3 Fundamental Algorithms

3.1 Linear Regression

Linear regression is a popular regression learning algorithm that learns a model which is a linear combination of features of the input example.

3.1.1 Problem Statement

We have a collection of labeled examples $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, where N is the size of the collection, $\mathbf{x}_i$ is the D-dimensional feature vector of example $i = 1, \ldots, N$, y_i is a real-valued[1] target and every feature $x_i^{(j)}$, $j = 1, \ldots, D$, is also a real number.

We want to build a model $f_{\mathbf{w},b}(\mathbf{x})$ as a linear combination of features of example $\mathbf{x}$:

$$f_{\mathbf{w},b}(\mathbf{x}) = \mathbf{w}\mathbf{x} + b, \tag{1}$$

where $\mathbf{w}$ is a D-dimensional vector of parameters and b is a real number. The notation $f_{\mathbf{w},b}$ means that the model f is parametrized by two values: $\mathbf{w}$ and b.

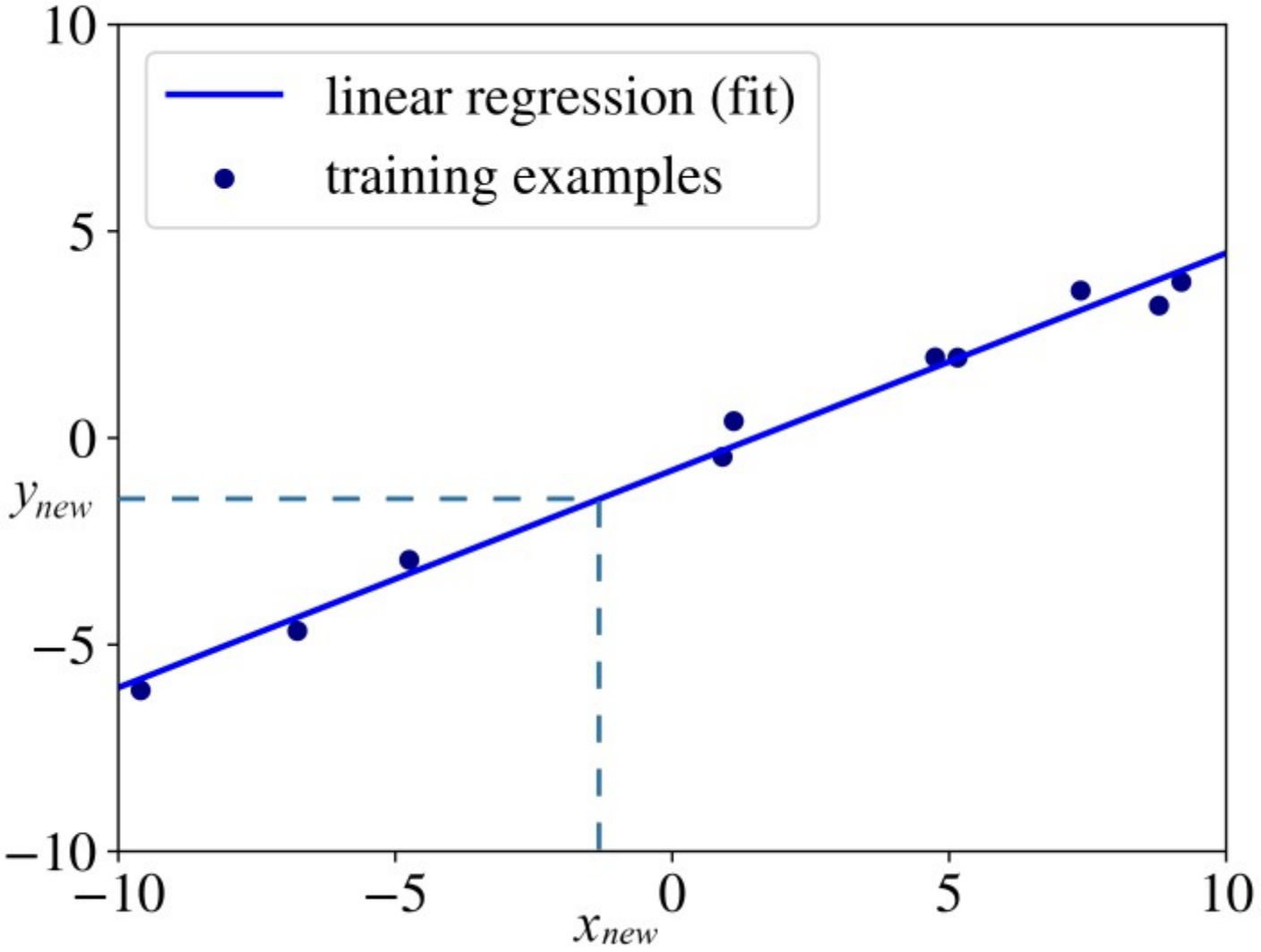
We will use the model to predict the unknown y for a given $\mathbf{x}$ like this: $y \leftarrow f_{\mathbf{w},b}(\mathbf{x})$. Two models parametrized by two different pairs $(\mathbf{w}, b)$ will likely produce two different predictions when applied to the same example. We want to find the optimal values $(\mathbf{w}^*, b^*)$. Obviously, the optimal values of parameters define the model that makes the most accurate predictions.

You could have noticed that the form of our linear model in eq. 1 is very similar to the form of the SVM model. The only difference is the missing sign operator. The two models are indeed similar. However, the hyperplane in the SVM plays the role of the decision boundary: it's used to separate two groups of examples from one another. As such, it has to be as far from each group as possible.

On the other hand, the hyperplane in linear regression is chosen to be as close to all training examples as possible.

You can see why this latter requirement is essential by looking at the illustration in fig. 1. It displays the regression line (in light-blue) for one-dimensional examples (dark-blue dots). We can use this line to predict the value of the target y_{new} for a new unlabeled input example x_{new}. If our examples are D-dimensional feature vectors (for $D > 1$), the only difference with the one-dimensional case is that the regression model is not a line but a plane (for two dimensions) or a hyperplane (for $D > 2$).

Now you see why it's essential to have the requirement that the regression hyperplane lies as close to the training examples as possible: if the blue line in fig. 1 was far from the blue dots, the prediction y_{new} would have fewer chances to be correct.


linear regression (fit)
training examples
10
5
0
−5
−10
y_{new}
−10
−5
0
5
10
x_{new}

3.1.2 Solution

To get this latter requirement satisfied, the optimization procedure which we use to find the optimal values for $\mathbf{w}^*$ and b^* tries to minimize the following expression:

$$\frac{1}{N}\sum_{i=1\ldots N}(f_{\mathbf{w},b}(\mathbf{x}_i)-y_i)^2. \tag{2}$$

In mathematics, the expression we minimize or maximize is called an objective function, or, simply, an objective. The expression $(f(\mathbf{x}_i)-y_i)^2$ in the above objective is called the **loss function**. It's a measure of penalty for misclassification of example i. This particular choice of the loss function is called **squared error loss**. All model-based learning algorithms have a loss function and what we do to find the best model is we try to minimize the objective known as the **cost function**. In linear regression, the cost function is given by the average loss, also called the **empirical risk**. The average loss, or empirical risk, for a model, is the average of all penalties obtained by applying the model to the training data.

Why is the loss in linear regression a quadratic function? Why couldn't we get the absolute value of the difference between the true target y_i and the predicted value $f(\mathbf{x}_i)$ and use that as a penalty? We could. Moreover, we also could use a cube instead of a square.

Now you probably start realizing how many seemingly arbitrary decisions are made when we design a machine learning algorithm: we decided to use the linear combination of features to predict the target. However, we could use a square or some other polynomial to combine the values of features. We could also use some other loss function that makes sense: the absolute difference between $f(\mathbf{x}_i)$ and y_i makes sense, the cube of the difference too; the **binary loss** (1 when $f(\mathbf{x}_i)$ and y_i are different and 0 when they are the same) also makes sense, right?

If we made different decisions about the form of the model, the form of the loss function, and about the choice of the algorithm that minimizes the average loss to find the best values of parameters, we would end up inventing a different machine learning algorithm. Sounds easy, doesn't it? However, do not rush to invent a new learning algorithm. The fact that it's different doesn't mean that it will work better in practice.

People invent new learning algorithms for one of the two main reasons:

1. The new algorithm solves a specific practical problem better than the existing algorithms.
2. The new algorithm has better theoretical guarantees on the quality of the model it produces.

One practical justification of the choice of the linear form for the model is that it's simple. Why use a complex model when you can use a simple one? Another consideration is that linear models rarely overfit. **Overfitting** is the property of a model such that the model predicts very well labels of the examples used during training but frequently makes errors when applied to examples that weren't seen by the learning algorithm during training.

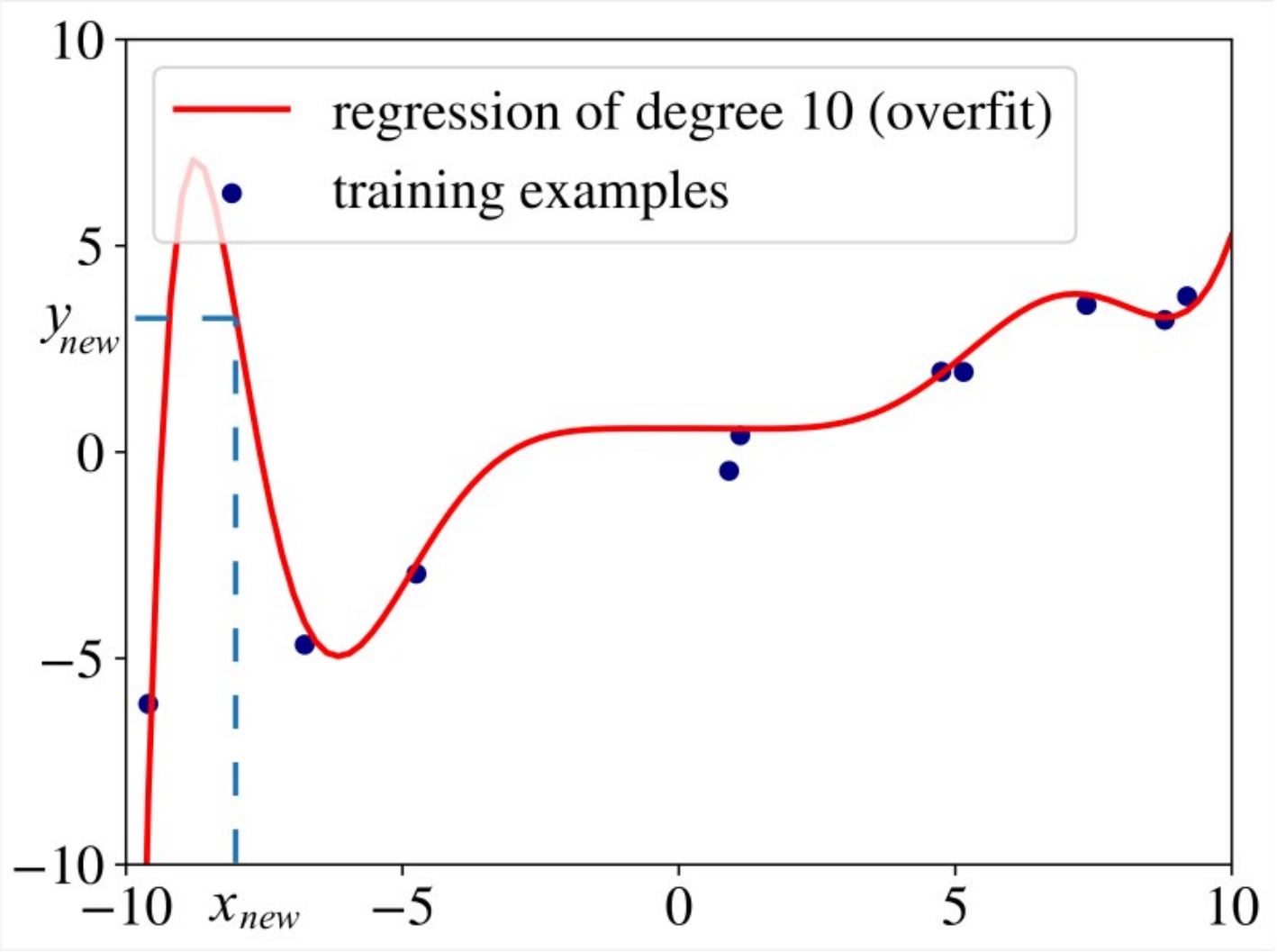
An example of overfitting in regression is shown in fig. 2. The data used to build the red regression line is the same as in fig. 1. The difference is that this time, this is the polynomial regression with a polynomial of degree 10. The regression line predicts almost perfectly the targets almost all training examples, but will likely make significant errors on new data, as you can see in fig. 1 for x_{new}. We talk more about overfitting and how to avoid it Chapter 5.

Now you know why linear regression can be useful: it doesn't overfit much. But what about the squared loss? Why did we decide that it should be squared? In 1705, the French mathematician Adrien-Marie Legendre, who first published the sum of squares method for gauging the quality of the model stated that squaring the error before summing is *convenient*. Why did he say that? The absolute value is not convenient, because it doesn't have a continuous derivative, which makes the function not smooth. Functions that are not smooth create unnecessary difficulties when employing linear algebra to find closed form solutions to optimization problems. Closed form solutions to finding an optimum of a function are simple algebraic expressions and are often preferable to using complex numerical optimization methods, such as **gradient descent** (used, among others, to train neural networks).

Intuitively, squared penalties are also advantageous because they exaggerate the difference between the true target and the predicted one according to the value of this difference. We

might also use the powers 3 or 4, but their derivatives are more complicated to work with.

Finally, why do we care about the derivative of the average loss? Remember from algebra that if we can calculate the gradient of the function in eq. 2, we can then set this gradient to zero[2] and find the solution to a system of equations that gives us the optimal values $\mathbf{w}^*$ and b^*.



3.2 Logistic Regression

The first thing to say is that logistic regression is not a regression, but a classification learning algorithm. The name comes from statistics and is due to the fact that the mathematical formulation of logistic regression is similar to that of linear regression.

I explain logistic regression on the case of binary classification. However, it can naturally be extended to multiclass classification.

3.2.1 Problem Statement

In logistic regression, we still want to model y_i as a linear function of $\mathbf{x}_i$, however, with a binary y_i this is not straightforward. The linear combination of features such as $\mathbf{w}\mathbf{x}_i + b$ is a function that spans from minus infinity to plus infinity, while y_i has only two possible values.

At the time where the absence of computers required scientists to perform manual calculations, they were eager to find a linear classification model. They figured out that if we define a negative label as 0 and the positive label as 1, we would just need to find a simple continuous function whose codomain is $(0, 1)$. In such a case, if the value returned by the model for input $\mathbf{x}$ is closer to 0, then we assign a negative label to $\mathbf{x}$; otherwise, the example is labeled as positive. One function that has such a property is the **standard logistic function** (also known as the **sigmoid function**):

$$f(x) = \frac{1}{1 + e^{-x}},$$

where e is the base of the natural logarithm (also called *Euler's number*; e^x is also known as the *exp(x)* function in Excel and many programming languages). Its graph is depicted in fig. 3.

By looking at the graph of the standard logistic function, we can see how well it fits our classification purpose: if we optimize the values of $\mathbf{x}$ and b appropriately, we could interpret the output of $f(\mathbf{x})$ as the probability of y_i being positive. For example, if it's higher than or equal to the threshold 0.5 we would say that the class of $\mathbf{x}$ is positive; otherwise, it's negative. In practice, the choice of the threshold could be different depending on the problem. We return to this discussion in Chapter 5 when we talk about model performance assessment.

So our logistic regression model looks like this:

$$f_{\mathbf{w},b}(\mathbf{x}) \stackrel{\text{def}}{=} \frac{1}{1+e^{-(\mathbf{w}\mathbf{x}+b)}}. \tag{3}$$

You can see the familiar term $\mathbf{w}\mathbf{x}+b$ from linear regression. Now, how do we find the best values $\mathbf{w}^*$ and b^* for our model? In linear regression, we minimized the empirical risk which was defined as the average squared error loss, also known as the **mean squared error** or MSE.

3.2.2 Solution

In logistic regression, instead of using a squared loss and trying to minimize the empirical risk, we maximize the *likelihood* of our training set according to the model. In statistics, the likelihood function defines how likely the observation (an example) is according to our model.

For instance, assume that we have a labeled example $(\mathbf{x}_i, y_i)$ in our training data. Assume also that we have found (guessed) some specific values $\hat{\mathbf{w}}$ and $\hat{b}$ of our parameters. If we now apply our model $f_{\hat{\mathbf{w}},\hat{b}}$ to $\mathbf{x}_i$ using eq. 3 we will get some value $0 < p < 1$ as output. If y_i is the positive class, the likelihood of y_i being the positive class, according to our model, is given by p. Similarly, if y_i is the negative class, the likelihood of it being the negative class is given by $1 - p$.

The optimization criterion in logistic regression is called **maximum likelihood**. Instead of minimizing the average loss, like in linear regression, we now maximize the likelihood of the training data according to our model:

$$L_{\mathbf{w},b} \overset{\text{def}}{=} \prod_{i=1\ldots N} f_{\mathbf{w},b}(\mathbf{x}_i)^{y_i}(1 - f_{\mathbf{w},b}(\mathbf{x}_i))^{(1-y_i)}. \tag{4}$$

The expression $f_{\mathbf{w},b}(\mathbf{x})^{y_i}(1 - f_{\mathbf{w},b}(\mathbf{x}))^{(1-y_i)}$ may look scary but it's just a fancy mathematical way of saying: "$f_{\mathbf{w},b}(\mathbf{x})$ when $y_i = 1$ and $(1 - f_{\mathbf{w},b}(\mathbf{x}))$ otherwise". Indeed, if $y_i = 1$, then $(1 - f_{\mathbf{w},b}(\mathbf{x}))^{(1-y_i)}$ equals 1 because $(1 - y_i) = 0$ and we know that anything power 0 equals 1. On the other hand, if $y_i = 0$, then $f_{\mathbf{w},b}(\mathbf{x})^{y_i}$ equals 1 for the same reason.

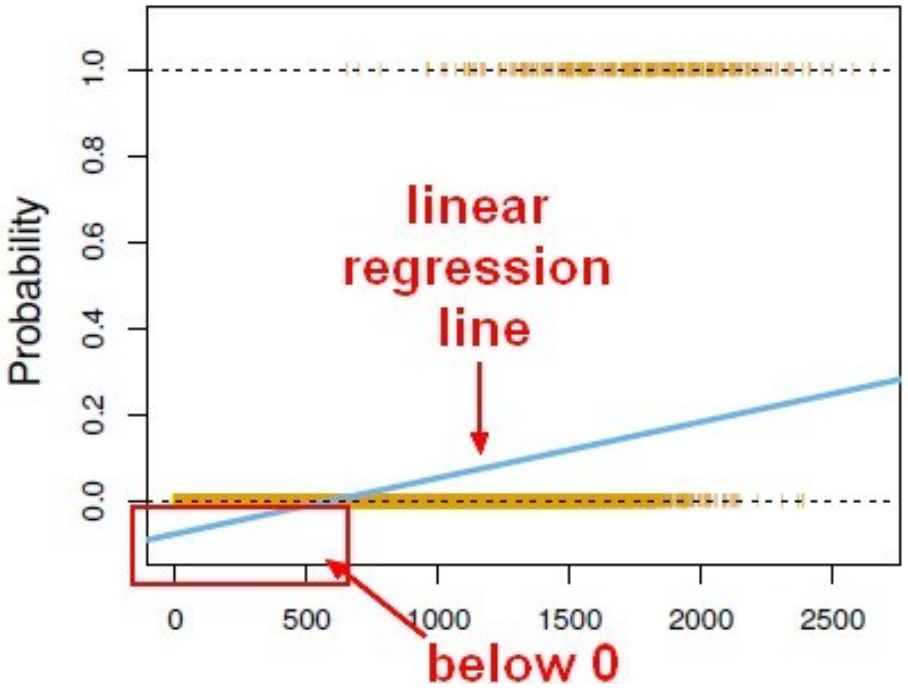
You may have noticed that we used the product operator $\prod$ in the objective function instead of the sum operator $\sum$ which was used in linear regression. It's because the likelihood of observing N labels for N examples is the product of likelihoods of each observation (assuming that all observations are independent of one another, which is the case). You can draw a parallel with the multiplication of probabilities of outcomes in a series of independent experiments in the probability theory.

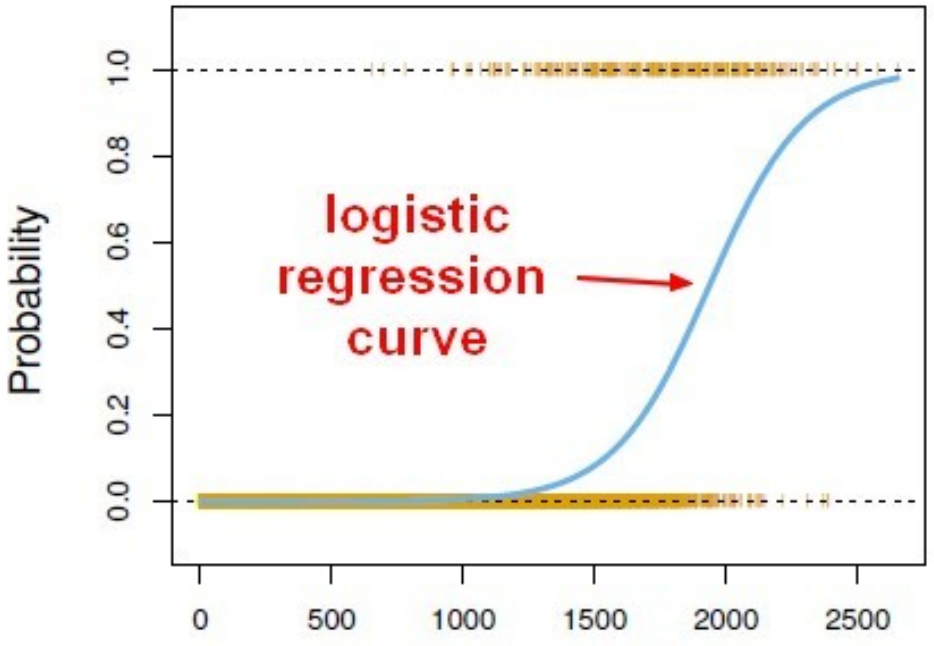
Because of the *exp* function used in the model, in practice, it's more convenient to maximize the *log-likelihood* instead of likelihood. The log-likelihood is defined like follows:

$$LogL_{\mathbf{w},b} \stackrel{\text{def}}{=} \ln(L(_{\mathbf{w},b}(\mathbf{x})) = \sum_{i-1}^{N} y_i \ln f_{\mathbf{w},b}(\mathbf{x}) + (1 - y_i) \ln (1 - f_{\mathbf{w},b}(\mathbf{x})).$$

Because ln is a *strictly increasing function*, maximizing this function is the same as maximizing its argument, and the solution to this new optimization problem is the same as the solution to the original problem.

Contrary to linear regression, there's no closed form solution to the above optimization problem. A typical numerical optimization procedure used in such cases is **gradient descent**. I talk about it in the next chapter.





Linear versus Logistic Regression

■Linear Regression	■Logistic Regression
■Target is an interval variable.	■Target is a discrete (binary or ordinal) variable.
■Input variables have any measurement level.	■Input variables have any measurement level.
■Predicted values are the mean of the target variable at the given values of the input variables.	■Predicted values are the probability of a particular level(s) of the target variable at the given values of the input variables.