Notes for Machine Learning

Wyatt Ho

November 23, 2021

Contents

Chapter 1

Introduction

1.1 What is Machine Learning?

- Arthur Samuel described it as: "the field of study that gives computers the ability to learn without being explicitly programmed." This is an older, informal definition.
- Tom Mitchell provides a more modern 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."

1.2 Supervised Learning

- Given a data set and already know the correct output.
- Are categorized into "regression" and "classification" problems.
- In a regression problem, we are trying to predict results within a continuous output, meaning that we are trying to map input variables to some continuous function.
- In a classification problem, we are instead trying to predict results in a discrete output. In other words, we are trying to map input variables into discrete categories.

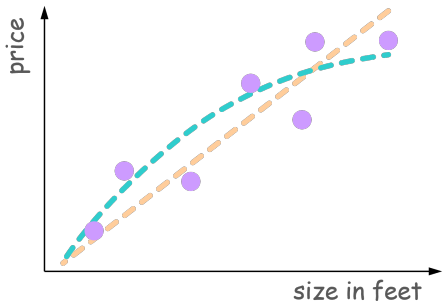

Figure 1.1: House price

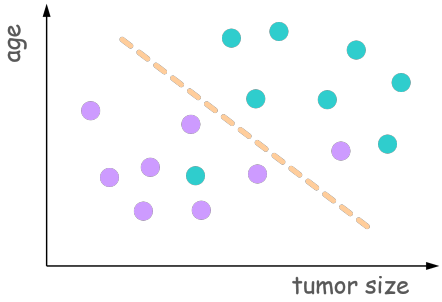

Figure 1.2: Tumor size

1.3 Unsupervised Learning

- Approach problems with little or no idea what our results should look like.
- We can derive this structure by clustering the data based on relationships among the variables in the data.
- There is no feedback based on the prediction results.

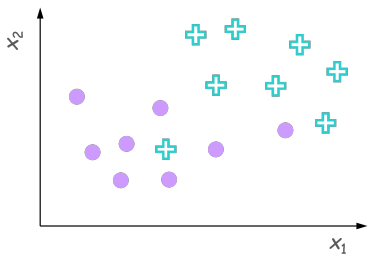

Figure 1.3: Supervised Learning

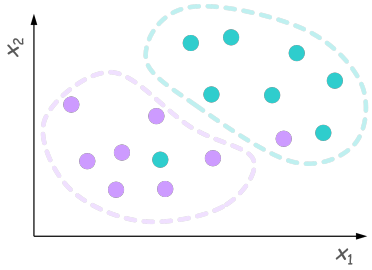

Figure 1.4: Unsupervised Learning

Chapter 2

Linear Regression

2.1 Introduction

- Try to predict results using a first-order linear equation

$$h(x) = \theta_0 + \theta_1 x \tag{2.1}$$

  To make a good prediction, choose appropriate *parameters* θ_0, θ_1 so that $h(x)$ is close to y for our training examples (x, y)

- Define the cost function

$$J_\theta(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h^{(i)} - y^{(i)}\right)^2 \tag{2.2}$$

- The way to choose the parameters is to find

$$\min_{\theta_0, \theta_1} J(\theta_0, \theta_1) \tag{2.3}$$

  Apply *gradient descent method*. Repeat the following until convergence

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \tag{2.4}$$

  where $j = 0, 1$, and α is the learning rate.

- If α is too large, it may fail to converge, or even diverge. As we approach a local minimum, gradient descent will automatically take smaller steps. So, no need to decrease α over time.

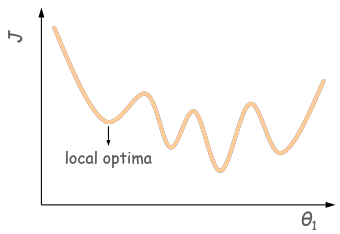

Figure 2.1: Local optima

2.2 Model Representation

- Assume there are m sets of data and each data has n features, then the raw input could be expressed as a $m \times n$ matrix $\mathbf{X}$:

$$\mathbf{X} = \begin{pmatrix} x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(m)} & x_2^{(m)} & \dots & x_n^{(m)} \end{pmatrix}_{m \times n} \tag{2.5}$$

  For polynomial functions, additional features can be created based on x_1,

$$h(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^3 \tag{2.6}$$

  Then, the new features could be assumed as $x_2 = x_1^2$, $x_3 = x_1^3$

- In this course, $\mathbf{X}$ is usually expanded as a $m \times (n+1)$ matrix:

$$\mathbf{X} = \begin{pmatrix} x_0^{(1)} & x_1^{(1)} & \dots & x_n^{(1)} \\ x_0^{(2)} & x_1^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ x_0^{(m)} & x_1^{(m)} & \dots & x_n^{(m)} \end{pmatrix}_{m \times (n+1)} \tag{2.7}$$

  Normally, let $x_0^{(i)} = 1$ so that the first item of the hypothesis would be the constant θ_0 (also called **bias**). The row component $\mathbf{x}^{(i)}$ of $\mathbf{X}$ means the i^{th} set of training examples [1]:

$$\mathbf{x}^{(i)} = \begin{pmatrix} x_0^{(i)} & x_1^{(i)} & \dots & x_n^{(i)} \end{pmatrix} \tag{2.8}$$

  The columm component $\mathbf{x}_j$ of $\mathbf{X}$ means the j^{th} feature of training examples:

$$\mathbf{x}_j = \begin{pmatrix} x_j^{(1)} \\ x_j^{(2)} \\ \vdots \\ x_j^{(n)} \end{pmatrix} \tag{2.9}$$

- The multivariable hypothesis function:

$$\begin{aligned} h^{(i)} &= \theta_0 x_0^{(i)} + \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \dots + \theta_n x_n^{(i)} \\ &= \begin{bmatrix} x_0^{(i)} & x_1^{(i)} & x_2^{(i)} & \dots & x_n^{(i)} \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \\ &= \mathbf{x}^{(i)} \theta \end{aligned} \tag{2.10}$$

[1] In this document, a component is shown in a normal font such as $x_j^{(i)}$, a vector is shown in a bold font with a lowercase letter such as $\mathbf{x}^{(i)}$, a matrix is shown in a bold font with an uppercase letter such as $\mathbf{X}$.

The relation between the hypothesis $h^{(i)}$ (seemed as **output**), the **input** $x_j^{(i)}$ and the **target** $y^{(i)}$ is:

$$\begin{pmatrix} h^{(1)} \\ h^{(2)} \\ \vdots \\ h^{(m)} \end{pmatrix} = \begin{pmatrix} 1 & x_1^{(1)} & \dots & x_n^{(1)} \\ 1 & x_1^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(m)} & \dots & x_n^{(m)} \end{pmatrix} \begin{pmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{pmatrix} \rightarrow \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{pmatrix} \tag{2.11}$$

2.3 Cost Function

- Define the cost function and derive a vectorized implementation

$$\begin{aligned} J(\theta) &= \frac{1}{2m} \sum_{i=1}^{m} \left(h^{(i)} - y^{(i)}\right)^2 \\ &= \frac{1}{2m} \sum_{i=1}^{m} \left(\theta_0 x_0^{(i)} + \theta_1 x_1^{(i)} + \cdots + \theta_n x_n^{(i)} - y^{(i)}\right)^2 \\ &= \frac{1}{2m} \sum_{i=1}^{m} \left(\sum_{j=1}^{n} \theta_j x_j^{(i)} - y^{(i)}\right)^2 \\ &= \frac{1}{2m} \sum_{i=1}^{m} \left(\mathbf{x}^{(i)}\theta - y^{(i)}\right)^2 \\ &= \frac{1}{2m} (\mathbf{X}\theta - \mathbf{y}) \cdot (\mathbf{X}\theta - \mathbf{y}) \end{aligned} \tag{2.12}$$

2.4 Gradient Descent

- Repeat until convergence

$$\begin{aligned} \theta_j &:= \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \\ &:= \theta_j - \frac{\alpha}{2m} \frac{\partial}{\partial \theta_j} \sum_{i=1}^{m} \left(h^{(i)} - y^{(i)}\right)^2 \\ &:= \theta_j - \frac{\alpha}{m} \sum_{i=1}^{m} \left(h^{(i)} - y^{(i)}\right) \left(x_j^{(i)}\right) \end{aligned} \tag{2.13}$$

for $j = 0, 1, \dots, n$

- In vectorized implementation

$$\theta := \theta - \frac{\alpha}{m} \mathbf{X}^T (\mathbf{X}\theta - \mathbf{y}) \tag{2.14}$$

Proof. Let

$$\nabla_j = \frac{1}{m} \sum_{i=1}^{m} (h^{(i)} - y^{(i)}) x_j^{(i)} \tag{2.15}$$

which is equivalent to

$$\nabla_j = \frac{1}{m}(\mathbf{h} - \mathbf{y})^T \mathbf{x}_j \tag{2.16}$$

so

$$\begin{aligned} \begin{bmatrix} \nabla_0 & \nabla_1 & \dots & \nabla_n \end{bmatrix} &= \frac{1}{m}(\mathbf{h} - \mathbf{y})^T \begin{bmatrix} \mathbf{x}_0 & \mathbf{x}_1 & \dots & \mathbf{x}_n \end{bmatrix} \\ &= \frac{1}{m}(\mathbf{h} - \mathbf{y})^T \mathbf{X} \end{aligned} \tag{2.17}$$

thus

$$\begin{bmatrix} \nabla_0 \\ \nabla_1 \\ \vdots \\ \nabla_n \end{bmatrix} = \left(\frac{1}{m}(\mathbf{h} - \mathbf{y})^T \mathbf{X} \right)^T = \frac{1}{m} \mathbf{X}^T (\mathbf{h} - \mathbf{y}) \tag{2.18}$$

□

2.5 Scaling and Normalization

- We can speed up gradient descent by having each of our input values in roughly the same range. Ideally:

$$-1 \leq x_j^{(i)} \leq 1 \tag{2.19}$$

- Two techniques to help with this are **feature scaling** and **mean normalization**.
- **Feature scaling** involves dividing the input values by the range of the input variable, resulting in a new range of just 1.
- **Mean normalization** involves subtracting the average value for that input variable resulting in a new average value of just zero.
- To implement both of these techniques, adjust input values as shown in this formula:

$$x_j^{(i)} := \frac{x_j^{(i)} - \mu^{(i)}}{s^{(i)}} \tag{2.20}$$

where $\mu^{(i)}$ is the average of all the values for feature i and $s^{(i)}$ is the range of values, or $s^{(i)}$ is the standard deviation.

2.6 Normal Equation Method

- This method minimize J by **explicitly** taking its derivatives with respect to the θ_j, and setting them to zero. The normal equation formula is given below

$$\theta = \left(\mathbf{X}^T \mathbf{X}\right)^{-1} \mathbf{X}^T \mathbf{y} \tag{2.21}$$

- If $\left(\mathbf{X}^T \mathbf{X}\right)^{-1}$ is not invertible, the common causes might be having **redundant features** (i.e. there are linearly dependent features), or **too many features** (e.g. $m \leq n$). Using **pinv** function rather than **inv** when implenting in Octave in this case.

- *Proof.* The derivation of equation 2.21 is presented at Eli Bendersky's website:

$$\begin{aligned} J(\theta) &= \frac{1}{2m} \left(\mathbf{X}\theta - \mathbf{y}\right)^T \left(\mathbf{X}\theta - \mathbf{y}\right) \\ &= \frac{1}{2m} \left(\theta^T \mathbf{X}^T - \mathbf{y}^T\right) \left(\mathbf{X}\theta - \mathbf{y}\right) \\ &= \frac{1}{2m} \left(\theta^T \mathbf{X}^T \mathbf{X}\theta - \theta^T \mathbf{X}^T \mathbf{y} - \mathbf{y}^T \mathbf{X}\theta + \mathbf{y}^T \mathbf{y}\right) \\ &= \frac{1}{2m} \left(\theta^T \mathbf{X}^T \mathbf{X}\theta - 2\left(\mathbf{X}\theta\right)^T \mathbf{y} + \mathbf{y}^T \mathbf{y}\right) \end{aligned} \tag{2.22}$$

  Derive by θ and compare to 0, $\frac{\partial J}{\partial \theta} = 0$

$$\begin{aligned} 2\mathbf{X}^T \mathbf{X}\theta - 2\mathbf{X}^T \mathbf{y} &= 0 \\ 2\mathbf{X}^T \mathbf{X}\theta &= 2\mathbf{X}^T \mathbf{y} \\ \theta &= \left(\mathbf{X}^T \mathbf{X}\right)^{-1} \mathbf{X}^T \mathbf{y} \end{aligned} \tag{2.23}$$

  □

- The following is a comparison of the algorithmic efficiency between *gradient descent* and *normal equation*

Table 2.1: Comparison of the algorithmic efficiency

Items	Gradient Descent	Normal Equation
Complexity	$O\left(kn^2\right)$	$O\left(n^3\right)$
Situation	Works well when n is large	Slow if n is very large

Chapter 3

Logistic Regression

3.1 Classification and Hypothesis

- A **sigmoid function** is a mathematical function having a characteristic **s-shaped** curve. A common example of a sigmoid function is the **logistic function** shown in the figure 3.1.

$$g(z) = \frac{1}{1 + e^{-z}} \tag{3.1}$$

- When $z \geq 0$, the output will be greater than or equal 0.5.

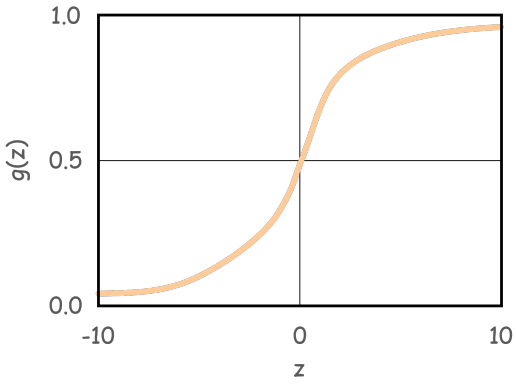

Figure 3.1: Logistic function

- For a **binary classification problem**, $y \in (0, 1)$, change the form of the hypothesis $h^{(i)}$ as a logistic function to satify $0 \leq h^{(i)} \leq 1$. Substitute z into $\mathbf{x}^{(i)}\theta$

$$h^{(i)} = \frac{1}{1 + e^{-\mathbf{x}^{(i)}\theta}} \tag{3.2}$$

where

$$\mathbf{x}^{(i)} = \begin{bmatrix} x_0^{(i)} & x_1^{(i)} & \dots & x_n^{(i)} \end{bmatrix}, \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \tag{3.3}$$

Let

$$\mathbf{h} = \begin{bmatrix} h^{(1)} \\ h^{(2)} \\ \vdots \\ h^{(m)} \end{bmatrix} \tag{3.4}$$

- The hypothesis $h^{(i)}$ will give the **probability**. For example, $h^{(i)} = 0.7$ means the probability of 70% that the output is "1".

$$h^{(i)} = P(y = 1|x; \theta) = 1 - P(y = 0|x; \theta) \tag{3.5}$$

3.2 Cost Function

- If the cost function of a logistic function is used the same as for linear regression, it will cause many local optima. (Not a **convex** function) Instead, the cost function for logistic regression looks like:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} L(h^{(i)}, y^{(i)}) \tag{3.6}$$

where,

$$L(h^{(i)}, y^{(i)}) = \begin{cases} -\log(h^{(i)}) & \text{if } y^{(i)} = 1 \\ -\log(1 - h^{(i)}) & \text{if } y^{(i)} = 0 \end{cases} \tag{3.7}$$

These two cases are shown as Figure 3.2.

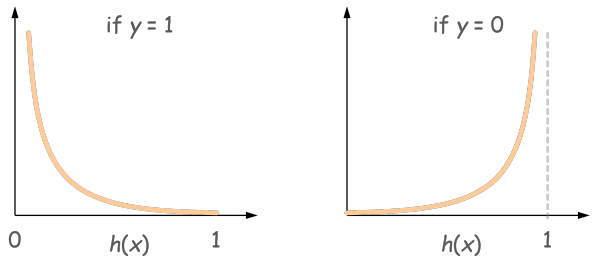

Figure 3.2: Cost of logistic function

The function L can be compressed into one case

$$L(h^{(i)}, y^{(i)}) = -y^{(i)} \log(h^{(i)}) - (1 - y^{(i)}) \log(1 - h^{(i)}) \tag{3.8}$$

Thus, the entire cost function as follows:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log(h^{(i)}) + (1 - y^{(i)}) \log(1 - h^{(i)}) \right] \tag{3.9}$$

A vectorized implementation is:

$$J(\theta) = -\frac{1}{m} \left(\mathbf{y}^T \log(\mathbf{h}) + (\mathbf{1} - \mathbf{y})^T \log(\mathbf{1} - \mathbf{h}) \right) \tag{3.10}$$

3.3 Gradient Descent

- Repeat until convergence

$$\begin{aligned}\theta_j &:= \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \\ &:= \theta_j - \frac{\alpha}{m} \sum_{i=1}^{m} \left[\left(h^{(i)} - y^{(i)} \right) x_j^{(i)} \right]\end{aligned} \tag{3.11}$$

  for $j = 0, 1, \ldots, n$.

- Surprisingly, the update rule is the same as the one derived in linear regression. As a result, the same gradient descent formula can be used for logistic regression as well.

- *Proof.* The derivative for the cost function is

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{-1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial h^{(i)}} L\left(h^{(i)}, y^{(i)} \right) \frac{\partial h^{(i)}}{\partial \theta_j} \tag{3.12}$$

  Compute the first derivative term, consider the base number of logarithm is e

$$\begin{aligned}\frac{\partial}{\partial h^{(i)}} L\left(h^{(i)}, y^{(i)} \right) &= y^{(i)} \frac{\partial}{\partial h^{(i)}} \log\left(h^{(i)} \right) + \left(1 - y^{(i)} \right) \frac{\partial}{\partial h^{(i)}} \log\left(1 - h^{(i)} \right) \\ &= \frac{y^{(i)}}{h^{(i)}} - \frac{1 - y^{(i)}}{1 - h^{(i)}} \\ &= -\frac{h^{(i)} - y^{(i)}}{h^{(i)} \left(1 - h^{(i)} \right)}\end{aligned} \tag{3.13}$$

  And the second derivative term,

$$\frac{\partial h^{(i)}}{\partial \theta_j} = \frac{\partial}{\partial \theta_j} \frac{1}{1 + e^{-\mathbf{x}^{(i)} \theta}} \tag{3.14}$$

  Let

$$\begin{aligned}\alpha &= 1 + e^{-\mathbf{x}^{(i)} \theta} \\ \beta &= -\mathbf{x}^{(i)} \theta\end{aligned} \tag{3.15}$$

  Then

$$\begin{aligned}\frac{\partial h^{(i)}}{\partial \theta_j} &= \frac{\partial h^{(i)}}{\partial \alpha} \frac{\partial \alpha}{\partial \beta} \frac{\partial \beta}{\partial \theta_j} \\ &= \frac{-1}{\left(1 + e^{-\mathbf{x}^{(i)} \theta} \right)^2} e^{-\mathbf{x}^{(i)} \theta} \left(-x_j^{(i)} \right) \\ &= \frac{1}{1 + e^{-\mathbf{x}^{(i)} \theta}} \left(\frac{-e^{-\mathbf{x}^{(i)} \theta}}{1 + e^{-\mathbf{x}^{(i)} \theta}} \right) \left(-x_j^{(i)} \right) \\ &= h^{(i)} \left(1 - h^{(i)} \right) \left(-x_j^{(i)} \right)\end{aligned} \tag{3.16}$$

So

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{-1}{m} \sum_{i=1}^{m} \left[\left(h^{(i)} - y^{(i)} \right) x_j^{(i)} \right] \tag{3.17}$$

□

- The vectorized implementation is

$$\theta := \theta - \frac{\alpha}{m} \mathbf{X}^T \left(\mathbf{h} - \mathbf{y} \right) \tag{3.18}$$

3.4 Decision Boundary

- After the optimization of parameters θ_j where $j = 0, \ldots, n$, the decision boundary is

$$\theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_n x_n = 0 \tag{3.19}$$

- For example. If

$$\theta = \begin{bmatrix} -3 & 1 & 1 \end{bmatrix}^T \tag{3.20}$$

The decision boundary is

$$-3x_0 + x_1 + x_2 = 0 \tag{3.21}$$

For those points located at the upper-right side of the decision boundary, the predictions are $y = 1$. On the contrary, for those points located at the lower-left side are $y = 0$.

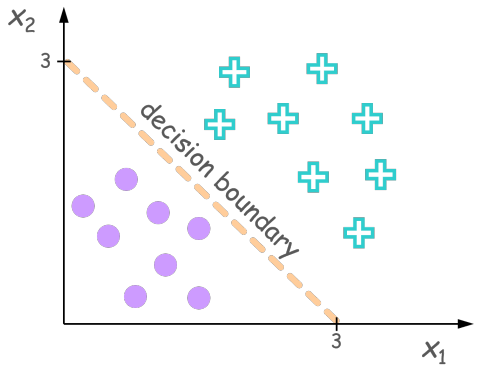

Figure 3.3: Logistic function

- An non-linear decision boundary can also be created by adding polynomial terms in the array of parameters, such as

$$\theta = \begin{bmatrix} \theta_0 & \theta_1 & \theta_2 & \theta_1^2 & \theta_2^2 \end{bmatrix}^T \tag{3.22}$$

For this case, the optimized parameters would be like

$$\theta = \begin{bmatrix} -1 & 0 & 0 & 1 & 1 \end{bmatrix}^T \tag{3.23}$$

Then the decision boundary is

$$-1 + x_1^2 + x_2^2 = 0 \tag{3.24}$$

The prediction would be

$$y = \begin{cases} 1, \text{ for } x_1^2 + x_2^2 \geq 1 \\ 0, \text{ for } x_1^2 + x_2^2 < 1 \end{cases} \tag{3.25}$$

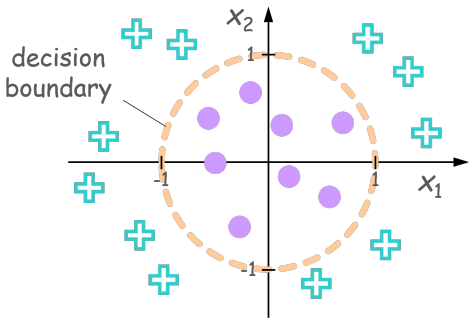

Figure 3.4: Logistic function

Chapter 4

Regularization

4.1 Overfitting

- The hypothesis in Figure 4.1 are:
  - $h(x) = \theta_0 + \theta_1 x \Rightarrow$ underfitting
  - $h(x) = \theta_0 + \theta_1 x + \theta_2 x^2$
  - $h(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4 \Rightarrow$ overfitting

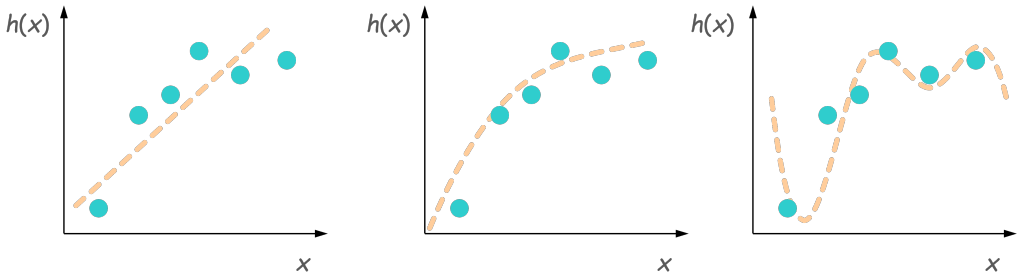

Figure 4.1: Three different regression models

Two main options to address these issue, **reducing features** and **regularization**.

- **Reduce the number of features**
  - Manually select
  - Use a model selection algorithm
- **Regularization**
  - Keep all the features, but reduce the magnitude of parameters θ_j
  - Works well when slightly features are useful.

4.2 Regularized Linear Regression

- The intuition of regularization is shown as 4.2

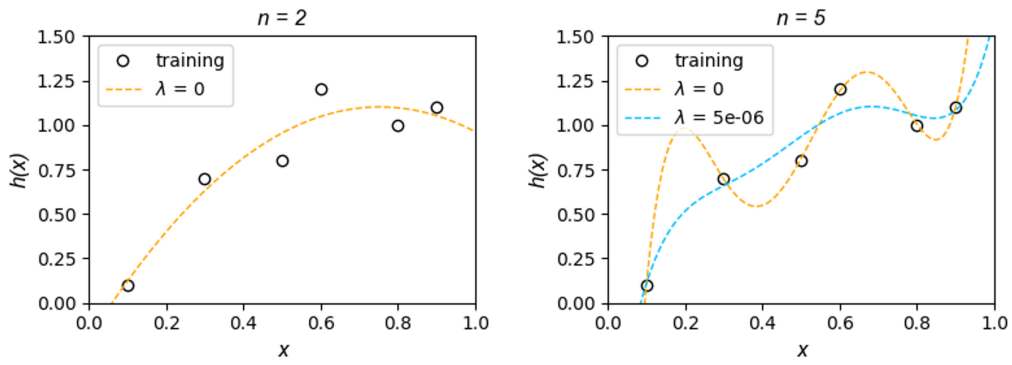

Figure 4.2: The intuition of regularization

- A regularization term (or regularizer) $\sum_{j=1}^{n} \theta_j^2$ is added to the cost function to impose a penalty on the complexity of $h^{(i)}$.

$$J(\theta) = \frac{1}{2m} \left[\sum_{i=1}^{m} (h^{(i)} - y^{(i)})^2 + \lambda \sum_{j=1}^{n} \theta_j^2 \right] \tag{4.1}$$

  The λ is the regularization parameter which controls the importance of the regularization term. To reduce the function variety, only the higher-order terms would be penalized, so θ_0 is excluded from the regularization term.

- The way to choose the parameters is to find

$$\min_{\theta} J(\theta) \tag{4.2}$$

  Because the regularization term is add to the cost function, the minimization process would not find out the optimum which make $h^{(i)}$ closest to $y^{(i)}$. The errors between each $h^{(i)}$ and $y^{(i)}$ would be bigger, so the shape of function would not be too *overfitting*.

- To prevent penalize θ_0, it is separated out from the rest of parameters of **gradient descent** function.

$$\begin{cases} \theta_0 := \theta_0 - \dfrac{\alpha}{m} \displaystyle\sum_{i=1}^{m} \left(h^{(i)} - y^{(i)}\right) x_0^{(i)} \\ \theta_j := \theta_j - \dfrac{\alpha}{m} \displaystyle\sum_{i=1}^{m} \left[\left(h^{(i)} - y^{(i)}\right) x_j^{(i)} + \lambda \theta_j \right] \end{cases} \tag{4.3}$$

  for $j = 1, \ldots, n$ and $j \neq 0$. The vectorized implementation would be

$$\theta := \theta - \frac{\alpha}{m} \left[\mathbf{X}^T (\mathbf{h} - \mathbf{y}) + \lambda \mathbf{L} \theta \right] \tag{4.4}$$

  where

$$\mathbf{L} = \begin{bmatrix} 0 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{bmatrix} \tag{4.5}$$

- And the regularized **normal equation** becomes

$$\theta = \left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{L}\right)^{-1}\mathbf{X}^T\mathbf{y} \tag{4.6}$$

Recall that if $m < n$, then $\mathbf{X}^T\mathbf{X}$ is non-invertible. However, when we add the term $\lambda\mathbf{L}$, then $\mathbf{X}^T\mathbf{X} + \lambda\mathbf{L}$ becomes invertible.

Proof.

$$\begin{aligned} J(\theta) &= \frac{1}{2m}\left[(\mathbf{X}\theta - \mathbf{y})^T(\mathbf{X}\theta - \mathbf{y}) + \lambda\theta^T\mathbf{L}\theta\right] \\ &= \frac{1}{2m}\left[\theta^T\mathbf{X}^T\mathbf{X}\theta - 2(\mathbf{X}\theta)^T\mathbf{y} + \mathbf{y}^T\mathbf{y} + \lambda\theta^T\mathbf{L}\theta\right] \end{aligned} \tag{4.7}$$

Derive by θ and compare to 0, $\frac{\partial J}{\partial\theta} = 0$

$$\begin{aligned} 2\mathbf{X}^T\mathbf{X}\theta - 2\mathbf{X}^T\mathbf{y} + 2\lambda\mathbf{L}\theta &= 0 \\ (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{L})\theta &= \mathbf{X}^T\mathbf{y} \\ \theta &= \left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{L}\right)^{-1}\mathbf{X}^T\mathbf{y} \end{aligned} \tag{4.8}$$

□

4.3 Regularized Logistic Regression

- The way to regularize a logistic regression is similar to linear regression.

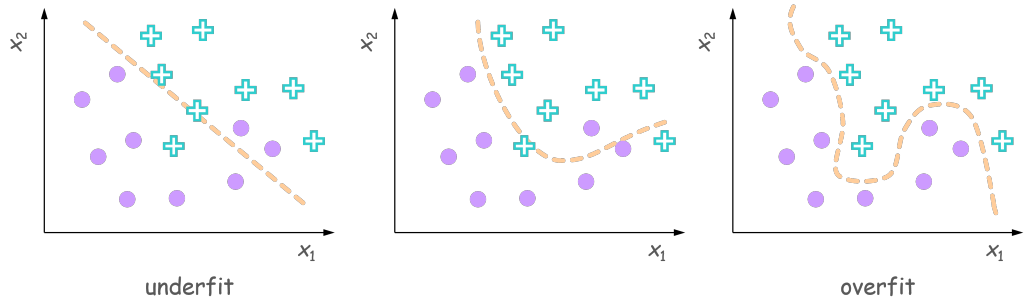

Figure 4.3: Three different regression models

- Define the cost function

$$J(\theta) = \frac{-1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(h^{(i)}\right) + \left(1 - y^{(i)}\right)\log\left(1 - h^{(i)}\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2 \tag{4.9}$$

The regularization term is added and it is the same as which in (4.1).

- Gradient descent

$$\begin{cases} \theta_0 \coloneqq \theta_0 - \dfrac{\alpha}{m} \displaystyle\sum_{i=1}^{m} \left(h^{(i)} - y^{(i)}\right) x_0^{(i)} \\ \theta_j \coloneqq \theta_j - \dfrac{\alpha}{m} \displaystyle\sum_{i=1}^{m} \left[\left(h^{(i)} - y^{(i)}\right) x_j^{(i)} + \lambda \theta_j\right] \end{cases} \tag{4.10}$$

for $j = 1, \ldots, n$ and $j \neq 0$. The vectorized implementation would be

$$\theta \coloneqq \theta - \frac{\alpha}{m} \left[\mathbf{X}^T \left(\mathbf{h} - \mathbf{y}\right) + \lambda \mathbf{L} \theta\right] \tag{4.11}$$

Chapter 5

Neural Networks

5.1 Model Representation

- Neural networks were developed as simulating neurons or networks of neurons in the brain.
- Neurons are basically computational units that take inputs (dendrites) as electrical inputs (called "spikes") that are channeled to outputs (axons).

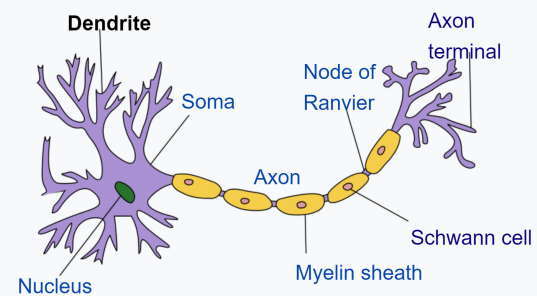

Figure 5.1: Neuron in the brain

- Visually, a simple forward propagation looks like:

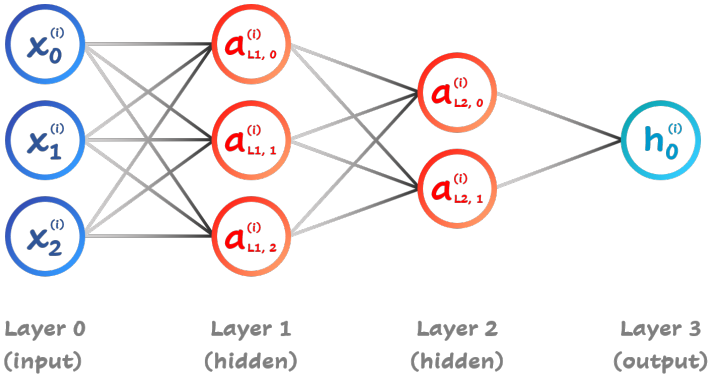

Figure 5.2: Neural networks architecture

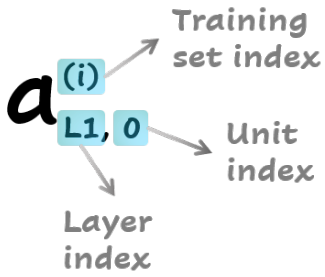

Figure 5.3: The convention of the superscript and subscript in neural networks

- Neural networks can also do multiclass classification

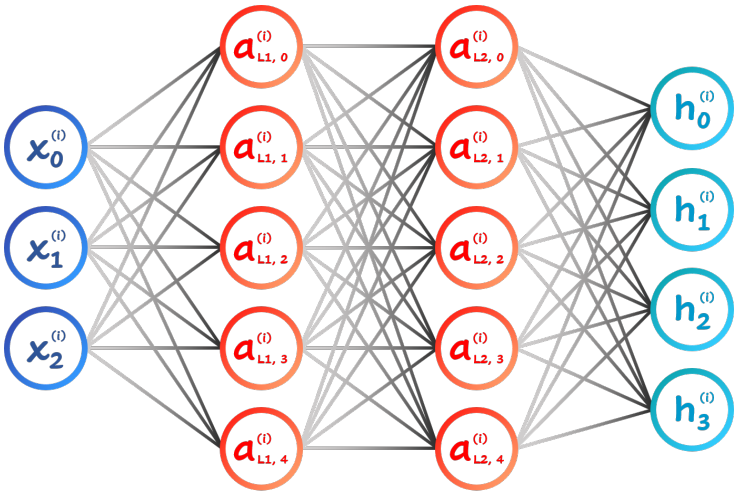

Figure 5.4: neural network for multi-class classification

$$\mathbf{h} \approx \begin{cases} \begin{bmatrix}1 & 0 & 0 & 0\end{bmatrix}^T, \text{for pedestrian} \\ \begin{bmatrix}0 & 1 & 0 & 0\end{bmatrix}^T, \text{for car} \\ \begin{bmatrix}0 & 0 & 1 & 0\end{bmatrix}^T, \text{for motocycle} \\ \begin{bmatrix}0 & 0 & 0 & 1\end{bmatrix}^T, \text{for truck} \end{cases} \tag{5.1}$$

- In neural networks, the logistic function is sometimes called a **activation** function. In this situation, parameters are sometimes called **weights**. The values for each of the activation nodes is obtained as follows:

$$\begin{aligned} a_{L1,0} &= g(\theta_{L1,00}x_0 + \theta_{L1,01}x_1 + \theta_{L1,02}x_2 + \theta_{L1,03}x_3) &&= g(z_{L1,0}) \\ a_{L1,1} &= g(\theta_{L1,10}x_0 + \theta_{L1,11}x_1 + \theta_{L1,12}x_2 + \theta_{L1,13}x_3) &&= g(z_{L1,0}) \\ a_{L1,2} &= g(\theta_{L1,20}x_0 + \theta_{L1,21}x_1 + \theta_{L1,22}x_2 + \theta_{L1,23}x_3) &&= g(z_{L1,0}) \end{aligned} \tag{5.2}$$

  Where $a_{Lj,i}$ means the activation of **unit** i in layer j, and Θ_{Lj} means a mapping matrix of weights from layer $j-1$ to layer j. If there are s_j units in layer j and s_{j-1} units in layer $j-1$,

then.

$$\Theta_{Lj} = \begin{bmatrix} \theta_{Lj,00} & \theta_{Lj,01} & \cdots & \theta_{Lj,0s_{j-1}} \\ \theta_{Lj,10} & \theta_{Lj,11} & \cdots & \theta_{Lj,1s_{j-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \theta_{Lj,s_j0} & \theta_{Lj,s_j1} & \cdots & \theta_{Lj,s_js_{j-1}} \end{bmatrix}_{(s_j+1)\times(s_{j-1}+1)} \tag{5.3}$$

5.2 Simplified Expressions

- The neural network looks too fancy. Let's try to make it simpler but more boring. The network could be expressed like:

$$\begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ x_2^{(i)} \end{bmatrix}_{3\times1} \xrightarrow{[\Theta_{L1}]_{5\times3}} \begin{bmatrix} a_{L1,0}^{(i)} \\ a_{L1,1}^{(i)} \\ a_{L1,2}^{(i)} \\ a_{L1,3}^{(i)} \\ a_{L1,4}^{(i)} \end{bmatrix}_{5\times1} \xrightarrow{[\Theta_{L2}]_{5\times5}} \begin{bmatrix} a_{L2,0}^{(i)} \\ a_{L2,1}^{(i)} \\ a_{L2,2}^{(i)} \\ a_{L2,3}^{(i)} \\ a_{L2,4}^{(i)} \end{bmatrix}_{5\times1} \xrightarrow{[\Theta_{L3}]_{4\times5}} \begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \end{bmatrix}_{4\times1} \tag{5.4}$$

- Or in a vectorized form:

$$\mathbf{x}^{(i)} \xrightarrow{\Theta_{L1}} \mathbf{a}_{L1}^{(i)} \xrightarrow{\Theta_{L2}} \mathbf{a}_{L2}^{(i)} \xrightarrow{\Theta_{L3}} \mathbf{h}^{(i)} \tag{5.5}$$

- Actually, only three linear algebra equations are needed for this neural network

$$\begin{aligned} \mathbf{a}_{L1}^{(i)} &= g(\Theta_{L1} \times \mathbf{x}^{(i)}) \\ \mathbf{a}_{L2}^{(i)} &= g(\Theta_{L2} \times \mathbf{a}_{L1}^{(i)}) \\ \mathbf{h}^{(i)} &= g(\Theta_{L3} \times \mathbf{a}_{L2}^{(i)}) \end{aligned} \tag{5.6}$$

5.3 Cost Function

- The cost function of a neural network with m data sets, K classifications and L layers

$$J(\theta) = \frac{-1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} \left[y_k^{(i)} \log\left(h_k^{(i)}\right) + \left(1 - y_k^{(i)}\right) \log\left(1 - h_k^{(i)}\right) \right] + \frac{\lambda}{2m} \sum_{l=1}^{L} ||\Theta_{Ll}||_2^2 \tag{5.7}$$

  Where $||\Theta_{Ll}||_2$ means Frobenius norm of the matrix Θ_{Ll}

5.4 Backpropagation Algorithm

- Consider a simple neural network with only one data set

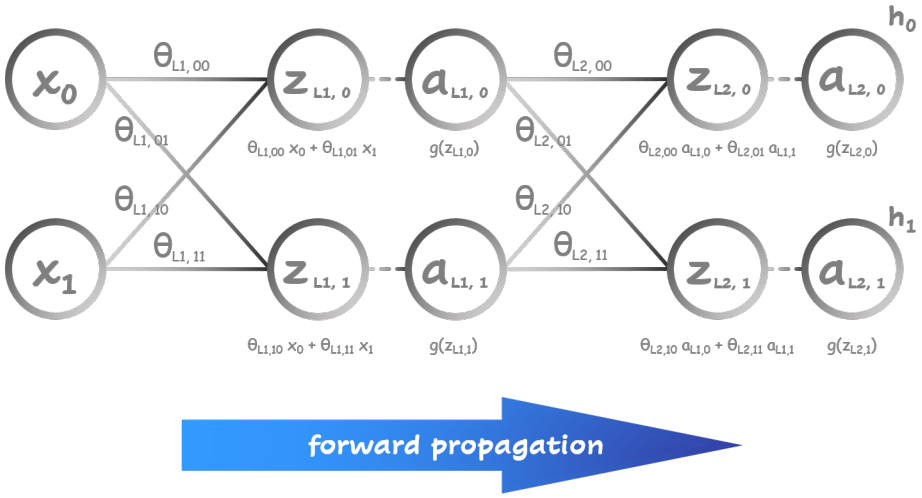

Figure 5.5: A simple neural network

The cost function would be

$$J(\theta) = [y_0 \log(h_0) + (1 - y_0) \log(1 - h_0) + y_1 \log(h_1) + (1 - y_1) \log(1 - h_1)] \tag{5.8}$$

And the gradient descent iteration would be:

$$\theta_{l,ij} := \theta_{l,ij} - \alpha \frac{\partial J(\theta)}{\partial \theta_{l,ij}} \tag{5.9}$$

To compute the gradient descent for $\theta_{l,ij}$, it's needed to find out the regulation of $\frac{\partial J}{\partial \theta_{l,ij}}$. First, try to do the partial derivatives at the second layer.

$$\frac{\partial J}{\partial \theta_{L2,00}} = \frac{\partial J}{\partial a_{L2,0}} \frac{\partial a_{L2,0}}{\partial z_{L2,0}} \frac{\partial z_{L2,0}}{\partial \theta_{L2,00}} = \frac{\partial J}{\partial a_{L2,0}} g'(z_{L2,0}) a_{L1,0} \tag{5.10}$$

$$\frac{\partial J}{\partial \theta_{L2,01}} = \frac{\partial J}{\partial a_{L2,0}} \frac{\partial a_{L2,0}}{\partial z_{L2,0}} \frac{\partial z_{L2,0}}{\partial \theta_{L2,01}} = \frac{\partial J}{\partial a_{L2,0}} g'(z_{L2,0}) a_{L1,1} \tag{5.11}$$

$$\frac{\partial J}{\partial \theta_{L2,10}} = \frac{\partial J}{\partial a_{L2,1}} \frac{\partial a_{L2,1}}{\partial z_{L2,1}} \frac{\partial z_{L2,1}}{\partial \theta_{L2,10}} = \frac{\partial J}{\partial a_{L2,1}} g'(z_{L2,1}) a_{L1,0} \tag{5.12}$$

$$\frac{\partial J}{\partial \theta_{L2,11}} = \frac{\partial J}{\partial a_{L2,1}} \frac{\partial a_{L2,1}}{\partial z_{L2,1}} \frac{\partial z_{L2,1}}{\partial \theta_{L2,11}} = \frac{\partial J}{\partial a_{L2,1}} g'(z_{L2,1}) a_{L1,1} \tag{5.13}$$

Then, try to do the partial derivatives at the first layer.

$$\begin{aligned} \frac{\partial J}{\partial \theta_{L1,00}} &= \left(\frac{\partial J}{\partial a_{L2,0}} \frac{\partial a_{L2,0}}{\partial z_{L2,0}} \frac{\partial z_{L2,0}}{\partial a_{L1,0}} + \frac{\partial J}{\partial a_{L2,1}} \frac{\partial a_{L2,1}}{\partial z_{L2,1}} \frac{\partial z_{L2,1}}{\partial a_{L1,0}} \right) \frac{\partial a_{L1,0}}{\partial z_{L1,0}} \frac{\partial z_{L1,0}}{\partial \theta_{L1,00}} \\ &= \left(\frac{\partial J}{\partial a_{L2,0}} g'(z_{L2,0}) \theta_{L2,00} + \frac{\partial J}{\partial a_{L2,1}} g'(z_{L2,1}) \theta_{L2,10} \right) g'(z_{L1,0}) x_0 \\ &= \frac{\partial J}{\partial a_{L1,0}} g'(z_{L1,0}) x_0 \end{aligned} \tag{5.14}$$

So the others would be like

$$\frac{\partial J}{\partial \theta_{L1,01}} = \frac{\partial J}{\partial a_{L1,0}} g'(z_{L1,0}) x_1 \tag{5.15}$$

$$\frac{\partial J}{\partial \theta_{L1,10}} = \frac{\partial J}{\partial a_{L1,1}} g'(z_{L1,1}) x_0 \tag{5.16}$$

$$\frac{\partial J}{\partial \theta_{L1,11}} = \frac{\partial J}{\partial a_{L1,1}} g'(z_{L1,1}) x_1 \tag{5.17}$$

It can be seen that $\frac{\partial J}{\partial a_{l,i}}$ are necessary for $\frac{\partial J}{\partial \theta_{l,ij}}$. However, if using a forward propagation to do the gradient descent process, it would be unavailable for $\frac{\partial J}{\partial a_{l,i}}$ at a previous layer because of the dependency in a neural network. To solve this inconvenience, backpropagation should be adopted.

That is to say the training processes of a neural network containing one forward process to get the values of each activations ($a_{l,i}$) and one backpropagation process to get the values of derivatives corresponded to each activations ($\frac{\partial J}{\partial a_{L,i}}$). Then the values derived from the backpropagation are necessary for the gradient descent process.

Observe equation 5.14, there is a regulation of $\frac{\partial J}{\partial a_{l,i}}$ can be found. The regulation can be seen as actions of the backpropagation process in Figure 5.4.

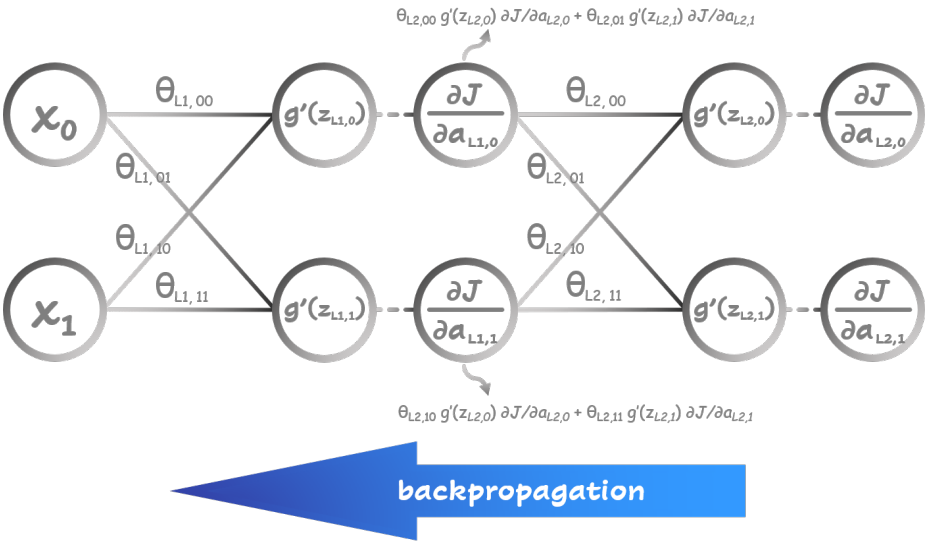

Figure 5.6: The backpropagation in a neural network

Once the backpropagation process was completed, the partial derivatives of each weight ($\frac{\partial J}{\partial \theta_{l,ij}}$) can computed which is shown as figure 5.4 and equation 5.18

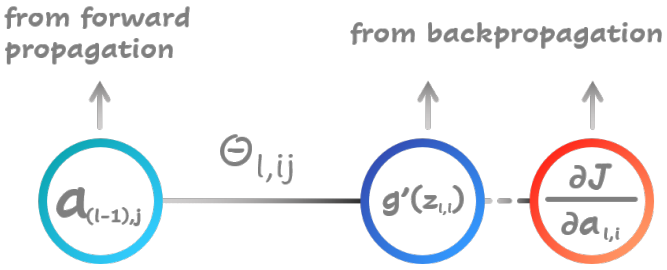

Figure 5.7: The backpropagation in a neural network

$$\frac{\partial J}{\partial \theta_{l,ij}} = \frac{\partial J}{\partial a_{l,i}} g'(z_{l,i}) a_{(l-1),j} \tag{5.18}$$

5.5 Non-linear Classification Example

- Logical functions in figure 5.5 would be examples to show how neural networks really work.

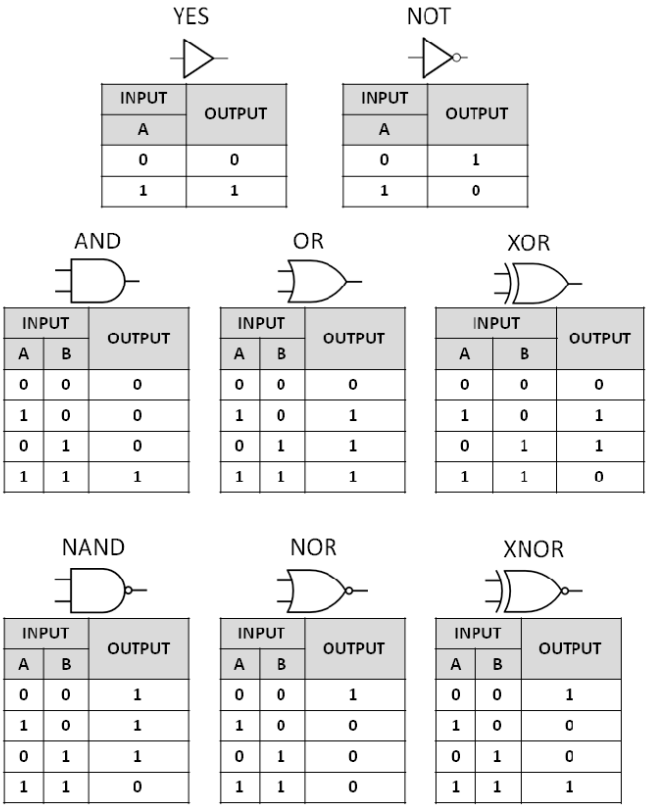

Figure 5.8: The common Boolean logic gates with symbols and truth tables

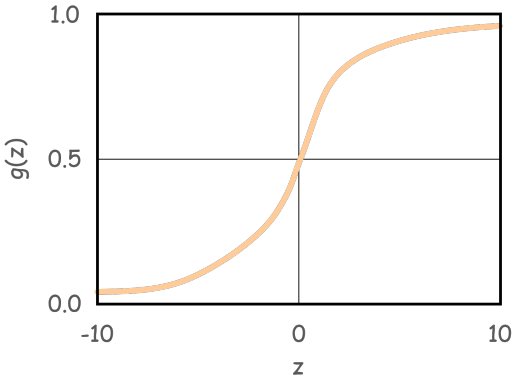

Figure 5.9: Logistic function

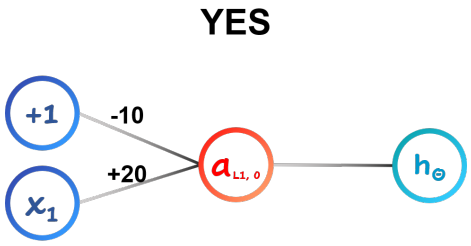

Figure 5.10: neural network of logical function YES

Table 5.1: YES calculation

Inputs		Activations	Outputs
x_0	x_1	$a_0^{(1)}$	h
1	0	$g(-10) \approx 0$	0
1	1	$g(+10) \approx 1$	1

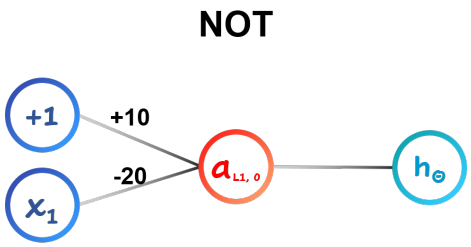

Figure 5.11: neural network of logical function NOT

Table 5.2: NOT calculation

Inputs		Activations	Outputs
x_0	x_1	$a_0^{(1)}$	h
1	0	$g(+10) \approx 1$	1
1	1	$g(-10) \approx 0$	0

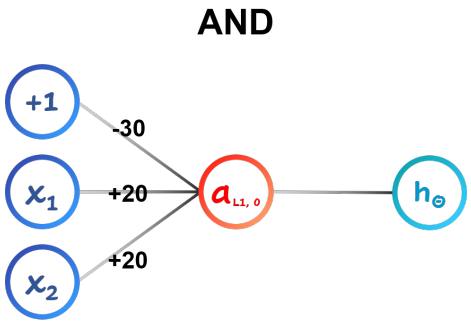

Figure 5.12: neural network of logical function AND

Table 5.3: AND calculation

Inputs			Activations	Outputs
x_0	x_1	x_2	$a_0^{(1)}$	h
1	0	0	$g(-30) \approx 0$	0
1	0	1	$g(-10) \approx 0$	0
1	1	0	$g(-10) \approx 0$	0
1	1	1	$g(+10) \approx 1$	1

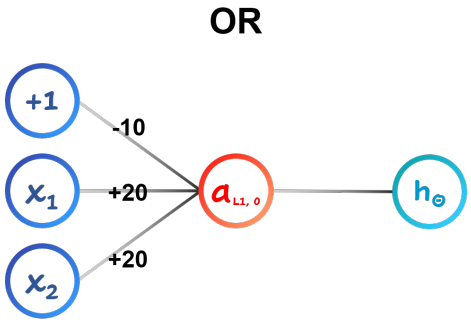

Figure 5.13: neural network of logical function OR

Table 5.4: OR calculation

Inputs			Activations	Outputs
x_0	x_1	x_2	$a_0^{(1)}$	h
1	0	0	$g(-10) \approx 0$	0
1	0	1	$g(+10) \approx 1$	1
1	1	0	$g(+10) \approx 1$	1
1	1	1	$g(+30) \approx 1$	1

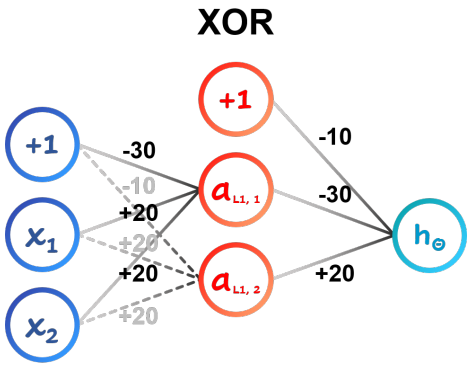

Figure 5.14: neural network of logical function XOR

Table 5.5: XOR calculation

Inputs			Activations						Outputs
x_0	x_1	x_2	$a_0^{(1)}$	$a_1^{(1)}$	$a_2^{(1)}$				h
1	0	0	1	$g(-30)$	$g(-10)$	$\approx$	1	0	0	$g(-10) \approx 0$
1	0	1	1	$g(-10)$	$g(+10)$	$\approx$	1	0	1	$g(+10) \approx 1$
1	1	0	1	$g(-10)$	$g(+10)$	$\approx$	1	0	1	$g(+10) \approx 1$
1	1	1	1	$g(+10)$	$g(+30)$	$\approx$	1	1	1	$g(-10) \approx 0$

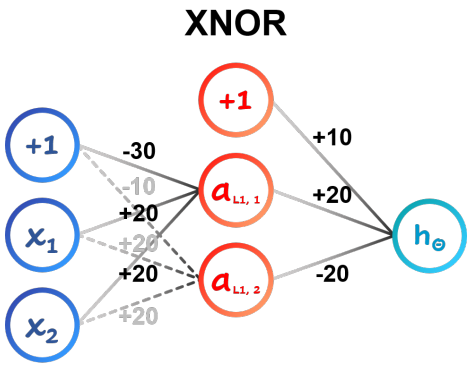

Figure 5.15: neural network of logical function XNOR

Table 5.6: XNOR calculation

Inputs			Activations						Outputs
x_0	x_1	x_2		$a_0^{(1)}$	$a_1^{(1)}$	$a_2^{(1)}$			h
1	0	0	$\begin{bmatrix} 1 & g(-30) & g(-10) \end{bmatrix}$				$\approx$	$\begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$	$g(+10) \approx 1$
1	0	1	$\begin{bmatrix} 1 & g(-10) & g(+10) \end{bmatrix}$				$\approx$	$\begin{bmatrix} 1 & 0 & 1 \end{bmatrix}$	$g(-10) \approx 0$
1	1	0	$\begin{bmatrix} 1 & g(-10) & g(+10) \end{bmatrix}$				$\approx$	$\begin{bmatrix} 1 & 0 & 1 \end{bmatrix}$	$g(-10) \approx 0$
1	1	1	$\begin{bmatrix} 1 & g(+10) & g(+30) \end{bmatrix}$				$\approx$	$\begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$	$g(+10) \approx 1$

Chapter 6

Machine Learning Diagnostic

6.1 Splitting dataset

- To implement a diagonostic test, data sets can be divided into three parts: **training set** 60%, **cross validation set** 20% and **test set** 20%.
  - **training set**: to train the different candidate models during the learning process
  - **validation set**: to compare performances of these models and decide which one to take, that is to tune the hyperparameters
  - **test set**: only to assess the performance such as accuracy, sensitivity, specificity, F-measure, and so on
- The cost functions of each sets would be J_{train}, J_{cv}, J_{test}

6.2 Model selection

- Diagnose degree of polynomial

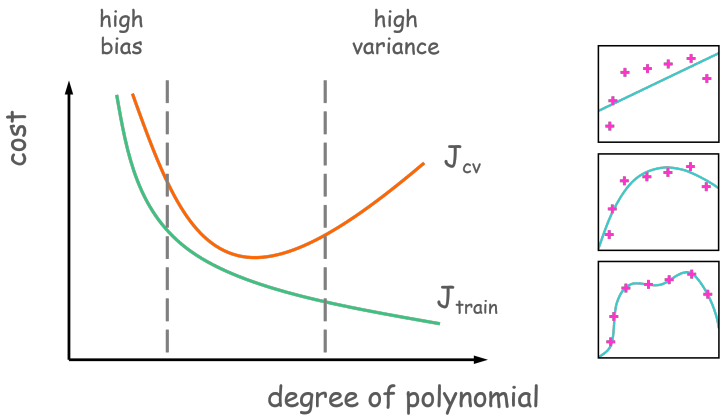

Figure 6.1: Diagnose degree of polynomial d

- Diagnose regularization parameter

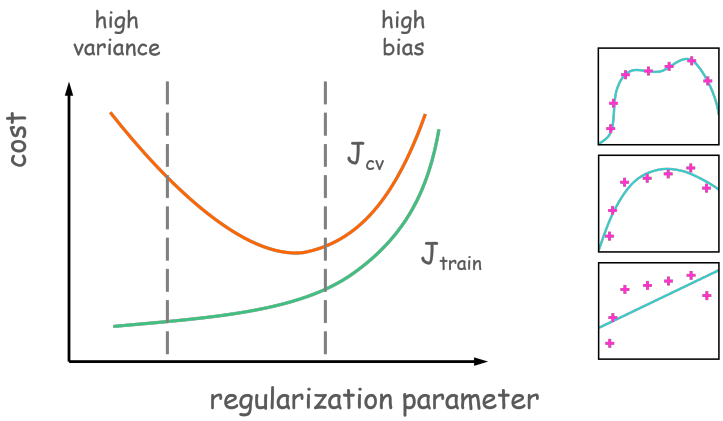

Figure 6.2: Diagnose regularization parameter λ

- Diagnose training set size

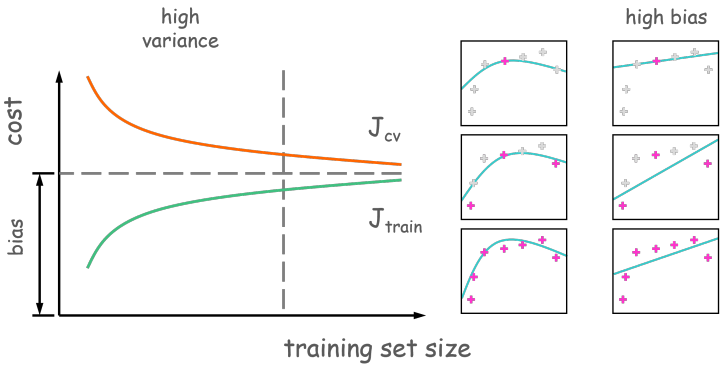

Figure 6.3: Diagnose training set size m

6.3 Error analysis

- Define the error metrics

Table 6.1: Error metrics

Cases	Actual 1	Actual 0
Predict 1	True positive	False positive
Predict 0	False negative	True negative

- Define precision P and recall R

$$\begin{aligned} P &= \frac{\text{True pos.}}{\text{True pos.} + \text{False pos.}} \\ R &= \frac{\text{True pos.}}{\text{True pos.} + \text{False neg.}} \end{aligned} \tag{6.1}$$

We want both precision P and recall R could approximate 1, but there is a trade-off between these two values. For a higher threshold, the precision P would increase but the recall R would decrease

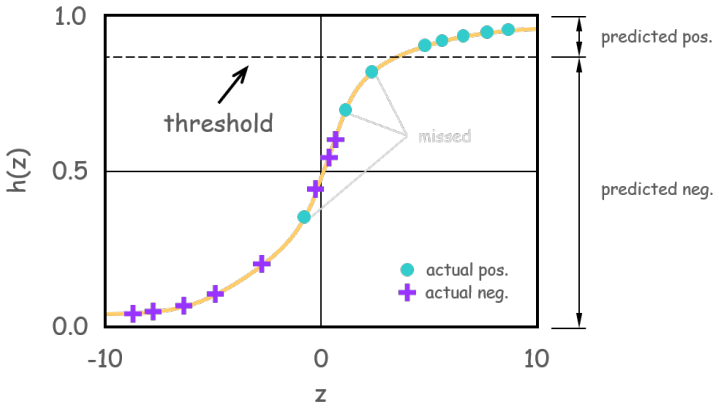

Figure 6.4: High precision low recall

Table 6.2: Higher threshold

Cases	Before (threshold $= 0.5$)	After (threshold > 0.5)
Precision	$\frac{7}{7+2} = 0.778$	$\frac{5}{5+0} = 1.000$
Recall	$\frac{7}{7+1} = 0.875$	$\frac{5}{5+3} = 0.625$

For a lower threshold, the recall R would increase but the precision P would decrease

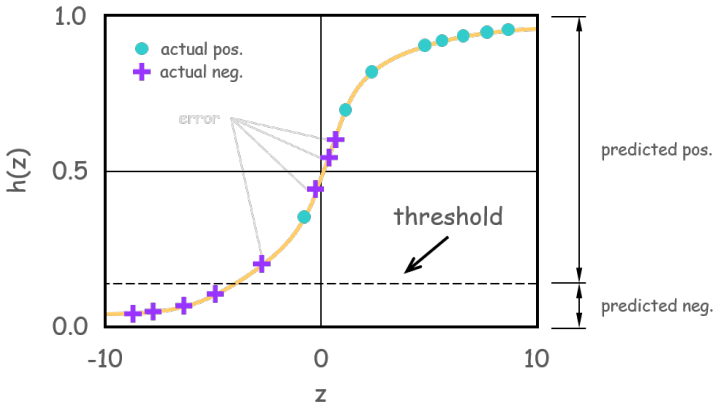

Figure 6.5: Low precision high recall

Table 6.3: Lower threshold

Cases	Before (threshold $= 0.5$)	After (threshold < 0.5)
Precision	$\frac{7}{7+2} = 0.778$	$\frac{8}{8+4} = 0.667$
Recall	$\frac{7}{7+1} = 0.875$	$\frac{8}{8+0} = 1.000$

- The F-score or F-measure is a measure of a test's accuracy. It is calculated from the precision and recall of the test,

$$F_1 = 2\frac{PR}{P+R} \tag{6.2}$$

  This is better than using an average equation like $\frac{P+R}{2}$.

Chapter 7

Support Vector Machine

7.1 Introduction

- The cost of the i^{th} item in the logistic function is

$$J^{(i)} = y^{(i)} \log h^{(i)} + (1 - y^{(i)}) \log (1 - h^{(i)}) \tag{7.1}$$

where

$$h^{(i)} = \frac{1}{1 + e^{-\mathbf{z}}} \tag{7.2}$$

- Plot the cost functions as $y = 1$ and $y = 0$. SVM use the approximated costs named as L_1 and L_0.

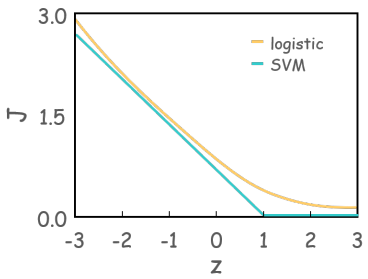

Figure 7.1: The cost function as $y = 1$

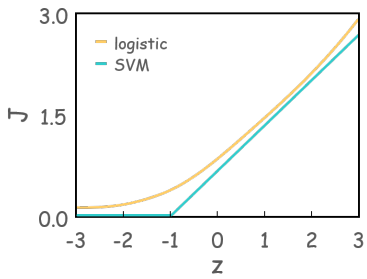

Figure 7.2: The cost function as $y = 0$

7.2 The optimization of the cost function

- The object of the optimization is

$$\min_{\theta} \frac{1}{m} \sum_{i=0}^{m} \left[y^{(i)} L_1(z^{(i)}) + (1 - y^{(i)}) L_0(z^{(i)}) \right] + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2 \tag{7.3}$$

As $z^{(i)} = \mathbf{x}^{(i)}\theta$. When $z^{(i)} \geq 1$ or $z^{(i)} \leq -1$, the first term in the cost is zero, only the regularization term will remain. As a result, the total cost would only remain

$$\min_{\theta} \sum_{j=1}^{n} \theta_j^2 \tag{7.4}$$

And that is to say

$$\min_{\theta} \sum_{j=1}^{n} \theta_j^2 = \min_{\theta} \|\theta\|^2 = \min_{\theta} \|\theta\| \tag{7.5}$$

- Observe $\mathbf{x}^{(i)}\theta$. This can be seen as an inner product of two vectors θ and $\mathbf{x}^{(i)}$. Where

$$\theta = \begin{bmatrix} \theta_0 & \theta_1 & \dots & \theta_n \end{bmatrix}^T \tag{7.6}$$

$$\mathbf{x}^{(i)} = \begin{bmatrix} x_0^{(i)} & x_1^{(i)} & \dots & x_n^{(i)} \end{bmatrix} \tag{7.7}$$

Assum the angle between θ and $\mathbf{x}^{(i)}$ is $\varphi^{(i)}$. That is to say

$$\begin{aligned} \mathbf{x}^{(i)}\theta &= \left\|\mathbf{x}^{(i)}\right\| \|\theta\| \cos\varphi^{(i)} \\ &= p^{(i)} \|\theta\| \operatorname{sign} \varphi^{(i)} \end{aligned} \tag{7.8}$$

Where $p^{(i)}$ is the projection of $\mathbf{x}^{(i)}$ onto θ

- Now the question is how to minimize the cost function. In cases of margin classification. The following must be satisfied

$$\begin{cases} \mathbf{x}^{(i)}\theta = \|\theta\| p^{(i)} & \geq 1, \text{ for } x^{(i)} \\ \mathbf{x}^{(j)}\theta = -\|\theta\| p^{(j)} & \leq 1, \text{ for } x^{(j)} \end{cases} \tag{7.9}$$

It's obvious to see that as $p^{(i)}$ and $p^{(j)}$ get larger, a smaller $\|\theta\|$ would be needed to satisfy the inequality. That implies *when θ in the direcction to make a maximized margin, then the total cost would be minimized.*

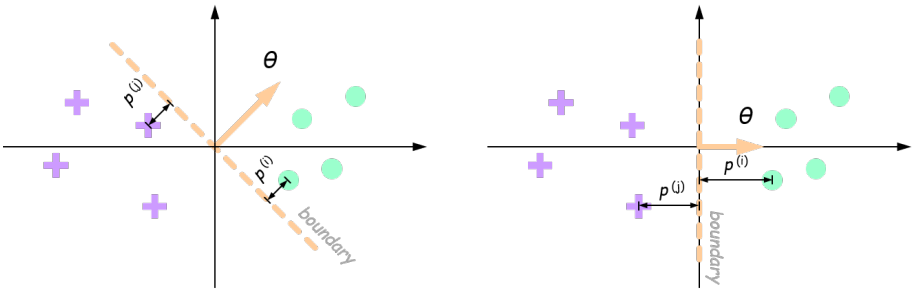

Figure 7.3: Margin classification with different θ

7.3 Gaussian kernel

- Gaussian kernel can make prediction of input data with a non-linear decision boundary. Given m data, then choose these data as landmarks.

$$l^{(1)} = x^{(1)} \quad l^{(2)} = x^{(2)} \quad \dots \quad l^{(m)} = x^{(m)} \tag{7.10}$$

- Define the *similarity function*

$$\begin{aligned} f_j^{(i)} &= f(x^{(i)}, l^{(j)}) \\ &= \exp\left(-\frac{\left\|x^{(i)} - l^{(j)}\right\|^2}{2\sigma^2}\right) \end{aligned} \tag{7.11}$$

  Then the values of the similarity fuctions would be

$$f_j^{(i)} \approx \begin{cases} 1, \text{ for } x^{(i)} \approx l^{(j)} \\ 0, \text{ for } x^{(i)} \neq l^{(j)} \end{cases} \tag{7.12}$$

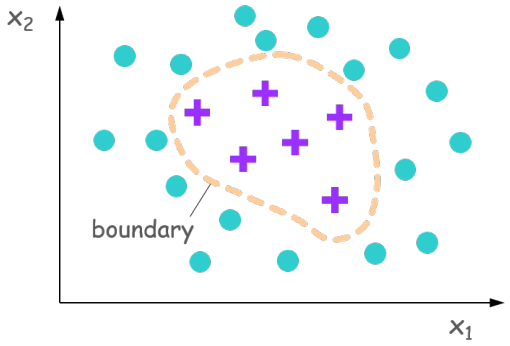

Figure 7.4: Non-linear decision boundary

- For each similarity functions, its distribution looks like

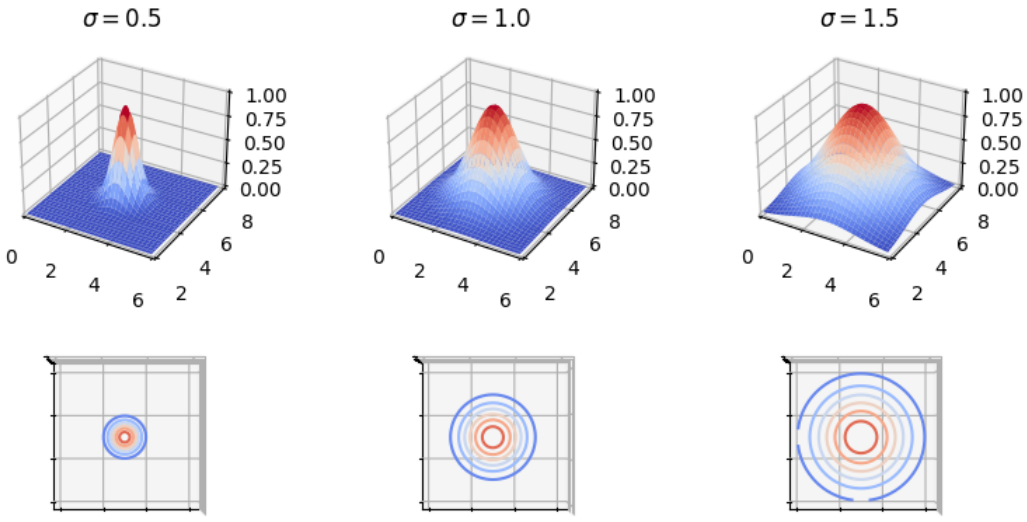

Figure 7.5: Gaussian kernel functions

- For training example $x^{(i)}$, all of its similarity would be

$$\mathbf{f}^{(i)} = \begin{bmatrix} f_0^{(i)} \\ f_1^{(i)} \\ f_2^{(i)} \\ \vdots \\ f_m^{(i)} \end{bmatrix} = \begin{bmatrix} 1 \\ f(x^{(i)}, l^{(1)}) \\ f(x^{(i)}, l^{(2)}) \\ \vdots \\ f(x^{(i)}, l^{(m)}) \end{bmatrix} \tag{7.13}$$

  The i^{th} item in the array is

$$f_i^{(i)} = f(x^{(i)}, l^{(i)}) = 1 \tag{7.14}$$

- So, how does Gaussian kernel work? Assume

$$z^{(i)} = \theta^T \mathbf{f}^{(i)} \tag{7.15}$$

  Put $z^{(i)}$ to the sigmoid function.

$$g^{(i)} = \frac{1}{1 + e^{-z^{(i)}}} \tag{7.16}$$

  Then the decision boundary would be

$$\begin{aligned} z^{(i)} = \theta^T \mathbf{f}^{(i)} \geq 0 \Rightarrow g^{(i)} \geq 0.5 \Rightarrow y^{(i)} = 1 \\ z^{(i)} = \theta^T \mathbf{f}^{(i)} < 0 \Rightarrow g^{(i)} < 0.5 \Rightarrow y^{(i)} = 0 \end{aligned} \tag{7.17}$$

  Then the total cost become

$$\min_\theta \frac{1}{m} \sum_{i=0}^{m} \left[y^{(i)} L_1(z^{(i)}) + (1 - y^{(i)}) L_0(z^{(i)}) \right] + \frac{\lambda}{2m} \sum_{j=1}^{m} \theta_j^2 \tag{7.18}$$

Chapter 8

K-means Algorithm

8.1 Introduction

- Randomly initialize K cluster centroids. Normally, K *training example* were picked and treated as the initial clusters.

$$\mu_k \in \mathbb{R}^n \text{ , for } k = 1, 2, \ldots, K \tag{8.1}$$

- Find the closet cluster for each $x^{(i)}$ and partition the m observations into K sets according to their closet cluster

$$\mathbf{S} = \{S_1, S_2, \ldots, S_K\} \tag{8.2}$$

- Find the center of geometry (mean) and then assign this value as a new μ_k for next iteration

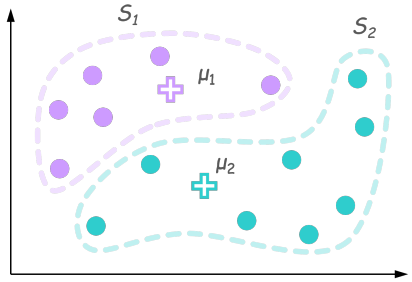

Figure 8.1: Initial cluster centroids

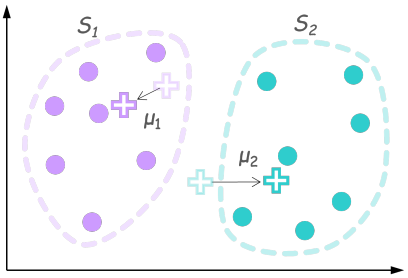

Figure 8.2: Cluster centroids at first iteration

- Optimization objective

$$\arg\min_{\mathbf{S}} \sum_{k=1}^{K} \sum_{x^{(i)} \in S_i} \left\| x^{(i)} - \mu_k \right\|^2 \tag{8.3}$$

8.2 Choosing the number of clusters

- Actually, there isn't a great way to choose the number of clusters automatically. By far, the most common way is still choosing it manually by looking at visualizations or by looking at the output of the clustering algorithm.

- One common method is Elbow Method. The intuition is that increasing the number of clusters will naturally improve the fit. Once the line chart resembles an arm, then the "elbow" (the point of inflection on the curve) is a good indication that the underlying model fits best at that point.

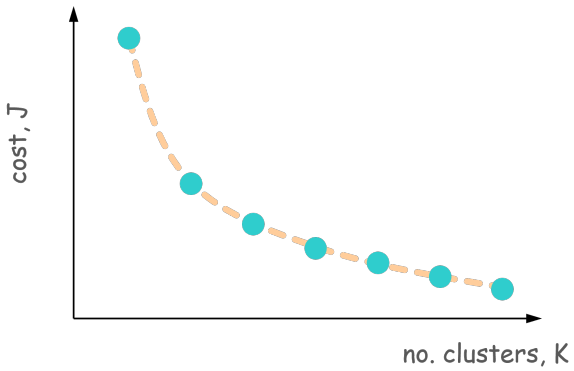

Figure 8.3: The cost value differ with the number of clusters

Chapter 9

Principal Component Analysis

9.1 Introduction

- Dimensionality Reduction

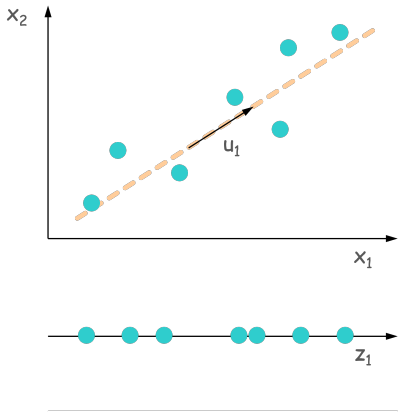

Figure 9.1: Reduction from 2D to 1D

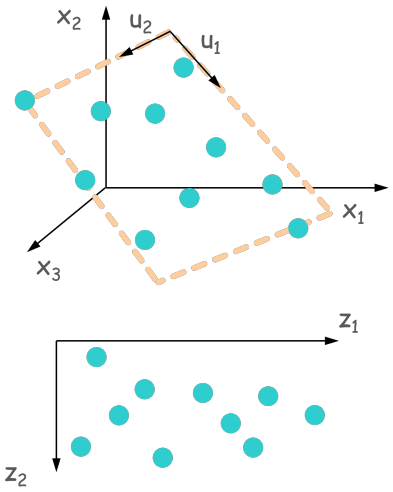

Figure 9.2: Reduction from 3D to 2D

- To reduce training data from n-dimension to k-dimension, find k vectors

$$u^{(1)}, u^{(2)}, \ldots, u^{(k)} \tag{9.1}$$

  onto which to project the data, so as to minimize the projection error.

- PCA is not regression

9.2 Algorithm

1. Mean normaliza1on and feature scaling

$$x^{(i)} := \frac{x^{(i)} - \mu^{(i)}}{s^{(i)}} \tag{9.2}$$

   where $\mu^{(i)}$ is the average of all the values for feature i and $s^{(i)}$ is the range of values, or $s^{(i)}$ is the standard deviation.

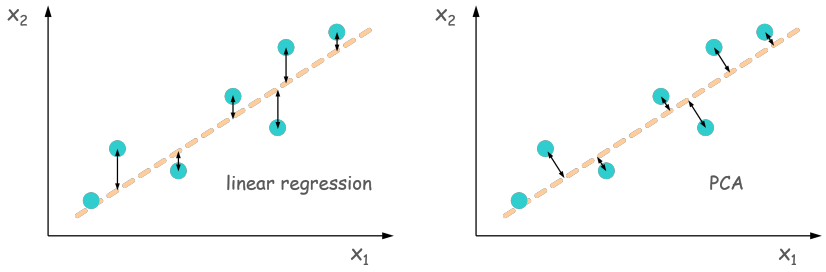

Figure 9.3: PCA versus linear regression

2. Compute *covariance matrix*

$$\mathbf{C}_{n\times n} = \frac{1}{m}\sum_{i=1}^{m} \mathbf{x}^{(i)^T}\mathbf{x}^{(i)} = \frac{1}{m}\mathbf{X}^T\mathbf{X} \tag{9.3}$$

3. Compute *eigenvectors* by *singular value decomposition*

$$\mathbf{C}_{n\times n} = \mathbf{U}_{n\times n}\mathbf{\Sigma}_{n\times n}\mathbf{V}^T_{n\times n} \tag{9.4}$$

Here $\mathbf{U}$ is a complex unitary matrix, the columns of which are orthogonal unit vectors called the left singular vectors of $\mathbf{C}$; and $\mathbf{\Sigma}$ is an rectangular diagonal matrix of positive numbers $\sigma(k)$ called the singular values of $\mathbf{C}$; and $\mathbf{V}$ is a complex unitary matrix whose columns are orthogonal unit vectors either called the right singular vectors of $\mathbf{C}$.

4. Rearrange the eigenvectors and eigenvalues. Sort the columns of the eigenvector matrix $\mathbf{U}$ and eigenvalue matrix $\mathbf{\Sigma}$ in order of decreasing eigenvalue.

5. Select a subset of the eigenvectors as basis vectors

$$\mathbf{U}_{n\times n} = \begin{bmatrix} | & | & | & & | & & | \\ u_1 & u_2 & u_3 & \dots & u_k & \dots & u_n \\ | & | & | & & | & & | \end{bmatrix} = \begin{bmatrix} & & | \\ \mathbf{U}_k & \dots & u_n \\ & & | \end{bmatrix} \tag{9.5}$$

Typically, choose k to be smallest value so that

$$\frac{\sum_{i=1}^{k}\sigma_{ii}}{\sum_{i=1}^{m}\sigma_{ii}} \geq 0.9 \tag{9.6}$$

It means 90% of variance retained.

6. Project the data onto the new basis

$$\mathbf{Z}_{m\times k} = \mathbf{X}_{m\times n}\mathbf{U}_{k,n\times k} \tag{9.7}$$

7. Recover an approximation of the original data that has been reduced to K dimensions

$$\mathbf{X}_{m\times n} = \mathbf{Z}_{m\times k}\mathbf{U}^T_{k,n\times k} \tag{9.8}$$

Chapter 10

Anomaly Detection

10.1 Gaussian Distribution

- Say $x \in \mathbb{R}$. If x is a distributed Gaussian with mean μ, variance σ^2 (where σ called "standard deviation")

$$P(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{10.1}$$

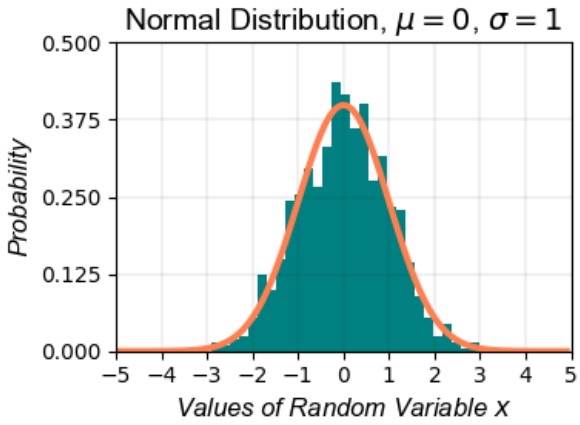

Figure 10.1: Gaussian distribution

- Algorithm

  1. Choose features x_i that you think might be indicative of anomalous examples, such as "memory use", "CPU load" for monitoring computers
  2. Fit parameters for $\mu_1, \ldots, \mu_n$ and $\sigma_1^2, \ldots, \sigma_n^2$

$$\begin{aligned} \mu_j &= \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)} \\ \sigma_j^2 &= \frac{1}{m}\sum_{i=1}^{m} \left(x^{(i)_j} - \mu_j\right)^2 \end{aligned} \tag{10.2}$$

3. Compute the probability density function

$$P(x) = \prod_{j=1}^{n} P(x_j; \mu_j, \sigma_j^2) = \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}\right) \tag{10.3}$$

4. Predict

$$y = \begin{cases} 1 & \text{if} \quad p(x) < \varepsilon \quad \text{(anomaly)} \\ 0 & \text{if} \quad p(x) \geq \varepsilon \quad \text{(nomal)} \end{cases} \tag{10.4}$$

Parameter ε can be choose by observation from cross validation set. There are other ways, such as *error metrices*, *precision/recall* or, *F1 score*.

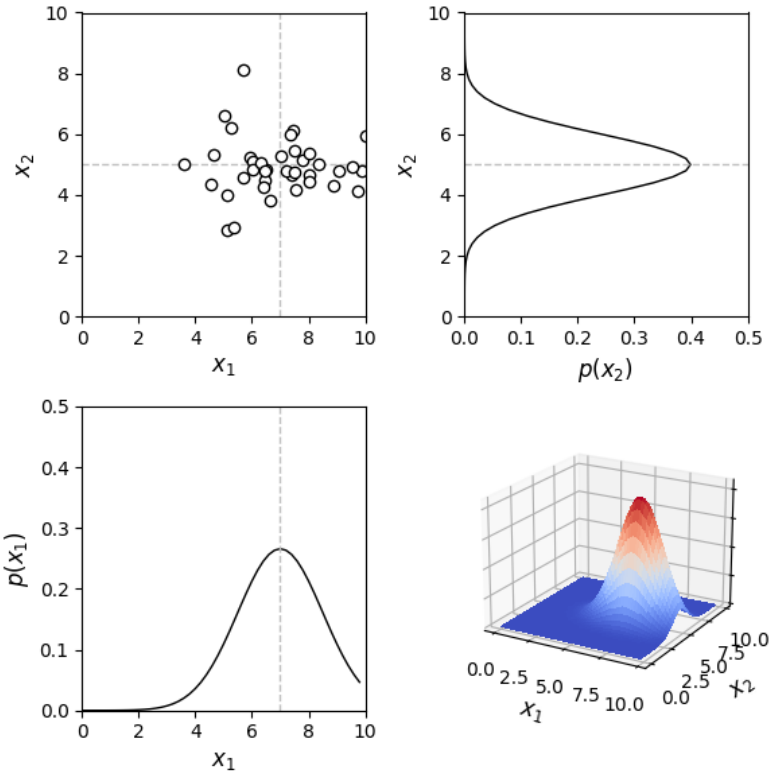

Figure 10.2: Gaussian distribution

10.2 Multivariate Gaussian Distribution

- The multivariate normal distribution is often used to describe any set of *correlated* random variables. If using single-variable distribution to describe a variable-correlated dataset, anomalies could be hard to excluded.
- Algorithm

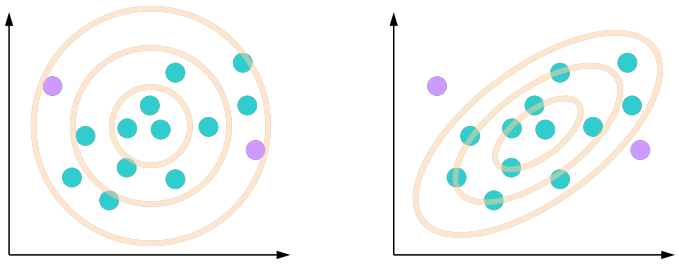

Figure 10.3: Gaussian distribution with single-variate and multivariate

1. Fit model $p(x)$ by setting mean vector $\mu \in \mathbb{R}^n$ and covariance matrix $\Sigma \in \mathbb{R}^{n \times n}$

$$\begin{aligned} \mu &= \frac{1}{m} \sum_{i=1}^{m} x^{(i)} \\ \Sigma &= \frac{1}{m} \sum_{i=1}^{m} \left(x^{(i)} - \mu\right) \left(x^{(i)} - \mu\right)^T \end{aligned} \tag{10.5}$$

2. Given a new example x, compute probability density function

$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu)\right) \tag{10.6}$$

3. Flag an anomaly if $p(x) < \varepsilon$

- Examples.

$$\begin{aligned} \mu_1 &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \Sigma_1 &= \begin{bmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \end{bmatrix} \\ \mu_2 &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \Sigma_2 &= \begin{bmatrix} 1.0 & 0.5 \\ 0.5 & 1.0 \end{bmatrix} \\ \mu_3 &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \Sigma_3 &= \begin{bmatrix} 1.0 & -0.5 \\ -0.5 & 1.0 \end{bmatrix} \end{aligned} \tag{10.7}$$

10.3 Non Gaussian Features

- Given the data set that looks like a *log-normal* distribution, take a log transformation of the data, then the histogram looks much more Gaussian.
- Probability density function for log-normal distribution

$$f_X(x) = \frac{1}{x\sigma\sqrt{2\pi}} \exp\left(-\frac{(\ln x - \mu)^2}{2\sigma^2}\right) \tag{10.8}$$

- Cumulative distribution function for log-normal distribution

$$F_X(x) = \frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{\ln x - \mu}{\sigma\sqrt{2}}\right)\right] \tag{10.9}$$

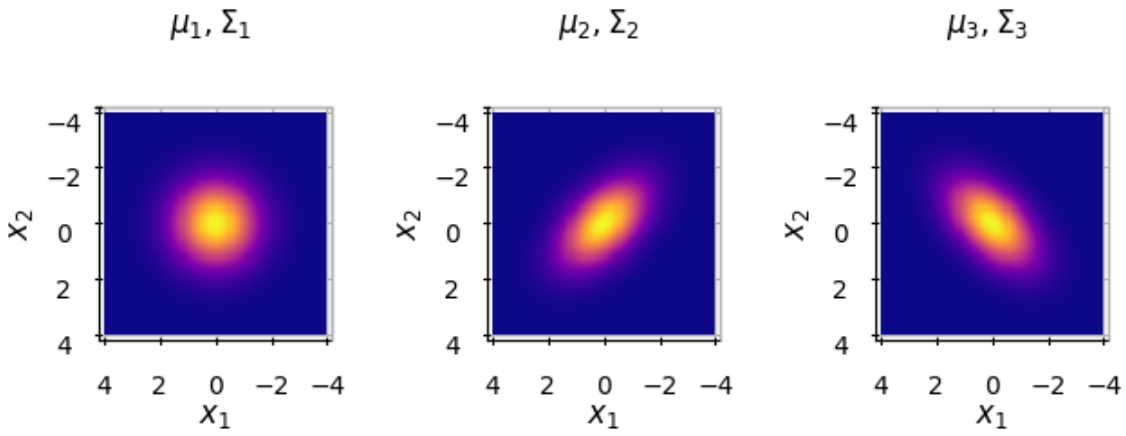

Figure 10.4: Gaussian distribution with multivariate

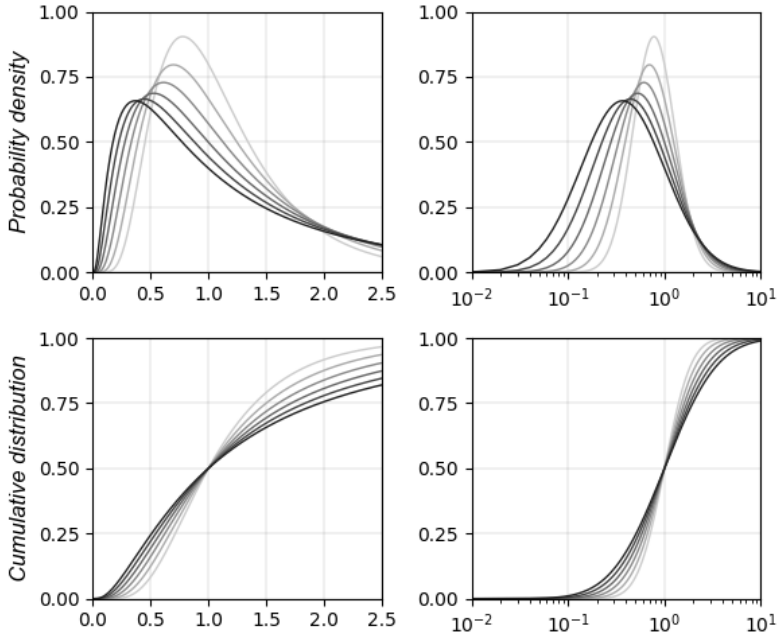

Figure 10.5: Log-normal distribution

Chapter 11

Recommender Systems

11.1 Given Features To Learn Parameters

Table 11.1: Movie ratings

Movie	Alice	Bob	Carol	Dave
Love at last	5	5	0	0
Romance forever	5	?	?	0
Cute puppies of love	?	4	0	?
Nonstop car chases	0	0	5	4
Swords vs. karate	0	0	5	?

Table 11.2: Movie features

Movie	romance	action
Love at last	0.90	0.00
Romance forever	1.00	0.01
Cute puppies of love	0.99	0.00
Nonstop car chases	0.10	1.00
Swords vs. karate	0.00	0.90

Table 11.3: User preference

Movie	Alice	Bob	Carol	Dave
romance	?	?	?	?
action	?	?	?	?

Define

$$\mathbf{Y} = \begin{bmatrix} 5 & 5 & 0 & 0 \\ 5 & & & 0 \\ & 4 & 0 & \\ 0 & 0 & 5 & 4 \\ 0 & 0 & 5 & \end{bmatrix}, \mathbf{R} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}, \mathbf{X} = \begin{bmatrix} 1 & 0.9 & 0.0 \\ 1 & 1.0 & 0.01 \\ 1 & 0.99 & 0.0 \\ 1 & 0.1 & 1.0 \\ 1 & 0.0 & 0.9 \end{bmatrix}, \mathbf{\Theta} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix} \quad (11.1)$$

Optimize $\theta^{(1)}, \ldots, \theta^{(q)}$

$$\min_{\theta^{(1)},\ldots,\theta^{(q)}} \frac{1}{2} \sum_{j=1}^{q} \sum_{i:r_{ij}=1} \left(x^{(i)}\theta^{(j)} - y_{ij}\right)^2 + \frac{\lambda}{2} \sum_{j=1}^{q} \left\|\theta^{(j)}\right\|^2 \tag{11.2}$$

Where

$$\begin{array}{ll} p & = \text{the number of movies} \\ q & = \text{the number of users} \\ \theta^{(j)} & = \text{column vector for } \mathbf{\Theta} \\ x^{(i)} & = \text{row vector for } \mathbf{X} \end{array} \tag{11.3}$$

11.2 Given Parameters To Learn Features

Table 11.4: Movie ratings

Movie	Alice	Bob	Carol	Dave
Love at last	5	5	0	0
Romance forever	5	?	?	0
Cute puppies of love	?	4	0	?
Nonstop car chases	0	0	5	4
Swords vs. karate	0	0	5	?

Table 11.5: Movie features

Movie	romance	action
Love at last	?	?
Romance forever	?	?
Cute puppies of love	?	?
Nonstop car chases	?	?
Swords vs. karate	?	?

Table 11.6: User preference

Movie	Alice	Bob	Carol	Dave
romance	5	5	0	0
action	0	0	5	5

Define

$$\mathbf{Y} = \begin{bmatrix} 5 & 5 & 0 & 0 \\ 5 & & & 0 \\ & 4 & 0 & \\ 0 & 0 & 5 & 4 \\ 0 & 0 & 5 & \end{bmatrix}, \mathbf{R} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}, \mathbf{X} = \begin{bmatrix} 1 & ? & ? \\ 1 & ? & ? \\ 1 & ? & ? \\ 1 & ? & ? \\ 1 & ? & ? \end{bmatrix}, \mathbf{\Theta} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 5 & 5 & 0 & 0 \\ 0 & 0 & 5 & 5 \end{bmatrix} \tag{11.4}$$

Optimize $x^{(1)}, \ldots, x^{(p)}$

$$\min_{x^{(1)},\ldots,x^{(p)}} \frac{1}{2} \sum_{i=1}^{p} \sum_{j:r_{ij}=1} \left(x^{(i)}\theta^{(j)} - y_{ij}\right)^2 + \frac{\lambda}{2} \sum_{i=1}^{p} \left\|x^{(i)}\right\|^2 \tag{11.5}$$

Where

$$\begin{array}{ll} p & = \text{the number of movies} \\ q & = \text{the number of users} \\ \theta^{(j)} & = \text{column vector for } \boldsymbol{\Theta} \\ x^{(i)} & = \text{row vector for } \mathbf{X} \end{array} \tag{11.6}$$

11.3 Collaborative Filtering Algorithm

Optimize $\theta^{(1)}, \ldots, \theta^{(q)}$ and $x^{(1)}, \ldots, x^{(p)}$ simultaneously

1. Initialize $\theta^{(1)}, \ldots, \theta^{(q)}$ and $x^{(1)}, \ldots, x^{(p)}$ to small random values
2. Compute cost function

$$J = \frac{1}{2} \sum_{(i,j):r_{ij}=1} \left(x^{(i)}\theta^{(j)} - y_{ij}\right)^2 + \frac{\lambda}{2} \sum_{j=1}^{q} \left\|\theta^{(j)}\right\|^2 + \frac{\lambda}{2} \sum_{i=1}^{p} \left\|x^{(i)}\right\|^2 \tag{11.7}$$

3. Minimize cost using gradient descent method

$$\begin{aligned} x_k^{(i)} &:= x_k^{(i)} - \alpha \left[\sum_{j:r_{ij}=1} \left(x^{(i)}\theta^{(j)} - y_{ij}\right) \theta_k^{(j)} + \lambda x_k^{(i)} \right] \\ \theta_k^{(j)} &:= \theta_k^{(j)} - \alpha \left[\sum_{i:r_{ij}=1} \left(x^{(i)}\theta^{(j)} - y_{ij}\right) x_k^{(i)} + \lambda \theta_k^{(j)} \right] \end{aligned} \tag{11.8}$$

4. Predict star rating of $\mathbf{X}\boldsymbol{\Theta}$

11.4 Applications

- Finding related movies
  - For each product i, its feature vector $x^{(i)} \in \mathbb{R}^n$. As $\left\|x^{(i)} - x^{(j)}\right\|$ decrease, the similarity between product i and j is increase.
- Recommendation for new users

$$\mathbf{Y} = \begin{bmatrix} 5 & 5 & 0 & 0 \\ 5 & & & 0 \\ & 4 & 0 & \\ 0 & 0 & 5 & 4 \\ 0 & 0 & 5 & \end{bmatrix}, \mu = \begin{bmatrix} 2.5 \\ 2.5 \\ 2.0 \\ 2.25 \\ 1.25 \end{bmatrix}, \tag{11.9}$$

Assume the rating by a new user Eve is μ, so the new rating matrix would be

$$\mathbf{Y} = \begin{bmatrix} 5 & 5 & 0 & 0 & 2.5 \\ 5 & & & 0 & 2.5 \\ & 4 & 0 & & 2.0 \\ 0 & 0 & 5 & 4 & 2.25 \\ 0 & 0 & 5 & & 1.25 \end{bmatrix} \tag{11.10}$$

If mean normalization is a pre-processing step for collaborative filtering

$$\bar{\mathbf{Y}} = \mathbf{Y} - \mu = \begin{bmatrix} 2.5 & 2.5 & -2.5 & -2.5 & 0 \\ 2.5 & & & -2.5 & 0 \\ & 2.0 & -2.0 & & 0 \\ -2.25 & -2.25 & 2.75 & 1.75 & 0 \\ -1.25 & -1.25 & 3.75 & & 0 \end{bmatrix} \tag{11.11}$$

Because the values at the last column of $\bar{\mathbf{Y}}$ are all zeros, it means

$$\theta^{(5)} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{11.12}$$

That implied the preferences of a new user would be neutral.

Appendix A

Matrices Analysis

A.1 The Gradient Field

- For a scalar function of three independent variables $f(x_1, x_2, x_3)$ the gradient is given by the vector equation

$$\nabla f = \frac{\partial f}{\partial x_1}\hat{x}_1 + \frac{\partial f}{\partial x_2}\hat{x}_2 + \frac{\partial f}{\partial x_3}\hat{x}_3 \tag{A.1}$$

- This can be seen as the derivative of a scalar with respect to a vector, and its result can be easily collected in vector form

$$\nabla f = \frac{\partial f}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \frac{\partial f}{\partial x_3} \end{pmatrix} \tag{A.2}$$

A.2 Inner Products

Let's proof the inner product of vectors. Recall the law of cosines

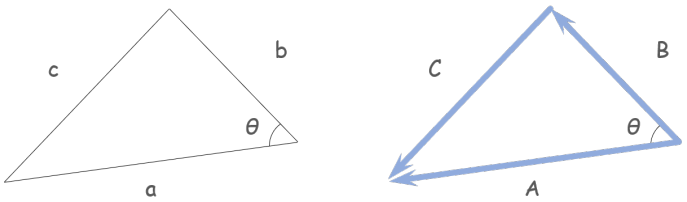

Figure A.1: The law of cosine and the angle between vectors

$$a^2 + b^2 - 2ab\cos\theta = c^2 \tag{A.3}$$

Apply this to vector triangle

$$\|\mathbf{A}\|^2 + \|\mathbf{B}\|^2 - 2\|\mathbf{A}\|\|\mathbf{B}\|\cos\theta = \|\mathbf{A}-\mathbf{B}\|^2 \tag{A.4}$$

Where

$$\begin{aligned} \|\mathbf{A}-\mathbf{B}\|^2 &= (\mathbf{A}-\mathbf{B})\cdot(\mathbf{A}-\mathbf{B}) \\ &= \|\mathbf{A}\|^2 + \|\mathbf{B}\|^2 - 2\mathbf{A}\cdot\mathbf{B} \end{aligned} \tag{A.5}$$

This gives

$$\cos\theta = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \, \|\mathbf{B}\|} \tag{A.6}$$

A.3 Eigenvalues And Eigenvectors

- For a scalar function of three independent variables $f(x_1, x_2, x_3)$ the gradient is given by the vector equation

A.4 Singular Value Decomposition

- For a scalar function of three independent variables $f(x_1, x_2, x_3)$ the gradient is given by the vector equation

Appendix B

Cheat sheet

B.1 Linear Regression vs Logistic Regression

Table B.1: Linear Regression vs Logistic Regression (w/ regularizer)

Items	Linear Regression	Logistic Regression
Cost (original)	$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(\mathbf{x}^{(i)}\theta - y^{(i)}\right)^2$	$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(h^{(i)}\right) + (1-y^{(i)})\log\left(1 -\right.\right.$
Cost (vectorized)	$J(\theta) = \frac{1}{2m}(\mathbf{X}\theta - \mathbf{y})^T(\mathbf{X}\theta - \mathbf{y})$	$J(\theta) = -\frac{1}{m}\left(\mathbf{y}^T\log(\mathbf{h}) + (\mathbf{1}-\mathbf{y})^T\log(\mathbf{1}-\mathbf{h})\right)$
Gradient descent (original)	$\theta_j \coloneqq \theta_j - \frac{\alpha}{m}\sum_{i=1}^{m}\left(h^{(i)} - y^{(i)}\right)x_j^{(i)}$	Same as linear regression
Gradient descent (vectorized)	$\theta \coloneqq \theta - \frac{\alpha}{m}\mathbf{X}^T(\mathbf{X}\theta - \mathbf{y})$	Same as linear regression
Normal equation	$\theta = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$	None