Chapter 4

Classification and Logistic Regression

In the previous chapter, we studied linear regression that predicts a label whose value is continuous. In this chapter, we will learn another type of supervised learning – classification, where the label has discrete (categorical) values. In a classification task, the model maps a specific input to a specific category. *Logistic regression*, an extension of linear regression, can perform some simple classification tasks. However, a typical classification task in practice, for example, image recognition, may require larger models such as deep neural networks. In subsequent chapters, we will see that a logistic regression classifier is a neuron that is the basic element of neural networks. Thus, logistic regression is a foundation of neural networks.

In this chapter, you will learn

- The settings of a classification task
- Logistic regression model
- Cost function and gradient descent algorithm for training logistic regression models
- Metrics of classification performance
- How to build and train a regression model in Python

4.1 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 **x** and to map it to one of K discrete categories or classes, C_k, $k=1,2,..,K$. Thus, classification is a function that maps data examples to finite categories. In other words, some data examples with certain common characteristics are grouped into the same category. The input space is thereby divided into decision regions by boundaries, which are called *decision boundaries* or *decision surfaces*. For example, the label variable y for two-category classification can be coded in binary, i.e. $y \in \{0,1\}$, which implies that y=0 represents class C_1 and y=1 represents class C_2. 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 C_3 would be $y=[0,0,1,0,0]^T$ in a one-hot code. 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 feature	y=0 (benign), y=1 (malignant)

4.2.2 Logistic regression model

Consider a binary classification problem (i.e., K=2). If the examples in the training dataset are linearly separable, there exists a linear surface, called decision boundary whose one side is associated with one class and another side with another class, as illustrated in Fig.4.1. This decision boundary can be represented by $\boldsymbol{\theta}^T\mathbf{x} = 0$, where $\boldsymbol{\theta}$ is the parameter vector and $\mathbf{x}$ is the input feature vector variable.

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

where x_j, *j=1,2,…, n* is the jth feature. We define a dummy feature $x_0 = 1$. For example, we consider *n*=2, and let $\boldsymbol{\theta}^T\mathbf{x} = \mathbf{0}$ be the decision boundary, shown in Fig.1. An input vector $\mathbf{x}$ is assigned to one class (say class2) if $\boldsymbol{\theta}^T\mathbf{x} \geq 0$, or 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 $\mathbf{x}$ to the decision boundary by

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

If data examples are linearly separable and an optimal decision boundary has been found, illustrated in Fig.4.1, 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.

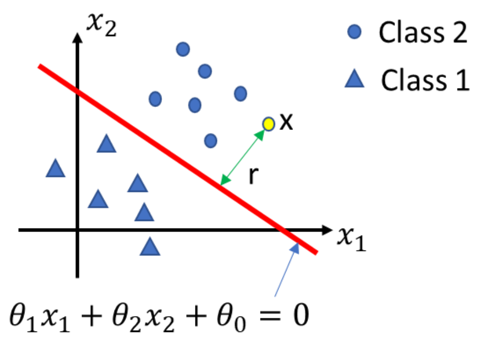


Fig. 4.1 Illustration of linear decision boundary for a two-class classification with two features.

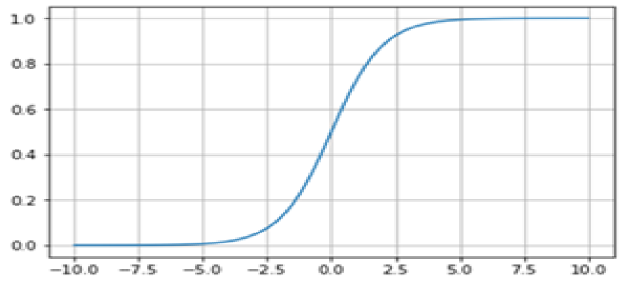


Fig.2 Sigmoid function g(z)

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*. A popular choice of activation function is sigmoid function, shown in Fig.4.2, defined as

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

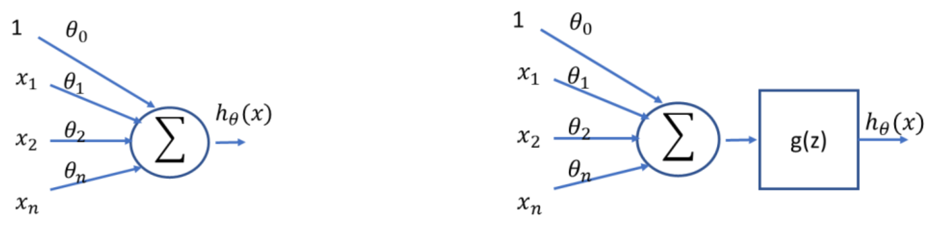
When *z=0, g(z)* is equal to 0.5. When *z* departures from zero to the positive side, *g(z)* quickly approaches 1. When *z* moves from zero to the negative side, *g(z)* will quickly approaches 0. It is worthy to note that any other functions with similar shape, e.g. tanh(), can be used for activation function.

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 $h_{\boldsymbol{\theta}}(\mathbf{x}) = g\left(\boldsymbol{\theta}^{T}\mathbf{x}\right)$ 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 instead we pass it through an activation function that maps the input linear combination to the probabilistic space. Fig.4.3 illustrates the block diagrams for linear regression and logistic regression, where **θ** is the weight parameter vector.



(a) Linear regression (b) Logistic regression

Fig.4.3 Comparison between linear regression and logistic regression

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 $\mathbf{x}$ and $\boldsymbol{\theta}$, the hypothesis for the logistic regression, which is defined as the output of logistic regression, is given by

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

For a two-class classification task, we usually label one class as y=0 and another class as y=1. For a sake of mathematic convenience, the class labeled as y=1 corresponds to $h_{\boldsymbol{\theta}}(\mathbf{x}) > 0.5$. Thus, $h_{\boldsymbol{\theta}}(\mathbf{x})$ can be interpreted as the estimated probability of *y=1*, denoted by $p(y = 1|\mathbf{x};\theta)$. The probability of x belongs to class y=0 can be predicted as $p(y = 0|\mathbf{x};\theta) = 1 - h_{\boldsymbol{\theta}}(\mathbf{x})$.

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., $\boldsymbol{\theta}^{T}\mathbf{x} = 0$, as shown in Fig.4.4 (a). If the input examples are nonlinear separable as shown in Fig.4.4(b) and (c), we can generalize (4.5) to a nonlinear function of $\mathbf{x}$.

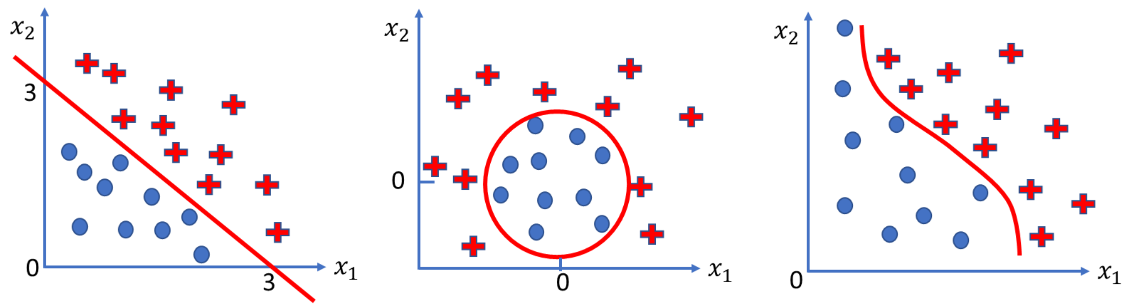


Fig.4(a) a linear decision boundary, (b) a circle decision boundary, (c) high-order non-linear decision boundary.

Fig.4(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, for example,

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix} = \begin{bmatrix} -3 \\ 1 \\ 1 \end{bmatrix}$$

We will predict y=1 for any data point (x_1, x_2) with $\theta_0 + \theta_1 x_1 + \theta_2 x_2 \geq 0$, 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.4(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.4(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.

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

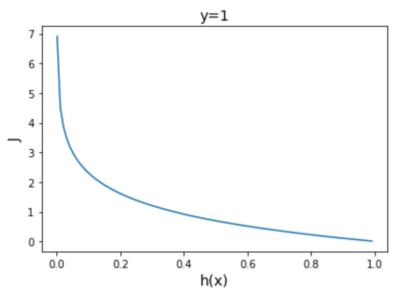
In this section, we will discuss how to fit the parameter **θ** so that the decision boundary can effectively classify the data examples. Given a value of **θ** and a value of **x**, the corresponding label *y* is predicted according to (4.7). In general, for a dataset, some predictions match the true labels while other predictions do not. This implies that the value of **θ** is not the optimal one or/and the data examples are not perfectly separable by the decision boundary. 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** (given a dataset), 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 **θ**, 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 **θ** space. The cost function of logistic regression, for a single data example (*x*,*y*), is defined as

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

where *y* is the value of the label. We can understand the cost function in an intuitive 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, $-\ln\big(h_{\boldsymbol{\theta}}(\mathbf{x})\big)$, has this characteristic, shown in Fig.4.5(a). Similarly, if the label y is 0, the penalty should be smaller when the hypothesis $h_{\boldsymbol{\theta}}(\mathbf{x})$ is close to 0. The function, $-\ln\big(1 - h_{\boldsymbol{\theta}}(\mathbf{x})\big)$, has this characteristic, shown in Fig.4.5(b).



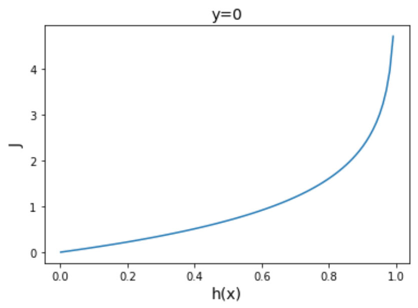


Fig.4.5. cost function $J(h_{\boldsymbol{\theta}}(\mathbf{x}), y)$, a) y=1, b) y=0

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

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

The cost function for *m* examples, $\big(\mathbf{x}^{(1)}, y^{(1)}\big)$, $\big(\mathbf{x}^{(2)}, y^{(2)}\big)$, … , $\big(\mathbf{x}^{(m)}, y^{(m)}\big)$, will be the average cost over the training set, given by

$$\begin{aligned} J(\boldsymbol{\theta}) &= \frac{1}{m}\sum_{i=1}^{m} J\big(h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}), y^{(i)}\big) \\ &= -\frac{1}{m}\sum_{i=1}^{m}\Big[y^{(i)} \ln\Big(h_{\boldsymbol{\theta}}\big(\mathbf{x}^{(i)}\big)\Big) + (1-y^{(i)})\ln\Big(1 - h_{\boldsymbol{\theta}}\big(\mathbf{x}^{(i)}\big)\Big)\ \Big] \end{aligned} \tag{4.10}$$

where

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

Note that the cost (4.10) is the binary cross entropy loss defined by (2.123).

To fit parameters, we minimize the cost function with respect to $\boldsymbol{\theta}$, i.e., $\boldsymbol{\theta}^* = arg \min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$. 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 a problem of two-class classification with the following assumptions: 1) the probability $p(y=1|\mathbf{x};\boldsymbol{\theta}) = h_{\boldsymbol{\theta}}(\mathbf{x})$ and $p(y=0|\mathbf{x};\boldsymbol{\theta}) = 1 - h_{\boldsymbol{\theta}}(\mathbf{x})$, 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**

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

The logarithm likelihood function can be obtained by applying ln(.) operation to (2.13)

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

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

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

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

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

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

Gradient descent algorithm

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

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

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

It can be shown that the partial derivative of J($\boldsymbol{\theta}$) 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 the ith example, and $x_0^{(i)} = 1$. Please note that (4.19) has the same form as (3.14) 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 $\mathbf{X}$. In matrix X, each row represents a data example while each column corresponds to a feature. Y is the label vector. The sigmoid function $g(\quad)$ is applied to $\mathbf{X} \cdot \boldsymbol{\theta}$ in an element wise style. Like the gradient decent algorithm for linear regression, we can have a gradient decent algorithm for logistic regression as follows:

Gradient decent algorithm for logistic regression

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

There are two limitations for logistic regression models:

1) The decision boundary is linear with input features. This will severely limit its applications in practice.
2) The model is limited to two-class classification tasks.

Thus, logistic regression model has very limited applications. The significance of logistic regression model is that it is the basic element, called neuron, in neural networks. As we will see in the next chapter, a neural network is a network of many logistic regression units. A neural network can achieve a complicated nonlinear decision boundary, and multi-class classification tasks.

4.2 Performance Metrics for Classification

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 for new data examples. Thus, to test the capability of the generalization, we need a testing dataset that has never been used for training.

4.2.1 Metrics for 2-class classification

First, let’s look at performance metrics in two-class classification tasks. Suppose a classifier has been tested on a testing dataset. The testing dataset consists of *m* examples. The testing result is illustrated by Fig.4.6. The “ground truth” box shows the distribution of examples among two classes (blue for class 1, and red for class 2). The “results” box shows how many examples from each class are classified into two classes. For a convenience, we refer examples in class 1 as positive and examples in class 2 as negative. Fig.4.6 tells us that

1) TP (true positive) examples in class 1 are correctly classified into class 1 by the classifier,

2) FP (false positive) examples in class 2 are incorrectly classified into class 1,

3) TN (true negative) examples in class 2 are correctly classified into class 2,

4) FN (false negative) examples in class 1 are incorrectly classified into class 2.

Note that the total number of examples $m = TP + FP + TN + FN$. There are $TP + FN$ examples in class 1 (positive) while $FP + TN$ examples in class 2 (negative).

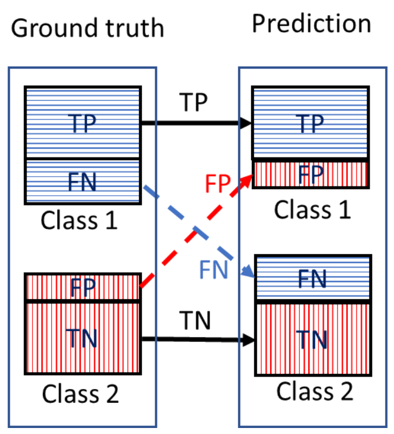


Fig.4.6 A binary classification result

There are several metrics to measure the classification performance with different emphases. A basic performance metric is *classification accuracy*, defined as the percentage of examples correctly classified

$$accuracy = \frac{TP+TN}{m} \tag{4.22}$$

However, it makes no sense to use the metric *accuracy* for some scenarios. For example, consider the classification of a very rare disease as 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%. When the costs of two types of misclassifications (positive classified as negative and negative classified as positive) are very different, the overall accuracy is less meaningful. 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 over treated if a false positive happens. Obviously, the false negative diagnosis will lead to a much more serious consequence than the false positive. The following metrics may be more suitable for some special scenarios.

Recall, also called *true positive rate* or *sensitivity*, is defined as the percentage of positive examples which are correctly predicted as positive

$$recall = \frac{TP}{TP+FN} \tag{4.23}$$

Recall indicates the probability of a positive example being correctly detected.

Precision is defined as the ratio of the number of examples correctly classified as positive to the total number of examples which are predicted as positive. It indicates the purity of true positive examples in all predicted positive examples.

$$precision = \frac{TP}{TP+FP} \tag{4.24}$$

There is usually a trade-off between recall and precision. “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.

While *recall* is the percentage of positive examples which are correctly detected, *Specificity* is the percentage of negative examples which are correctly detected.

$$specificity = \frac{TN}{FP+TN} \tag{4.25}$$

False positive rate (FPR) is the percentage of negative examples which are incorrectly classified as positive. It indicates the probability of predicting an actual negative as a positive.

$$FPR = 1 - specificity = \frac{FP}{FP+TN} \tag{4.26}$$

F-score (or F-measure) is an integrated metric of recall and precision and defined as the harmonic mean of the precision and recall.

$$F = \frac{2 \times recall \times precision}{recall + precision} = \frac{2}{recall^{-1} + precision^{-1}} \tag{4.27}$$

The F-score will always be nearer to the smaller value of recall or precision. The highest possible value of an F-score is 1.0, indicating perfect precision and recall, and the lowest possible value is 0, if either precision or recall are zero.

Among the above metrics, recall (or true positive rate), specificity, and false positive rate can be interpreted as probabilities of predictions, illustrated in Fig. 4.7.

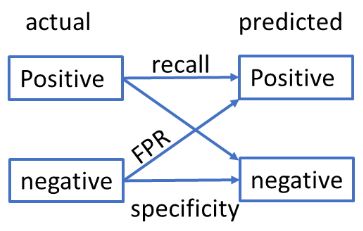


Fig.4.7 Metrics as probabilities

4.2.2 Metrics for multi-class classification

Confusion matrix is defined as a square matrix that includes the information on how many examples in each class are classified into each class. 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).

For example, Table 4.1 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 percentage of total examples correctly classified, can be calculated as dividing the sum of diagonal elements by the sum of all elements in the confusion matrix. The accuracy is 0.7576 for the table.

Table 4.1 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

The definitions for recall, precision, F-score for the binary classification can be generalized for multi-classifications. In a multi-classification task, there is a recall, precision, F-score defined for each class. When we define these metrics for a particular class, we treat this class as positive and all other classes as negative.

In many multi-classification applications, the classifier outputs the probabilities over classes and the predicted class is the one with the highest probability. Sometimes we use the top-1 accuracy and the top-5 accuracy to measure the classification accuracy. For the top-1 accuracy (the strictest match), we check if the top class (the one with the highest probability) is the same as the target label. For the top-5 accuracy (the relaxed match), we check if the target label matches one of your top 5 predictions (the 5 classes with the top-5 probabilities). In both cases, the error rate is computed as the number of times a predicted label NOT matched the target label, divided by the number of data points evaluated.

4.2.3 Receive operating characteristic (ROC) curve

ROC stands for Receiver Operating Characteristic. It originates from sonar back in the 1940s; ROCs were used to measure how well a sonar signal could be detected from noise. Regardless of the origin, in machine learning an ROC curve is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters: true positive rate (i.e., recall) and false positive rate.

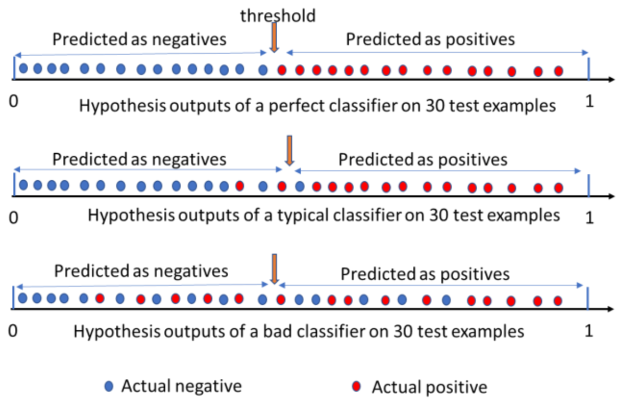

threshold
Predicted as negatives
Predicted as positives
0
Hypothesis outputs of a perfect classifier on 30 test examples
1
Predicted as negatives
Predicted as positives
0
Hypothesis outputs of a typical classifier on 30 test examples
1
Predicted as negatives
Predicted as positives
0
Hypothesis outputs of a bad classifier on 30 test examples
1
Actual negative
Actual positive

Fig.4.8 Probability outputs of three classifiers on 30 examples

Consider a binary classification task. A typical classification model outputs an estimated probability of "the current input example belongs to the positive class", and then a threshold is applied to decide the predicted class based on the probability. A default threshold is 0.5 for our logistic regression model. However, in general, an alternative threshold may exist to achieve the best classification performance, depending on the probability distribution over examples. Furthermore, different models may generate different probability distribution over a particular testing set of examples. For example, Fig.4.8 illustrates the outputs of three different models on the same 30 testing examples. The examples are labeled as red for positive class and blue for negative class. The position of each example indicates the probability of a data example being positive. The first model 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 model has a large number of misclassifications even with a best threshold.

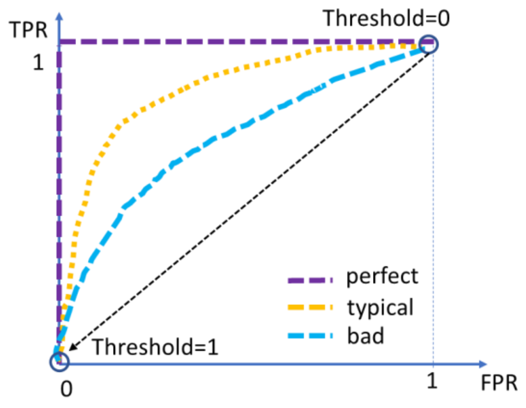


Fig.4.9 ROC curves for three classifiers

In general, when we decrease the threshold, we get more positive predictions thus the recall (TPR) is increased, but at the same time the FPR (false positive rate) is also undesirably increased. The separability characteristics of three models in Fig.4.8 can be distinguished by plotting their ROC curves as Fig.8. The ROC curve is obtained by plotting TPR vs. FPR when changing the 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 randomly selected positive (red) example is located to the right of a random negative (blue) example. The AUC of a perfect classifier is equal to 1. The lowest AUC is equal to 0 for the case that all predictions are wrong. Thus, the higher the AUC of the ROC curve, the better the performance.

4.2 Implementation of Logistic Regression in Python

In this section, by an example, we will demonstrate how to build and train a logistic regression classifier in Python. The problem is to predict whether a student will get a grade of A or not, based on the scores of two exams during the semester for a particular course. Please note that the actual grade will be calculated as the weighted sum of the scores of two exams, final exam, and homework assignments. Suppose we have the grades and scores of students in the previous semester for the same course. This information is used to construct a training dataset and a testing dataset, as shown below. Thus, each example in training dataset is represented by triplet (x_1, x_2, y), where (x_1, x_2)

are the scores of two exams for one student, and y is the label ($y = 1$ indicates that the student got A). For convenience, we save the examples as *grades_train.txt* and *grades_test.txt*, for training set and testing set, respectively.

Training set

60,83,1	65,87,0	85,90,1
65,84,1	90,75,0	57,78,1
85,10,0	48,75,1	33,63,0
60,91,1	23,72,0	60,75,1
70,62,0	43,71,0	92,97,1
100,96,1	62,96,1	60,44,0
40,83,1	75,95,0	

Testing set

62,73,0	12,57,0	85,77,1
39,79,0	51,55,0	39,51,0
50,86,1	68,52,0	
45,83,1	57,90,1	

Import packages

```python
import numpy as np
import matplotlib.pyplot as plt
```

Then, we load the training data to the Numpy array `dataset`. The input features are assigned to `X`, and labels are assigned to `y`. Furthermore, inputs for label `1` are saved in `X_1` and inputs for label `0` are saved in `X_0`. For this dataset, we can check that X.shape is (20,2), y.shape is (20,), X_1.shape is (11,2), X_0.shape is (9,2).

```python
# load training data
dataset = np.loadtxt("grades_train.txt", delimiter=",")
X = dataset[:, :-1]
y = dataset[:, -1]
X_1 = X[np.where(dataset[:,-1]==1)]
X_0 = X[np.where(dataset[:,-1]==0)]
```

Similarly, we load the testing data.

```python
# load testing data

dataset_test = np.loadtxt("grades_test.txt", delimiter=",")
X_test = dataset_test[:, :-1]
y_test = dataset_test[:, -1]
X_1_test = X_test[np.where(dataset_test[:,-1]==1)]
X_0_test = X_test[np.where(dataset_test[:,-1]==0)]
```

The following functions are defined for the gradient descent algorithm for logistic regression in a hierarchical style.

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

def h(theta, x):
    # Returns the probability after passing through sigmoid
    return sigmoid(np.matmul(x,theta))
```

```
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.matmul(x.T, h(theta,x) - y)
```

```
def gradient_descent(alpha, x, y, numIterations):
    # x.shape=(m,n+1), the first column is one
    # y.shape=(m,1)
    # theta.shape (n+1,1)
    cost=[]

    theta = np.zeros((x.shape[1], 1))

    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;
```

According to the above functions, the input feature matrix x includes the extra column with all "one" (thus x has a shape of *m* rows and *n+1* columns), and y is a matrix of (*m*, 1), where *m* is the number of examples. Before we call the `gradient_descent()`, we need to prepare the input feature matrix `X_add_ones` and label matrix `y_2d`. We should select appropriate values for learning rate (alpha and the number of iterations) to achieve desirable results. The cost is plotted in Fig. 4. 10 to monitor the convergence speed of the algorithm.

```
# prepare data in np.ndarray format
X_add_ones=np.c_[np.ones((X.shape[0],1)), X]    # add 1 to X: np.ndarray (m,n+1)
y_2d=np.expand_dims(y, axis=1) # y: (m,), y_2d: (m,1)

# learn the model by gradient descent
theta, cost=gradient_descent(0.0009,X_add_ones,y_2d,400000)
# X_add_ones: (m, n+1)
# y_2d: (m,1)
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()
```

```
theta by gradient_decent: [[-6.55274552]
 [-0.00865777]
 [ 0.09343879]]
```

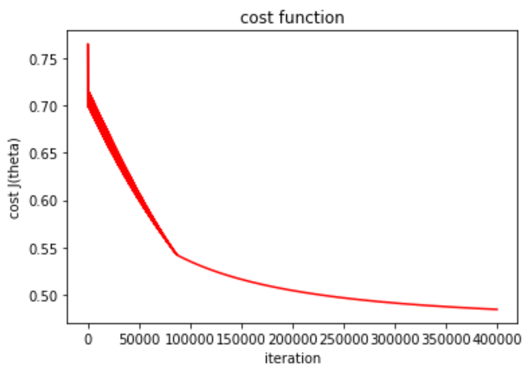


Fig.4.10 Cost plot during the training.

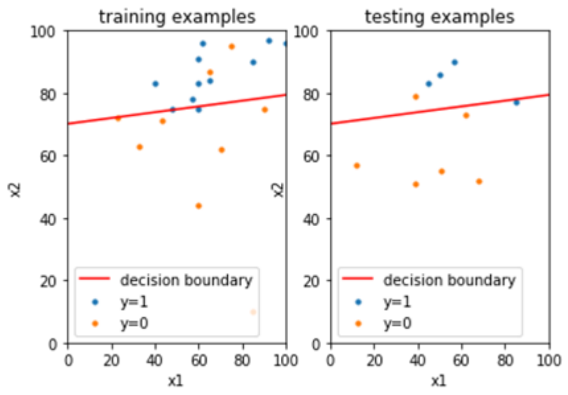


Fig.4.11 Plots for decision boundary

To visualize the results, we plot the decision boundary: $\theta_0 + \theta_1 x_1 + \theta_2 x_2 = 0$, along with data examples, as Fig.4.11. From Fig.4.11, we can see how the data examples are separated by the decision boundary.

```
# plot decision boundaries
x_values = [0, 100]
y_values_my = - (theta[0][0] + np.dot(theta[1][0], x_values)) / theta[2][0]

plt.subplot(1, 2, 1)
plt.plot(x_values, y_values_my, label='decision boundary', color='red')
plt.scatter(X_1[:,0], X_1[:,1], s=10, label='y=1')
plt.scatter(X_0[:,0], X_0[:,1], s=10, label='y=0')
plt.xlim([0,100])
plt.ylim([0,100])
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend(loc='lower left')
plt.title('training examples')

plt.subplot(1, 2, 2)
plt.plot(x_values, y_values_my, label='decision boundary', color='red')
plt.scatter(X_1_test[:,0], X_1_test[:,1], s=10, label='y=1')
plt.scatter(X_0_test[:,0], X_0_test[:,1], s=10, label='y=0')
plt.xlim([0,100])
plt.ylim([0,100])
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend(loc='lower left')
plt.title('testing examples')
plt.show()
```

The package *sklearn* provides many convenient functions to compute the metrics of a model on a dataset. As an example, we calculate the major metrics of our model on the training set, as below. The details of the functions in sklearn.metrics can be found at the sklearn website.

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
```

```
X_add_ones=np.c_[np.ones((X.shape[0],1)), X]    # add 1 to X:(m,n+1)
prob=np.ndarray.flatten(h(theta,X_add_ones))
# predicted probabilities for all examples

temp=np.ndarray.flatten(np.asarray([prob>0.5]))
# predicted labels (true/false) for all examples

y_pred=temp.astype(int)
# predicted labels (1 or 0) for all examples

results = confusion_matrix(y, y_pred)
print ('Confusion Matrix :')
print(results)
print ('Accuracy Score :',accuracy_score(y, y_pred))
print ('Report : ')
print (classification_report(y, y_pred))
```

```
Confusion Matrix :
[[ 7  2]
 [ 1 10]]
Accuracy Score : 0.85
Report :
              precision    recall  f1-score   support

         0.0       0.88      0.78      0.82         9
         1.0       0.83      0.91      0.87        11

    accuracy                           0.85        20
   macro avg       0.85      0.84      0.85        20
weighted avg       0.85      0.85      0.85        20
```

Given the labels of examples and their predicted probabilities, we can plot the ROC curve and calculate the area under the curve.

```
fpr, recall, thresholds = roc_curve(y,prob)
#create ROC curve
plt.plot(fpr,recall)
plt.ylabel('recall')
plt.xlabel('False Positive Rate')
plt.grid()
plt.show()
auc = roc_auc_score(y, prob)
```

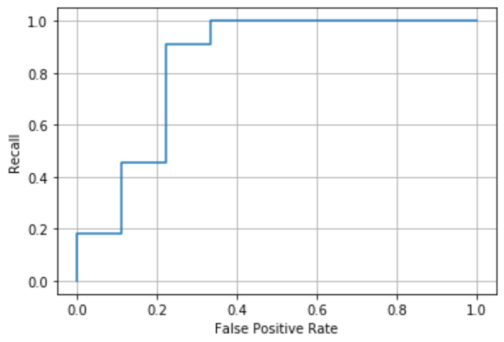


Fig.4.12 ROC curve

Summary and Further Reading

In this chapter, we began with the settings of classification problems. Logistic regression is a simple model that can perform a binary classification task. It is essential to understand the similarity and difference between linear regression and logistic regression. A logistic regression classifier is composed of a linear summer and a non-linear activation function (e.g. sigmoid 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 a similar way as linear regression. The ln()-based cost function is the core for the algorithm development.

The performance metrics of a classification model 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, an example of logistic regression using python is detailed.

File: \Users\weido\ch2_log_reg\log_reg23.ipynb, grades_train.txt, grades_test.txt.

Exercises

1. Suppose that you have trained a logistic regression classifier, and for a given new example x it delivers 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

2. 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) $P(y=0|x;\theta) = 0.7^2$

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

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

   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.

4. Derive the equation (4.21).

5. 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?

   1) 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.
   2) 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.

6. If we use tanh() as the activation function for logistic regression, instead of sigmoid function,

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

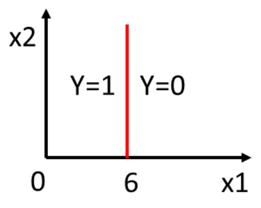
then the output of the regression model is

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

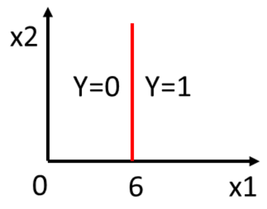
   1) Plot the curve of tanh(z). How should we predict the labels based on the value of $h_{\boldsymbol{\theta}}(\mathbf{x})$?
   2) Find the derivative of tanh(z).
   3) Equation (4.9) defines the cost for one example
$$J(h_{\boldsymbol{\theta}}(\mathbf{x}), y) = -y\ln\left(h_{\boldsymbol{\theta}}(\mathbf{x})\right) - (1-y)\ln\left(1 - h_{\boldsymbol{\theta}}(\mathbf{x})\right) \qquad (4.9)$$
      Is equation (4.9) still effective when the activation function is tanh()?

   4) Can you propose a cost function for the logistic regression with a tanh() activation function?
   5) Based on your cost function in d), develop a gradient descent algorithm for the logistic regression.

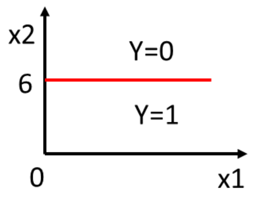
7. Suppose you train a logistic classifier
$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$
and obtain $\theta_0 = -6, \theta_1 = 0, \theta_2 = 1$. Which of the following figures represents the decision boundary of your classifier?



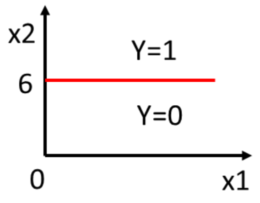
a)



b)

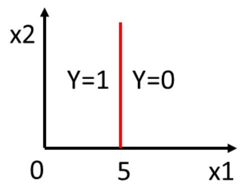


c)

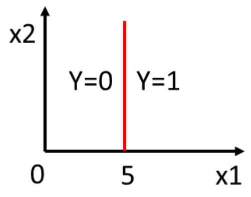


d)

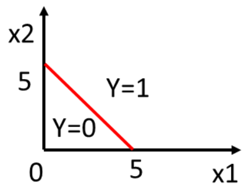
8. Consider logistic regression with two features *x₁* and *x₂*. After training, we obtain the result for paremters: $\theta_0 = 5, \theta_1 = -1, \theta_2 = -1$. Which of following figures shows the decision boundary?



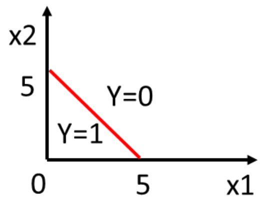
a)



b)



c)

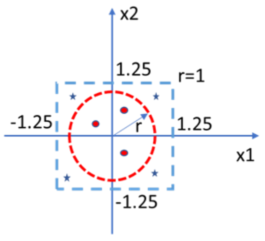


d)

9. (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 $(x^{(i)}, y^{(i)}), i = 1,2, \ldots, m$ 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 of 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 classification 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 classification accuracy on the testing set.
   4) Plot the ROC curve of your trained model on the testing set.
10. (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.

4) Train your model, plot the decision boundary along with the training examples, and print the classification accuracy.

11. 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-score is always nearer to the bigger value among recall and precision.
    5) F-score is an arithmetic average of recall and precision.
    6) F-score is always nearer to the smaller value among recall and precision.
    7) The range of F-score is from 0 to 1.

12. In a testing set, there are 999 data examples of negative class and one 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-score.

13. A testing result of a classifier is summarized by the following confusion matrix

	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-score.

14. 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.

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

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	

16. 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) Suppose the decision threshold is 0.4, calculate accuracy, recall (or sensitivity or TPR), precision, specificity, FPR, F-score.
    2) Suppose the decision threshold is 0.6, calculate accuracy, recall (or sensitivity or TPR), precision, specificity, FPR and F-score.
    3) Sketch the ROC of the classifier.

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