Machine Learning

Bruce Zhang

2024.12.3

Contents

1 What is Machine Learning?

Typically, Machine Learning is divided into two kinds of parts, Supervised learning and Unsupervised learning.

Supervised learning: Learns from being given "right answers".

$$\text{Supervised learning} = \begin{cases} \text{Regression(回归): Predict a number, infinitely many possible outputs.} \\ \text{Classification(分类): Predict categories, small number of possible outputs.} \end{cases}$$

Unsupervised learning: Data only comes with inputs x, but not outputs labels y. Algorithm has to find structure in the data.

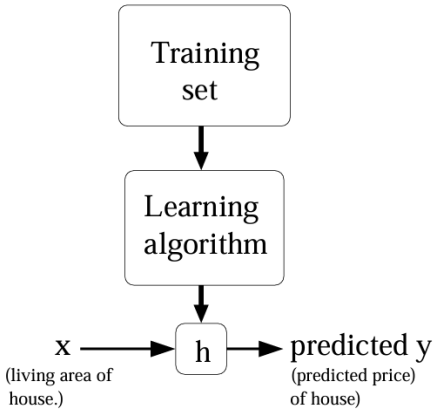
$$\text{Unsupervised learning} = \begin{cases} \text{Clustering(聚类): Group similar data points together.} \\ \text{Dimensionality reduction(降维): Compress data using fewer number.} \\ \text{Anomaly detection(异常检测): Find unusual data points.} \end{cases}$$

2 Supervised Learning

The main **notation** we will use are as follows

$$\begin{aligned} x &= \text{"input" variable or "input" feature} \\ y &= \text{"target" variable or "output" variable} \\ m &= \text{number of training examples} \\ (x^{(i)}, y^{(i)}) &= i^{\text{th}} \text{ training examples} \end{aligned}$$

Seen pictorially, the process is therefore like this:



When the target variable that we're trying to predict is continuous, such as in our housing example, we call the learning problem a **regression** problem. When y can take on only a small number of discrete values (such as if, given the living area, we wanted to predict if a dwelling is a house or an apartment, say), we call it a **classification** problem.

2.1 Linear Regression

Living area (feet^2)	**#bedrooms**	**Price (1000$s)**
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540
⋮	⋮	⋮

Given this training set, obvisouly, the input feature, namely the $\boldsymbol{x}$ is two-dimensional vector in $\mathbb{R}^2$. Recalling the previous picture, what we need to do is representing a function h, which predicts the output $\hat{y}$. Let's say we decide to approximate y as a linear function of x:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 \tag{2.1.1}$$

The θ_i's are the parameters (also called **weight**) parameterizing the space of linear functions. To simplify our notation, we introduce the convention of letting $x_0 = 1$ (**intercept term**).

$$h(\boldsymbol{x}) = \sum_{i=0}^{n} \theta_i x_i = \boldsymbol{\theta}^T \boldsymbol{x}$$

A problem is that how can we identify the parameters so that the output of $h(x)$, $\hat{y}$ close to y. **So we actually need another function that can help us to evaluate how good a $\boldsymbol{\theta}$ parameter-based function is**. We define the **cost function**:

$$J(\boldsymbol{\theta}) = \frac{1}{2m} \sum_{i=1}^{m} \left(h(x^{(i)}) - y^{(i)} \right)^2 \tag{2.1.2}$$

2.1.1 Gradient Descent Algorithm

Just like the name of "cost function", we hope to choose $\boldsymbol{\theta}$ so as to minimize $J(\boldsymbol{\theta})$. What we really want is an efficient algorithm that we can write in code for automatically finding the values of parameters $\boldsymbol{\theta}$, they give us the best fit line, that minimizes the cost function J.

Let's consider the **gradient descent algorithm**, which start with some initial $\boldsymbol{\theta}$, and repeatedly performs the update:

$$\boldsymbol{\theta}_j = \boldsymbol{\theta}_j - \alpha \frac{\partial}{\partial \boldsymbol{\theta}_j} J(\boldsymbol{\theta}) \tag{2.1.3}$$

α is called the **learning rate**.

(Note: This update is simultaneously performed for all values of $j = 0, \ldots, n$.)

$$\begin{aligned}\frac{\partial}{\partial \boldsymbol{\theta}_j} J(\boldsymbol{\theta}) &= \frac{1}{2m} \frac{\partial}{\partial \boldsymbol{\theta}_j} \left[\sum_{i=1}^{m} \left(h(x^{(i)}) - y^{(i)} \right)^2 \right] \\ &= \frac{1}{2m} \sum_{i=1}^{m} \frac{\partial}{\partial \boldsymbol{\theta}_j} \left(h(x^{(i)}) - y^{(i)} \right)^2 \\ &= \frac{1}{m} \sum_{i=1}^{m} \left(h(x^{(i)}) - y^{(i)} \right) x_j^{(i)}\end{aligned}$$

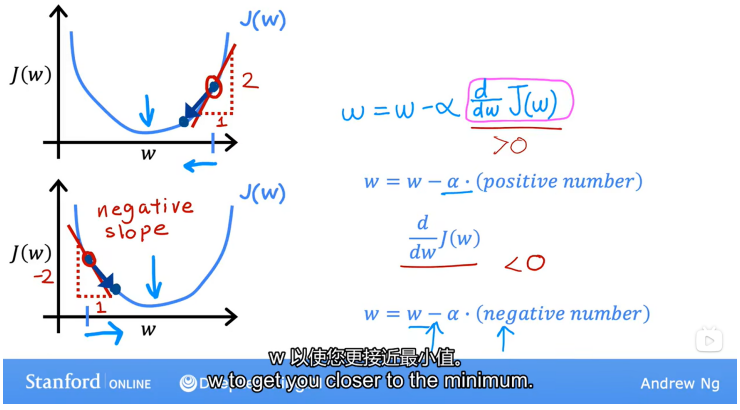
In conclusion, this gives the update rule:

$$\boldsymbol{\theta}_j = \boldsymbol{\theta}_j + \alpha \frac{1}{m} \sum_{i=1}^{m} \left[\left(y^{(i)} - h(x^{(i)}) \right) x_j^{(i)} \right] \tag{2.1.4}$$

The rule is called the **LMS** update rule (LMS stands for "least mean squares"), and is also known as the **Widrow-Hoff** learning rule.

When I was reading the explanation of the equation above from the note of CS-229, I prefer Andrew Ng's explanation in the video of Coursera.

In a moment, if the partial of J is a positive number in a place, obvisouly with the increase of the parameter θ_j, the value of J will alse increase, thus we minus this positive number so that the value of J will decrease. Accordingly, if the partial of J is a negative number in a place, with the increase of the parameter θ_j, the value of J will decrease, thus we minus this negative number so that the value of J will decrease.



This method looks at every example in the entire training set on every

step, and is called **batch gradient descent**. In fact, no mater where the initial value is chosen, the optimization problem we have posed here for linear regression has only one global, and no other local, optima, thus gradient descent always converges because J **is a convex quadratic function**.

There is an alternative to batch gradient descent that also works very well. Consider the following algorithm:

Loop {
  for $i = 1$ to m, {
    $\theta_j = \theta_j + \alpha\big(y^{(i)} - h_\theta(x^{(i)})\big)x_j^{(i)}$ (for every j).
  }
}

In this algorithm, we repeatedly run through the training set, and each time we encounter a training example, we update the parameters according to the gradient of the error with respect to that single training example only. This algorithm is called **stochastic gradient descent** (also incremental gradient descent). Whereas batch gradient descent has to scan through the entire training set before taking a single step—a costly operation if m is large—stochastic gradient descent can start making progress right away, and continues to make progress with each example it looks at. Often, stochastic gradient descent gets θ "close" to the minimum much faster than batch gradient descent.

2.1.2 The Normal Equation

Gradient descent gives one way of minimizing J. And it is time to perform the minimization explicitly and without resorting to an iterative algorithm by explicitly taking J's derivatives with respect to the θ_j's, and setting them to zero.However, we need to introduce some notation before showing.

Matrix derivatives

For a function f: $\mathbb{R}^{m\times n} \to \mathbb{R}$ mapping from m-by-n matrices to the real numbers, we define the derivative of f with respect to A to be:

$$\nabla_A f(A) = \begin{bmatrix} \frac{\partial f}{\partial A_{11}} & \cdots & \frac{\partial f}{\partial A_{1n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial A_{m1}} & \cdots & \frac{\partial f}{\partial A_{mn}} \end{bmatrix} \tag{2.1.5}$$

We also introduce the **trace** operator. For an square matrix A, the trace of A is defined to be the sum of its diagonal entries:

$$trA = \sum_{i=1}^{n} A_{ii}$$

There are lots of properties of the trace operator.

$$\begin{aligned} trAB &= trBA \\ trA &= trA^T \\ tr(A+B) &= trA + trB \\ traA &= atrA \\ \nabla_A trAB &= B^T \\ \nabla_{A^T} f(A) &= \left(\nabla_A f(A)\right)^T \\ \nabla_A trABA^TC &= CAB + C^TAB^T \\ \nabla_A |A| &= |A| \left(A^{-1}\right)^T \end{aligned}$$

Least squares revisited

Given a training set, define the design matrix X to be the m-by-n matrix that contains the training examples' input values in its rows:

$$X = \begin{bmatrix} - \left(x^{(1)}\right)^T - \\ - \left(x^{(2)}\right)^T - \\ \vdots \\ - \left(x^{(m)}\right)^T - \end{bmatrix}$$

$\boldsymbol{y}$ is the m-dimensional vector containing all the target values from the training set:

$$\boldsymbol{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

Notice that $h(x^{(i)}) = \left(x^{(i)}\right)^T \theta$:

$$X\theta - \boldsymbol{y} = \begin{bmatrix} \left(x^{(1)}\right)^T \theta \\ \vdots \\ \left(x^{(m)}\right)^T \theta \end{bmatrix} - \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix} = \begin{bmatrix} h(x^{(1)}) - y^{(1)} \\ \vdots \\ h(x^{(m)}) - y^{(m)} \end{bmatrix}$$

using the fact that for a vector z, we have that $z^T z = \sum_i z_i^2$:

$$\begin{aligned} \frac{1}{2m} \left(X\theta - \boldsymbol{y}\right)^T \left(X\theta - \boldsymbol{y}\right) &= \frac{1}{2m} \sum_{i=1}^{m} \left(h(x^{(i)}) - y^{(i)}\right)^2 \\ &= J(\theta) \end{aligned}$$

Finally, to minimize J, let's find its derivatives with respect to θ. Combining some of the previous function, we find that

$$\nabla_{A^T} trABA^TC = B^TA^TC^T + BA^TC \tag{2.1.6}$$

Hence,

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \frac{1}{2m}(X\theta - \vec{y})^T(X\theta - \vec{y}) \\
&= \frac{1}{2m}\nabla_\theta \left(\theta^T X^T X\theta - \theta^T X^T \vec{y} - \vec{y}^T X\theta + \vec{y}^T\vec{y}\right) \\
&= \frac{1}{2m}\nabla_\theta \operatorname{tr}\left(\theta^T X^T X\theta - \theta^T X^T \vec{y} - \vec{y}^T X\theta + \vec{y}^T\vec{y}\right) \\
&= \frac{1}{2m}\nabla_\theta \left(\operatorname{tr}\theta^T X^T X\theta - 2\operatorname{tr}\vec{y}^T X\theta\right) \\
&= \frac{1}{2m}\left(X^T X\theta + X^T X\theta - 2X^T\vec{y}\right) \\
&= X^T X\theta - X^T\vec{y}
\end{aligned}
$$

(IN FACT, I don't want to explain the details of the deduce, refering to CS229-note1)

To minimize J, we set its derivatives to zero, and obtain the **normal equation**:

$$X^T X\theta = X^T\vec{y} \tag{2.1.7}$$

thus

$$\theta = \left(X^T X\right)^{-1} X^T\vec{y}$$

As for the part of linear regression, there are still two subdivisions, Probabilistic interpretation and Locally weight linear regression, which just have a role of supplement and expanding, thus I don't want to write them into my note, refering to CS229-note1.

2.1.3 Probabilistic Interpretation

Now we need to prove the rationality of the cost function strictly mathematically, that is, from the point of view of probability statistics.

Before the proof, it's nessary to retrospect two important mathematical terms, **Probability and Likelihood**. In informal contexts likelihood and Probability are almost synonymous, but in statistics likelihood and probability are two different concepts. **Probability** is the possibility of something

happening under a specific environment, that is, the possibility of predicting something happening according to the parameters corresponding to the environment before the result is produced. For example, before flipping a coin, we do not know which side will end up facing up, but according to the nature of the coin, we can infer that the probability of any side facing up is 50%. The probability is meaningful only before the coin is flipped, after which the outcome is certain; **Likelihood** is just the opposite, to speculate on the possible circumstances (parameters) that produced the result from the certain result. Still the example of flipping a coin, suppose we randomly toss a coin 1,000 times, the result is 500 times heads, 500 times numbers heads (the actual situation is not usually so ideal, here is just an example), We can easily judge that this is a standard coin, the probability of both sides being up is 50%, and the process is that we judge the nature (parameter) of the event itself according to the result, that is, the likelihood.

If we represent the parameter corresponding to the environment by θ, the result by x, then here is the probability:

$$P(x|\theta)$$

The probability of event x occurring on the premise of θ

The corresponding likelihood can be expressed as

$$L(\theta|x)$$

The probability that the parameter is θ on the premise of x

Probabilty is equal to likelihood numerically when the result corresponds to the parameter.

$$L(\theta|x) = P(x|\theta)$$

Let us assume that the target variables and the inputs are related via the equation

$$y^{(i)} = \theta^T x^{(i)} + \epsilon^{(i)}$$

where $\epsilon^{(i)}$ is an error term that captures something weird wrong, like un-

modeled effects or random noise. Let's further assume that the $\epsilon^{(i)}$ are distributed IID (independently and identically distributed) according to a Gaussian distribution with mean zero and some variance σ^2. We can write this assumption as $\epsilon^{(i)} \sim \mathcal{N}(0, \sigma^2)$. The density of $\epsilon^{(i)}$ is given by

$$p(\epsilon^{(i)}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(\epsilon^{(i)})^2}{2\sigma^2}\right)$$

This implies that

$$p(y^{(i)}|x^{(i)};\theta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right)$$

When we wish to explicitly view this as a function of θ, we will instead call it the likelihood function:

$$L(\theta) = L(\theta; X, y) = p(y|X;\theta)$$

Note that by the independence assumption on the $\epsilon^{(i)}$'s, this can be written

$$\begin{aligned} L(\theta) &= \prod_{i=1}^{m} p(y^{(i)}|x^{(i)};\theta) \\ &= \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) \end{aligned} \tag{2.1.8}$$

Now, given this probabilistic model relating the $y^{(i)}$'s and the $x^{(i)}$'s, what is a reasonable way of choosing our best guess of the parameters θ? The principal of maximum likelihood says that we should should choose θ so as to make the data as high probability as possible. I.e., we should choose θ to maximize $L(\theta)$.

Obvisouly it's hard to maximize the original $L(\theta)$, which is made up of lots of complicated parts. It will be simpler if we instead maximize the log

likelihood $\ell(\theta)$.

$$\begin{aligned}\ell(\theta) &= \log L(\theta) \\ &= \log \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{\left(y^{(i)} - \theta^T x^{(i)}\right)^2}{2\sigma^2}\right) \\ &= \sum_{i=1}^{m} \log \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{\left(y^{(i)} - \theta^T x^{(i)}\right)^2}{2\sigma^2}\right) \\ &= m \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{\sigma^2} \cdot \frac{1}{2} \sum_{i=1}^{m} \left(y^{(i)} - \theta^T x^{(i)}\right)^2.\end{aligned} \tag{2.1.9}$$

Hence, maximize $\ell(\theta)$ gives the same answer as minimizing:

$$\frac{1}{2} \sum_{i=1}^{m} \left(y^{(i)} - \theta^T x^{(i)}\right)^2$$

which we recognize to be $J(\theta)$, our original least-squares cost function.

So that's a relatively complete mathematical proof!

2.2 Classification and Logistic Regression

2.2.1 Logistic Regression

If we are trying to build a spam classifier for email, the $x^{(i)}$ may be some features of a piece of email, and y may be 1 if it is a piece of spam mail, and 0 otherwise. 0 is also called the **negative class**, and 1 the **positive class**. Given $x^{(i)}$, the corresponding $y^{(i)}$ is also called the **label** for the training example.

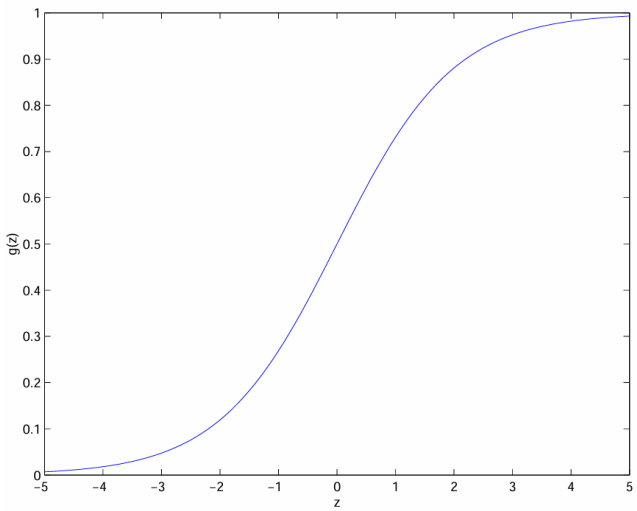
We could approach the classification problem considering the hypotheses $h(x)$ as **the Probability** of x's label, namely y. We will choose

$$h(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}} \tag{2.2.1}$$

where

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

is called the **logistic function** or the **sigmoid function**.Here is a plot of $g(z)$



$\theta^T x$ is a linear transformation of the input x. Moreover, $g(z)$, and hence alse $h(x)$, is always bounded between 0 and 1. We are trying to map the input to a range of 0 to 1, representing the probability that the input belongs to the given label.

Other functions that smoothly increase from 0 to 1 can also be used, but for a couple of reasons that we'll see later (when we talk about GLMs, and when we talk about generative learning algorithms), the choice of the logistic function is a fairly natural one. Before moving on, here's a useful property of the derivative of the sigmoid function:

$$\begin{aligned} g'(z) &= \frac{d}{dz}\frac{1}{1+e^{-z}} \\ &= \frac{1}{(1+e^{-z})^2}(e^{-z}) \\ &= \frac{1}{1+e^{-z}}\left(1-\frac{1}{1+e^{-z}}\right) \\ &= g(z)\left(1-g(z)\right) \end{aligned}$$

2.2.2 Logistic Loss Function

Previously we introduce the cost function of linear regression as follows

$$J(\boldsymbol{\theta}) = \frac{1}{2m}\sum_{i=1}^{m}\left(h(x^{(i)}) - y^{(i)}\right)^2$$

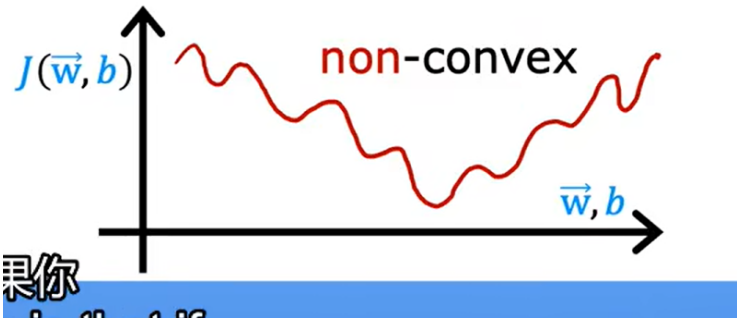
then write it in another way:

$$J(\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{m}\frac{1}{m}\left(h(x^{(i)}) - y^{(i)}\right)^2 \tag{2.2.2}$$

We regard the part to the right of the summation symbol as a whole, called **the loss on a single training example**.

$$L\left(h(x^{(i)}), y^{(i)}\right)$$

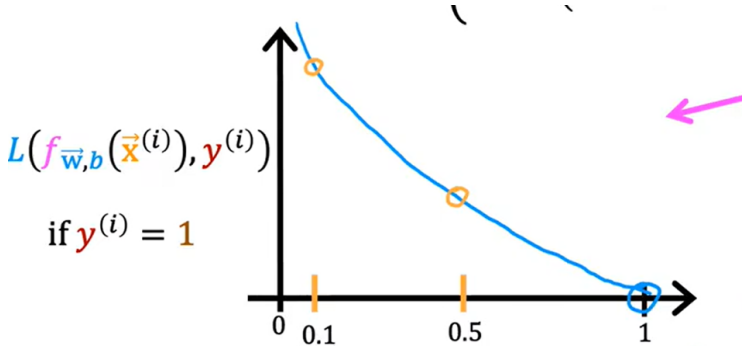
As for the linear regression part, it's ok to use the previous form, meanly because that is a convex function for $J(\boldsymbol{\theta})$. However, because of the introduction of logistic function, the previous form will lead to a non-convex function like this:



It's hard to converge. WE NEED TO INTRODUCE ANOTHER FORM!

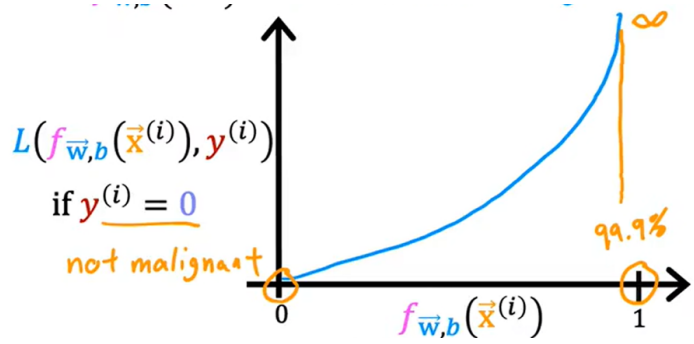
$$L\left(h(x^{(i)}), y^{(i)}\right) = \begin{cases} -\ln(h(x^{(i)})) & \text{if } y^{(i)} = 1 \\ -\ln(1 - h(x^{(i)})) & \text{if } y^{(i)} = 0 \end{cases} \tag{2.2.3}$$

For the first equation that $y^{(i)} = 1$, here is the plot.



As $h(x^{(i)}) \to 1$ then loss $\to 0$
As $h(x^{(i)}) \to 0$ then loss $\to 1$

It fulfils that loss is lowest when $h(x^{(i)})$ predict close to true label $y^{(i)}$. For the first equation that $y^{(i)} = 1$, here is the plot.



As $h(x^{(i)}) \to 0$ then loss $\to 0$
As $h(x^{(i)}) \to 1$ then loss $\to 1$

The further prediction $h(x^{(i)})$ is from target $y^{(i)}$, the higher the loss.

This is what we actually want to use as the part of the new cost function. Notice both two of them are easy to converge. Let's simplify cost function.

The loss function can be represented as the following form:

$$L\left(h(x^{(i)}), y^{(i)}\right) = -y^{(i)} \ln(h(x^{(i)})) - (1 - y^{(i)}) \ln(1 - h(x^{(i)}))$$

Thus the final cost function:

$$\begin{aligned} J(\boldsymbol{\theta}) &= \frac{1}{m}\sum_{i=1}^{m}\left[L\left(h(x^{(i)}), y^{(i)}\right)\right] \\ &= -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\ln(h(x^{(i)})) + (1-y^{(i)})\ln(1-h(x^{(i)}))\right] \end{aligned} \tag{2.2.4}$$

The equation above is derived from **the maximum likelihood**, we'll talk about it later.

2.2.3 Gradient Descent

We can derived the formula as following from cost function. (However the process is a tiny little complex, I don't want to type them out)

$$\frac{\partial}{\partial \boldsymbol{\theta}_j} J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m}\left(h(x^{(i)}) - y^{(i)}\right)x_j^{(i)}$$

Surprise! It seems like it's the same with the former one.
But in fact, is it? NO! They have different $h(\boldsymbol{x})$.

$$\begin{cases} \text{Linear regression} \quad h(\boldsymbol{x}) = \boldsymbol{\theta}^T\boldsymbol{x} \\ \text{Logistic regression} \quad h(\boldsymbol{x}) = \frac{1}{1+e^{(\boldsymbol{\theta}^T\boldsymbol{x})}} \end{cases}$$

This therefore gives us the gradient descent rule

$$\boldsymbol{\theta}_j = \boldsymbol{\theta}_j + \alpha\frac{1}{m}\sum_{i=1}^{m}\left[\left(y^{(i)} - h(x^{(i)})\right)x_j^{(i)}\right] \tag{2.2.5}$$

The form is the same as the former, except for $h(\boldsymbol{x})$. Is this coincidence, or is there a deeper reason behind this? We'll answer this when get get to GLM models.

2.2.4 Again: Probabilistic Interpretation

Let's assume that

$$P(y = 1|x; \theta) = h(x)$$
$$P(y = 0|x; \theta) = 1 - h(x)$$

which can be written compactly as

$$p(y|x; \theta) = (h(x))^y (1 - h(x))^{1-y}$$

Assuming that the m training examples were generated independently, we can then write down the likelihood of the parameters as

$$\begin{aligned} L(\theta) &= p(y|x; \theta) \\ &= \prod_{i=1}^{m} p(y^{(i)}|x^{(i)}; \theta) \\ &= \prod_{i=1}^{m} \left(h(x^{(i)})\right)^{y^{(i)}} \left(1 - h(x^{(i)})\right)^{1-y^{(i)}} \end{aligned}$$

As before, it will be easier to maximize the log likelihood:

$$\begin{aligned} \ell(\theta) &= \ln L(\theta) \\ &= \sum_{i=1}^{m} y^{(i)} \ln(h(x^{(i)})) + (1 - y^{(i)}) \ln(1 - h(x^{(i)})) \end{aligned}$$

To maximize it, namely minimize the previous one. Here the proof is!

2.3 Generalized Linear Models

We can figure out that there are some somehow similarities between linear regression and classification regression. In this section, we will show that both of these methods are special cases of a broader family of models, called Generalized Linear Models (GLMs). We will also show how other models in the GLM family can be derived and applied to other classification and regression problems.

2.3.1 The Exponential Family

We say that a class of distributions is in the exponential family if it can be written in the form

$$p(y;\eta) = b(y)\exp\left(\eta^T T(y) - a(\eta)\right) \tag{2.3.1}$$

η: **natural parameter**
$T(y)$: **sufficient statistic**
$a(\eta)$: **log partition function**

The quantity $e^{-a(\eta)}$ essentially plays the role of a normalization constant, that makes sure the distribution $p(y;\eta)$ sums/integrates over y to 1.

We see that the probability of y is determined by parameter η, so a fixed choice of T, a and b defines a family of distributions; as we vary η, we get different distributions within this family.

Considering the **Bernoulli distribution**, as we vary ϕ, which will contribute to η, we obtain Bernoulli distribution with different means.

We write the Bernoulli distribution as:

$$\begin{aligned} p(y;\phi) &= \phi^y (1-\phi)^{1-y} \\ &= \exp\left(y\ln\phi + (1-y)\ln(1-\phi)\right) \\ &= \exp\left(\ln\left(\frac{\phi}{1-\phi}\right) y + \ln(1-\phi)\right) \end{aligned}$$

Thus, the natural parameter is given by $\eta = \ln(\phi/(1-\phi))$.

$$\begin{aligned} T(y) &= y \\ a(\phi) &= -\ln(1-\phi) \\ &= \ln(l + e^{\phi}) \\ b(\phi) &= 1 \end{aligned}$$

Let's now move on to consider the **Gaussian distribution**. Recall that when deriving linear regression, the value of σ^2 had no effect on our

final choice of θ. To simplify the derivation below, we set $\sigma^2 = 1$

$$\begin{aligned} p(y;\mu) &= \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}(y-\mu)^2\right) \\ &= \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}y^2\right) \cdot \exp\left(\mu y - \frac{1}{2}\mu^2\right) \end{aligned}$$

Thus, we see that the Gaussian is in the exponential family.

$$\begin{aligned} \eta &= \mu \\ T(y) &= y \\ a(\eta) &= \mu^2/2 \\ &= \eta^2/2 \\ b(y) &= (1/\sqrt{2\pi}) \exp\left(-y^2/2\right) \end{aligned}$$