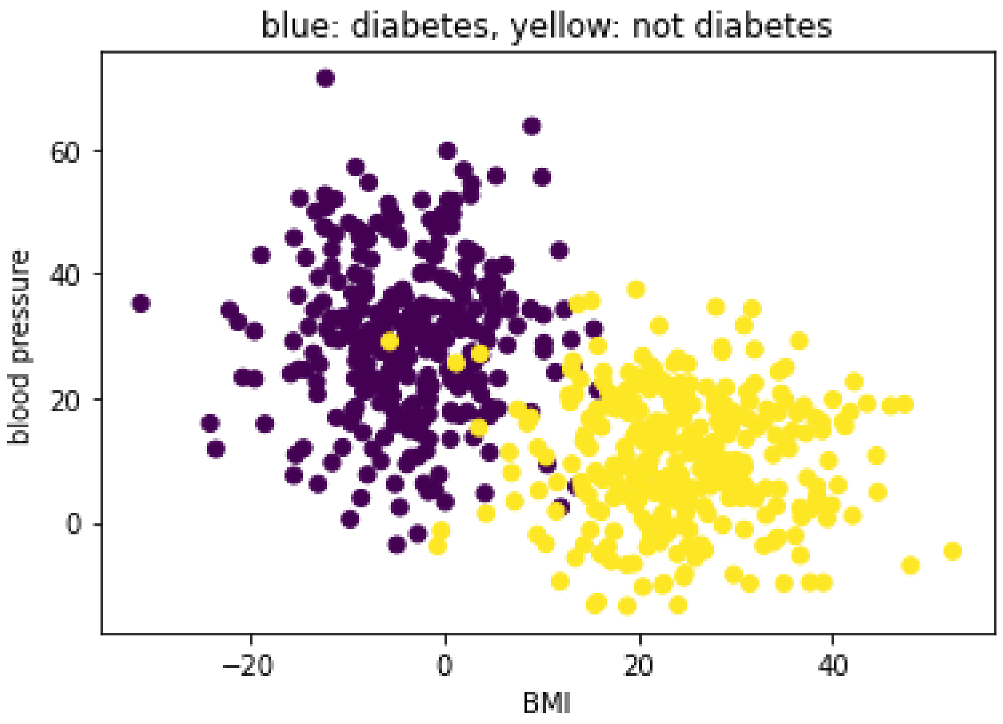
```
for i = 0 to k:
 for j = 0 to n
  $\theta_j = \theta_j - \eta \frac{\partial}{\partial \theta_j} L(\hat{y}, y)$
 end
end
```

0.6 Regression Problem and Logistic Regression

Let's start with a simple binary problem. We have patients, we measured their BMI (Body mass index) and blood pressure. After that, test results are recieved and some of are diabetes and some of are not diabetes. After those BMI, blood pressure, and diabetes/non-diabetes data; we need to model what we have. From now on, we will have a model that tell us a patient whether a diabetes or not depends on BMI and blood pressure. For example, when we plot our data, it looks like:

[115]:
```python
import numpy as np
import matplotlib.pyplot as plt
x_1 = np.concatenate([np.random.normal(-3, 8, 300),np.random.normal(25, 9,␣
 ↪300)])
x_2 = np.concatenate([np.random.normal(30, 13, 300), np.random.normal(11, 11,␣
 ↪300)])
```

```
y = np.concatenate([np.zeros(300),np.ones(300)])
df = pd.DataFrame({'x_1':x_1,'x_2':x_2,'y':y})
plt.title('blue: diabetes, yellow: not diabetes')
plt.xlabel('BMI')
plt.ylabel('blood pressure')
plt.scatter(df['x_1'],df['x_2'],c=df['y']);
```



Let's say that we have an classifier $f(x)$. For binary classification we have classes $y = \{0, 1\}$. If diabetes 1, else 0. From probabilistic perspective:

$$\begin{aligned} \text{if } f(x) > 0.5: &\quad \text{patient is diabetes} \\ \text{else if } f(x) < 0.5: &\quad \text{patient is not diabetes} \end{aligned}$$

From probabilistic perspective, if we had access to other pieces of knowledge such as diabetes full detail and full knowledge about blood pressure and BMI, whether someone is a low-risk or high-risk diabetes could have been deterministically calculated. The risk of diabetes of a patient is denoted by a Bernoulli random variable C conditioned on the observables $X = [x_1, x_2]$, where $C = 1 $ indicates a high-risk patient and $C = 0$ indicates a low-risk patient. Thus if we know $P(C|X1, X2)$, when a new application arrives with $X_1 = x_1$ and $X_2 = x_2$, we can

$$\text{choose} \begin{cases} C = 1, & \text{if } p(C = 1|x_1, x_1) > 0.5 \\ C = 0, & \text{else if } p(C = 1|x_1, x_1) < 0.5 \end{cases}$$

Applying linear regression to a classification problem usually isnt a great idea. Because $f(x)$ can be greater or less than 1 or 0. We want $f(x) \in [0, 1]$.

We can complete this normalization task with some kind of 'activation functions'. Activation functions are non-linear functions. Formally defined, a linear funcion $\sigma(z)$ is defined as:

$$h(c_1x + x_2y) = h(c_1x) + h(c_2y) = c_1h(x) + c_2h(y)$$

If a function does not follow the above definition, we call this function 'non-linear function'.

The sigmoid function

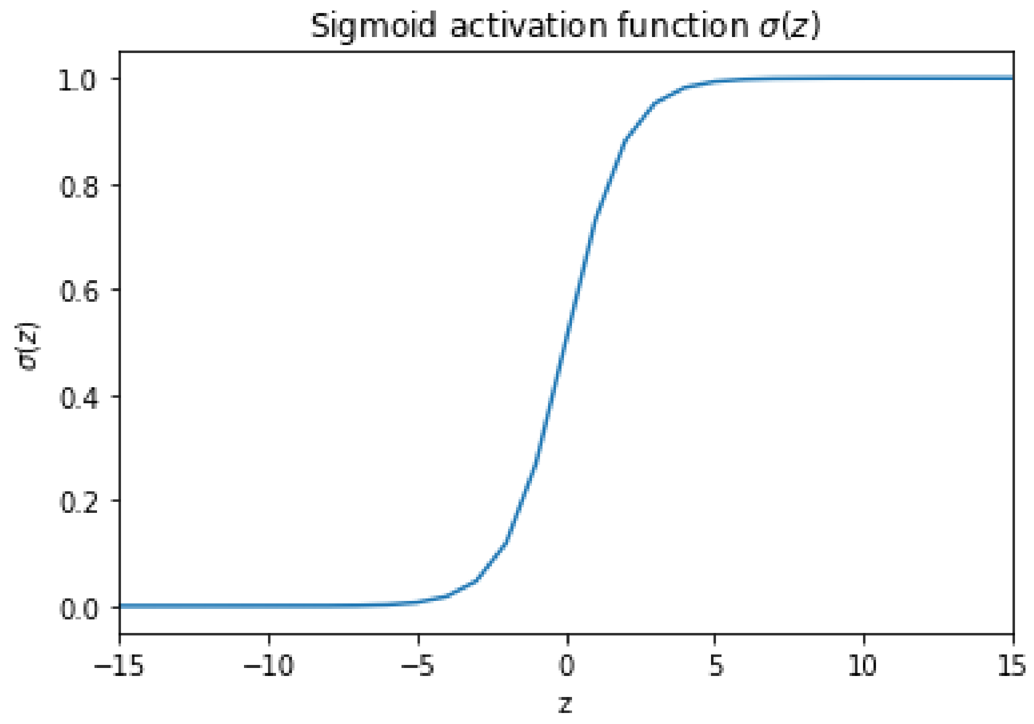
$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

is a non-linear activation function.

We can use this activation function for nonlinearity. So our estimator becomes:

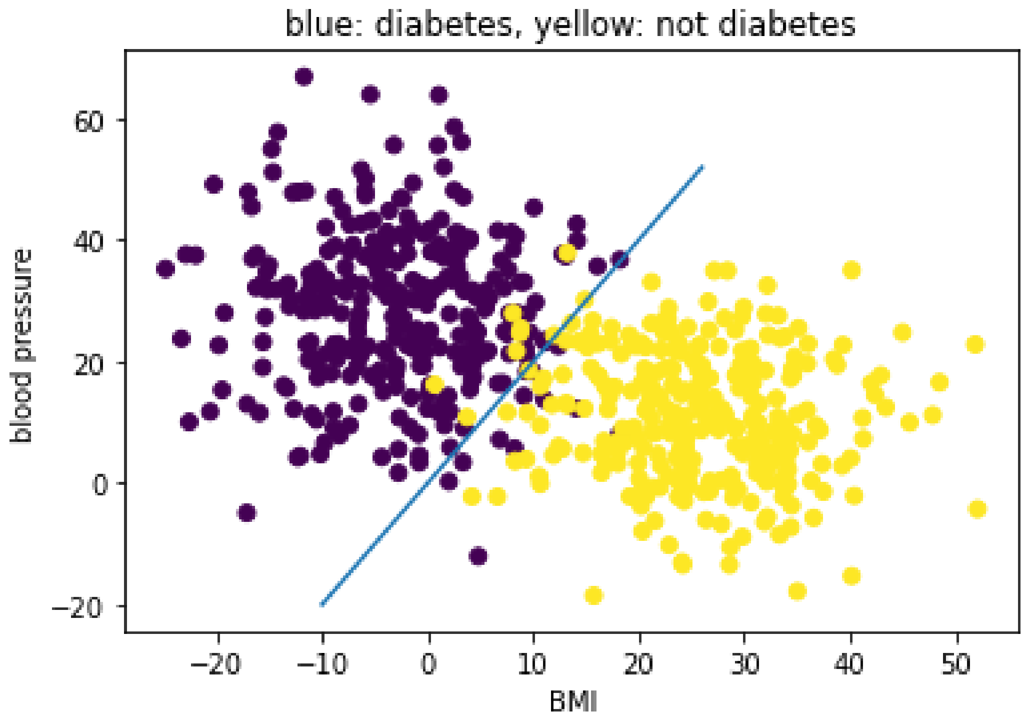
$$f(\sigma(x_0 + \theta_1x_1 + \theta_2x_2 + \cdots + \theta_nx_n)) = \frac{1}{1 + \exp(-(x_0 + \theta_1x_1 + \theta_2x_2 + \cdots + \theta_nx_n))}$$

[125]:
```
plt.plot(np.array([point for point in range (-15,300)]),1/(1+np.exp(-np.
 ↪array([point for point in range (-15,300)]))));
plt.xlim(right=15,left=-15);
plt.title('Sigmoid activation function $\sigma(z)$');
plt.xlabel('z');
plt.ylabel('$\sigma(z)}$');
```



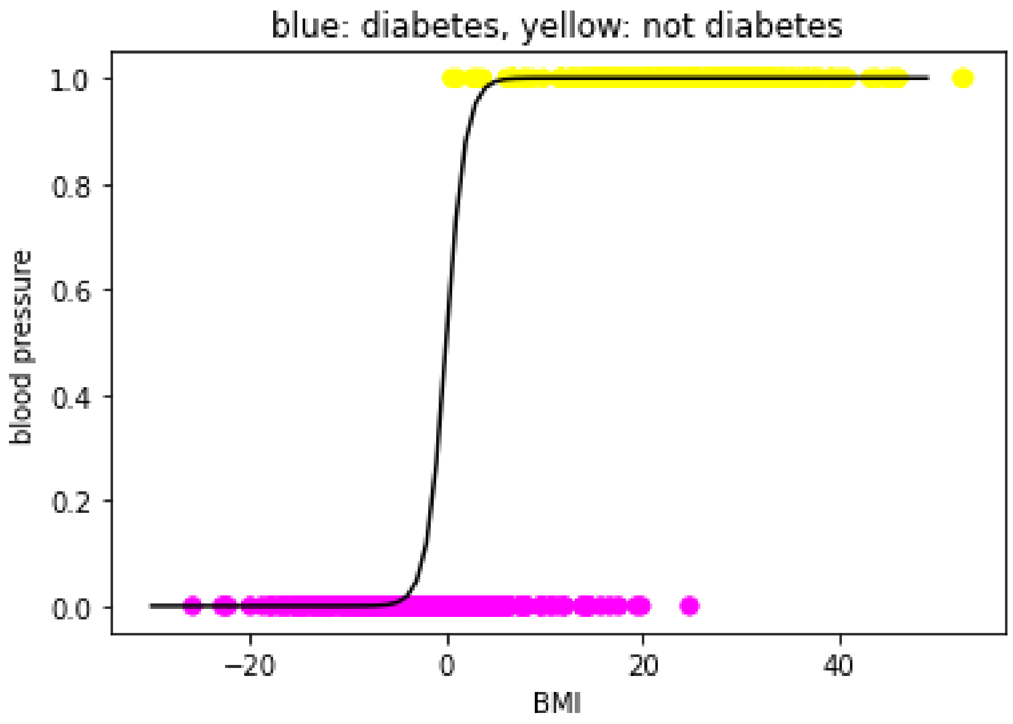
Our optimal decision boundary will look like:

```
[136]: import numpy as np
       import matplotlib.pyplot as plt
       x_1 = np.concatenate([np.random.normal(-3, 8, 300),np.random.normal(25, 9,␣
        ↪300)])
       x_2 = np.concatenate([np.random.normal(30, 13, 300), np.random.normal(11, 11,␣
        ↪300)])
       y = np.concatenate([np.zeros(300),np.ones(300)])
       df = pd.DataFrame({'x_1':x_1,'x_2':x_2,'y':y})
       plt.title('blue: diabetes, yellow: not diabetes')
       plt.xlabel('BMI')
       plt.ylabel('blood pressure')
       plt.scatter(df['x_1'],df['x_2'],c=df['y']);
       plt.plot(np.array([point for point in range(-10,27)]),2*np.array([point for␣
        ↪point in range(-10,27)]));
```



If a point at the left side of the line, then it is diabetes; otherwise not diabetes. Below graph shows as probabilistic mapping way:

[139]:
```
import numpy as np
import matplotlib.pyplot as plt
x_1 = np.concatenate([np.random.normal(-3, 8, 300),np.random.normal(25, 9,␣
 ↪300)])
x_2 = np.concatenate([np.random.normal(30, 13, 300), np.random.normal(11, 11,␣
 ↪300)])
y = np.concatenate([np.zeros(300),np.ones(300)])
df = pd.DataFrame({'x_1':x_1,'x_2':x_2,'y':y})
plt.title('blue: diabetes, yellow: not diabetes')
plt.xlabel('BMI')
plt.ylabel('blood pressure')
plt.scatter(df['x_1'],df['y'],c=df['y'],cmap='spring');
plt.scatter(df['x_1'],df['y'],c=df['y'],cmap='spring');
plt.plot(np.array([point for point in range (-30,50)]),1/(1+np.exp(-np.
 ↪array([point for point in range (-30,50)]))),color='black');
```



So our problem now again, defining the loss function and estimating the optimal parameters with partial derivatives. We proved that the objective function of linear regression comes from Gaussian Distribution (assuming that the data is randomly sampled form a Gaussian Distribution).

The objective function of logistic regression comes from a distribution as well. The Bernoulli Distribution Bernoulli $= p^x(1-p)^{(1-x)}$.

Let us see how can we learn the optimal parameters of logistic regression. We are given a sample of two classes, the data is data $= \{x^i, y^i\}$, if $x^i \in C_1$ we say that patient is diabetes and $y^i = 1$, if

$x \in C_0$ we say that patient is not diabetes and $y^i = 0$.

We assume that y^i, given x^i, is Bernoulli with probability $p^i = p(C_1|x^i)$, in other words

$p(y^i|x^i) \sim \text{Bernoulli}(p^i)$

Then, for m samples, the likelihood function will be

$$\mathcal{L}(\theta|x) = \prod_i^m (p^i)^{y^i}(1-p^i)^{1-y^i}$$

We know that when we have a likelihood function to maximize, we canalways turn it into an error function to be minimized as negative log-likelihood.

$$-\log \mathcal{L}(\theta|x) = -\log \prod_i^m [(p^i)^{y^i}(1-p^i)^{1-y^i}]$$

$$= -\sum_i^m \log[(p^i)^{y^i}(1-p^i)^{1-y^i}]$$

$$= -\sum_i^m \log[(p^i)^{y^i}] + log[(1-p^i)^{1-y^i}]$$

$$= -\sum_i^m y^i \log(p^i) + (1-y^i)\log(1-p^i)$$

This function our loss function on logistic regression! We proved that where this loss function comes from.

As we defined above, the probability is actually

$$p = \sigma(x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n) = \frac{1}{1+\exp(-(x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n))}$$

which is predicted $\hat{y}$

To sum up, the loss becomes:

$$L(\hat{y}, y) = -\sum_i^m y^i \log(p^i) + (1-y^i)\log(1-p^i) = -\sum_i^m y^i \log(\hat{y}^i) + (1-y^i)\log(1-\hat{y}^i)$$

We call this loss ‘Binary Cross Entropy Loss’. Binary crossentropy is a loss function that is used in binary classification tasks. In information theory, the cross entropy between two probability distributions $p = y$ and $q = \hat{y}$ over the same underlying set of events measures the average number of bits needed to identify an event drawn from the set if a coding scheme used for the set is optimized for an estimated probability distribution q, rather than the true distribution p.

Not, let’s minimize this loss.

$$\nabla_\theta L(\hat{y}, y) = \nabla_\theta \left(- \sum_i^m y^i \log(p^i) + (1 - y^i) \log(1 - p^i) \right)$$

$$= - \sum_i^m y^i \nabla_\theta \log(p^i) + (1 - y^i) \nabla_\theta \log(1 - p^i)$$

$$= - \sum_i^m \frac{y^i}{p^i} \nabla_\theta p^i + \frac{(1 - y^i)}{(1 - p^i)} \nabla_\theta (1 - p^i)$$

$$= - \sum_i^m \frac{y^i}{p^i} \nabla_\theta p^i - \frac{(1 - y^i)}{(1 - p^i)} \nabla_\theta p^i$$

$$= - \sum_i^m \left(\frac{y^i}{p^i} - \frac{(1 - y^i)}{(1 - p^i)} \right) \nabla_\theta p^i$$

From this point, all we need to do is find the gradient of p. Note that the derivative of the sigmoid function:

$$\nabla_z \sigma(z) = \sigma(z)(1 - \sigma(z))$$

and from the chain rule we have,

$$\nabla_\theta p^i = \nabla_\theta \sigma(\theta^T x^i) = \sigma(\theta^T x^i)(1 - \sigma(\theta^T x^i)) x^i$$

$$= p^i (1 - p^i) x^i$$

Plugging in this gradient value, we have

$$= - \sum_i^m \left(\frac{y^i}{p^i} - \frac{(1 - y^i)}{(1 - p^i)} \right) \nabla_\theta p^i$$

$$= - \sum_i^m \left(\frac{y^i}{p^i} - \frac{(1 - y^i)}{(1 - p^i)} \right) p^i (1 - p^i) x^i$$

$$= - \sum_i^m (y^i (1 - p^i) - (1 - y^i) p^i) x^i$$

$$= - \sum_i^m (y^i - p^i) x^i$$

From now we can easily update our parameters with Gradient Descent Algorithm again.

```
for i = 0 to k:
    $\theta = \theta + \eta \frac{\partial}{\partial \theta} L(\hat{y}, y) = \theta + \eta \sum_{i}^{m} (y^i - p^i) x^i$
end
```

[ ]: