Week 1

Introduction

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.

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.

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.

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.

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. It’s not just about clustering. For example, associative memory is unsupervised learning.

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.

Associative: Suppose a doctor over years of experience forms associations in his mind between patient characteristics and illnesses that they have. If a new patient shows up then based on this patient’s characteristics such as symptoms, family medical history, physical attributes, mental outlook, etc the doctor associates possible illness or illnesses based on what the doctor has seen before with similar patients. This is not the same as rule based reasoning as in expert systems. In this case we would like to estimate a mapping function from patient characteristics into illnesses.

Linear Regression with One Variable

Model Representation

Recall that in *regression problems*, we are taking input variables and trying to map 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 from a **single input** value. We're doing **supervised learning** here, so that means we already have an idea what the input/output cause and effect should be.

The Hypothesis Function

Our hypothesis function has the general form:

$$h_\theta(x)=\theta_0+\theta_1 x$$

We give to h_θ values for θ_0 and θ_1 to get our output 'y'. In other words, we are trying to create a function called h_θ that is able to reliably map our input data (the x's) to our output data (the y's).

Example:

x (input)	y (output)
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 $h_\theta(x) = 2+2x$.

So for input of 1 to our hypothesis, y will be 4. This is off by 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.

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

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 ($\frac{1}{2m}$) 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.

Gradient Descent

So we have our hypothesis function and we have a way of measuring how accurate it is. Now what we need is a way to automatically improve our 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 for the combinations of parameters). 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 guesses of our hypothesis function.

We put θ_0 on the x axis and θ_1 on the z axis, with the cost function on the vertical y 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 line tangent 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 that derivative by the parameter α, called the **learning rate**.

The gradient descent equation is:

repeat until convergence:

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

for j = 0 and j = 1

Intuitively, this could be thought of as:

repeat until convergence:

$\theta_j := \theta_j - \alpha$[Slope of tangent aka derivative]

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: {

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

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

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 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.

What's Next

Instead of using linear regression on just one input variable, we'll generalize and expand our concepts so that we can predict data with multiple input variables. Also, we'll solve for θ_0 and θ_1 exactly without needing an iterative function like gradient descent.

Linear Algebra Review

Khan Academy has excellent Linear Algebra Tutorials.

This online Linear Algebra text is also an excellent resource, particularly for a proof of the normal equation.

Matrices and Vectors

Matrices are 2-dimensional arrays:

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

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.
* 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
* $\mathbb{R}^{\mathbb{n}}$ refers to the **set** of n-dimensional vectors of real numbers

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:

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

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 **rows** of the vector must equal the number of **columns** of the matrix.

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

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.

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 and 0's elsewhere.

When multiplying the identity matrix after some matrix, the square identity matrix should match the other matrix's **columns**. When multiplying the identity matrix before some other matrix, the square identity matrix should match the other matrix's **rows**.

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].

The **transposition** of a matrix is like rotating the matrix once clockwise and then reversing it:

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

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

In other words:

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

Footnotes

[1]: As described in the course video, this octave function computes the pseudo inverse for singular matrices which do not have inverses.