Notes on Machine Learning from Andrew Ng's Coursera Course

Simon Zahn

August 30, 2016

Contents

Preface

Hello, this is a preface.

Chapter 1

Introduction

Two definitions of Machine Learning are offered at the start of the course:

Arthur Samuel 'The field of study that gives computers the ability to learn without being explicitely programmed.'

Tom Mitchell 'A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks T, as measured by P, improves with experience E.'

1.1 Supervised Learning

In supervised learning, we have a data set and already know what the correct output should be, knowing that there is an existing relationship between the input and output. There are two types of supervised learning: **classification**, and **regression**.

In a regression problem, we're trying to predict results with a continuous output; i.e. we are trying to map input variables to some continuous function. In a classification problem, we are instead trying to predict results with a discrete output. Let's look at some examples:

Regression If we're trying to predict the price of a house given data about the house such as its size, location, etc., this is a regression problem.

Regression Given a picture of a person, try to predict his/her age.

Classification Given a picture of a person, try to predict if he/she is of high school, college, or graduate age.

Classification As a bank, decide whether or not to give a loan to a potential borrower.

1.2 Unsupervised Learning

Unsupervised learning allows us to approach problems with little or no idea about what our results should look like. We can try and derive structure from data where we don't necessarily know the effect of the variables. We can derive this structure by clustering the data based on relationships among the variables in the data. With unsupervised learning, there is no feedback based on the prediction results.

One example of unsupervised learning is clustering. Imagine we take 1000 essays written by US college students. We can try and automatically group these essays into a smaller number that are somehow similar or related by different variables; word frequency, sentence length, page count, etc.

Here is an example of unsupervised learning that isn't clustering: the cocktail party algorithm. This is a way to find structure in messy data, such as the identification of individual voices and music from a mesh of sounds at a cocktail party.[1]

[1] For more information about auditory filtering, look at Wikipedia's Cocktail Party Effect article.

Chapter 2

Linear Regression

In regression problems, we take a variable (or multiple variables) as input, and try to fit the output to a continuous expected result function.

2.1 Univariate Linear Regression

In univariate linear regression, we want to predict a single output value $\hat{y}$ from a single input value x. Since this is supervised learning, we already have an idea about what the input/output relationship should look like.

2.1.1 The Hypothesis Function

Imagine we have a problem where the input is x and the output is y. In order to do machine learning, there should exist a relationship (a pattern) between the input and output variables. Let's say this function is $y = f(x)$. In this situation, f is known as the target function. However, this function f is unknown to us, so we need to try and guess what it is. To do that, we form a *hypothesis* function $h(x)$ that approximates the unknown $f(x)$.

For single variable linear regression, our hytothesis function takes two parameters: θ_0 and θ_1. As such, we often write it as $h_\theta(x)$, and it takes the form

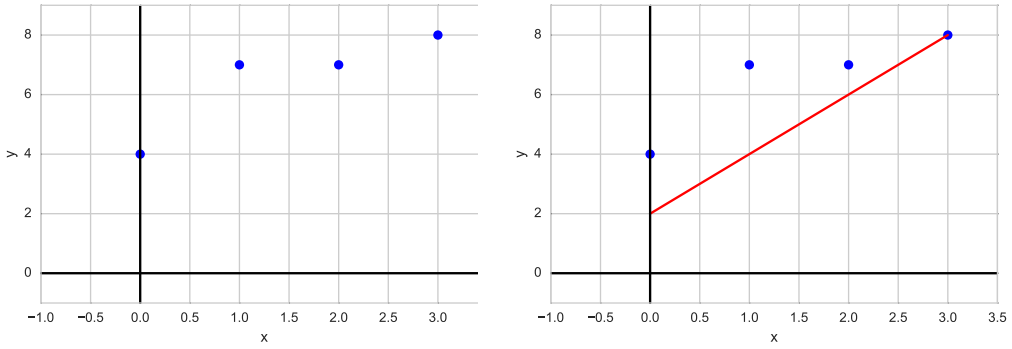
$$\hat{y} = h_\theta(x) = \theta_0 + \theta_1 x \tag{2.1}$$

Note that this is the equation of a straight line ($y = mx + b$). We're trying to find the values for θ_0 and θ_1 to get our estimated output $\hat{y}$. In other words, we're trying to determine the function h_θ that maps our input data (the x's) to our output data (the y's).

Suppose we have the following set of training data:

Input x	Output y
0	4
1	7
2	7
3	8

We can plot these points, as shown in Figure 2.1a. Let's make a random guess at our hypothesis function: $\theta_0 = 2$ and $\theta_1 = 2$, making our hypothesis function $h_\theta(x) = 2 + 2x$, as shown in Figure 2.1b.



(a) Plotting our example points.

(b) Plotting our example points.

Using this hypothesis function for $x = 1$, we have $\hat{y} = h_\theta(1) = 2 + 2 \cdot 1 = 4$. In this case, $\hat{y} = 4$, but $y = 7$, so mabe this isn't the best fit hypothesis.

2.1.2 The Cost Function

The cost function,[1] is a function used for parameter estimation, where the input to the cost function is some function of the difference between estimated and the true values for an instance of data. In this case, we can use the cost function to measure the accuracy of our hypothesis function.

The cost function looks at something similar to an average[2] of all the results of the hypothesis with inputs from the x's compared to the actual output y's. We define our cost function as follows:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2 = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2 \tag{2.2}$$

This is known as the **mean squared error**. If we set $\bar{x}$ equal to the mean of the squares all the $h_\theta(x_i) - y_i$, then the cost function is just the mean of $\bar{x}$. The term $\frac{1}{2m}$ is merely a convenience for the computation of gradient descent, which we'll see very shortly.

[1] The cost function can also be called the loss function.

[2] It's actually something a bit fancier than a standard average.

2.1.3 Gradient Descent

We now have our hypothesis function defined, as well as a way of measuring how well it fits the data. Now, we have to estimate the parameters in the hypothesis function, and that's where gradient descent comes in.

Let's graph our cost function as a function of the parameter estimates. This can be somewhat confusing, as we are moving up to a higher level of abstraction. We are not graphing x and y itself, but the parameter range of our hypothesis function and the cost resulting from selecting particular sets of parameters. We put θ_0 on the x-axis, and θ_1 on the y-axis, with the cost function on the vertical z-axis.

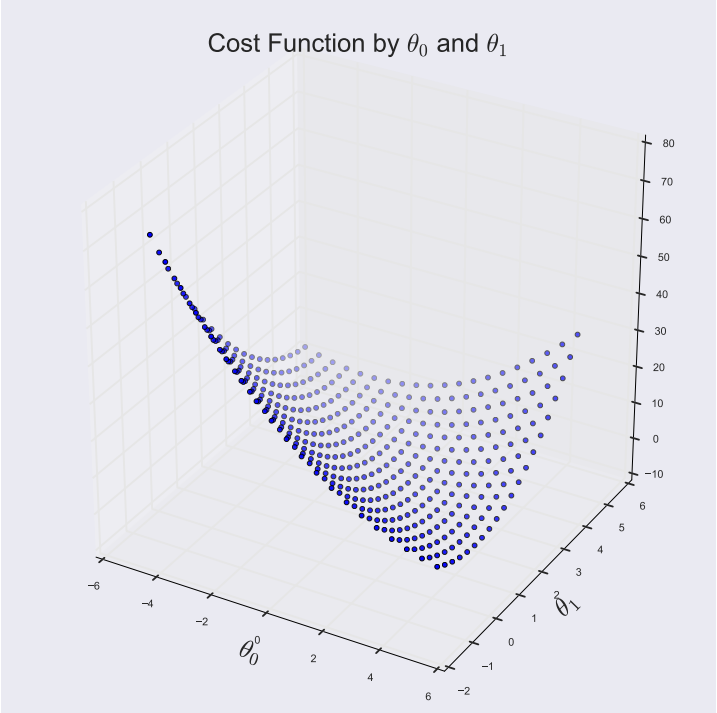


Figure 2.2: Plot of the cost function $J(\theta_0, \theta_1)$ using our hypothesis $h_\theta(x)$.

Our goal is to take the parameters θ_0 and θ_1 for when the cost function is at its minimum. We can calculate this value by taking the derivative of the cost function, which gives us direction of the steepest gradient to move towards. Take a step in that direction, and repeat.The step size is determined by the parameter α, which is called the **learning rate**. The gradient descent algorithm is:

Repeat until convergence:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \tag{2.3}$$

where $j = 0, 1$ represents the feature index number.

Gradient Descent for Linear Regression

When specifically applied to the case of univariate linear regression, we can derive another form of the gradient descent equation. If we substitute our actual hypothesis function and cost function, we can modify the equation to

Repeat until convergence: {

$$\begin{aligned} \theta_0 &:= \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i) \\ \theta_1 &:= \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} ((h_\theta(x_i) - y_i) x_i) \end{aligned} \tag{2.4}$$

}

where m is the size of the training set, θ_0 is a constant that will be changing simultaneously with θ_1, and x_1, y_i are values of the given training set. Note that we have separated out the two cases for θ_j into separate equations for θ_0 and θ_1, and that for θ_1 we are multiplying x_i at the end due to the derivative.

2.2 Multivariate Linear Regression

Let's start by looking at some sample housing data with multiple features.

Size (feet^2)	# of Bedrooms	# of Floors	Age (years)	Price (in 1000's of \$)
x_1	x_2	x_3	x_4	y
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
560	1	1	12	155

In this, we can introduce some notation:

- The variables x_1, x_2, etc. are the features.
- The variable y is the output variable.
- The number of input features is denoted n. In this example, $n = 4$.
- m specifies the number of training examples (rows). Here, $m = 5$.
- $x^{(i)}$ is the input (features) of the i^{th} training example. So $x^{(2)}$ is the column vector $[1416, 3, 2, 40, 232]$.

- $x_j^{(i)}$ is feature j in the i^{th} training example. Here, $x_1^{(4)} = 852$.

At this point, we can define the multivariable form of the hypothesis function for linear regression:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \cdots + \theta_n x_n \tag{2.5}$$

For convenience of notation, we will define $x_0 = 1$ for all feature vectors ($x_0^{(i)} = 1$). So now, if we include x_0, our hypothesis function takes the form:

$$h_\theta(x) = \sum_{i=0}^{n} \theta_i x_i \tag{2.6}$$

Now, we can also write the x values and θ values as vectors:

$$\vec{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1} \qquad \text{and} \qquad \vec{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

In vector notation, this is

$$h_\theta(x) = \vec{\theta}^{\mathsf{T}} \vec{x} \tag{2.7}$$

where we transpose $\vec{\theta}$ into a row vector so we're able to take the inner product.

Now that we have our vector $\vec{\theta} \in \mathbb{R}^{n+1}$, the cost function is

$$J\left(\vec{\theta}\right) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 \tag{2.8}$$

2.2.1 Gradient Descent for Multiple Variables

Using our expanded hypothesis and cost functions, the gradient descent algorithm becomes:

Repeat until convergence: {

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) \cdot x_j^{(i)} \qquad \text{for } j = 0, 1, \cdots, n \tag{2.9}$$

}

2.2.2 Feature Scaling

When features are in very different ranges, it can slow down gradient descent dramatically (and also mess up our machine learning algorithms!), because θ will descend quickly on small ranges and slowly on large ranges, and so will oscillate inefficiently down to

the minimum. The way to prevent this is to ensure that all the ranges are roughly the same, ideally:

$$-1 \leq x \leq 1$$

Two techniques to accomplish this are **feature scaling** and **mean normalization**. Feature scaling involved dividing the input values by the range (max value minus the min value) of the input variable, resulting in a new range of just 1.

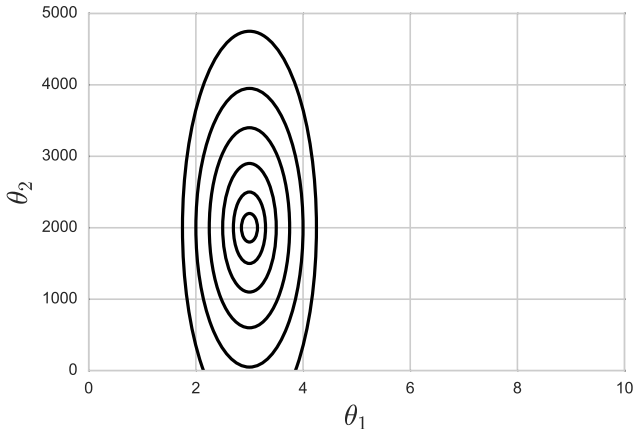


Figure 2.3: When one feature is on a much larger scale than the other, the plot of the cost function will be stretched out in the direction of the larger feature. Here, imagine that θ_1 is the number of bedrooms a house has, and θ_2 is the size in square feet.

Mean normalization involves subtracting the mean value for an input variable from the value for that input variable, resulting in a new mean of zero. To implement both of these simultaneously, use the following formula:

$$x_i := \frac{x_i - \mu_i}{s_i} \tag{2.10}$$

where μ_i is the average value of x_i, and s_i can either be the range ($x_{\max} - x_{\min}$) or the standard deviation.

2.2.3 Tips for Gradient Descent

Here are some of Professor Andrew Ng's tips on implementing gradient descent.

1. **Plot $J(\theta)$.** If you plot $J(\theta)$ as the ordinate and the number of iterations as the abscissa,[3] the graph should be steadily decreasing with increasing number of iterations. If $J(\theta)$ ever increases, then α is probably too large.

2. If $J(\theta)$ decreases by less then E in one iteration, where E is some very small number, such as 10^{-3}, then you can declare convergence.

[3] On a Cartesian coordinate plane, the ordinate is the y-axis and the abscissa is the x-axis.

3. For sufficiently small α, $J(\theta)$ should decrease with every iteration. To choose α, try a range of values for α with threefold increases, such as:

$$\cdots \to 0.001 \to 0.003 \to 0.01 \to 0.03 \to 0.1 \to 0.3 \to 1 \to 3 \to \cdots$$

4. Sometimes, it's better to define new features instead of using the ones given. For example, if we have a house with features frontage[4] and depth,[5] you can combine these into a new feature called area, which is how much land the house sits on.

2.2.4 Polynomial Regression

The form of the hypothesis doesn't necessarily need to be linear of that doesn't fit the data well. We can change the behavior or curve of our hypothesis function by making it quadratic, cubic, square root, or some other form.

For example, if our hypothesis function is $h_\theta(x) = \theta_0 + \theta_1 x_1$, we can create additional features based on x_1, to get the quadratic function $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2$, or the cubic function $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^3$.

When thinking about nonlinear features, it is important to keep in mind that features scaling becomes even more essential than it was for linear regression. If x_1 has a range of 1 to 1000, the x_1^2 has a range of 1 to $1,000,000$.

2.3 Vectorized Equations

Let's revisit our housing example from §2.2. Recall that we looked at the the following example data, and we'll add an extra column for x_0 that always takes a value of one:

	Size (feet^2)	# Bedrooms	# Floors	Age (years)	Price (in \$1000's)
x_0	x_1	x_2	x_3	x_4	y
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178
1	560	1	1	12	155

From this, we construct a matrix X that contains all of the features from the training data, and a vector $\vec{y}$ of all the output data.

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \\ 1 & 560 & 1 & 1 & 12 \end{bmatrix} \quad \text{and} \quad \vec{y} = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \\ 155 \end{bmatrix}$$

[4] The width of the land in the front of the house.
[5] The width of the land on the side of the house.

Here, X is a $m \times (n+1)$ matrix, and $\vec{y}$ is a m-dimensional vector.

Let's go through this again, but this time in full abstraction. Say we have m examples $\left(x^{(1)}, y^{(1)}\right), \left(x^{(2)}, y^{(2)}\right), \ldots, \left(x^{(m)}, y^{(m)}\right)$ and each $x^{(i)}$ has n features. Then, we have an $(n+1)$-dimensional feature vector:

$$x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} \in \mathbb{R}^{n+1} \tag{2.11}$$

The matrix X, which is also called the **design matrix**, is constructed by taking the transpose of each vector $x^{(i)}$. Each feature vector $x^{(i)}$ becomes a row in the design matrix. Just as previously, the output vector $\vec{y}$ is obtained by taking all the labels and stacking them up into an m-dimensional vector, and the vector $\vec{\theta}$ is created from stacking all of the parameters for the hypothesis function.

$$X = \begin{bmatrix} -- & -- & \left(x^{(1)}\right)^\intercal & -- & -- \\ -- & -- & \left(x^{(2)}\right)^\intercal & -- & -- \\ -- & -- & \left(x^{(3)}\right)^\intercal & -- & -- \\ -- & -- & \left(x^{(4)}\right)^\intercal & -- & -- \\ -- & -- & \vdots & -- & -- \\ -- & -- & \left(x^{(m)}\right)^\intercal & -- & -- \end{bmatrix} \qquad \vec{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ y^{(3)} \\ y^{(4)} \\ \vdots \\ y^{(m)} \end{bmatrix} \qquad \vec{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \\ \vdots \\ \theta_n \end{bmatrix} \tag{2.12}$$

Think back to the start of this section when we separated our table into the design matrix X and the output vector y. The design matrix is simply the data as stored in a table put into a matrix.

In our matrix notation for multivariate regression, the hypothesis function takes the form

$$h_\theta(X) = X\vec{\theta} \tag{2.13}$$

This will always work since X is an $m \times n$ matrix, and $\vec{\theta}$ is an $n \times 1$ vector. In a similar fashion, the cost function in matrix notation is

$$J\left(\vec{\theta}\right) = \frac{1}{2m}\left(X\vec{\theta} - \vec{y}\right)^\intercal \left(X\vec{\theta} - \vec{y}\right) \tag{2.14}$$

The gradient descent rule can be expressed as

$$\vec{\theta} := \vec{\theta} - \alpha \nabla J(\theta) \tag{2.15}$$

There ∇ is the gradient (vector derivative) operator. If we solve this out using our vectorized hypothesis function, we get

$$\vec{\theta} := \vec{\theta} - \frac{\alpha}{m} X^\intercal \left(X\vec{\theta} - \vec{y}\right) \tag{2.16}$$

2.4 The Normal Equation

The normal equation is a method of solving for the optimal θ analytically, that is, without iteration. From calculus, if we want to find the minimum of a quadratic equation, we set the derivative equal to zero, and solve. We can apply the same logic to the cost function. If we take the partial derivative $\partial/\partial\theta_j J(\theta)$ and set this equal to zero for all values of j, we'll analytically solve for the minimum.

The derivation of the normal equation is fairly involved from a linear algebra perspective, so at this point just take it as a fact:

$$\vec{\theta} = (X^\intercal X)^{-1} X^\intercal \vec{y} \tag{2.17}$$

When deciding whether to use gradient descent of the normal equation, consider the following:

Gradient Descent	**Normal Equation**
Need to choose α	No need to choose α
Needs many iterations	No need to iterate
$O\left(kn^2\right)$	$O\left(n^3\right)$, need to calculate $X^\intercal X$
Works well when n is large	Slow if n is very large

With the normal equation, computing the inverse has complexity $O\left(n^3\right)$. If we have a large number of features, this will cause the normal equation to perform slowly. In practice, when n exceeds $10,000$, it would probably be a good idea to use gradient descent.

2.4.1 Normal Equation Noninvertibility

When implementing the normal equation, sometimes the matrix $X^\intercal X$ is noninvertible. The common causes are:

- Redundant features, where two or more features are linearly dependent
- Too many features (i.e. $m \leq n$). In this case, delete some features or use regularization (which we'll get to later)

We typically avoid this problem by coding a pseudoinverse, instead of taking the actual inverse.

Chapter 3

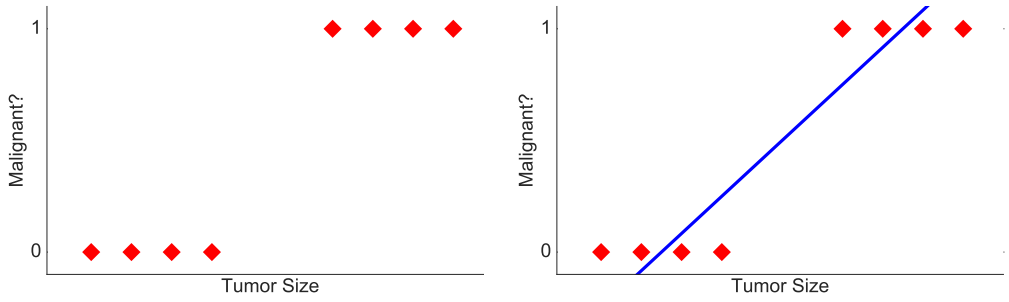
Logistic Regression

Now we turn away from regression to classification problems. Don't be confused by the name 'logistic regression,' it's actually just named for the mathematical function and it's a common approach to classification.

3.1 Binary Classification

In classification problems, instead of our output being continuous, we expect it to fall into discrete classes. We'll start with the simplest case: binary classification. Here, we have our output variable $y \in \{0, 1\}$. Typically, we take 0 as the negative class and 1 as the positive class.

Consider the following example: we have a sample of eight tumors, and we want to determine if they're malignant based on the tumor size. These are plotted in Figure 3.1a. One thing we can do is assume a linear relationship with hypothesis $h_\theta(\vec{x}) = \vec{\theta}^\intercal \vec{x}$. This shown in Figure 3.1b.



(a) Sample tumor data.

(b) Plot of tumors by size with linear regression line $y = \frac{x-1}{6} - 0.25$.

Figure 3.1: Plots of tumors by size.

To try and make predictions, we can threshold the output at $h_\theta(x) = 0.5$, and then:

- If $h_\theta(x) \geq 0.5$, then predict $y = 1$
- If $h_\theta(x) < 0.5$, then predict $y = 0$

and you can see this in Figure 3.2.

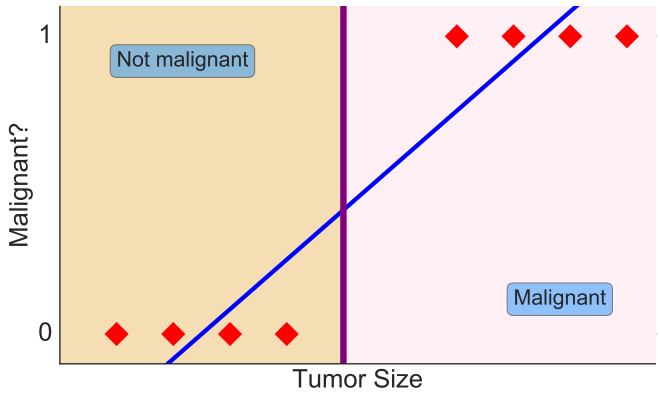


Figure 3.2: Linear regression plotted with classification regions.

In this example, it would seem like linear regression is a good classifier. However, what if we add a new data point for a large tumor. Suddenly, our results look like this

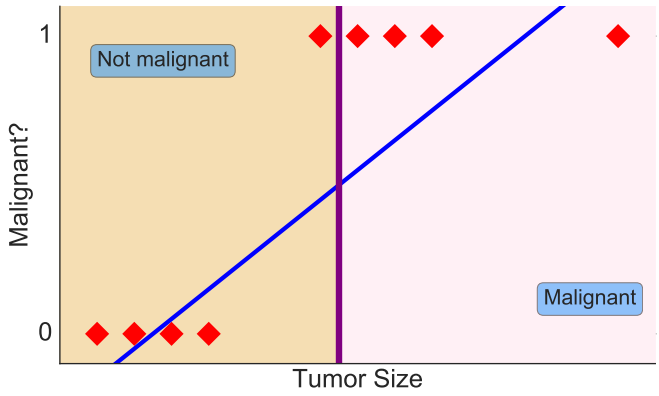


Figure 3.3: Linear regression plotted with classification regions after a new data point is added. Notice how one of the malignant tumors is now being misclassified as benign.

and now, we have a malignant tumor being misclassified as benign. Ergo, maybe linear regression isn't the best way to build a binary classifier.

3.2 Hypothesis Representation

In linear regression, our hypothesis was $h_\theta(\vec{x}) = \vec{\theta}^\intercal \vec{x}$ For logistic regression, we want our hypothesis to satisfy $0 \leq h_\theta(x) \leq 1$. To do this, we use the sigmoid function.[1] To make this work, we modify our hypothesis to be

$$h_\theta(x) = g\left(\vec{\theta}^\intercal \vec{x}\right) \tag{3.1}$$

where the function $g(z)$ is defined as

$$g(z) = \frac{1}{1 + e^{-z}} \tag{3.2}$$

Thus, to get the hypothesis function using the sigmoid function, just set $z = \vec{\theta}^\intercal \vec{x}$.

The sigmoid function, shown in Figure 3.4, maps any real number onto the interval $(0, 1)$. This makes it immensely useful for transforming an arbitrary function for use with classification.

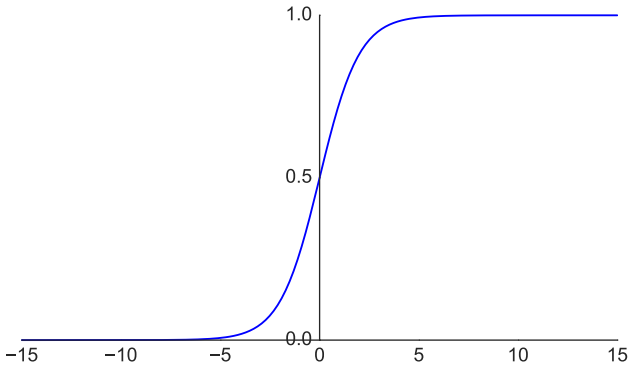


Figure 3.4: Sample plot of the sigmoid function.

3.2.1 Interpretation of the Logistic Hypothesis Function

When examining the hypothesis function output for logistic regression, we interpret $h_\theta(x)$ is the estimated probability that $y = 1$ on an input example x. For example, let's revisit the tumor size question from above. We have

$$\vec{x} = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ \text{tumor size} \end{bmatrix}$$

If our hypothesis $h_\theta(x) = 0.7$, then we can tell the patient that there is a 70% chance of the tumor being malignant.

[1] This is also called the logistic function, and is the namesake for logistic regression.

Slightly more formally, we interpret $h_\theta(x)$ as:[2]

$$h_\theta(x) = P(y = 1|x;\theta) \tag{3.3}$$

Thus, by the rules of probability:

$$P(y = 0|x;\theta) + P(y = 1|x;\theta) = 1 \tag{3.4}$$
$$P(y = 0|x;\theta) = 1 - P(y = 1|x;\theta) \tag{3.5}$$

3.2.2 Fitting Logistic Regression to a Binary Classifier

Now, we need to fit our hypothesis function into a binary classfier: 0 or 1. Using our probabilistic interpretation of the logistic hypothesis function, we can make the following supposition:

$$y = 1 \text{ given that } h_\theta(x) \geq 0.5 \tag{3.6}$$
$$y = 0 \text{ given that } h_\theta(x) < 0.5 \tag{3.7}$$

Recall the plot of the sigmoid function in Figure 3.4. We see that $g(z) \geq 0.5$ when $z \geq 0$. In our case, if we're setting $z = \vec{\theta}^\intercal \vec{x}$, then we have:

$$h_\theta(x) = g\left(\vec{\theta}^\intercal \vec{x}\right) \geq 0.5 \quad \text{when} \quad \vec{\theta}^\intercal \vec{x} \geq 0 \tag{3.8}$$

From this, we can now state

$$\vec{\theta}^\intercal \vec{x} \geq 0 \implies y = 1 \tag{3.9}$$
$$\vec{\theta}^\intercal \vec{x} < 0 \implies y = 0 \tag{3.10}$$

When utilizing the sigmoid function, keep the following in mind:

- When $z = 0$, then $e^0 = 1$ so $g(x) = \frac{1}{2}$
- When z goes to ∞, we have $e^{-\infty} \to 0$, and this implies $g(x) = 1$
- As $z \to -\infty$, $e^{\infty} \to \infty \implies g(z) = 0$

3.3 Decision Boundary

Consider the data plotted in Figure 3.5. Suppose our hypothesis is given by

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

[2] This is read as "the probability that $y = 1$, given x, parameterized by θ.

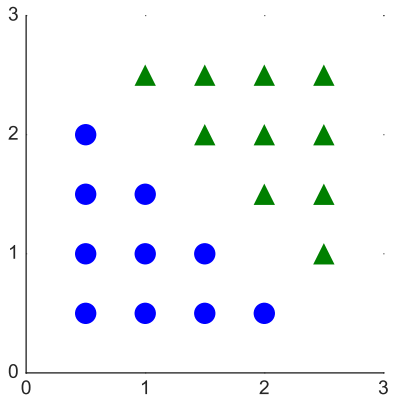


Figure 3.5: Sample data.

We haven't yet discussed how to fit the parameters of this model (that's coming up next), but suppose we choose the following values for the parameters

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

Given this choice of parameters, let's figure out where $y = 1$ and where $y = 0$. From §3.2.2, recall that we predict $y = 1$ when $\vec{\theta}^{\intercal}\vec{x} \geq 0$, so here, we predict $y = 1$ if $-3 + x_1 + x_2 \geq 0$. If we solve this for $x_1 + x_2$ we get

$$x_1 + x_2 \geq 3 \implies y = 1$$

If we change this to a pure equality, $x_1 + x_2 = 3$, we have the equation of a straight line (shown on Figure 3.6). The line drawn, is called the **decision boundary**. The decision boundary is the line created by the hypothesis function that separates the area where we classify $y = 0$ and where $y = 1$.

To be clean, the decision boundary is a property of the hypothesis function, and not a property of the dataset. We fit the parameters of the hypothesis based on the training data, but once those parameters are set, the decision boundary is a property solely of the hypothesis function.

Now, suppose we have data as shown below in Figure 3.7a. It's fairly obvious that no straight line decision boundary will work for this data. Again, we don't know how to fit the parameters for this model yet, but say our hypothesis function looks like this

$$h_\theta(x) = g\left(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2\right)$$

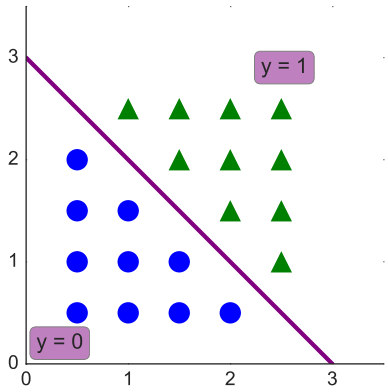


Figure 3.6: Sample data with plotted decision boundary.

Imagine we fit the parameters appropriately, and we get

$$\vec{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

Then, our hypothesis predicts that $y = 1$ when $x_1^2 + x_2^2 \geq 1$. This is the equation for a circle of radius 1, centered at the origin (see Figure 3.7b). In this case, we predict $y = 1$ everywhere outside the purple circle, and $y = 0$ everywhere inside the circle.

With even higher order polynomials, we can get even more complicated decision boundaries.

3.4 Cost Function

Imagine we have a training set of data with m examples

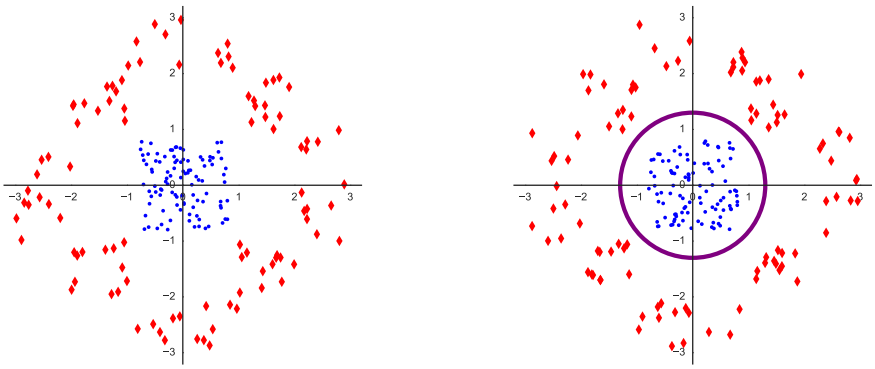
$$\left\{ \left(x^{(1)}, y^{(1)} \right), \left(x^{(2)}, y^{(2)} \right), \cdots, \left(x^{(m)}, y^{(m)} \right) \right\}$$

and n features, represented by an $n+1$-dimensional feature vector

$$x \in \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}$$

where $x_0 = 1$ and our output $y \in \{0, 1\}$. Our hypothesis is given by

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^\intercal x}} \tag{3.11}$$



(a) This is a sample of data with no clear linear decision boundary.

(b) By altering our hypothesis function to include polynomial terms, we can have a non-linear decision boundary.

Figure 3.7: Data that can't be fit with a linear decision line.

How do we choose the parameters for this model? For linear regression, we had the following cost function (adjusted slightly, we moved the $\frac{1}{2}$ to the inside of the summation)

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 \tag{3.12}$$

but we're going to change how we write this function a little bit. Instead, we'll write

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\text{cost}\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right) \tag{3.13}$$

where we'll define the cost function to be

$$\text{cost}\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right) = \frac{1}{2}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 \tag{3.14}$$

This allows us to see more clearly that the cost function is really the sum over the cost term. To simplify even further, we'll remove the superscripts (i)

$$\text{cost}\left(h_\theta(x), y\right) = \frac{1}{2}\left(h_\theta(x) - y\right)^2 \tag{3.15}$$

If we try an minimize this cost function, this turns out to be a non-convex function. That means that the may be several local minima, which would prevent our gradient descent algorithm from working well. You can see a sample non-convex function in Figure 3.8. What we want instead, is a convex function (like a parabola) that only has a single minimum that is the global minimum. The sigmoid function is a non-linear signal function, so $J(\theta)$ ends up being non-convex.

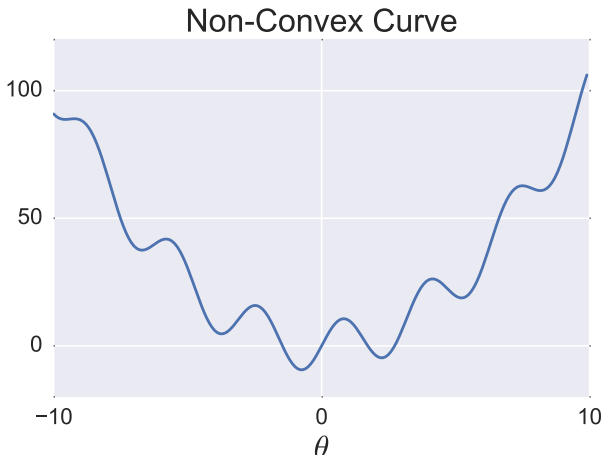
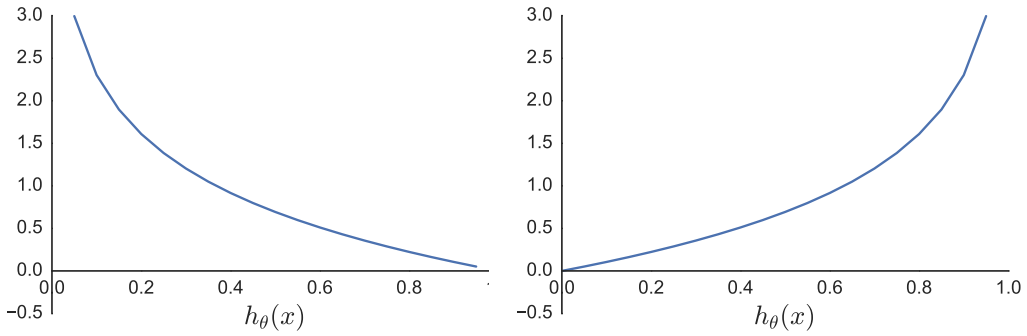


Figure 3.8: A non-convex curve. Notice all of the local minima.

We need to define a new (convex) cost function that will allow us to determine the parameters in our hypothesis. For logistic regression, we use the following cost function

$$\text{cost}\left(h_\theta\left(x\right), y\right) = \begin{cases} -\log\left(h_\theta\left(x\right)\right) & \text{if } y = 1 \\ -\log\left(1 - h_\theta\left(x\right)\right) & \text{if } y = 0 \end{cases} \tag{3.16}$$

We plot this cost function below in Figure 3.9.



(a) $y = 1$. (b) $y = 0$

Figure 3.9: The piecewise cost function for logistic regression.

The shape of the curve comes from standard plot of $\log\left(x\right)$, and we just use a negative to flip it upside-down. This cost function, has some very desirable properties for us right now.

- If $y = 1$, then $h_\theta(x) = 1$ and the cost is zero. However, as $h_\theta(x) \to 0$ then cost $\to \infty$. This captures the intuition that if $h_\theta(x) = 0$, but $y = 1$, we'll penalize the learning algorithm by a very large cost.
- For $y = 0$, this is reversed. If we have $y = 0$ and $h_\theta(x) = 0$, then the cost is 0. If $y = 0$, the cost grows very large as $h_\theta(x)$ increases towards 1.

Appendix A

Notation

$\hat{}$	predicted output
$\hat{y}$	predicted output of the variable y

$\mathbb{Z}$	the set of integers; zahlen is the German word for numbers
$\bigoplus$	the earth
$\odot$	the sun
c	speed of light
G	Newton′s gravitational constant
C	circumference
S	distance

Appendix B

Linear Algebra Review