Machine Learning Lecture Notes

by Andrew Ng

Transcripted by: Fernando García de la Cruz

Chapter 1

Week 1

1.1 What is Machine Learning?

Two definitions of Machine Learning are offered. **Arthur Samuel** described it as: **"the field of study that gives computers the ability to learn without being explicitly programmed."** This is an older, informal definition.

Tom Mitchell provides a more modern definition:

> "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

Example: playing checkers.

E = the experience of playing many games of checkers
T = the task of playing checkers.
P = the probability that the program will win the next game.

In general, any machine learning problem can be assigned to one of two broad classifications: **supervised learning, or unsupervised learning**.

1.2 Supervised Learning

In supervised learning, we are given a data set and already know what our correct output should look like, having the idea that there is a relationship between the input and the output.

Supervised learning problems are categorized into "regression" and "classification" problems. In a regression problem, we are trying to predict results within a continuous output, meaning that we are trying to map input variables to some continuous function. In a classification problem, we are instead trying to predict results in a discrete output. In other

words, we are trying to map input variables into discrete categories. Here is a description on Math is Fun on Continuous and Discrete Data.

1.2.1 Example 1

Given data about the size of houses on the real estate market, try to predict their price. Price as a function of size is a continuous output, so this is a regression problem.

We could turn this example into a classification problem by instead making our output about whether the house "sells for more or less than the asking price." Here we are classifying the houses based on price into two discrete categories.

1.2.2 Example 2

A) **Regression** - Given a picture of Male/Female, We have to predict his/her age on the basis of given picture.

B) **Classification** - Given a picture of Male/Female, We have to predict whether He/She is of High school, College, Graduate age. Another Example for Classification - Banks have to decide whether or not to give a loan to someone on the basis of his credit history.

1.3 Unsupervised Learning

Unsupervised learning, on the other hand, allows us to approach problems with little or no idea what our results should look like. We can 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, i.e., **there is no teacher to correct you**.

1.3.1 Example

Clustering: Take a collection of 1000 essays written on the US Economy, and find a way to automatically group these essays into a small number that are somehow similar or related by different variables, such as word frequency, sentence length, page count, and so on.

Non-clustering: The "Cocktail Party Algorithm", which can find structure in messy data (such as the identification of individual voices and music from a mesh of sounds at a cocktail party) . Here is an answer on Quora to enhance your understanding: click here!.

1.4 ML:Linear Regression with One Variable

1.4.1 Model Representation

Recall that in regression problems, we are taking input variables and trying to fit the output onto a continuous expected result function.

Linear regression with one variable is also known as "univariate linear regression".

Univariate linear regression is used when you want to predict a **single output** value y from a **single input** value x. We're doing **supervised learning** here, so that means we already have an idea about what the input/output cause and effect should be.

1.4.2 The Hypothesis Function

Our hypothesis function has the general form:

$$\hat{y} = h_\theta(x) = \theta_0 + \theta_1 x \tag{1.1}$$

Note that this is like the equation of a straight line. We give to $h_\theta(x)$ values for θ_0 and θ_1 to get our estimated output output $\hat{y}$ In other words, we are trying to create a function called h_θ that is trying to map our input data (the x's) to our output data (the y's).

Example:
Suppose we have the following set of training data:

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

Now we can make a random guess about our h_θ function $\theta_0 = 2$ and $\theta_1 = 2$. The hypothesis function becomes $h_\theta = 2 + 2x$

So for input of 1 to our hypothesis, y will be 4. This is off by 3. Note that we will be trying out various values of θ_0 and θ_1 to try to find values which provide the best possible "fit" or the most representative "straight line" through the data points mapped on the x-y plane.

1.4.3 Cost Function

We can measure the accuracy of our hypothesis function by using a **cost function**. This takes an average (actually a fancier version of an average) of all the results of the hypothesis with inputs from x's compared to the actual output y's.

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2 = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2 \tag{1.2}$$

To break it apart, it is $\frac{1}{2}\bar{x}$ where $\bar{x}$ is the mean of the squares of $(h_\theta(x_i) - y_i)$, or the difference between the predicted value and the actual value.

This function is otherwise called the **"Squared error function"**, or **"Mean squared error"**. The mean is halved $\left(\frac{1}{2m}\right)$ as a convenience for the computation of the gradient descent, as the derivative term of the square function will cancel out the $\frac{1}{2}$ term.

Now we are able to concretely measure the accuracy of our predictor function against the correct results we have so that we can predict new results we don't have.

If we try to think of it in visual terms, our training data set is scattered on the x-y plane. We are trying to make straight line (defined by $h_\theta(x)$) which passes through this scattered set of data. Our objective is to get the best possible line. The best possible line will be such so that the average squared vertical distances of the scattered points from the line will be the least. In the best case, the line should pass through all the points of our training data set. In such a case the value of $J(\theta_0, \theta_1)$ will be 0.

1.5 ML: Gradient Descent

So we have our hypothesis function and we have a way of measuring how well it fits into the data. Now we need to estimate the parameters in hypothesis function. That's where gradient descent comes in.

Imagine that we graph our hypothesis function based on its fields θ_0 and θ_1 (actually we are graphing the cost function as a function of the parameter estimates). This can be kind of confusing; 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 set 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. The points on our graph will be the result of the cost function using our hypothesis with those specific theta parameters.

We will know that we have succeeded when our cost function is at the very bottom of the pits in our graph, i.e. when its value is the minimum.

The way we do this is by taking the derivative (the tangential line to a function) of our cost function. The slope of the tangent is the derivative at that point and it will give us a direction to move towards. We make steps down the cost function in the direction with the steepest descent, and the size of each step is determined by the parameter α, which is called the learning rate.

The gradient descent algorithm is:

$$\text{repeat until convergence: } \{$$

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

$$\}$$

where

j=0,1 represents the feature index number.

Intuitively, this could be thought of as:

repeat until convergence:

$$\theta_j := -\alpha\delta$$

where: δ = [Slope of tangent aka derivative in j dimension]

1.5.1 Gradient Descent for Linear Regression

When specifically applied to the case of linear regression, a new form of the gradient descent equation can be derived. We can substitute our actual cost function and our actual hypothesis function and modify the equation to (the derivation of the formulas are out of the scope of this course, but a really great one can be found here):

repeat until convergence: {

$$\theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)$$

$$\theta_1 := \theta_1 - \alpha\frac{1}{m}\sum_{i=1}^{m}((h_\theta(x_i) - y_i)x_i)$$

}

where m is the size of the training set, θ_0 a constant that will be changing simultaneously with θ_1 and x_i, y_i are values of the given training set (data).

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.

The point of all this is that if we start with a guess for our hypothesis and then repeatedly apply these gradient descent equations, our hypothesis will become more and more accurate.

1.5.2 Gradient Descent for Linear Regression: visual worked example

Some may find the following video useful as it visualizes the improvement of the hypothesis as the error function reduces.

1.6 ML:Linear Algebra Review

Khan Academy has excellent Linear Algebra Tutorials: click here!

1.6.1 Matrices and Vectors

Matrices are 2-dimensional arrays:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \\ j & k & l \end{bmatrix}$$

The above matrix has four rows and three columns, so it is a 4 x 3 matrix.

A vector is a matrix with one column and many rows:

$$\begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix}$$

So vectors are a subset of matrices. The above vector is a 4 x 1 matrix.

Notation and terms:

- A_{ij} refers to the element in the ith row and jth column of matrix A.
- A vector with 'n' rows is referred to as an 'n'-dimensional vector
- v_i refers to the element in the ith row of the vector.
- In general, all our vectors and matrices will be 1-indexed. Note that for some programming languages, the arrays are 0-indexed.
- Matrices are usually denoted by uppercase names while vectors are lowercase.
- "Scalar" means that an object is a single value, not a vector or matrix.
- $\mathbb{R}$ refers to the set of scalar real numbers
- $\mathbb{R}^n$ refers to the set of n-dimensional vectors of real numbers

1.6.2 Addition and Scalar Multiplication

Addition and subtraction are **element-wise**, so you simply add or subtract each corresponding element:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} w & x \\ y & z \end{bmatrix} = \begin{bmatrix} a+w & b+x \\ c+y & d+z \end{bmatrix}$$

To add or subtract two matrices, their dimensions **must be the same**.

In scalar multiplication, we simply multiply every element by the scalar value:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} * x = \begin{bmatrix} a*x & b*x \\ c*x & d*x \end{bmatrix}$$

1.6.3 Matrix-Vector Multiplication

We map the column of the vector onto each row of the matrix, multiplying each element and summing the result.

$$\begin{bmatrix} a & b \\ c & d \\ e & f \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a*x+b*y \\ c*x+d*y \\ e*x+f*y \end{bmatrix}$$

The result is a vector. The vector must be the second term of the multiplication. The number of columns of the matrix must equal the number of rows of the vector.

An m x n matrix multiplied by an n x 1 vector results in an m x 1 vector.

1.6.4 Matrix-Matrix Multiplication

We multiply two matrices by breaking it into several vector multiplications and concatenating the result

$$\begin{bmatrix} a & b \\ c & d \\ e & f \end{bmatrix} * \begin{bmatrix} w & x \\ y & z \end{bmatrix} = \begin{bmatrix} a*w+b*y & a*x+b*z \\ c*w+d*y & c*x+d*z \\ e*w+f*y & e*x+f*z \end{bmatrix}$$

An m x n matrix multiplied by an n x o matrix results in an m x o matrix. In the above example, a 3 x 2 matrix times a 2 x 2 matrix resulted in a 3 x 2 matrix.

To multiply two matrices, the number of columns of the first matrix must equal the number of rows of the second matrix.

1.6.5 Matrix Multiplication Properties

- Not commutative. $A * B \neq B * A$
- Associative. $(A * B) * C = A * (B * C)$

The identity matrix, when multiplied by any matrix of the same dimensions, results in the original matrix. It's just like multiplying numbers by 1. The identity matrix simply has 1's on the diagonal (upper left to lower right diagonal) and 0's elsewhere.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

When multiplying the identity matrix after some matrix $(A*I)$, the square identity matrix should match the other matrix's columns. When multiplying the identity matrix before some other matrix $(I * A)$, the square identity matrix should match the other matrix's rows.

1.6.6 Inverse and Transpose

The inverse of a matrix A is denoted A-1. Multiplying by the inverse results in the identity matrix.

A non square matrix does not have an inverse matrix. We can compute inverses of matrices in octave with the `pinv(A)` function [1] and in matlab with the `inv(A)` function. Matrices that don't have an inverse are singular or degenerate.

The transposition of a matrix is like rotating the matrix 90° in clockwise direction and then reversing it. We can compute transposition of matrices in matlab with the `transpose(A)` function or A':

$$A = \begin{bmatrix} a & b \\ c & d \\ e & f \end{bmatrix}$$

$$A^T = \begin{bmatrix} a & c & e \\ b & d & f \end{bmatrix}$$

In other words:

$$A_{ij} = A^T_{ji}$$

Chapter 2

Week 2

2.1 ML:Linear Regression with Multiple Variables

Linear regression with multiple variables is also known as "multivariate linear regression".

We now introduce notation for equations where we can have any number of input variables.

$$\begin{aligned} x_j^{(i)} &= \text{value of feature } j \text{ in the } i^{th} \text{ training example} \\ x^{(i)} &= \text{the column vector of all the feature inputs of the } i^{th} \text{ training example} \\ m &= \text{the number of training examples} \\ n &= \left|x^{(i)}\right| ; (\text{the number of features}) \end{aligned}$$

Now define the multivariable form of the hypothesis function as follows, accommodating these multiple features:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \cdots + \theta_n x_n \tag{2.1}$$

In order to develop intuition about this function, we can think about θ_0 as the basic price of a house, θ_1 as the price per square meter, θ_2 as the price per floor, etc. x_1 will be the number of square meters in the house, x_2 the number of floors, etc.

Using the definition of matrix multiplication, our multivariable hypothesis function can be concisely represented as:

$$h_\theta(x) = \begin{bmatrix} \theta_0 & \theta_1 & \dots & \theta_n \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} = \theta^T x$$

This is a vectorization of our hypothesis function for one training example; see the lessons on vectorization to learn more.

Remark: Note that for convenience reasons in this course Mr. Ng assumes:

$$x_0^{(i)} = 1 \text{ for } (i \in 1, \dots, m)$$

[Note: So that we can do matrix operations with theta and x, we will set $x_0^{(i)} = 1$, for all values of i. This makes the two vectors 'theta' and $x_{(i)}$ match each other element-wise (that is, have the same number of elements: n+1).]

The training examples are stored in X row-wise, like such:

$$X = \begin{bmatrix} x_0^{(1)} & x_1^{(1)} \\ x_0^{(2)} & x_1^{(2)} \\ x_0^{(3)} & x_1^{(3)} \end{bmatrix}, \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$

You can calculate the hypothesis as a column vector of size (m x 1) with:

$$h_\theta(X) = X\theta \tag{2.2}$$

For the rest of these notes, and other lecture notes, X will represent a matrix of training examples $x_{(i)}$ **stored row-wise.**

2.2 Cost Function

For the parameter vector θ (of type $\mathbb{R}^{n+1}$ or in $\mathbb{R}^{(n+1)\times 1}$), the cost function is:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 \tag{2.3}$$

The vectorized version is:

$$J(\theta) = \frac{1}{2m} (X\theta - \vec{y})^T (X\theta - \vec{y}) \tag{2.4}$$

Where $\vec{y}$ denotes the vector of all y values.

2.3 Gradient Descent for Multiple Variables

The gradient descent equation itself is generally the same form; we just have to repeat it for our "n" features:

repeat until convergence: {

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

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

$$\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_2^{(i)}$$

$\dots$

}

In other words:

repeat until convergence: {

$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$ for j := 0..n

}

2.4 Matrix Notation

The Gradient Descent rule can be expressed as:

$$\theta := \theta - \alpha \nabla J(\theta) \tag{2.5}$$

Where $\nabla J(\theta)$ is a column vector of the form:

$$\nabla J(\theta) = \begin{bmatrix} \frac{\partial J(\theta)}{\partial \theta_0} \\ \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{bmatrix} \tag{2.6}$$

The j-th component of the gradient is the summation of the product of two terms:

$$\begin{aligned} \frac{\partial J(\theta)}{\partial \theta_j} &= \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x_j^{(i)} \\ &= \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)} \cdot \left(h_\theta(x^{(i)}) - y^{(i)} \right) \end{aligned}$$

Sometimes, the summation of the product of two terms can be expressed as the product of two vectors.

Here, $x_j^{(i)}$, for i = 1,...,m, represents the m elements of the j-th column, $\vec{x_j}$, of the training set X.

The other term $\left(h_\theta(x^{(i)}) - y^{(i)} \right)$ is the vector of the deviations between the predictions $h_\theta(x^{(i)})$ and the true values $y^{(i)}$. Re-writing $\frac{\partial J(\theta)}{\partial \theta_j}$, we have:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \vec{x_j}^T (X\theta - \vec{y})$$

$$\nabla J(\theta) = \frac{1}{m} X^T (X\theta - \vec{y})$$

Finally, the matrix notation (vectorized) of the Gradient Descent rule is:

$$\theta := \theta - \frac{\alpha}{m} X^T (X\theta - \vec{y}) \tag{2.7}$$

2.5 Feature Normalization

We can speed up gradient descent by having each of our input values in roughly the same range. This is because θ will descend quickly on small ranges and slowly on large ranges, and so will oscillate inefficiently down to the optimum when the variables are very uneven.

The way to prevent this is to modify the ranges of our input variables so that they are all roughly the same. Ideally:

$$-1 \leq x_{(i)} \leq 1$$

or

$$-0.5 \leq x_{(i)} \leq 0.5$$

These aren't exact requirements; we are only trying to speed things up. The goal is to get all input variables into roughly one of these ranges, give or take a few.

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

$$x_i := \frac{x_i - \mu_i}{s_i} \tag{2.8}$$

Where μ_i is the **average** of all the values for feature (i) and s_i is the range of values (max - min), or s_i is the standard deviation.

Note that dividing by the range, or dividing by the standard deviation, give different results. The quizzes in this course use range - the programming exercises use standard deviation.

Example: x_i is housing prices with range of 100 to 2000, with a mean value of 1000.

Then, $x_i := \frac{price - 1000}{1900}$

2.6 Gradient Descent Tips

Debugging gradient descent. Make a plot with number of iterations on the x-axis. Now plot the cost function, $J(\theta)$ over the number of iterations of gradient descent. If $J(\theta)$ ever increases, then you probably need to decrease α.

Automatic convergence test. Declare convergence if $J(\theta)$ decreases by less than E in one iteration, where E is some small value such as 10-3. However in practice it's difficult to choose this threshold value.

It has been proven that if learning rate α is sufficiently small, then $J(\theta)$ will decrease on every iteration. Andrew Ng recommends decreasing α by multiples of 3.

2.6.1 Features and Polynomial Regression

We can improve our features and the form of our hypothesis function in a couple different ways.

We can combine multiple features into one. For example, we can combine x_1 and x_2 into a new feature x_3 by taking $x_1 \cdot x_2$

Polynomial Regression

Our hypothesis function need not be linear (a straight line) if that does not fit the data well.

We can **change the behavior or curve** of our hypothesis function by making it a quadratic, cubic or square root function (or any other form).

For example, if our hypothesis function is $h_\theta(x) = \theta_0 + \theta_1 x_1$ then 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 $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^3$

In the cubic version, we have created new features x_2 and x_3 where $x_2 = x_1^2$ and $x_3 = x_1^3$.

To make it a square root function, we could do: $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 \sqrt{x_1}$

Note that at 2:52 and through 6:22 in the "Features and Polynomial Regression" video, the curve that Prof Ng discusses about "doesn't ever come back down" is in reference to the hypothesis function that uses the sqrt() function (shown by the solid purple line), not the one that uses $size^2$ (shown with the dotted blue line). The quadratic form of the hypothesis function would have the shape shown with the blue dotted line if θ_2 was negative.

One important thing to keep in mind is, if you choose your features this way then feature scaling becomes very important.

eg. if x_1 has range 1 - 1000 then range of x_1^2 becomes 1 - 1000000 and that of x_1^3 becomes 1 - 1000000000.

2.7 Normal Equation

The "Normal Equation" is a method of finding the optimum theta **without iteration**.

$$\theta = (X^T X)^{-1} X^T y \tag{2.9}$$

There is no need to do feature scaling with the normal equation.

Mathematical proof of the Normal equation requires knowledge of linear algebra and is fairly involved, so you do not need to worry about the details.

Proofs are available at these links for those who are interested:

Wikipedia

thegreenplace

The following is a comparison of gradient descent and the normal equation:

Gradient Descent	**Normal Equation**
Need to choose alpha	No need to choose alpha
Needs may iterations	No need to iterate
$\mathcal{O}(kn^2)$	$\mathcal{O}(n^3)$, need to calculate inverse of X^TX
Works well when n is large	Slow if n is very large

With the normal equation, computing the inversion has complexity $\mathcal{O}(n^3)$. So if we have a very large number of features, the normal equation will be slow. In practice, when n exceeds 10,000 it might be a good time to go from a normal solution to an iterative process.

2.7.1 Normal Equation Noninvertibility

When implementing the normal equation in octave we want to use the `pinv` function rather than `inv`.

X^TX may be **noninvertible**. The common causes are:

- Redundant features, where two features are very closely related (i.e. they are linearly dependent)
- Too many features (e.g. $m \leq n$). In this case, delete some features or use "regularization" (to be explained in a later lesson).

Solutions to the above problems include deleting a feature that is linearly dependent with another or deleting one or more features when there are too many features.

Chapter 3

Week 3

3.1 Logistic Regression

Now we are switching from regression problems to **classification problems**. Don't be confused by the name "Logistic Regression"; it is named that way for historical reasons and is actually an approach to classification problems, not regression problems.

3.2 Binary Classification

Instead of our output vector y being a continuous range of values, it will only be 0 or 1.

$$y \in \{0, 1\}$$

Where 0 is usually taken as the "negative class" and 1 as the "positive class", but you are free to assign any representation to it.

We're only doing two classes for now, called a "Binary Classification Problem."

One method is to use linear regression and map all predictions greater than 0.5 as a 1 and all less than 0.5 as a 0. This method doesn't work well because classification is not actually a linear function.

Hypothesis Representation

Our hypothesis should satisfy:

$$0 \leq h_\theta(x) \leq 1$$

Our new form uses the **"Sigmoid Function"**, also called the **"Logistic Function"**:

$$h_\theta(x) = g(\theta^T x) \tag{3.1}$$

$$z = \theta^T x \tag{3.2}$$

$$g(z) = \frac{1}{1 + e^{-z}} \tag{3.3}$$

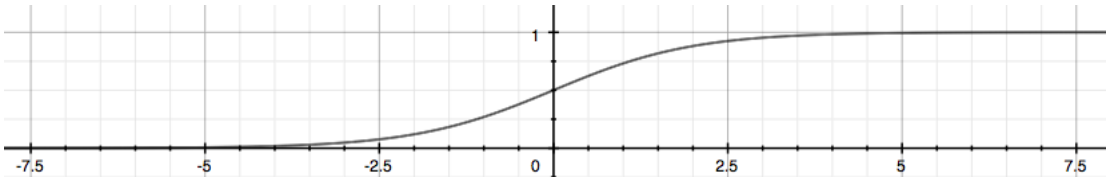

Figure 3.1: Sigmoid Function

The function g(z), shown here, maps any real number to the (0, 1) interval, making it useful for transforming an arbitrary-valued function into a function better suited for classification. Try playing with interactive plot of sigmoid function: click here!.

We start with our old hypothesis (linear regression), except that we want to restrict the range to 0 and 1. This is accomplished by plugging θ^T into the Logistic Function.

h_θ will give us the probability that our output is 1. For example, $h_\theta(x) = 0.7$ gives us the probability of 70% that our output is 1.

$$h_\theta(x) = P(y = 1|x; \theta) = 1 - P(y = 0|x; \theta)$$
$$P(y = 0|x; \theta) + P(y = 1|x; \theta) = 1$$

Our probability that our prediction is 0 is just the complement of our probability that it is 1 (e.g. if probability that it is 1 is 70%, then the probability that it is 0 is 30%).

3.3 Decision Boundary

In order to get our discrete 0 or 1 classification, we can translate the output of the hypothesis function as follows:

$$h_\theta(x) \geq 0.5 \rightarrow y = 1$$
$$h_\theta(x) < 0.5 \rightarrow y = 0$$

The way our logistic function g behaves is that when its input is greater than or equal to zero, its output is greater than or equal to 0.5:

$$g(z) \geq 0.5$$
$$\text{when } z \geq 0$$

Remember:

$$z = 0, e^0 = 1 \Rightarrow g(z) = 1/2$$
$$z \rightarrow \infty, e^{-\infty} \rightarrow 0 \Rightarrow g(z) = 1$$
$$z \rightarrow -\infty, e^{\infty} \rightarrow \infty \Rightarrow g(z) = 0$$

So if our input to g is θ^T, then that means:

$$h_\theta(x) = g(\theta^T x) \geq 0.5$$
$$\text{when } \theta^T x \geq 0$$

From these statements we can now say:

$$\theta^T x \geq 0 \Rightarrow y = 1$$
$$\theta^T x < 0 \Rightarrow y = 0$$

The **decision boundary** is the line that separates the area where y = 0 and where y = 1. It is created by our hypothesis function.

Example:

$$\theta = \begin{bmatrix} 5 \\ -1 \\ 0 \end{bmatrix}$$
$$y = 1 \; if \; 5 + (-1)x_1 + 0x_2 \geq 0$$
$$5 - x_1 \geq 0$$
$$-x_1 \geq -5$$
$$x_1 \leq 5$$

In this case, our decision boundary is a straight vertical line placed on the graph where $x_1 = 5$, and everything to the left of that denotes y = 1, while everything to the right denotes y = 0.

Again, the input to the sigmoid function g(z) (e.g. $\theta^T X$) doesn't need to be linear, and could be a function that describes a circle (e.g.$z = \theta_0 + \theta_1 x_1^2 + \theta_2 x_2^2$) or any shape to fit our data.

3.4 Cost Function

We cannot use the same cost function that we use for linear regression because the Logistic Function will cause the output to be wavy, causing many local optima. In other words, it will not be a convex function.

Instead, our cost function for logistic regression looks like:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \text{Cost}(h_\theta(x^{(i)}), y^{(i)}) \tag{3.4}$$

$$\text{Cost}(h_\theta(x), y) = -\log(h_\theta(x)) \qquad \text{if y} = 1$$
$$\text{Cost}(h_\theta(x), y) = -\log(1 - h_\theta(x)) \qquad \text{if y} = 0$$

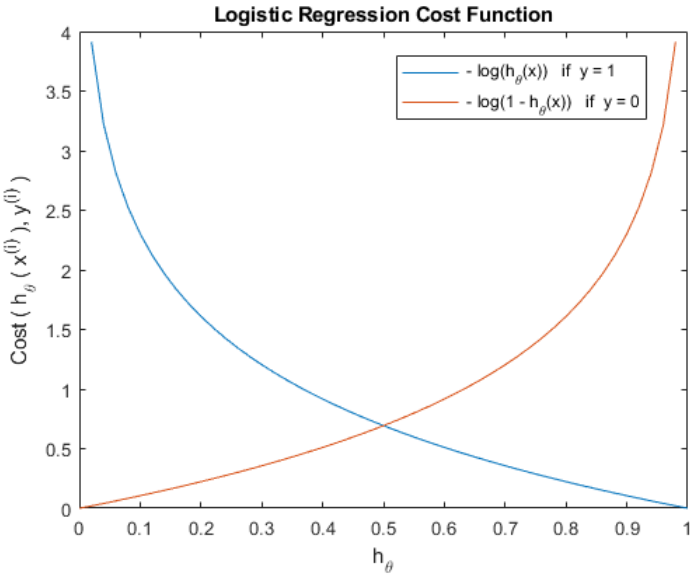

Figure 3.2: Logistic Regression Cost Function

The more our hypothesis is off from y, the larger the cost function output. If our hypothesis is equal to y, then our cost is 0:

$$\text{Cost}(h_\theta(x), y) = 0 \text{ if } h_\theta(x) = y$$
$$\text{Cost}(h_\theta(x), y) \to \infty \text{ if } y = 0 \text{ and } h_\theta(x) \to 1$$
$$\text{Cost}(h_\theta(x), y) \to \infty \text{ if } y = 1 \text{ and } h_\theta(x) \to 0$$

If our correct answer 'y' is 0, then the cost function will be 0 if our hypothesis function also outputs 0. If our hypothesis approaches 1, then the cost function will approach infinity.

If our correct answer 'y' is 1, then the cost function will be 0 if our hypothesis function outputs 1. If our hypothesis approaches 0, then the cost function will approach infinity.

Note that writing the cost function in this way guarantees that $J(\theta)$ is convex for logistic regression.

3.5 Simplified Cost Function and Gradient Descent

We can compress our cost function's two conditional cases into one case:

$$\text{Cost}(h_\theta(x), y) = -y \ \log(h_\theta(x)) - (1 - y) \log(1 - h_\theta(x)) \tag{3.5}$$

Notice that when y is equal to 1, then the second term $(1-y)\log(1-h_\theta(x))$ will be zero and will not affect the result. If y is equal to 0, then the first term $-y\log(h_\theta(x))$ will be zero and will not affect the result.

We can fully write out our entire cost function as follows:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))] \tag{3.6}$$

A vectorized implementation is:

$$h = g(X\theta) \tag{3.7}$$

$$J(\theta) = \frac{1}{m}\cdot\left(-y^T\log(h) - (1-y)^T\log(1-h)\right) \tag{3.8}$$

3.6 Week 4

This is week number 4

3.7 Week 5

This is week number 5

3.8 The Problem of Overfitting

Consider the problem of predicting y from $x \in R$. The leftmost figure 3.3 shows the result of fitting a $y = \theta_0 + \theta_1 x$ to a dataset. We see that the data doesn't really lie on straight line, and so the fit is not very good.

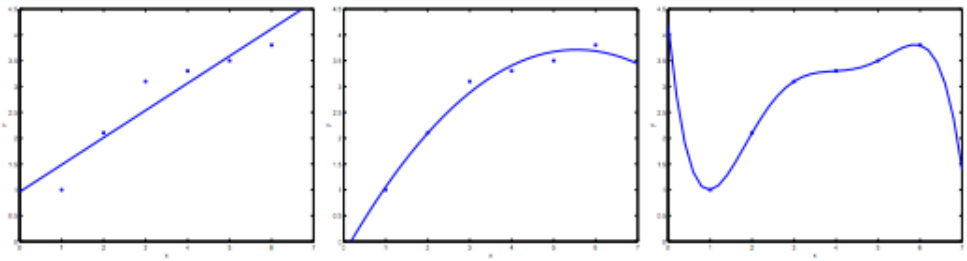

Figure 3.3: The Problem of Overfitting

Instead, if we had added an extra feature x^2, and fit $y = \theta_0 + \theta_1 x + \theta_2 x^2$, then we obtain a slightly better fit to the data (See middle figure). Naively, it might seem that the more features we add, the better. However, there is also a danger in adding too many features:

The rightmost figure is the result of fitting a 5^{th} order polynomial $y = \sum_{j=0}^{5} \theta_j$. We see that even though the fitted curve passes through the data perfectly, we would not expect this to be a very good predictor of, say, housing prices (y) for different living areas (x). Without formally defining what these terms mean, we'll say the figure on the left shows an instance of **underfitting**-in which the data clearly shows structure not captured by the model-and the figure on the right is an example of **overfitting**.

Underfitting, or high bias, is when the form of our hypothesis function h maps poorly to the trend of the data. It is usually caused by a function that is too simple or uses too few features. At the other extreme, overfitting, or high variance, is caused by a hypothesis function that fits the available data but does not generalize well to predict new data. It is usually caused by a complicated function that creates a lot of unnecessary curves and angles unrelated to the data.

This terminology is applied to both linear and logistic regression. There are two main options to address the issue of overfitting:

1) Reduce the number of features:

- Manually select which features to keep.
- Use a model selection algorithm (studied later in the course).

2) Regularization

- Keep all the features, but reduce the magnitude of parameters θ_j
- Regularization works well when we have a lot of slightly useful features.

Chapter 4

Summary

4.1 Lineal Regression

4.1.1 Hypothesis

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \cdots + \theta_n x_n \tag{4.1}$$

$$h_\theta(x) = \theta^T x \tag{4.2}$$

$$h_\theta(X) = X\theta \tag{4.3}$$

4.1.2 Cost Function

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 \tag{4.4}$$

$$J(\theta) = \frac{1}{2m} (X\theta - \vec{y})^T (X\theta - \vec{y}) \tag{4.5}$$

4.1.3 Gradient Descent

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)} \tag{4.6}$$

$$\theta := \theta - \frac{\alpha}{m} X^T (X\theta - \vec{y}) \tag{4.7}$$

for j=0, 1,..., n:

4.1.4 Normal Equation

$$\theta = (X^T X)^{-1} X^T y \tag{4.8}$$

4.2 Logistic Regression

4.2.1 Hypothesis

$$0 \leq h_\theta(x) \leq 1 \tag{4.9}$$

"Sigmoid Function," also called the "Logistic Function":

$$h_\theta(x) = g(\theta^T x) \tag{4.10}$$
$$z = \theta^T x \tag{4.11}$$
$$g(z) = \frac{1}{1+e^{-z}} \tag{4.12}$$

$$h_\theta(x) = P(y=1|x;\theta) = 1 - P(y=0|x;\theta) \tag{4.13}$$
$$P(y=0|x;\theta) + P(y=1|x;\theta) = 1 \tag{4.14}$$

4.2.2 Cost Function

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))] \tag{4.15}$$

$$h = g(X\theta) \tag{4.16}$$
$$J(\theta) = \frac{1}{m}\cdot\left(-y^T\log(h) - (1-y)^T\log(1-h)\right) \tag{4.17}$$

4.2.3 Gradient Descent

$$\theta_j := \theta_j - \frac{\alpha}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} \tag{4.18}$$
$$\theta := \theta - \frac{\alpha}{m}X^T(g(X\theta) - \vec{y}) \tag{4.19}$$