Chapter 4

Logistic Regression

4.1 Learning Objectives

In the previous chapter, we studied linear regression to predict a label with continuous values. In this chapter, we will learn another type of supervised learning – classification, where the label has discrete (categorical) values. A simple classification task can be implemented by *logistic regression*, which is the topic of this chapter. A complicated classification task, such as image recognition, may require deep learning neural networks. In subsequent chapters, we will see that a logistic regression classifier is a typical neuron that is the basic element of neural networks. Thus, logistic regression is a foundation of studying neural networks later.

After finishing the chapter, one should be able to

- Understand the concept of classification problem
- Understand why linear regression does not work well for a classification problem
- Understand the architecture and algorithm of logistic regression
- Understand sigmoid function and cost function in logistic regression
- Understand the vectorized gradient descent for logistic regression
- Be able to develop a logistic regression classifier from the scratch or using sklearn library
- Understand multi-class classification using one-vs-all strategy.
- Understand performance metrics: accuracy, recall, precision, specificity, ROC

4.2 Mathematical description of logistic regression

4.2.1 Classification

The linear regression previously discussed assumes that the target (or label) variable is quantitative (or continuous). But in many situations, the target variable is instead qualitative (or categorical, or discrete). The task for predicting a categorical variable is called *classification*. The goal in classification is to take an input vector $\mathbf{x}$ and to assign it to one of K discrete categories or classes, C_k, $k=1,2,..,K$. Thus, classification is a function that maps multiple data examples into finite categories. In other words, some data examples with certain common characteristics are grouped

into one category. The input space is thereby divided into decision regions where boundaries are called *decision boundaries* or *decision surfaces*. The label variable *y* for two-category classification can be coded in binary, i.e. $y \in \{0,1\}$, such at y=0 represents class C1 and y=1 represents class C2. For m-class classification tasks, $\boldsymbol{y}$ is typically represented by an integer or an m-bit one-hot code. For instance, if we have K=5 classes, then the label y for class 3 would be $y=[0,0,1,0,0]^T$.

[Example 4.1] The following table lists three examples of classification task.

Input (X)	Categorical label or class (y)
Email	y=0 (not spam), y=1 (spam)
Online Transaction	y=0 (not fraudulent), y=1 (fraudulent)
Tumor	y=0 (benign), y=1 (malignant)

4.2.2 Why linear regression does not work well for classification?

The following example is usually used to explain why linear regression can not be used for classification, shown in Fig.1. Suppose that we have a data set (*x*,*y*), *x* is the size of tumors and *y* indicates whether the tumor is malignant for a given *x*. For example, we have 9 data examples plotted in Fig.1 as 4 blue x and 5 blue o. it is assumed that a larger sized tumor is more likely malignant. If a linear regression is applied to the 9 data examples, a prediction line (in green),$y = \theta_0 + \theta_1 x$, is obtained. However, the labeled value of y is either 0 or 1. Thus a threshold value of 0.5 be used to predict the discrete value of *y*,

$$\hat{y} = \begin{cases} 0 & \theta_0 + \theta_1 x < 0.5 \\ 1 & \theta_0 + \theta_1 x \geq 0.5 \end{cases}$$

The problem of this method will happen when one data example (outlier in red) is added to the training set. As a result, the prediction line will change significantly, leading to an unacceptable classification error rate. In fact, a working prediction line should have a non-linear shape (e.g. a step shape) illustrated by the purple dot line in Fig.1. Thus, the learning model for a classification task should be non-linear.

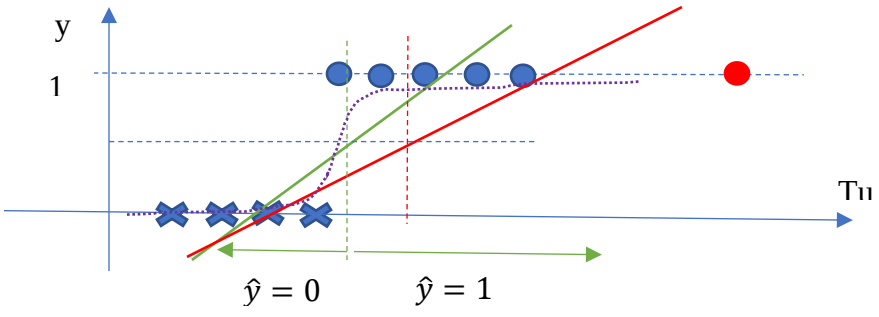


Fig.1 Why linear regression does not work well for classification

4.2.3 Logistic regression model

If the input vector space in a 2-class classification problem is linearly separable, there exists a linear decision boundary $\boldsymbol{\theta}^T\mathbf{x} = 0$, where $\boldsymbol{\theta}$ and $\mathbf{x}$ are the column vectors, defined as

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \in R^{(n+1)\times 1}, \qquad \mathbf{x} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \in R^{(n+1)\times 1}$$

where x_j, $j=1,2,\ldots,$ n is the jth feature. We can define a dummy feature $x_0 = 1$. θ is the parameter vector. To demonstrate this, we consider n=2, and let $\boldsymbol{\theta}^T\mathbf{x}=\mathbf{0}$ be the decision boundary, shown in Fig. 2. An input vector x is assigned to one class (say class2) if $\boldsymbol{\theta}^T\mathbf{x} \geq 0$ and to another class (say class1) otherwise. It can be proven that the value of $\boldsymbol{\theta}^T\mathbf{x}$ gives a signed measure of the perpendicular distance r of any point $\mathbf{x}$ from the decision boundary

$$r = \frac{\boldsymbol{\theta}^T\mathbf{x}}{\sqrt{\theta_1^2+\theta_2^2}} \tag{4.1}$$

If data examples are linearly separable and an optimal decision boundary has been found, illustrated in Fig.2, the signed distance r can be used to measure or predict *how likely* a data example comes from class 1 or class 2. If r>0, x is assigned to class 2, and the larger the r, the more likely x comes from class 2. If r<0, x is assigned to class 1, and the larger of the magnitude of r, the more likely x comes from class 1.

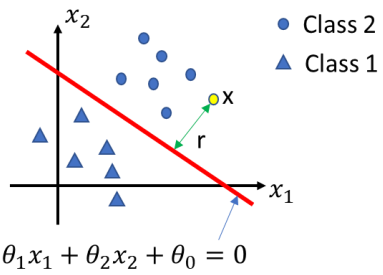


Fig. 2 Illustration of linear decision boundary in two-class classification with two features.

For classification problems, however, we wish to predict discrete class labels, or more generally posterior probabilities $p(C_k|\mathbf{x})$ that lie in the range of [0,1]. To achieve this, we apply a non-linear g(.)

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = g\left(\boldsymbol{\theta}^T\mathbf{x}\right) \tag{4.2}$$

to map the signed distance r to a probabilistic space [0,1], and thus $h_{\boldsymbol{\theta}}(\mathbf{x})$ can be interpreted as the probability of x belonging to a class. g(.) is known as an activation function. For example, one of the good candidates of activation function is sigmoid function shown in Fig.3, defined as

$$g(z) = \frac{1}{1+e^{-z}} \tag{4.3}$$

When $z=0$, $g(z)$ is equal to 0.5. When z departures from zero to the positive side, $g(z)$ quickly approaches 1. When z moves from zero to the negative side, $g(z)$ will quickly approaches 0. It is

worthy to note that any other functions with similar shape, e.g. tanh(), can be used for activation function, as you will see in neural networks.

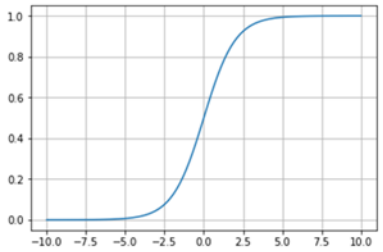


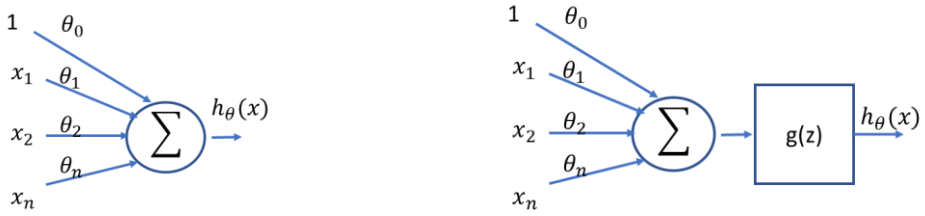
Fig.3 sigmoid function g(z)

Thus, based on the model (4.2), **x** would be predicted as class 1 if $h_{\boldsymbol{\theta}}(\mathbf{x}) < 0.5$ (i.e. $\boldsymbol{\theta}^T\mathbf{x} < \mathbf{0}$) , and as class 2 if $h_{\boldsymbol{\theta}}(\mathbf{x}) \geq 0.5$ (i.e. $\boldsymbol{\theta}^T\mathbf{x} > \mathbf{0}$). The decision boundary is defined by $h_{\boldsymbol{\theta}}(\mathbf{x}) = 0.5$, i.e.

$$\boldsymbol{\theta}^T\mathbf{x} = g^{-1}(0.5) = 0. \tag{4.4}$$

The hypothesis $h_{\boldsymbol{\theta}}(\mathbf{x}) = g(\boldsymbol{\theta}^T\mathbf{x})$ can be interpreted as the probability of x belonging to one of the classes (say class 2), and thus the probability of x belonging to another class (say class 1) is 1-$h_{\boldsymbol{\theta}}(\mathbf{x})$.

The model defined by (4.2) is called *logistic regression*, which is a generalized linear model in the sense that we don't output the weighted sum of inputs directly, but we pass it through an activation function that maps input linear combinations to the probabilistic space, as shown in Fig.4(b). The decision boundaries are linear functions of **x**, even if g(.) is generally nonlinear. It should be emphasized that logistic regression is a model for classification rather than regression.



(a) Linear regression (b) Logistic regression

Fig.4 Comparison between linear regression and logistic regression

Thus, the output of the linear combination is

$$z = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n = \boldsymbol{\theta}^T\mathbf{x} \tag{4.5}$$

Given **x** and **θ**, the hypothesis for the logistic regression, which is the output of logistic regression, is

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = g(z) = g(\boldsymbol{\theta}^T\mathbf{x}) = \frac{1}{1+e^{-\boldsymbol{\theta}^T\mathbf{x}}} \tag{4.6}$$

$h_{\boldsymbol{\theta}}(\mathbf{x})$ can be interpreted as the estimated probability of *y=1*, denoted by P[y=1|**x;θ**]. Class 1 is indicated by y=0, and class 2 is indicated by y=1. Thus, the posterior probability $p(C_2|x) = h_{\boldsymbol{\theta}}(\mathbf{x})$ and $p(C_1|x) = 1 - h_{\boldsymbol{\theta}}(\mathbf{x})$.

Intuitively, the prediction of *y* can be performed as

$$\hat{y} = \begin{cases} 1 & h_{\boldsymbol{\theta}}(\mathbf{x}) \geq 0.5 \\ 0 & h_{\boldsymbol{\theta}}(\mathbf{x}) < 0.5 \end{cases} \tag{4.7}$$

The decision boundary separating the two classes can be determined by setting the weighted sum of inputs to 0, i.e., $\boldsymbol{\theta}^T\mathbf{x} = 0$. If the input vector space is nonlinear separable (Fig.5(b) and (c) in Example 3.2), we can generalize (4.5) to a nonlinear function of **x**.

[Example 4.2] Decision boundaries

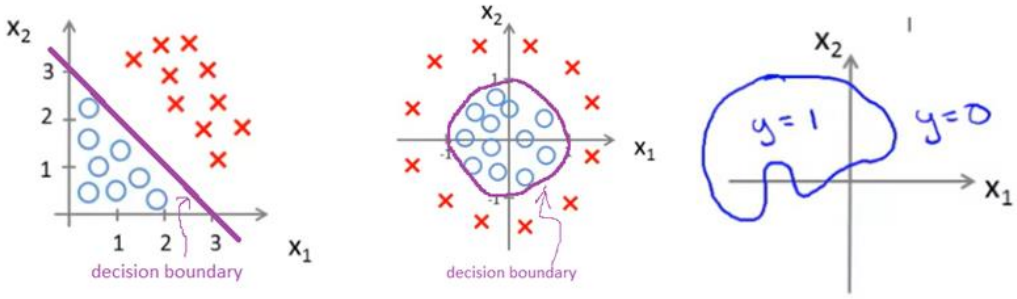


Fig.5(a) a linear decision boundary, (b) a circle decision boundary, (c) high-order non-linear decision boundary.

Fig.5(a) shows a linear decision boundary for a linearly separable dataset. The hypothesis is $h_{\boldsymbol{\theta}}(\mathbf{x}) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$, where

$$\theta = \begin{bmatrix} -3 \\ 1 \\ 1 \end{bmatrix}$$

We will predict y=1 if $\theta_0 + \theta_1 x_1 + \theta_2 x_2 \geq 0$, and predict y=0 if $\theta_0 + \theta_1 x_1 + \theta_2 x_2 < 0$. Thus, the decision boundary is the line $\theta_0 + \theta_1 x_1 + \theta_2 x_2 = 0$, i.e., $-3 + x_1 + x_2 = 0$ for this case. Any point located right-up the line will be classified as y=1, and a point below the line will be classified as y=0. The hypothesis values on the decision boundary is equal to 0.5.

Fig.5(b) shows a non-linear decision boundary when the data set is not linearly separable. The hypothesis $h_{\boldsymbol{\theta}}(\mathbf{x}) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$, with $\theta = [-1, 0, 0, 1, 1]^T$. The decision boundary is $\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 = 0$, i.e., $x_1^2 + x_2^2 = 1$. The model predicts y=1 if $x_1^2 + x_2^2 \geq 1$, and predicts y=0 if otherwise.

Fig.5(c) shows a more complicated decision boundary, based on the hypothesis $h_{\boldsymbol{\theta}}(\mathbf{x}) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2^2 + \theta_6 x_1^2 x_2 + \cdots)$ with higher-order polynomial features involved.

4.2.4 Learn the model: find optimal θ based on a data set

In this section, we will discuss how to fit the parameter θ so that the decision boundary can effectively classify the data examples. Given a value of θ and a value of *x*, the corresponding label *y* is predicted according to (4.7). In general, for a dataset, some predictions match the true labels while other predictions do not. This implies that the value of θ is not the optimal one or/and the data examples are theoretically not separable by the hypothesis model. This mismatch is measured by a cost function. Leaning the model is to adjust parameters θ such that the cost function is minimum. Since the cost function is a function of parameter θ, we can use optimization method to search for a value of θ to minimize the cost function. This optimization process is called model learning, training or fitting.

Cost function

To search for an optimal $\boldsymbol{\theta}$, we first quantify this mismatch between the current model and data examples, i.e., defining a cost function, and then minimize the cost function over the $\boldsymbol{\theta}$ space. The cost function of logistic regression, for a single data example (*x*,*y*), is defined as

$$J(h_{\boldsymbol{\theta}}(\mathbf{x}), y) = \begin{cases} -\ln\left(h_{\boldsymbol{\theta}}(\mathbf{x})\right) & if \;\; y = 1 \\ -\ln\left(1 - h_{\boldsymbol{\theta}}(\mathbf{x})\right) & if \;\; y = 0 \end{cases} \tag{4.8}$$

where *y* is the value of the label. We can understand the cost function in this way: if the label *y* is 1, the hypothesis $h_{\boldsymbol{\theta}}(\mathbf{x})$ is expected to be close 1. In other words, the value of $h_{\boldsymbol{\theta}}(\mathbf{x})$ near 0 will result in a larger penalty than $h_{\boldsymbol{\theta}}(\mathbf{x})$ near 1. The function, $-\ln\left(h_{\boldsymbol{\theta}}(\mathbf{x})\right)$, has this characteristic, shown in Fig.6(a). On the other hand, if the label y is 0, the hypothesis $h_{\boldsymbol{\theta}}(\mathbf{x})$ is expected to be close 0 with a smaller penalty. The function, $-\ln\left(1 - h_{\boldsymbol{\theta}}(\mathbf{x})\right)$, has this characteristic, shown in Fig.6(b).

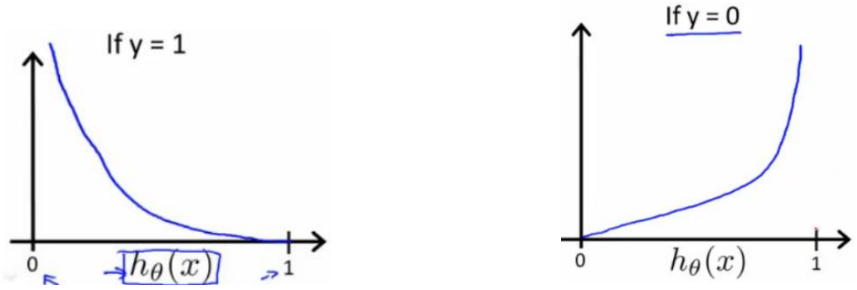


Fig.6. cost function $J(h_{\boldsymbol{\theta}}(\mathbf{x}), y)$, a) y=1, b) y=0

Remarks: 1) cost =0 if $h_{\boldsymbol{\theta}}(\mathbf{x}) = y$,

2) cost →∞, if y=1 but $h_{\boldsymbol{\theta}}(\mathbf{x})$→0,

3) cost→∞, if y=0 but $h_{\boldsymbol{\theta}}(\mathbf{x})$→1

The cost function in (4.8) can be equivalently represented by a more compact format

$$J(h_{\boldsymbol{\theta}}(\mathbf{x}), y) = -y\ln\left(h_{\boldsymbol{\theta}}(\mathbf{x})\right) - (1-y)\ln\left(1 - h_{\boldsymbol{\theta}}(\mathbf{x})\right) \tag{4.9}$$

The cost function for *m* examples, $\left(\mathbf{x}^{(1)}, y^{(1)}\right)$, $\left(\mathbf{x}^{(2)}, y^{(2)}\right)$, … , $\left(\mathbf{x}^{(m)}, y^{(m)}\right)$, will be

$$J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} J\left(h_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)}\right), y^{(i)}\right) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)} \ln\left(h_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)}\right)\right) + (1-y^{(i)}) \ln\left(1-h_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)}\right)\right)\ ] \tag{4.10}$$

where

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = g(z) = g\left(\boldsymbol{\theta}^{\boldsymbol{T}}\mathbf{x}\right) = \frac{1}{1+e^{-\boldsymbol{\theta}^{\boldsymbol{T}}\mathbf{x}}}$$

To fit parameters, we minimize the cost function with respect to θ, i.e., $\theta^* = arg\min_{\theta} J(\theta)$.

To predict, given a new value of x, the label can be predicted as

$$\hat{y} = \begin{cases} 1 & h_{\boldsymbol{\theta}^*}(\mathbf{x}) \geq 0.5 \\ 0 & h_{\boldsymbol{\theta}^*}(\mathbf{x}) < 0.5 \end{cases} \tag{4.11}$$

Logistic regression: maximum likelihood, a probabilistic view

Before we develop the gradient descent algorithm for training the logistic regression, let's look at logistic regression from a maximum likelihood point of view for a further understanding. In this section, we will see that minimizing the cost function of *m* examples (4.10), $J(\boldsymbol{\theta})$, is equivalent to maximizing the likelihood function.

Consider the problem of two-class classification with the following assumptions: 1) the probability $p(y=1|\mathbf{x};\theta) = h_{\boldsymbol{\theta}}(\mathbf{x})$ and $p(y=0|\mathbf{x};\theta) = 1-h_{\boldsymbol{\theta}}(\mathbf{x})$, where $h_{\boldsymbol{\theta}}(\mathbf{x})$ defined by (4.6) is the output of activation function and is interpreted/treated as the posterior probability of class; 2) all examples in the training dataset are independently drawn from the same probability distribution. The first assumption can be represented by a compact form

$$p(y|\mathbf{x};\theta) = h_{\theta}(\mathbf{x})^{y} \cdot (1-h_{\theta}(\mathbf{x}))^{1-y} \tag{4.12}$$

The likelihood function for *m* examples can be defined as the *m*-dimensional joint posterior probability of vector **y** given a matrix of features **X**

$$\mathcal{L}(\theta) = p(\mathbf{y}|\mathbf{X};\theta) = \prod_{i=1}^{m} p(y^{(i)}|\mathbf{x}^{(i)};\theta) = \prod_{i=1}^{m} h_{\theta}\left(\mathbf{x}^{(i)}\right)^{y^{(i)}} \cdot \left(1-h_{\theta}\left(\mathbf{x}^{(i)}\right)\right)^{1-y^{(i)}} \tag{4.13}$$

The logarithm likelihood function can be obtained by applying ln(.) operation to (4.13)

$$\ell(\theta) = \sum_{i=1}^{m}[y^{(i)} \ln\left(h_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)}\right)\right) + (1-y^{(i)}) \ln\left(1-h_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)}\right)\right)\ ] \tag{4.14}$$

By comparing (4.14) with (4.10), we have

$$J(\theta) = -\frac{1}{m}\ell(\theta) \tag{4.15}$$

(4.15) implies that minimizing the cost function $J(\theta)$ is equivalent to maximizing the likelihood function $\ell(\theta)$, i.e.,

$$\theta^* = arg\min_{\theta} J(\theta) = arg\max_{\theta} \ell(\theta) \tag{4.16}$$

This conclusion validates the effectiveness of cost function in (4.9) from a probabilistic point of view.

Gradient descent algorithm

To learn the parameter θ, we will use gradient descent algorithm to minimize the cost function (4.10) and thus equivalently to maximize the likelihood function (4.16). To calculate the partial derivative of J(θ) with respect to θ_j, we need to use the chain rule and the following derivative equations,

$$\frac{d}{dz}g(z) = g(z)\big(1 - g(z)\big), \qquad \text{g(z) is the sigmoid function} \tag{4.17}$$

$$\frac{d}{du}\ln(u) = \frac{1}{u}, \text{ ln(u) is the nature logarithm function} \tag{4.18}$$

It can be shown that the partial derivative of J(θ) is

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}[(h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)}) \cdot x_j^{(i)}] \tag{4.19}$$

where *j =0,1,2,….,n,* $x_j^{(i)}$ is the jth feature of th ith example and $x_0^{(i)} = 1$. Please note that (4.19) has the same form as (3.15) that was used for linear regression with a different definition of hypothesis function $h_{\boldsymbol{\theta}}(\mathbf{x})$.

If we format the above derivatives as a gradient vector

$$\mathbf{grad} = \frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \begin{bmatrix} \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_0} \\ \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_n} \end{bmatrix} \tag{4.20}$$

then it is easy to verify that the gradient vector can be represented in vectorized format

$$\mathbf{grad} = \frac{1}{m}\mathbf{X}^T(g(\mathbf{X} \cdot \boldsymbol{\theta}) - \boldsymbol{Y}) \tag{4.21}$$

where $\mathbf{X} = \begin{bmatrix} -- \mathbf{x}^{(1)^T} -- \\ -- \mathbf{x}^{(2)^T} -- \\ \vdots \\ -- \mathbf{x}^{(m)^T} -- \end{bmatrix} = \begin{bmatrix} 1 & x_1^{(1)} & \dots & x_n^{(1)} \\ 1 & x_1^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & & \vdots \\ 1 & x_1^{(m)} & \dots & x_n^{(m)} \end{bmatrix} \in R^{m\times(n+1)}$ $\qquad \boldsymbol{Y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$

$\mathbf{X}^T$ is the transpose of **X**. In matrix X, each row represents a data example while each column corresponds to a feature. Y is the label vector. Like the gradient decent algorithm for linear regression, we can have a gradient decent algorithm for logistic regression as follows:

<u>Gradient decent algorithm for logistic regression</u>

1) Set initial values for $\boldsymbol{\theta}$, select α
2) Repeat:
   (i) Compute gradient vector: $grad \in R^{(n+1)\times 1}$

   $$\mathbf{grad} = \frac{1}{m}\mathbf{X}^T \cdot (g(\mathbf{X} \cdot \boldsymbol{\theta}) - \boldsymbol{Y})$$

   (ii) Update simultaneously $\boldsymbol{\theta} \coloneqq \boldsymbol{\theta} - \alpha \cdot \mathbf{grad}$

(iii) Update cost function $J(\boldsymbol{\theta})$

$$J(\boldsymbol{\theta}) = -\frac{1}{m}[(\ln(g(\mathbf{X}\cdot\boldsymbol{\theta})))^T\boldsymbol{Y} + (\ln(1-g(\mathbf{X}\cdot\boldsymbol{\theta})))^T(1-\boldsymbol{Y})]$$

(iv) Terminate: if the predefined maximal iterations have been completed, then exit to 3), otherwise go back to (i). Or Compare the current cost with the previous cost. If they are close enough, then exit to 3).

3) Return $\boldsymbol{\theta}$

Note: all the functions, ln( ), g( ), perform in a broadcasting manner.

Gradient descent is one of the optimization algorithms to fit parameters θ for $\min_{\theta} J(\theta)$. Besides gradient descent, there are some advanced optimization algorithms available to fit parameters θ. These advanced algorithms include Conjugate gradient, BFGS, and L_BFGS. Their advantages are 1) No need to manually pick α, 2) often faster than gradient descent. The details of these algorithms are beyond the scope of the text. You are not expected to write code to implement these algorithms unless you are an expert in numerical optimization. But you can use the libraries for these algorithms if they are available in your development languages.

[**Example 4.3]** Function scipy.optimize.fmin_tnc ( ) can be used to minimize a function. Consider minimizing

$$J(x) = (x_1 - 4)^2 + (x_2 - 5)^2$$

```
1.  import numpy as np
2.  from scipy.optimize import fmin_tnc
3.  x = np.zeros((2,1))
4.  def cost(x):
5.      total_cost=(x[0]-4)**2+(x[1]-5)**2
6.      return total_cost
7.  def gradient(x):
8.      grad=np.zeros((2,1))
9.      grad[0]=2*(x[0]-4)
10.     grad[1]=2*(x[1]-5)
11.     return grad
12. weights = fmin_tnc(func=cost, x0=x, fprime=gradient)
13. #weights = fmin_tnc(func=cost, x0=x, approx_grad=True)
14. print(weights)
```

Output:

(array([4., 5.]), 3, 0)

The details about fmin_tnc can be found at
https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.fmin_tnc.html .

4.2.5 Multi-class classification: One -vs-All

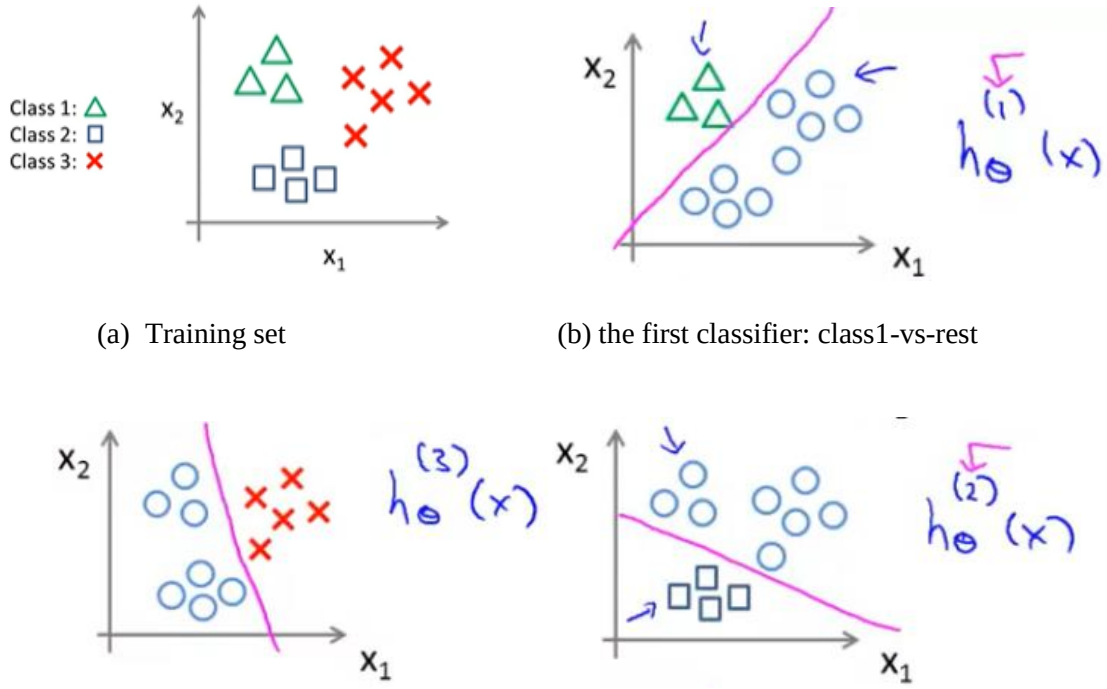
Now, let's discuss how to apply logistic regression to multi-class classification problems. Here are some examples of multi-class classification:

Email tagging: work (y=1), friends (y=2), family (y=3), hobby (y=4)

Medical diagnosis: Not ill (y=1), Cold (y=2), Flu (y=3)

Weather forecast: Sunny (y=1), Cloudy (y=2), Rain (y=3), Snow (y=4)

For an illustration purpose, Fig.7(a) shows a training set with three classes indicated by three symbols (Δ for class 1: y=1, □ for class 2: y=2, x for class 3: y=3). Fig.7(b),(c),(d) show three classifiers, each of which defines one class as positive label and the rest classes as negative label.



(a) Training set

(b) the first classifier: class1-vs-rest

(c) the second classifier: class2-vs-rest

(d) the third classifier: class3-vs-rest

Fig.7, three classifiers for multi-class classification using one-vs-all (or one-vs-rest)

Since the hypothesis of each classifier is interpreted as the probability of the positive class, we have

$$P(y = i|x;\theta) = h_\theta^{(i)}(x), \text{ i=1,2,3} \tag{4.22}$$

In summary, we train a logistic regression classifier $h_\theta^{(i)}(x)$ for class *i* to predict the probability that $y=i$. In total we need to build three logistic regression classifiers, as shown in Fig. 8. On a new input *x*, to make a prediction, pick the class *i* that maximizes

$$\max_i h_\theta^{(i)}(x) \tag{4.23}$$

However, in practice, multiple-class classifications are typically implemented by neural networks using softmax activation function in the output layer, which will be discussed in chapter XX.

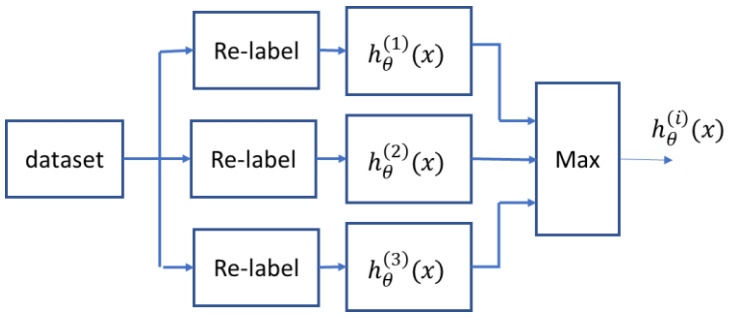


Fig.8. One-vs-All for multi-class classification

4.3 Performance Metrics

If we have developed a machine learning algorithm, how do we make sure it will work well? In this section, we will discuss how to evaluate the performance of a classifier. Sometimes an algorithm may work well on training set but not on new data examples which were not seen in the training set. However, we expect the algorithm to generalize to new data examples. Thus, to test the capability of the generalization, we need a testing set that has never been used for training. Some performance metrics are defined to measure the performance of an algorithm (or model).

4.3.1 Confusion matrix

After a machine learning algorithm has been tested using a test data set, we need to draw a conclusion on whether it works well by summarizing the test results. For example, in a classification task, we would like to know how many examples are classified correctly, and how many examples are misclassified into wrong classes. For a regression task, we may want to know the sum-of-squares error on the testing set. Here we focus on classification.

Confusion matrix is defined as a square matrix that includes the information on how many examples in each class are classified into different classes. Each row associated with an actual class in the data set while each column associated with a predicted class at the output of the algorithm. (Note that some textbook may define in opposite way: each row for a predicted class and each column associated with actual class).

Table 1 confusion matrix for a 3-class classification task

		Predicted output		
		Class 1	Class 2	Class 3
Actual class	Class 1	8	1	2
	Class 2	0	9	1
	Class 3	3	1	8

[Example 4.4] Table 1 shows an example of a 3×3 confusion matrix for 3-class classification task. The first row shows that, among 11 (8+1+2=11) examples of class 1, eight are correctly classified, one is incorrectly classified as class 2, and two are incorrectly classified as class 3. The second row similarly shows how class 2 examples are classified, and so on. Thus, the diagonal elements in the matrix are the numbers of examples correctly classified. The **overall accuracy**, defined as the

proportion of examples correctly classified, can be calculated through dividing the sum of diagonal elements by the sum of all elements in the confusion matrix. The accuracy is 0.7576 for the table.

[Example 4.5] For a 2-class classification problem, one class (e.g., class 1) is usually defined as positive while the other as negative. Thus, the confusion matrix can be interpreted as

Table 2. Confusion matrix for a 2-class classification task

	Predicted positive	**Predicted negative**
Actual class 1 (positive)	True positive (TP)	False negative (FN)
Actual class 2 (negative)	False positive (FP)	True negative (TN)

Class 1: positive
Class 2: negative
True positive (TP): the number of examples which are positive and are predicted to be positive
False negative (FN): the number of examples which are positive but are predicted to be negative
False positive (FP): the number of examples which are negative but are predicted to be positive
True negative (TN): the number of examples which are negative and are predicted to be negative

In summary, a confusion matrix is a summary of prediction results on a classification problem. The number of correct and incorrect predictions are summarized with count values and broken down by each class. The confusion matrix shows the ways in which your classification model is confused when it makes predictions. It gives us insight not only into the errors being made by a classifier but more importantly the types of errors that are being made. We will see in the next section that more important performance metrics can be derived from the confusion matrix.

4.3.2 Metrics for 2-class classification

Now let's look at more details on performance metrics in two-class classification tasks. The following metrics are defined and calculated based on the data in confusion matrix Table.2.

1) *Classification accuracy*:

$$accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{4.24}$$

Using accuracy assumes equal costs for both kinds of errors. In a situation where two types of error lead to different costs, accuracy may not be a good metric. For example, in a cancer diagnosis task, False negative and false positive will result in different costs. A patient will miss the required treatment if false negative occurs. A patient will be just be over treated if a false positive happens. The former case will have a larger cost than the latter case. Furthermore, accuracy is not a good metric when the dataset is very unbalanced. For instance, consider the classification of a very rare disease (positive or negative). In the dataset, there is only 1% patients in positive class and 99% in negative class. A bad algorithm that predicts any example to be negative will achieve an accuracy of 99%.

2) *Recall (true positive rate or sensitivity):*

$$recall = \frac{TP}{TP+FN} \tag{4.25}$$

Recall is the percentage of positive examples which are correctly predicted as positive. It indicates the probability of predicting an actual positive as a positive.

3) *Precision:*

$$precision = \frac{TP}{TP+FP} \tag{4.26}$$

Precision is defined as the ratio of the number of examples correctly classified as positive to the number of examples which are predicted as positive. It indicates the purity of true positive examples in the all predicted positive examples. High precision indicates an example predicted as positive is indeed positive (small FP).

"high recall and low precision" implies that most of the positive examples are correctly recognized, but there are a lot of false positive. "low recall and high precision" means that we miss a lot positive examples, but those we predict as positive are indeed truly positive.

4) *Specificity*: specificity is defined as the ratio of the total number of correctly classified negative examples to the total number of negative examples. In other words, specificity is the proportion of negative examples which are correctly recognized.

$$specificity = \frac{TN}{FP+TN} \tag{4.27}$$

5) *False positive rate*: this metric will be used to plot receiver operating characteristic (ROC) curve later. It indicates the probability of predicting an actual negative as a positive.

$$FPR = 1 - specificity = \frac{FP}{FP+TN} \tag{4.28}$$

6) *F-measure*:

We integrate two metrics (recall, precision) into one metric, called F-measure:

$$F = \frac{2 \times recall \times precision}{recall+precision} \tag{4.29}$$

The F-measure will always be nearer to the smaller value of recall or precision.

Among the above metrics, recall (or true positive rate), specificity, and false positive rate can be interpreted as probabilities of predictions, illustrated in Fig. 9.

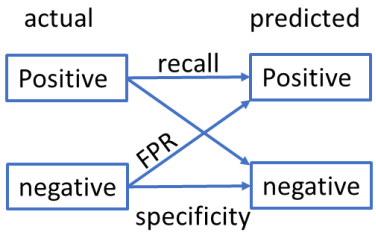


Fig.9 Metrics as probabilities

Below is the Python implementation of some metrics computation (optional):

```
# Python script for confusion matrix creation.
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
actual = [1, 1, 0, 1, 0, 0, 1, 0, 0, 0]
```

```
predicted = [1, 0, 0, 1, 0, 0, 1, 1, 1, 0]
results = confusion_matrix(actual, predicted)
print 'Confusion Matrix :'
print(results)
print 'Accuracy Score :',accuracy_score(actual, predicted)
print 'Report : '
print classification_report(actual, predicted)
```

4.3.3 Receive operating characteristic (ROC) curve

ROC stands for Receiver Operating Characteristic. Its origin is from sonar back in the 1940s; ROCs were used to measure how well a sonar signal could be detected from noise.

Like logistic regression, a typical classification algorithm outputs an estimated probability of "this example belongs to the positive class". For example, Fig.10 illustrates the outputs of three different classification algorithms for 30 testing examples. The position of each dot indicates the probability of a data example. The first classifier is perfect because the examples can be completely separated with an appropriate threshold. The second one is a typical classifier where there are always some misclassifications regardless the choice of threshold. The third classifier has a large number of misclassifications even with a best threshold.

In general, when we decrease the threshold, we get more positive predictions thus the recall (TPR) is increased, but FPR is also undesirably increased. The separability characteristics of three algorithms in Fig.10 can be distinguished by plotting their ROC curves as Fig.11. The ROC curve is obtained by plotting TPR vs. FPR when change threshold from 0 to 1. The area under the ROC curve (AUC) measures the separability of outputs. In other words, AUC represents the probability that a random positive (red) example is positioned to the right of a random negative (blue) example. The AUC of a perfect classifier is equal 1. The lowest AUC is equal to 0 for the case that all predictions are wrong. Thus, the higher the ROC curve, the better the performance.

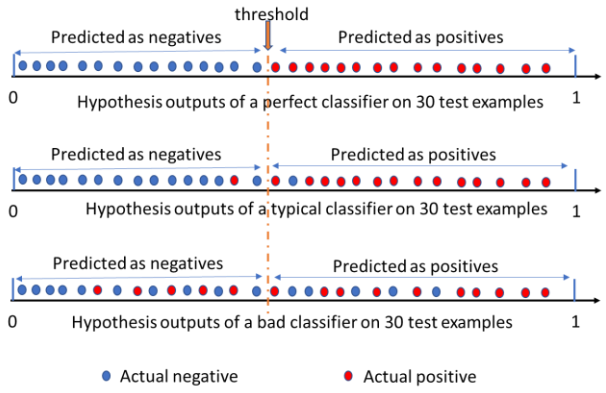


Fig.10 Probability outputs of three classifiers

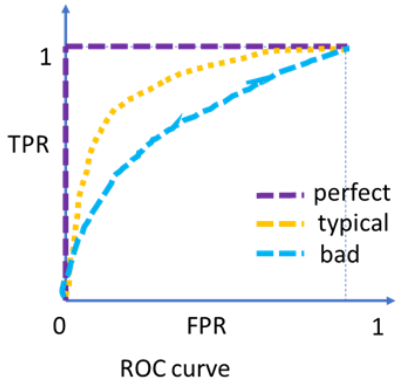


Fig.11 ROC curves

4.4 Implementation of Logistic Regression in Python

In this section, we will describe how to build a logistic regression classifier from scratch. The objective is to build a classifier that can predict whether an applicant will be admitted to the university or not, based on the two exam scores of the applicant. The data (marks.txt) has been taken from Andrew Ng's Machine Learning course on Coursera. The data consists of scores of two exams for 100 applicants. The target value takes on binary values 1 or 0. 1 means the applicant was admitted to the university whereas 0 means the applicant didn't get an admission. The complete code is attached as ***log_reg.py*** in the Appendix of this chapter.

The code development includes the following steps:

1) Load the dataset using pandas, and split the data set into two subsets (admitted and not_admitted) for visualization.
2) Prepare the data X, y: a "One" column vector is needed to add to the original data set.
3) Create functions for learning model: sigmoid(x), net_input(theta, x), h(theta,x), cost_function(theta, x, y), gradient(theta, x, y), gradient_descent(alpha, x, y, numIterations)
4) Learn the model by calling gradient_descent ( ) function
   theta, cost=gradient_descent(0.004,X,y,1000000)
   Note: be careful when selecting alpha, and numIterations. It is helpful to plot cost.
5) Plot the decision boundary line: $\theta_0 + \theta_1 x_1 + \theta_2 x_2 = 0$
6) An alternative way to learn the model by calling an advanced optimization function fmin_tnc() from scipy package.
7) Compare the two results. theta by gradient_decent is [[-24.70893451] [0.20261325] [0.19781088]], and theta by fmin_tnc is [-25.16143487 0.20623257 0.20147238]. The corresponding decision boundary lines are too close to distinguish. The gradient_decent approach is much slower than fmin_tnc, and alpha and the number of iterations should be carefully selected. However, when fmin_tnc is used, theta should be initialized to very small (zero or near zero) to avoid dividing by zero during log() computation
8) Make predictions for all examples and calculate accuracy of the model (0.89 implying 11 misclassifying out of 100 examples).
9) Finally, implement logistic regression using sklearn LogisticRegression with only two lines of code. Unlike the gradient descent code, it is noticed that the data X for fitting the model LogisticRegression should be m-by-n matrix, without ONE column being added to the original data.

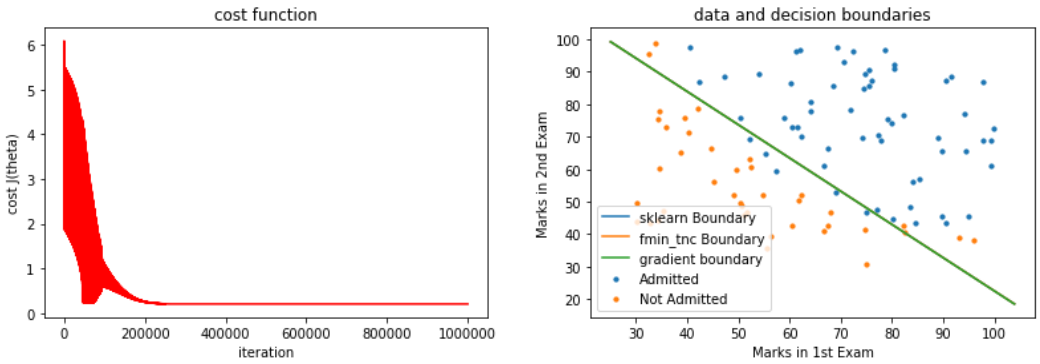


Fig.12 a) Cost plot versus iterations in gradient descent, b) data visualization and decision boundaries (three boundaries matched well).

The summary of results for the three approaches are printed by the program as follows:

```
Theta (θ)
___fmin_tnc: [-25.16131863   0.20623159   0.20147149]
___gradient: [-24.70893451   0.20261325   0.19781088]
___sklearn LogistiRegression: [-25.05219314] [[0.20535491 0.2005838 ]]

Accuracy:
___fmin_tnc: 0.89
___gradient: 0.89
___sklearn LogisticRegression: 0.89
```

Summary

In this chapter, we introduced the settings of classification problems and logistic regression. It is essential to understand the similarity and difference between linear regression and logistic regression, and why linear regression does not work well for a classification problem. A logistic regression classifier is composed of a linear regression and a non-linear function.

The non-linear function, sigmoid function, is the key to understanding the logistic regression. The output of the sigmoid function in a logistic regression, called hypothesis, $h_\theta(x)$, is interpreted as the probability of positive label (i.e., y=1). The gradient descent algorithm for logistic regression has been developed in the same way as for linear regression. The ln()-based cost function is the core for the algorithm development.

For multi-class classification problems, a strategy, called one-vs-all or one-vs-rest, was described. In a k-class classification problem, k logistic regression classifiers are designed. Each classifier labels a different class as positive (y=1) and the rest classes as negative (y=0). To predict the class for a new data example x, select the classifier that outputs a maximum hypothesis, and the corresponding positive labeled class is the predicted class.

The performance metrics of a classifier has been discussed. The accuracy of classification is the most commonly used metric. In addition, recall and precision are also important metrics especially for class-unbalanced data sets. ROC visualizes the performance of a classifier.

Finally, a project of logistic regression using python is detailed. Three ways of implementation are presented: 1) gradient descent from the scratch; 2) fmin_tnc optimization from scipy library; 3) LogisticRegression from sklearn library.

Important pieces of code are highlighted as follows:

1) fmin_tnc optimization

```
    # learn the model by scipy fmin_tnc
    from scipy.optimize import fmin_tnc
    def fit(self, x, y, theta):
        opt_weights = fmin_tnc(func=cost_function, x0=theta,
                      fprime=gradient,args=(x, y.flatten()))
        return opt_weights[0]

    parameters = fit(fit, X, y, theta)

```

2) LogisticRegression from sklearn

```
# sklearn LogisticRegression
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
model = LogisticRegression(solver='lbfgs')
model.fit(X[:,1:], np.ndarray.flatten(y))  # data array without ONE column
predicted_classes = model.predict(X[:,1:])
accuracy_sklearn = accuracy_score(y.flatten(),predicted_classes)
par_sklearn = model.coef_
intercept= model.intercept_
```

References

[1] www.coursera.org
[2] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani, Chapter 4, Classification, in *An introduction to Statistical Learning with Applications in R.* Springer.
[3] Scipy.optimize.fmin_tnc: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.fmin_tnc.html
[4] LogisticRegression: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

Exercises

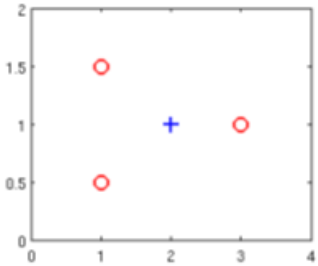
1. Suppose we want to predict, from data x about a tumor, whether it is malignant (y=1) or benign (y=0). Our logistic regression classifier outputs, for a specific tumor, $h_\theta(x) = P(y = 1|x;\theta) = 0.7$, so we estimate that there is a 70% chance of this tumor being malignant. What should be our estimate for $P(y = 0|x;\theta)$, the probability that the tumor is benign?
   a) $P(y = 0|x;\theta) = 0.3$
   b) $P(y = 0|x;\theta) = 0.7$
   c) $P(y = 0|x;\theta) = 0.3 \times 0.7$
   d) $P(y = 0|x;\theta) = 0.7^2$

2. Suppose that you have trained a logistic regression classifier, and it outputs on a new example x a prediction $h_\theta(x) = 0.4$. This means (check all the apply):
   a) Our estimate for $P(y = 0|x;\theta)$ is 0.4
   b) Our estimate for $P(y = 0|x;\theta)$ is 0.6
   c) Our estimate for $P(y = 1|x;\theta)$ is 0.4
   a) Our estimate for $P(y = 1|x;\theta)$ is 0.6

3. Suppose you have the following training set, and fit a logistic regression classifier

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

x_1	x_2	y
1	0.5	0
1	1.5	0
2	1	1
3	1	0



Which of the following statements are true? Check all that apply.

a) Adding polynomial features (e.g. instead using $h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_1 x_2 + \theta_5 x_2^2)$) could increase how well we can fit the training data.
b) At the optimal value of θ, we can have $J(\theta) \geq 0$.
c) Adding polynomial features (e.g. instead using $h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_1 x_2 + \theta_5 x_2^2)$) could increase $J(\theta)$ because we are now summing over more terms.
d) If we train gradient descent for enough iterations, for some examples $x^{(i)}$ in the training set it is possible to obtain $h_\theta(x^{(i)}) > 1$.

4. In logistic regression, the cost function for our hypothesis outputting (predicting) $h_\theta(x)$ on a training example that has label $y \in \{0,1\}$ is:

$$cost(h_\theta(x), y) = \begin{cases} -ln h_\theta(x) & if\ y = 1 \\ -ln(1 - h_\theta(x)) & if\ y = 0 \end{cases}$$

Which of the followings are true? Check all that apply.
a) If $h_\theta(x) = y$, then $cost(h_\theta(x), y) = 0$ $(for\ y = 0\ and\ y = 1)$
b) If y=0, then $cost(h_\theta(x), y) \to \infty\ as\ h_\theta(x) \to 1$
c) If y=0, then $cost(h_\theta(x), y) \to \infty\ as\ h_\theta(x) \to 0$
d) Regardless of whether y=0 or y=1, if $h_\theta(x) = 0.5$, then cost >0.

5. One iteration of gradient descent simultaneously performs these updates:

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)}) {x_0}^{(i)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)}) {x_1}^{(i)}$$

⋮

$$\theta_n := \theta_n - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right) x_n{}^{(i)}$$

We would like a vectorized implementation of the form $\theta := \theta - \alpha\delta$ (for some vector $\delta \in \mathbb{R}^{n+1}$) What should the vectorized implementation be? Denoting $x^{(i)} = [x_0{}^{(i)}\ x_1{}^{(i)} \ldots.\ x_n{}^{(i)}] \in \mathbb{R}^{1\times(n+1)}$

a) $\theta := \theta - \alpha \frac{1}{m} \sum_{i=1}^{m} [\left(h_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right)(x^{(i)})^T]$

b) $\theta := \theta - \alpha \frac{1}{m} [\sum_{i=1}^{m} \left(h_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right)]\ (x^{(i)})^T$

c) $\theta := \theta - \alpha \frac{1}{m} x^{(i)} [\sum_{i=1}^{m} \left(h_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right)]$

d) None of the above is correct implementation

6. For logistic regression, the gradient is given by

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^{m} [(h_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)}\right) - y^{(i)}) \cdot x_j^{(i)}]$$

Which of these is a correct gradient descent update for logistic regression with a learning rate of α? Check all that apply.

a) $\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right) \mathbf{x}^{(i)}$ (simultaneously update for all j).

b) $\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right) \mathbf{x}_j{}^{(i)}$ (simultaneously update for all j).

c) $\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(\frac{1}{1+e^{-\theta^T \mathbf{x}^{(i)}}} - y^{(i)}\right) \mathbf{x}_j{}^{(i)}$ (simultaneously update for all j).

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

7. Which of the following statements are true? Check all that apply.

a) The sigmoid function is never greater than one.

b) The cost function $J(\theta)$ for logistic regression trained with $m \geq 1$ examples is always greater than or equal to zero.

c) Linear regression always works well for classification if you classify by using a threshold on the prediction made by linear regress.

d) For logistic regression, sometimes gradient descent will converge to a local minimum (and fail to find the global minimum). This is the reason we prefer more advanced optimization algorithms such fmin_tnc().

8. Suppose you are running gradient descent to fit a logistic regression model with parameter $\theta \in \mathbb{R}^{n+1}$. Which of the following is a reasonable way to make sure the learning rate α is set properly and that gradient descent is running correctly?

a) Plot $J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$ as a function of the number of iterations and make sure $J(\theta)$ is decreasing on every iteration.

b) Plot $J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} [y^{(i)} \ln\left(h_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)}\right)\right) + (1 - y^{(i)}) \ln\left(1 - h_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)}\right)\right)]$ as a function of the number of iterations and make sure $J(\theta)$ is decreasing on every iteration.

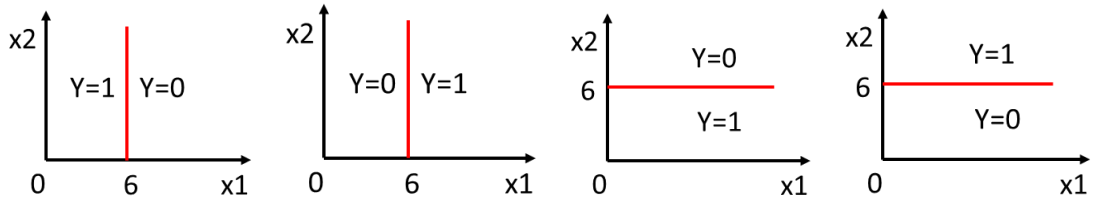
c) Plot $J(\theta)$ as a function of θ and make sure it is decreasing with θ.

d) Plot $J(\theta)$ as a function of θ and make sure it is convex.

9. Suppose you train a logistic classifier

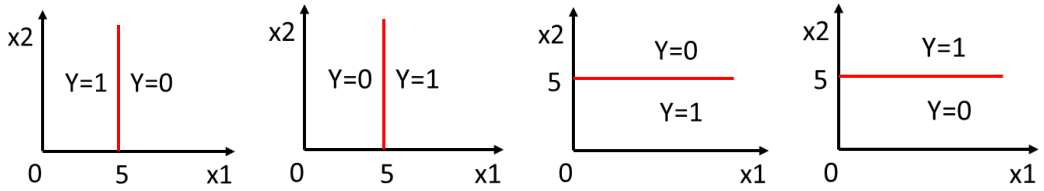
$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

and obtain $\theta_0 = -6, \theta_1 = 0, \theta_2 = 1$. Which of the following figures represents the decision boundary of your classifier?



a) b) c) d)

10. Consider logistic regression with two features x_1 and x_2. Suppose $\theta_0 = 5, \theta_1 = -1, \theta_2 = 0$, so that $h_\theta(x) = g(5 - x_1)$. Which of these shows the decision boundary of $h_\theta(x)$?



a) b) c) d)

11. Suppose you have a multi-class classification problem with *k* classes (so $y \in \{1,2,\dots,k\}$). Using the one-vs-all method, how many different logistic regression classifiers will you end up training?

a) k-1
b) k
c) k+1
d) Approximately $log_2(k)$

12. (Programming) In this exercise, you will generate a training dataset and a testing dataset from a probability model, and then learn a logistic regression classifier using the training set, and test your trained classifier on the testing set.

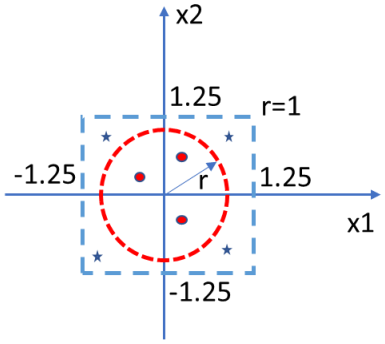
    1) The data examples in a form of $\left(x^{(i)}, y^{(i)}\right), i = 1,2,\dots,m$ are generated based on the following Gaussian models
    $p(x|y=0,\mu_0,\sigma_0^2) \sim N(\mu_0,\sigma_0^2)$ with $\mu_0 = 1, \sigma_0^2 = 4$
    $p(x|y=1,\mu_1,\sigma_1^2) \sim N(\mu_1,\sigma_1^2)$ with $\mu_1 = 4, \sigma_0^2 = 9$

    Generate a training set consisting of 50 examples for y=0 and 50 examples for y=1, a total of 100 examples.
    Generate a testing set consisting 25 examples for y=0 and 25 examples for y=1, a total of 50 examples.
    Note: you can plot the histogram of x for each set to validate your data.

2) Thus, the data set has two classes and one input feature x. Train a logistic regression model to fit the training dataset. The initial parameters are suggested as $\theta_0 = -2, \theta_1 = 1$. What are your resulting parameters θ_0, θ_1? What is the accuracy on the training set? Plot the cost function versus iteration index.
3) Test your logistic regression model using the testing set generated in 1), and find the accuracy on the testing set.

13. (programming) (hw3)
14. (Programming) In this exercise, you will train a logistic regression model using the training set generated in the way illustrated in the figure below.

    1) A data example $\left(x_1^{(i)}, x_2^{(i)}, y^{(i)}\right), i = 1,2,..,m$, can be generated by the following steps:
        a) Generate a pair of random numbers $\left(x_1^{(i)}, x_2^{(i)}\right)$, $x_1^{(i)}, x_2^{(i)}$ are independently drawn from the uniform distribution between -1.25 and 1.25.
        b) The label $y^{(i)}$ is assign to 1 (shown by star) if $\sqrt{\left(x_1^{(i)}\right)^2 + \left(x_2^{(i)}\right)^2} > 1$, and assigned to 0 (shown by red dot) otherwise.
    2) Generate a training set consisting of 100 examples, and plot the scatterplot of the training set.
    3) Since it is known that the decision boundary is quadratic (not linear), you will use the model $h_{\boldsymbol{\theta}}(\mathbf{x}) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 + \theta_5 x_2^2)$. Train this model, and find the resulting parameters $(\theta_0, \theta_1, \theta_2, \theta_3, \theta_4, \theta_5)$. Carefully select the initial parameter values but they should not be the optimal solution $(-1,0,0,0,1,1)$. Plot the cost function versus iteration index.



15. Which of the following statements are True?
    1) In binary classification, high recall and low precision implies that most of the positive examples are correctly recognized, but there are very few negative examples correctly recognized.
    2) In binary classification, high recall and low precision implies that most of the positive examples are correctly recognized, but there are a lot of false positive.
    3) In binary classification, low recall and high precision means that we miss a lot of positive examples, but those we predict as positive are indeed truly positive.

4) F-measure is always nearer to the bigger value of recall or precision.

5) F-measure is an arithmetic average of recall and precision.

6) F-measure is always nearer to the smaller value of recall or precision.

7) The range of F-measure is from 0 to 1.

16. In a testing set, we have 999 data examples of negative class and a single example of positive class, and the classifier is predicting all examples to be negative.

    1) Draw the confusion matrix.

    2) Calculate accuracy, recall, precision and F-measure.

17. In a testing set, we have 999 data examples of negative class and a single example of positive class, and the classifier is predicting all examples to be positive.

    1) Draw the confusion matrix.

    2) Calculate accuracy, recall, precision and F-measure.

18. After a binary classifier is tested using a testing data set, a confusion matrix is obtained as

	Predicted as positive	Predicted as negative
Actual positive	100	5
Actual negative	10	50

    1) How many examples are there in the testing set?

    2) Calculate classification accuracy, recall, precision, and F-measure.

19. The testing result of a binary classifier is shown in the following table by comparing the actual label y and the predicted label $\hat{y}$:

y	1	1	0	1	0	0	1	0	0	0
$\hat{y}$	1	0	0	1	0	0	1	1	1	0

    1: positive, 0: negative

    1) Find the confusion matrix.

    2) Calculate accuracy, recall, precision, and F-measure.

    3) (optional) Verify your above answers using Python sklearn library.

20. The following table shows the testing result for a classifier. Find accuracy, recall, sensitivity, true positive rate, specificity, false positive rate and F-measure.

index	actual	predicted	
1	0	0	
2	0	0	
3	0	0	
4	0	0	TN
5	0	0	
6	0	0	
7	0	0	
8	0	1	
9	0	1	FP
10	0	1	
11	1	0	
12	1	0	FN
13	1	1	
14	1	1	
15	1	1	
16	1	1	
17	1	1	TP
18	1	1	
19	1	1	
20	1	1	

21. The testing results of a binary classifier are shown in the following table. The labels and hypothesis outputs are listed for a test set of 20 examples.
    1) Calculate accuracy, recall (or sensitivity or TPR), precision, specificity, FPR, F-measure, when threshold is 0.4.
    2) Calculate accuracy, recall (or sensitivity or TPR), precision, specificity, FPR and F-measure, when threshold is 0.6.
    3) Sketch the ROC of the classifier.

y	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**
$h_\theta(x)$	**0.01**	**0.02**	**0.10**	**0.21**	**0.26**	**0.34**	**0.35**	**0.45**	**0.48**	**0.55**

y	**0**	**1**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
$h_\theta(x)$	**0.59**	**0.60**	**0.71**	**0.83**	**0.84**	**0.89**	**0.91**	**0.93**	**0.94**	**0.97**

Appendix

log_reg.py

```
"""
Created on Fri Apr  5 09:22:00 2019
log_reg.py
@author: mll444
"""

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# load data by pd.read_csv
dataset_pd = pd.read_csv("marks.txt", header=None)

# visualize the data
y_pd=dataset_pd.iloc[:,-1]
X_pd=dataset_pd.iloc[:,:-1]
X_admit = X_pd.loc[y_pd==1]
X_not_admit = X_pd.loc[y_pd==0]
plt.scatter(X_admit.iloc[:, 0], X_admit.iloc[:, 1], s=10, label='Admitted')
plt.scatter(X_not_admit.iloc[:, 0], X_not_admit.iloc[:, 1], s=10, label='Not Admitted')
plt.legend()
plt.xlabel('Marks in 1st Exam')
plt.ylabel('Marks in 2nd Exam')
plt.title('data visualization')
plt.show()

# prepare data in np.ndarray format
X=np.c_[np.ones((X_pd.shape[0],1)), X_pd]   # X: np.ndarray (100,3)  (m,n+1)
y = y_pd[:, np.newaxis]                     # y: np.ndarray (100,1)  (m,1)
theta = np.zeros((X.shape[1], 1))           # theta: np.ndarray (3,1) (n+1,1)
# to avoid dividing by zero when computing log() in cost_function,
# theta1, theta2... should be
# initialized to very small numbers.
# If dividing by zero happens, the fmin_tnc won't work,
# but my gradient_decent still work


def sigmoid(x):
   return 1 / (1 + np.exp(-x))

def net_input(theta, x):
    # Computes the weighted sum of inputs
    return np.dot(x, theta)
def h(theta, x):
    # Returns the probability after passing through sigmoid
    return sigmoid(net_input(theta, x))

#print (np.dot(np.transpose((np.log(h(theta,X)))),y))
# check log and h() function
#print (np.dot(np.transpose((np.log(1-h(theta,X)))),1-y))
#print(net_input(theta,X).shape)
def cost_function(theta, x, y):
    # Computes the cost function for all the training samples
```

```
        m = x.shape[0]
        t1=np.dot(np.transpose((np.log(h(theta,X)))),y)
        t2=np.dot(np.transpose((np.log(1-h(theta,X)))),1-y)
        total_cost = -(1 / m) * (t1+t2)
        return total_cost

    def gradient(theta, x, y):
        # Computes the gradient of the cost function at the point theta
        m = x.shape[0]
        return (1 / m) * np.dot(x.T, h(theta,x) - y)

    #print(cost_function(theta=theta,x=X,y=y))
    #check the cost function [[0.69314718]]

    #print(gradient(theta=theta,x=X,y=y))
    #check gradient function[[ -0.1       ]
    #[-12.00921659]
    #[-11.26284221]]

    def gradient_descent(alpha, x, y, numIterations):
        cost=[]
        #m = x.shape[0] # number of samples
        theta = np.zeros((x.shape[1], 1))
        #x_transpose = x.transpose()
        for iter in range(0, numIterations):
            theta=theta-alpha*gradient(theta,x,y)
            J=cost_function(theta,x,y)
            cost.append(J)
        return theta,cost;


    # learn the model by scipy fmin_tnc
    from scipy.optimize import fmin_tnc
    def fit(self, x, y, theta):
        opt_weights = fmin_tnc(func=cost_function, x0=theta,
                      fprime=gradient,args=(x, y.flatten()))
        return opt_weights[0]

    parameters = fit(fit, X, y, theta)
    #print("theta by fmin_tnc:", parameters)
    #[-25.16131863   0.20623159   0.20147149]


    # learn the model by gradient descent
    theta, cost=gradient_descent(0.004,X,y,1000000)
    #print("theta by gradient_decent:", theta)
    cost1=np.ndarray.flatten(np.array(cost))
    plt.plot(cost1, color='red')
    plt.xlabel('iteration')
    plt.ylabel('cost J(theta)')
    plt.title('cost function')
    plt.show()


    # predict and accuracy
    h_pred=np.dot(X,theta)
    y_pred=[h_pred>0]
    correct=[y_pred==y]
    accuracy_gradient=np.sum(correct)/len(correct[0][0])
    h_pred=np.dot(X,np.reshape(parameters,(3,1)))
    y_pred=[h_pred>0]
```

```
    correct=[y_pred==y]
    accuracy_fmin_tnc=np.sum(correct)/len(correct[0][0])

    # sklearn LogisticRegression
    from sklearn.linear_model import LogisticRegression
    from sklearn.metrics import accuracy_score
    model = LogisticRegression(solver='lbfgs')
    model.fit(X[:,1:], np.ndarray.flatten(y))  # data array without ONE column
    predicted_classes = model.predict(X[:,1:])
    accuracy_sklearn = accuracy_score(y.flatten(),predicted_classes)
    par_sklearn = model.coef_
    intercept= model.intercept_
    #print('accuracy of LogisiticRegression:', accuracy_sklearn)
    #print('theta and intercept:', par_sklearn, intercept)

    # plot decision boundaries
    x_values = [np.min(X[:, 1] - 5), np.max(X[:, 2] + 5)]
    y_values = - (parameters[0] + np.dot(parameters[1], x_values)) / parameters[2]
    x_values = [np.min(X[:, 1] - 5), np.max(X[:, 2] + 5)]
    y_values_my = - (theta[0][0] + np.dot(theta[1][0], x_values)) / theta[2][0]
    y_values_sklearn = - (intercept + np.dot(par_sklearn[0][0], x_values)) / par_skl
    earn[0][1]
    plt.plot(x_values, y_values_sklearn, label='sklearn Boundary')
    plt.plot(x_values, y_values, label='fmin_tnc Boundary')
    plt.plot(x_values, y_values_my, label='gradient boundary')
    plt.scatter(X_admit.iloc[:, 0], X_admit.iloc[:, 1], s=10, label='Admitted')
    plt.scatter(X_not_admit.iloc[:, 0], X_not_admit.iloc[:, 1], s=10, label='Not Adm
    itted')
    plt.xlabel('Marks in 1st Exam')
    plt.ylabel('Marks in 2nd Exam')
    plt.legend()
    plt.title('data and decision boundaries')
    plt.show()

    # Print the summary of resutls
    print('theta')
    print('___fmin_tnc:', parameters)
    print('___gradient:', theta.flatten())
    print('___LogistiRegression:', intercept, par_sklearn, '\n')
    print('Accuracy')
    print('___fmin_tnc:', accuracy_fmin_tnc)
    print('___gradient:', accuracy_gradient)
    print('___LogisticRegression:',accuracy_sklearn)
```