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 and Unsupervised learning.

## 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 a person, we have to predict their age on the basis of the given picture

(b) Classification - Given a patient with a tumor, we have to predict whether the tumor is malignant or benign.

## Unsupervised Learning

Unsupervised learning 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.

**Example:**

Clustering: Take a collection of 1,000,000 different genes, and find a way to automatically group these genes into groups that are somehow similar or related by different variables, such as lifespan, location, roles, and so on.

Non-clustering: The "Cocktail Party Algorithm", allows you to find structure in a chaotic environment. (i.e. identifying individual voices and music from a mesh of sounds at a [cocktail party](https://en.wikipedia.org/wiki/Cocktail_party_effect)). # 将复合的声音分类出来。

# Model Representation

To establish notation for future use, we’ll use *x*(*i*) to denote the “input” variables (living area in this example), also called input features, and *y*(*i*) to denote the “output” or target variable that we are trying to predict (price). A pair (*x*(*i*),*y*(*i*)) is called a training example, and the dataset that we’ll be using to learn—a list of m training examples (*x*(*i*),*y*(*i*));*i*=1,...,*m*—is called a training set. Note that the superscript “(i)” in the notation is simply an index into the training set, and has nothing to do with exponentiation. We will also use X to denote the space of input values, and Y to denote the space of output values. In this example, X = Y = ℝ.

To describe the supervised learning problem slightly more formally, our goal is, given a training set, to learn a function h : X → Y so that h(x) is a “good” predictor for the corresponding value of y. For historical reasons, this function h is called a hypothesis. Seen pictorially, the process is therefore like this:



When the target variable that we’re trying to predict is continuous, such as in our housing example, we call the learning problem a regression problem. When y can take on only a small number of discrete values (such as if, given the living area, we wanted to predict if a dwelling is a house or an apartment, say), we call it a classification problem.

# Cost Function

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

*J*(*θ*0,*θ*1)=12*m*∑*i*=1*m*(*y*^*i*−*yi*)2=12*m*∑*i*=1*m*(*hθ*(*xi*)−*yi*)2

To break it apart, it is 12 *x*¯ where *x*¯ is the mean of the squares of *hθ*(*xi*)−*yi* , 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 (12) as a convenience for the computation of the gradient descent, as the derivative term of the square function will cancel out the 12 term. The following image summarizes what the cost function does:

# Cost Function - Intuition I

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 a straight line (defined by *hθ*(*x*)) which passes through these scattered data points.

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. Ideally, the line should pass through all the points of our training data set. In such a case, the value of *J*(*θ*0,*θ*1) will be 0. The following example shows the ideal situation where we have a cost function of 0.

When *θ*1=1, we get a slope of 1 which goes through every single data point in our model. Conversely, when *θ*1=0.5, we see the vertical distance from our fit to the data points increase.

This increases our cost function to 0.58. Plotting several other points yields to the following graph:

Thus as a goal, we should try to minimize the cost function. In this case, *θ*1=1 is our global minimum.

# 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 the 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). We are not graphing x and y itself, but the parameter range of our hypothesis function and the cost resulting from selecting a 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. The graph below depicts such a setup.

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 red arrows show the minimum points in the graph.

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. The size of each step is determined by the parameter α, which is called the learning rate.

For example, the distance between each 'star' in the graph above represents a step determined by our parameter α. A smaller α would result in a smaller step and a larger α results in a larger step. The direction in which the step is taken is determined by the partial derivative of *J*(*θ*0,*θ*1). Depending on where one starts on the graph, one could end up at different points. The image above shows us two different starting points that end up in two different places.

The gradient descent algorithm is:

repeat until convergence:

*θj*:=*θj*−*α*∂∂*θjJ*(*θ*0,*θ*1)

where

j=0,1 represents the feature index number.

At each iteration j, one should simultaneously update the parameters *θ*1,*θ*2,...,*θn*. Updating a specific parameter prior to calculating another one on the *j*(*th*) iteration would yield to a wrong implementation.

# Gradient Descent Intuition

In this video we explored the scenario where we used one parameter *θ*1 and plotted its cost function to implement a gradient descent. Our formula for a single parameter was :

Repeat until convergence:

|  |
| --- |
| *θ*1:=*θ*1−*αddθ*1*J*(*θ*1) |

Regardless of the slope's sign for *ddθ*1*J*(*θ*1), *θ*1 eventually converges to its minimum value. The following graph shows that when the slope is negative, the value of *θ*1 increases and when it is positive, the value of *θ*1 decreases.



On a side note, we should adjust our parameter *α* to ensure that the gradient descent algorithm converges in a reasonable time. Failure to converge or too much time to obtain the minimum value imply that our step size is wrong.



### How does gradient descent converge with a fixed step size *α*?

The intuition behind the convergence is that *ddθ*1*J*(*θ*1) approaches 0 as we approach the bottom of our convex function. At the minimum, the derivative will always be 0 and thus we get:

|  |
| --- |
| *θ*1:=*θ*1−*α*∗0 |



Gradient Descent For Linear Regression

**Note:** [At 6:15 "h(x) = -900 - 0.1x" should be "h(x) = 900 - 0.1x"]

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 :

|  |
| --- |
| repeat until convergence: {*θ*0:=*θ*1:=}*θ*0−*α*1*m*∑*i*=1*m*(*hθ*(*xi*)−*yi*)*θ*1−*α*1*m*∑*i*=1*m*((*hθ*(*xi*)−*yi*)*xi*) |

where m is the size of the training set, *θ*0 a constant that will be changing simultaneously with *θ*1 and *xi*,*yi* are values of the given training set (data).

Note that we have separated out the two cases for *θj* into separate equations for *θ*0 and *θ*1 ; and that for *θ*1 we are multiplying *xi* at the end due to the derivative. The following is a derivation of ∂∂*θjJ*(*θ*) for a single example :



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.

So, this is simply gradient descent on the original cost function J. This method looks at every example in the entire training set on every step, and is called **batch gradient descent**. Note that, while gradient descent can be susceptible to local minima in general, the optimization problem we have posed here for linear regression has only one global, and no other local, optima; thus gradient descent always converges (assuming the learning rate α is not too large) to the global minimum. Indeed, J is a convex quadratic function. Here is an example of gradient descent as it is run to minimize a quadratic function.



The ellipses shown above are the contours of a quadratic function. Also shown is the trajectory taken by gradient descent, which was initialized at (48,30). The x’s in the figure (joined by straight lines) mark the successive values of θ that gradient descent went through as it converged to its minimum.

Matrices and Vectors

Matrices are 2-dimensional arrays:

|  |
| --- |
| ⎡⎣⎢⎢⎢*adgjbehkcfil*⎤⎦⎥⎥⎥ |

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:

|  |
| --- |
| ⎡⎣⎢⎢*wxyz*⎤⎦⎥⎥ |

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

**Notation and terms**:

* *Aij* 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.
* *vi* 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.
* R refers to the set of scalar real numbers.
* Rn refers to the set of n-dimensional vectors of real numbers.

% The ; denotes we are going back to a new row.

A = [1, 2, 3; 4, 5, 6; 7, 8, 9; 10, 11, 12]

% Initialize a vector

v = [1;2;3]

% Get the dimension of the matrix A where m = rows and n = columns

[m,n] = size(A)

% You could also store it this way

dim\_A = size(A)

% Get the dimension of the vector v

dim\_v = size(v)

% Now let's index into the 2nd row 3rd column of matrix A

A\_23 = A(2,3)

Addition and Scalar Multiplication

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

|  |
| --- |
| [*acbd*]+[*wyxz*]=[*a*+*wc*+*yb*+*xd*+*z*] |

Subtracting Matrices:

|  |
| --- |
| [*acbd*]−[*wyxz*]=[*a*−*wc*−*yb*−*xd*−*z*] |

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

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

|  |
| --- |
| [*acbd*]∗*x*=[*a*∗*xc*∗*xb*∗*xd*∗*x*] |

In scalar division, we simply divide every element by the scalar value:

|  |
| --- |
| [*acbd*]/*x*=[*a*/*xc*/*xb*/*xd*/*x*] |

Experiment below with the Octave/Matlab commands for matrix addition and scalar multiplication. Feel free to try out different commands. Try to write out your answers for each command before running the cell below.

% Initialize matrix A and B

A = [1, 2, 4; 5, 3, 2]

B = [1, 3, 4; 1, 1, 1]

% Initialize constant s

s = 2

% See how element-wise addition works

add\_AB = A + B

% See how element-wise subtraction works

sub\_AB = A - B

% See how scalar multiplication works

mult\_As = A \* s

% Divide A by s

div\_As = A / s

% What happens if we have a Matrix + scalar?

add\_As = A + s

Matrix-Vector Multiplication

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

|  |
| --- |
| ⎡⎣*acebdf*⎤⎦∗[*xy*]=⎡⎣⎢*a*∗*x*+*b*∗*yc*∗*x*+*d*∗*ye*∗*x*+*f*∗*y*⎤⎦⎥ |

The result is a **vector**. 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**.

Below is an example of a matrix-vector multiplication. Make sure you understand how the multiplication works. Feel free to try different matrix-vector multiplications.

% Initialize matrix A

A = [1, 2, 3; 4, 5, 6;7, 8, 9]

% Initialize vector v

v = [1; 1; 1]

% Multiply A \* v

Av = A \* v

Matrix Multiplication Properties

* Matrices are not commutative: *A*∗*B*≠*B*∗*A*
* Matrices are 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.

|  |
| --- |
| ⎡⎣100010001⎤⎦ |

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

% Initialize random matrices A and B

A = [1,2;4,5]

B = [1,1;0,2]

% Initialize a 2 by 2 identity matrix

I = eye(2)

% The above notation is the same as I = [1,0;0,1]

% What happens when we multiply I\*A ?

IA = I\*A

% How about A\*I ?

AI = A\*I

% Compute A\*B

AB = A\*B

% Is it equal to B\*A?

BA = B\*A

% Note that IA = AI but AB != BA

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 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*=⎡⎣*acebdf*⎤⎦ |

|  |
| --- |
| *AT*=[*abcdef*] |

In other words:

*Aij*=*ATji*

% Initialize matrix A

A = [1,2,0;0,5,6;7,0,9]

% Transpose A

A\_trans = A'

% Take the inverse of A

A\_inv = inv(A)

% What is A^(-1)\*A?

A\_invA = inv(A)\*A

**Access MATLAB Online**

A MATLAB Online license is being provided by MathWorks to students of Machine Learning for the duration of this course. MATLAB Online is the online version of the MATLAB desktop program and it retains most of the features of the original program in a web-based interface. No download or installation is required, and the program can be accessed from any computer running a common web browser.

To gain access to MATLAB Online, follow these steps:

1. If you do not already have one, create a MathWorks account at: <https://www.mathworks.com/mwaccount/register>

2. Click on the [Machine Learning license link](https://www.mathworks.com/licensecenter/classroom/machine_learning/) then provide your MathWorks account log-in information when requested.

3. Click on the blue 'Access MATLAB Online' button when it appears, and log-in to MATLAB Online with your MathWorks account information.

4. Bookmark <https://matlab.mathworks.com/> for quicker access to MATLAB Online in the future.

For help with access or technical issues with MATLAB Online, see the MATLAB Help discussion forum which is monitored by MathWorks.

**Upload the Programming Exercises to MATLAB Online（联系操作）**

The Machine Learning programming exercises consist of code files, data files, and instructions. They are each provided as a compressed .zip file in the programming assignment course pages later in the course. In order to upload the programming exercises to MATLAB Online, complete the following steps:

1. Download the programming exercise .zip file, e.g. machine-learning-ex**n**.zip (where **n** is the exercise number).

2. Log in to MATLAB Online, then drag and drop the exercise .zip file into your Current Folder (or use the ‘Upload’ button in the Home tab).\*

3. Enter and run the following command at the command line to unzip the exercise folder: **unzip machine-learning-exn;** (replace the **n** with the exercise number)

5. Confirm the exercise folder was unzipped correctly, then delete the zip file.

6. Right-click the unzipped machine-learning-ex**n** folder, and select 'Remove from Path -> Selected Folder and Subfolders'. This is to prevent MATLAB from using incorrect files from another exercise folder.

**\*DO NOT unzip the homework files on your computer and upload the files individually.**

Installing Octave on Windows

Installing Octave on Windows

**Use this link to install Octave for windows:** <http://wiki.octave.org/Octave_for_Microsoft_Windows>

**Octave on Windows can be used to submit programming assignments** in this course but will likely need a patch provided in the discussion forum. Refer to [https://www.coursera.org/learn/machine-learning/discussions/vgCyrQoMEeWv5yIAC00Eog?](https://www.coursera.org/learn/machine-learning/discussions/vgCyrQoMEeWv5yIAC00Eog?page=2) for more information about the patch for your version.

"**Warning:** Do not install Octave 4.0.0"; checkout the "Resources" menu's section of "Installation Issues".

Installing Octave on Mac OS X (10.10 Yosemite and 10.9 Mavericks)

Mac OS X [has a feature called Gatekeeper](http://support.apple.com/en-us/HT202491) that may only let you install applications from the Mac App Store. You may need to configure it to allow the Octave installer. Visit your System Preferences, click Security & Privacy, and check the setting to allow apps downloaded from Anywhere. You may need to enter your password to unlock the settings page.

2. Download [the Octave 3.8.0 installer](http://sourceforge.net/projects/octave/files/Octave%20MacOSX%20Binary/2013-12-30%20binary%20installer%20of%20Octave%203.8.0%20for%20OSX%2010.9.1%20%28beta%29/GNU_Octave_3.8.0-6.dmg/download) or the latest version that isn't 4.0.0. The file is large so this may take some time.

3. Open the downloaded image, probably named *GNU\_Octave\_3.8.0-6.dmg* on your computer, and then open *Octave-3.8.0-6.mpkg* inside.

4. Follow the installer’s instructions. You may need to enter the administrator password for your computer.

5. After the installer completes, Octave should be installed on your computer. You can find *Octave-cli* in your Mac’s Applications, which is a text interface for Octave that you can use to complete Machine Learning’s programming assignments.

Octave also includes an experimental graphical interface which is called *Octave-gui*, also in your Mac’s Applications, but we recommend using *Octave-cli* because it’s more stable.

Note: If you use a package manager (like MacPorts or Homebrew), we recommend you follow [the package manager installation instructions](http://wiki.octave.org/Octave_for_MacOS_X#Package_Managers).

"**Warning:** Do not install Octave 4.0.0"; checkout the "Resources" menu's section of "Installation Issues".

Installing Octave on GNU/Linux

Installing Octave on GNU/Linux

We recommend [using your system package manager to install Octave](http://wiki.octave.org/Octave_for_GNU/Linux).

On Ubuntu, you can use:

* *sudo apt-get update && sudo apt-get install octave*

On Fedora, you can use:

* *sudo yum install octave-forge*

Please consult [the Octave maintainer’s instructions](http://wiki.octave.org/Octave_for_GNU/Linux) for other GNU/Linux systems.

"**Warning:** Do not install Octave 4.0.0"; checkout the "Resources" menu's section of "Installation Issues".

More Octave/MATLAB resources

Octave Resources

At the Octave command line, typing **help** followed by a function name displays documentation for a built-in function. For example, **help plot** will bring up help information for plotting. Further documentation can be found at the Octave [documentation pages](http://www.gnu.org/software/octave/doc/interpreter/).

MATLAB Resources

At the MATLAB command line, typing help followed by a function name displays documentation for a built-in function. For example, help plot will bring up help information for plotting. Further documentation can be found at the MATLAB [documentation pages](http://www.mathworks.com/help/matlab/).

Introduction to MATLAB with Onramp

Made for MATLAB beginners or those looking for a quick refresh, the MATLAB Onramp is a 1-2 hour interactive introduction to the basics of MATLAB programming. **Octave users are also welcome to use Onramp** (requires creation of a free MathWorks account). To access Onramp:

1. If you don’t already have one, create a MathWorks account at: <https://www.mathworks.com/mwaccount/register>

2. Go to: <https://matlabacademy.mathworks.com/> and click on the MATLAB Onramp button to start learning MATLAB!

MATLAB Programming Tutorials

These short tutorial videos introduce MATLAB and cover various programming topics used in the assignments. Feel free to watch some now and return to reference them as you work through the programming assignments. Many of the topics below are also covered in MATLAB Onramp. **\*Indicates content covered in Onramp.**

Get Started with MATLAB and MATLAB Online

* [What is MATLAB?](https://youtu.be/WYG2ZZjgp5M)\*
* [MATLAB Variables](https://youtu.be/0w9NKt6Fixk)\*
* [MATLAB as a Calculator](https://youtu.be/aRSkNpCSgWY)\*
* [MATLAB Functions](https://youtu.be/RJp46UVQBic)\*
* [Getting Started with MATLAB Online](https://youtu.be/XjzxCVWKz58)
* [Managing Files in MATLAB Online](https://youtu.be/B3lWLIrYjC0)

Vectors

* [Creating Vectors](https://youtu.be/R5Mnkrk9Mos)\*
* [Creating Uniformly Spaced Vectors](https://youtu.be/_zqTOV5yl8Y)\*
* [Accessing Elements of a Vector Using Conditions](https://youtu.be/8D04GW_foQ0)\*
* [Calculations with Vectors](https://youtu.be/VQaZ0TvjF0c)\*
* [Vector Transpose](https://youtu.be/vgRLwjHBmsg)

Visualization

* [Line Plots](https://youtu.be/-hhJoveE4sY)\*
* [Annotating Graphs](https://youtu.be/JyovEGPSdoI)\*
* [Multiple Plots](https://youtu.be/fBx8EFuXFLM)\*

Matrices

* [Creating Matrices](https://youtu.be/qdTdwTh6jMo)\*
* [Calculations with Matrices](https://youtu.be/mzzJ9gnMrYE)\*
* [Accessing Elements of a Matrix](https://youtu.be/uWPHxpTuZRA)\*
* [Matrix Creation Functions](https://youtu.be/VPcbpVd_mPA)\*
* [Combining Matrices](https://youtu.be/ejTr3ekTTyA)
* [Determining Array Size and Length](https://youtu.be/IF9-ffmxuy8)
* [Matrix Multiplication](https://youtu.be/4hsx3bdNjGk)
* [Reshaping Arrays](https://youtu.be/UQpDIHlFo8A)
* [Statistical Functions with Matrices](https://youtu.be/Y97W3_u7cM4)

MATLAB Programming

* [Logical Variables](https://youtu.be/bRMg4GsFDQ8)\*
* [If-Else Statements](https://youtu.be/JZSuU-Laigo)\*
* [Writing a FOR loop](https://youtu.be/lg65bzgvI5c)\*
* [Writing a WHILE Loop](https://youtu.be/PKH5lCMJXbk)
* [Writing Functions](https://youtu.be/GrcNN04eqXU)
* [Passing Functions as Inputs](https://youtu.be/aNCwR9dRjHs)

Troubleshooting

* [Using Online Documentation](https://youtu.be/54n5zJwR8aM)\*
* [Which File or Variable Am I Using?](https://youtu.be/Z09BvGeYNdE)
* [Troubleshooting Code with the Debugger](https://youtu.be/DB4aJMnZtNQ)