Sessional Assignment

Data Mining

Marks: **20**

NAME: SHEHARYAR TAHIR
ID# 13484

Note:

- Every student must do it yourself; Plagiarized assignment will not be acceptable.
- Make a Proper Word Document/ PDF or PowerPoint Presentation of this assignment, Picture will not be acceptable.
- Must submit before deadline.

Complete the Machine Learning course at Coursera. Till 5th of June and show the result.

ANSWER:

Machine Learning Course from Coursera

Lecture 1: Introduction to Machine learning:

Machine learning is basically the science of getting computers to learn, without being explicitly programmed. We probably use it dozens of times a day without even knowing it. Each time you do a web search on Google or Bing that works so well because their machine learning software has figured out how to rank what pages. When Facebook or Apple's photo application recognizes your friends in your pictures, that's also machine learning. Each time you read your email and a spam filter saves you from having to wade through tons of spam, again, that's because your computer has learned to distinguish spam from non-spam email. So, that's machine learning. There's a science of getting computers to learn without being explicitly programmed. One of the research projects that I'm working on is getting robots to tidy up the house. How do you go about doing that? Well what you can do is have the robot watch you demonstrate the task and learn from that. The robot can then watch what objects you pick up and where to put them and try to do the same thing even when you aren't there. For me, one of the reasons I'm excited about this is the AI, or artificial intelligence problem. Building truly intelligent machines, we can do just about anything that you or I can do. Many scientists think the best way to make progress on this is through learning algorithms called neural networks, which mimic how the human brain works

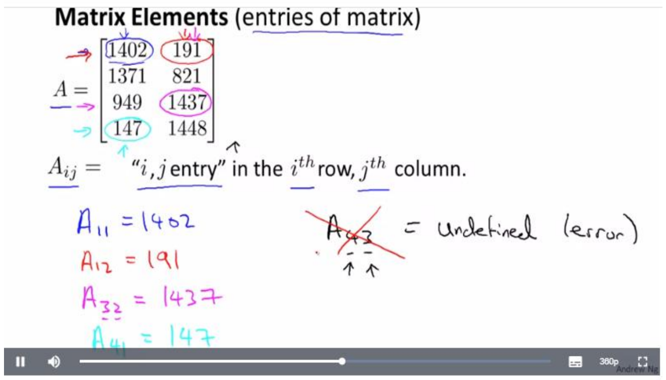
Lecture 2: Supervised algorithm that is Linear Regression:

They have used some motivating example of predicting housing prices. They have used a data set of housing prices from the city of Portland, Oregon. And they plot their data set of a number of houses that were different sizes that were sold for a range of different prices. Let's say that given this data set, we have a friend that's trying to sell a house and let's see if friend's house is size of 1250 square feet and you want to tell them how much they might be able to sell the house for. Well one thing we could do is fit a model. Maybe fit a straight line to this data. Looks something like that and based on that, maybe you could tell your friend that let's say maybe he can sell the house for around $220,000. So this is an example of a supervised learning algorithm. And it's supervised learning because we're given the, quotes, "right answer" for each of our examples. Namely they told what was the actual house, what was the actual price of each of the houses in our data set were sold for and moreover, this is an example of a regression problem where the term regression refers to the fact that we are predicting a real-valued output namely the price. The other most common type of supervised learning problem is called the classification problem where we predict discrete-valued outputs such as if we are looking at cancer tumors and trying to decide if a tumor is malignant or benign. So that's a zero-one valued discrete output. More formally, in supervised learning, they had a data set and this data set is called a training set. So for housing prices example, we have a training set of different housing prices and our job is to learn from this data how to predict prices of the houses. They used lower case m throughout this course to denote the number of training examples. So in the data set, if we have, let's say 47 rows in this table. Then we have 47 training examples and m equals 47. They use lowercase x to denote the input variables often also called the features. That would be the x is here, it would the input features. And used y to denote my output variables or the target variable to predict and so that's the second column here. They used (x, y) to denote a single training example. So, a single row in this table corresponds to a single training example and to refer to a specific training example, they used this notation x(i) comma gives me y(i) And, we're going to use this to refer to the ith training example. So this superscript i over here, this is not exponentiation right? This (x(i), y(i)), the superscript i in parentheses that's just an index into my training set and refers to the ith row in this table, okay? So this is not x to the power of i, y to the power of i. Instead (x(i), y(i)) just refers to the ith row of this table. So for example, x(1) refers to the input value for the first training example so that's 2104. That's this x in the first row. x(2) will be equal to 1416 right? That's the second x and y(1) will be equal to 460. The first, the y value for my first training example, that's what that (1) refers to. So as mentioned, occasionally I'll ask you a question to let you check your understanding and a few seconds in this video a multiple-choice question will pop up in the video. When it does, please use your mouse to select what you think is the right answer. What defined by the training set is. So here's how this supervised learning algorithm works. We saw that with the training set like our training set of housing prices and we feed that to our learning algorithm. Is the job of a learning algorithm to then output a function which by convention is usually denoted lowercase h and h stands for hypothesis And what the job of the hypothesis is, is, is a function that takes as input the size of a house like maybe the size of the new house your friend's trying to sell so it takes in the value of x and it tries to output the estimated value of y for the corresponding house. So h is a function that maps from x's to y's. People often ask me, you know, why is this function called hypothesis. Some of you may know the meaning of the term hypothesis, from the dictionary or from science or whatever. It turns out that in machine learning, this is a name that was used in the early days of machine learning and it kinda stuck. 'Cause maybe not a great name for this sort of function, for mapping from sizes of houses to the predictions, that you know...

When designing a learning algorithm, the next thing we need to decide is how do we represent this hypothesis h. For this they choose their initial choice , for representing the hypothesis, will be the following. We're going to represent h as follows. And we will write this as h<u>theta(x) equals theta<u>0</u></u> plus theta<u>1 of x. And as a shorthand, sometimes instead of writing, you</u> know, h subscript theta of x, sometimes there's a shorthand, they just write as a h of x. But more often I'll write it as a subscript theta over there. And plotting this in the pictures, all this means is that, we are going to predict that y is a linear function of x. Right, so that's the data set and what this function is doing, is predicting that y is some straight line function of x. That's h of x equals theta 0 plus theta 1 x, okay? And why a linear function? Well, sometimes we'll want to fit more complicated, perhaps non-linear functions as well. But since this linear case is the simple building block, we will start with this example first of fitting linear functions, and we will build on this to eventually have more complex models, and more complex learning algorithms. Let me also give this particular model a name. This model is called linear regression or this, for example, is actually linear regression with one variable, with the variable being x. Predicting all the prices as functions of one variable X. And another name for this model is univariate linear regression. And univariate is just a fancy way of saying one variable. So, that's linear regression

Lecture 3: Linear Algebra Review:

A matrix is a rectangular array of numbers written between square brackets. So, for example, here is a matrix on the right, a left square bracket. And then, write in a bunch of numbers. These could be features from a learning problem or it could be data from somewhere else, but the specific values don't matter, and then I'm going to close it with another right bracket on the right. And so that's one matrix. And, here's another example of the matrix, let's write 3, 4, 5,6. So matrix is just another way for saying, is a 2D or a two dimensional array. And the other piece of knowledge that we need is that the dimension of the matrix is going to be written as the number of row times the number of columns in the matrix. So, concretely, this example on the left, this has 1, 2, 3, 4 rows and has 2 columns, and so this example on the left is a 4 by 2 matrix - number of rows by number of columns. So, four rows, two columns. This one on the right, this matrix has two rows. That's the first row, that's the second row, and it has three columns. That's the first column, that's the second column, that's the third column So, this second matrix we say it is a 2 by 3 matrix. So we say that the dimension of this matrix is 2 by 3. Sometimes you also see this written out, in the case of left, you will see this written out as R4 by 2 or concretely what people will sometimes say this matrix is an element of the set R 4 by 2. So, this thing here, this just means the set of all matrices that of dimension 4 by 2 and this thing on the right, sometimes this is written out as a matrix that is an R 2 by 3. So if you ever see, 2 by 3. So if you ever see something like this are 4 by 2 or are 2 by 3, people are just referring to matrices of a specific dimension. Next, let's talk about how to refer to specific elements of the matrix. And by matrix elements, other than the matrix I just mean the entries, so the numbers inside the matrix. So, in the standard notation, if A is this matrix here, then A sub-strip IJ is going to refer to the i, j entry, meaning the entry in the matrix in the ith row and jth column. So for example a1-1 is going to refer to the entry in the 1st row and the 1st column, so that's the first row and the first column and so a1-1 is going to be equal to 1, 4, 0, 2. Another example, 8 1 2 is going to refer to the entry in the first row and the second column and so A 1 2 is going to be equal to one nine one. This come from a quick examples. Let's see, A, oh let's say A 3 2, is going to refer to the entry in the 3rd row, and second column, right, because that's 3 2 so that's equal to 1 4 3 7. And finally, 8 4 1 is going to refer to this one right, fourth row, first column is equal to 1 4 7 and if, hopefully you won't, but if you were to write and say well this A 4 3, well, that refers to the fourth row, and the third column that, you know, this matrix has no third column so this is undefined, you know, or you can think of this as an error. There's no such element as 8 4 3, so, you know, you shouldn't be referring to 8 4 3. So, the matrix gets you a way of letting you quickly organize, index and access lots of data. In case I seem to be tossing up a lot of concepts, a lot of new notations very rapidly, you don't need to memorize all of this, but on the course website where we have posted the lecture notes, we also have all of these definitions written down. So you can always refer back, you know, either to these slides, possible coursework, so audible lecture notes if you forget well, A41 was that? Which row, which column was that? Don't worry about memorizing everything now. You can always refer back to the written materials on the course website, and use that as a reference. So that's what a matrix is. Next, let's talk about what is a vector. A vector turns out to be a special case of a matrix. A vector is a matrix that has only 1 column so you have an N x 1 matrix, then that's a remember, right? N is the number of rows, and 1 here is the number of columns, so, so matrix with just one column is what we call a vector. So here's an example of a vector, with I guess I have N equals four elements here. so we also call this thing, another term for this is a four dmensional vector, just means that this is a vector with four elements, with four numbers in it. And, just as earlier for matrices you saw this notation R3 by 2 to refer to 2 by 3 matrices, for this vector we are going to refer to this as a vector in the set R4. So this R4 means a set of four-dimensional vectors. Next let's talk about how to refer to the elements of the vector. We are going to use the notation yi to refer to the ith element of the vector y. So if y is this vector, y subscript i is the ith element. So y1 is the first element,four sixty, y2 is equal to the second element, two thirty two -there's the first. There's the second. Y3 is equal to 315 and so on, and only y1 through y4 are defined consistency 4-dimensional vector. Also it turns out that there are actually 2 conventions for how to index into a vector and here they are. Sometimes, people will use one index and sometimes zero index factors. So this example on the left is a one in that specter where the element we write is y1, y2, y3, y4. And this example in the right is an example of a zero index factor where we start the indexing of the elements from zero. So the elements go from a zero up to y three. And this is a bit like the arrays of some primary languages where the arrays can either be indexed starting from one. The first element of an array is sometimes a Y1, this is sequence notation I guess, and sometimes it's zero index depending on what programming language you use. So it turns out that in most of math, the one index version is more common For a lot of machine learning applications, zero index vectors gives us a more convenient notation. So what you should usually do is, unless otherwised specified, you should assume we are using one index vectors. In fact, throughout the rest of these videos on linear algebra review, I will be using one index vectors. But just be aware that when we are talking about machine learning applications, sometimes I will explicitly say when we need to switch to, when we need to use the zero index vectors as well. Finally, by convention, usually when writing matrices and vectors, most people will use upper case to refer to matrices. So we're going to use capital letters like A, B, C, you know, X, to refer to matrices, and usually we'll use lowercase, like a, b, x, y, to refer to either numbers, or just raw numbers or scalars or to vectors. This isn't always true but this is the more common notation where we use lower case "Y" for referring to vector and we usually use upper case to refer to a matrix. So, you now know what are matrices and vectors.



Matrix Elements (entries of matrix)

$$A = \begin{bmatrix} 1402 & 191 \\ 1371 & 821 \\ 949 & 1437 \\ 147 & 1448 \end{bmatrix}$$

$A_{ij} =$ "i, j entry" in the i^{th} row, j^{th} column.

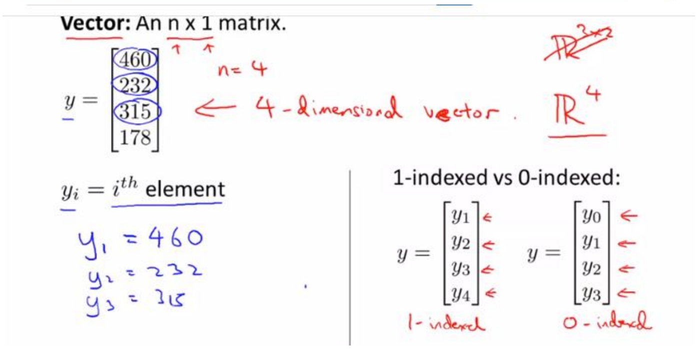
$A_{11} = 1402$

$A_{12} = 191$

$A_{32} = 1437$

$A_{41} = 147$

$\cancel{A_{43}}$ = undefined (error)



Vector: An n x 1 matrix.

$$y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix} \leftarrow \text{4-dimensional vector.} \quad \mathbb{R}^4$$

$n = 4$

$\cancel{\mathbb{R}^{3\times2}}$

$y_i = i^{th}$ element

$y_1 = 460$

$y_2 = 232$

$y_3 = 315$

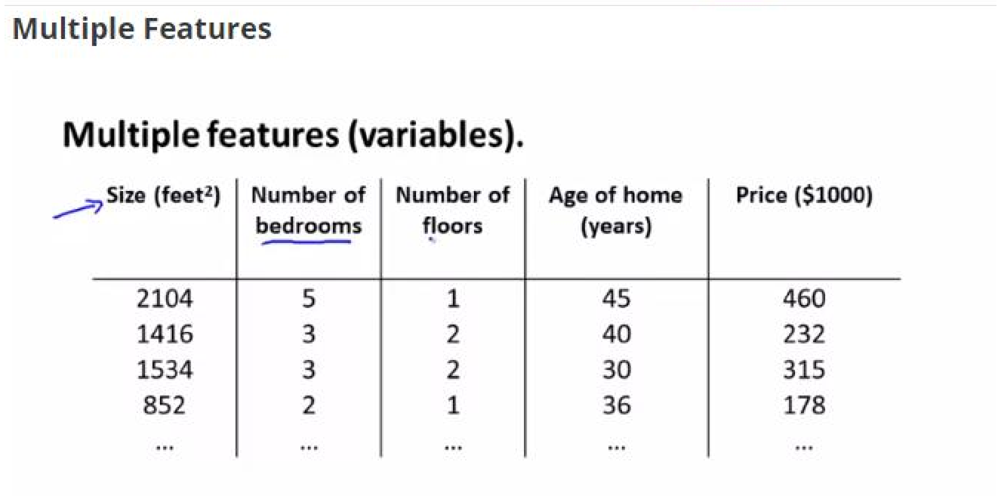
1-indexed vs 0-indexed:

$$y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} \qquad y = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

1-indexed   0-indexed

.

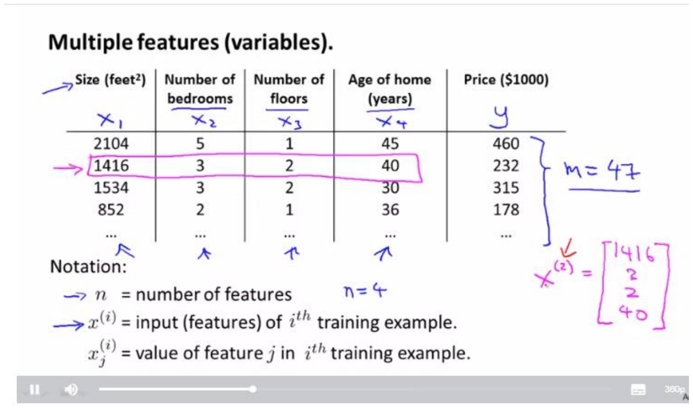
Lecture 4: Linear Regression with Multiple Variables:


Multiple Features

Multiple features (variables).

Size (feet²)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
...	...	...	...	...

If we had not only the size of the house as a feature or as a variable of which to try to predict the price, but that we also knew the number of bedrooms, the number of house and the age of the home and years. It seems like this would give us a lot more information with which to predict the price. To introduce a little bit of notation, we sort of started to talk about this earlier, I'm going to use the variables X subscript 1 X subscript 2 and so on to denote my, in this case, four features and I'm going to continue to use Y to denote the variable, the output variable price that we're trying to predict. Let's introduce a little bit more notation. Now that we have four features I'm going to use lowercase "n" to denote the number of features. So in this example we have n4 because we have, you know, one, two, three, four features. And "n" is different from our earlier notation where we were using "n" to denote the number of examples. So if you have 47 rows "M" is the number of rows on this table or the number of training examples. So I'm also going to use X superscript "I" to denote the input features of the "I" training example. As a concrete example let say X2 is going to be a vector of the features for my second training example. And so X2 here is going to be a vector 1416, 3, 2, 40 since those are my four features that I have to try to predict the price of the second house. So, in this notation, the superscript 2 here. That's an index into my training set. This is not X to the power of 2. Instead, this is, you know, an index that says look at the second row of this table. This refers to my second training example. With this notation X2 is a four dimensional vector. In fact, more generally, this is an in-dimensional feature back there. With this notation, X2 is now a vector and so, I'm going to use also Xi subscript J to denote the value of the J, of feature number J and the training example. So concretely X2 subscript 3, will refer to feature number three in the x factor which is equal to 2,right? That was a 3 over there, just fix my handwriting. So x2 subscript 3 is going to be equal to 2. Now that we have multiple features, let's talk about what the form of our hypothesis should be. Previously this was the form of our hypothesis, where x was our single feature, but now that we have multiple features, we aren't going to use the simple representation any more. Instead, a form of the hypothesis in linear regression is going to be this, can be theta 0 plus theta 1 x1 plus theta 2 x2 plus theta 3 x3 plus theta 4 X4. And if we have N features then rather than summing up over our four features, we would have a sum over our N features. Concretely for a particular setting of our parameters we may have H of X 80 + 0.1 X1 + 0.01x2 + 3x3 - 2x4. This would be one example of a hypothesis and you remember a hypothesis is trying to predict the price of the house in thousands of dollars, just saying that, you know, the base price of a house is maybe 80,000 plus another open 1, so that's an extra, what, hundred dollars per square feet, yeah, plus the price goes up a little bit for each additional floor that the house has. X two is the number of floors, and it goes up further for each additional bedroom the house has, because X three was the number of bedrooms, and the price goes down a little bit with each additional age of the house. With each additional year of the age of the house. Here's the form of a hypothesis rewritten on the slide. And what I'm gonna do is introduce a little bit of notation to simplify this equation. For convenience of notation, let me define x subscript 0 to be equals one. Concretely, this means that for every example i I have a feature vector X superscript I and X superscript I subscript 0 is going to be equal to 1. You can think of this as defining an additional zero feature. So whereas previously I had n features because x1, x2 through xn, I'm now defining an additional sort of zero feature vector that always takes on the value of one. So now my feature vector X becomes this N+1 dimensional vector that is zero index. So this is now a n+1 dimensional feature vector, but I'm gonna index it from 0 and I'm also going to think of my parameters as a vector. So, our parameters here, right that would be our theta zero, theta one, theta two, and so on all the way up to theta n, we're going to gather them up into a parameter vector written theta 0, theta 1, theta 2, and so on, down to theta n. This is another zero index vector. It's of index signed from zero. That is another n plus 1 dimensional vector. So, my hypothesis cannot be written theta 0x0 plus theta 1x1+ up to theta n Xn. And this equation is the same as this on top because, you know, eight zero is equal to one. Underneath and I now take this form of the hypothesis and write this as either transpose x, depending on how familiar you are with inner products of vectors if you write what theta transfers x is what theta transfer and this is theta zero, theta one, up to theta N. So this thing here is theta transpose and this is actually a N plus one by one matrix. [It should be a 1 by (n+1) matrix] It's also called a row vector and you take that and multiply it with the vector X which is X zero, X one, and so on, down to X n. And so, the inner product that is theta transpose X is just equal to this. This gives us a convenient way to write the form of the hypothesis as just the inner product between our parameter vector theta and our theta vector X. And it is this little bit of notation, this little excerpt of the notation convention that let us write this in this compact form. So that's the form of a hypthesis when we have multiple features. And, just to give this another name, this is also called multivariate linear regression. And the term multivariable that's just maybe a fancy term for saying we have multiple features, or multivariables with which to try to predict the value Y.



Multiple features (variables).

Size (feet^2) x_1	Number of bedrooms x_2	Number of floors x_3	Age of home (years) x_4	Price ($1000) y
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
…	…	…	…	…

$m = 47$

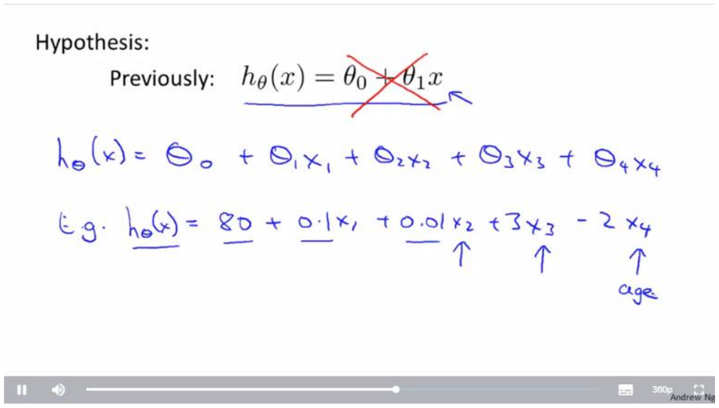
Notation:

n = number of features $\quad n = 4$

$x^{(i)}$ = input (features) of i^{th} training example.

$x_j^{(i)}$ = value of feature j in i^{th} training example.

$$x^{(2)} = \begin{bmatrix} 1416 \\ 3 \\ 2 \\ 40 \end{bmatrix}$$



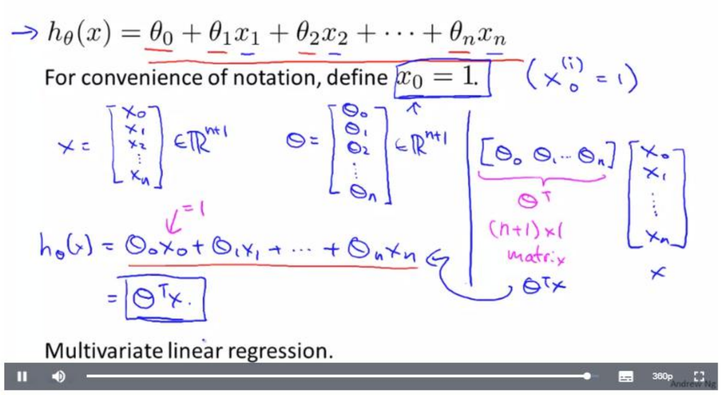
Hypothesis:

Previously: $h_\theta(x) = \theta_0 \cancel{+ \theta_1 x}$

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$$

$$\text{E.g. } h_\theta(x) = 80 + 0.1 x_1 + 0.01 x_2 + 3 x_3 - 2 x_4$$

age



$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

For convenience of notation, define $\boxed{x_0 = 1}$. $(x_0^{(i)} = 1)$

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

$$\underbrace{[\theta_0\ \theta_1 \cdots \theta_n]}_{\theta^T} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}$$

$(n+1) \times 1$ matrix $\quad \theta^T x \quad x$

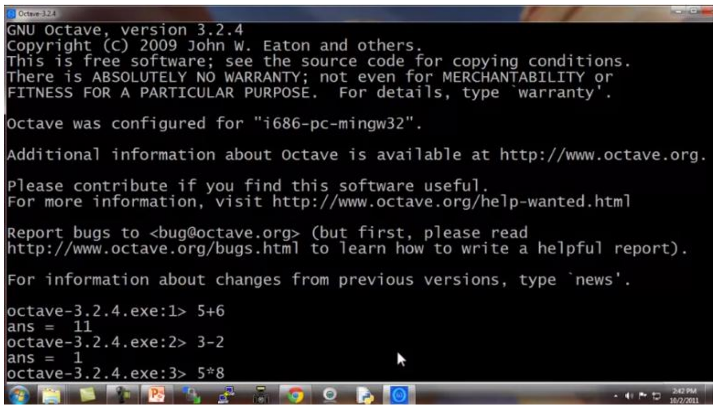
$$h_\theta(x) = \theta_0 \overset{=1}{x_0} + \theta_1 x_1 + \cdots + \theta_n x_n$$

$$= \boxed{\theta^T x.}$$

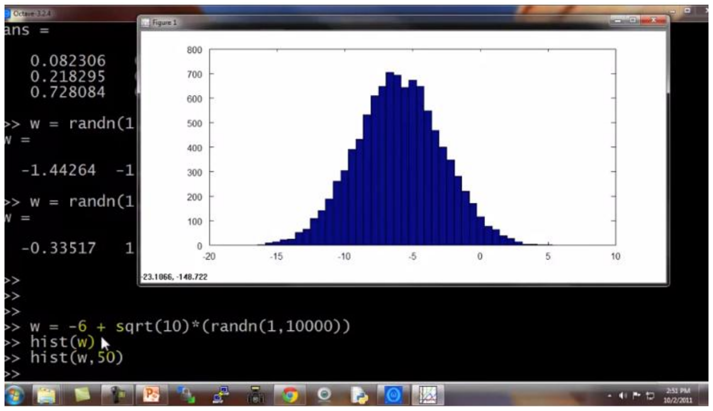
Multivariate linear regression.

Lecture 5: Octave/Matlab Tutorial:

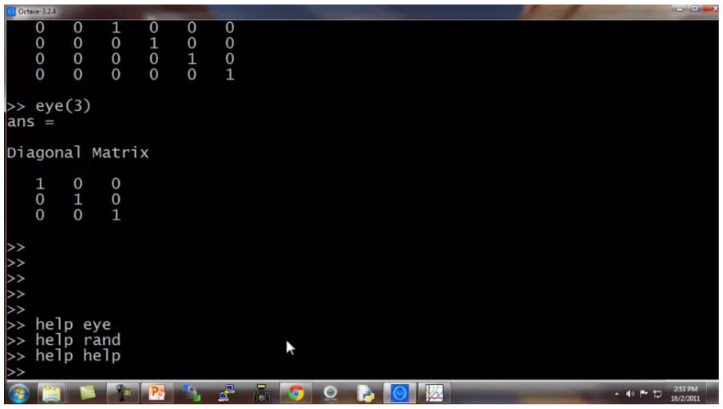
In Octave. And that's my Octave prompt. Let me first show the elementary operations you can do in Octave. So you type in 5 + 6. That gives you the answer of 11. 3 - 2. 5 x 8, 1/2, 2^6 is 64. So those are the elementary math operations. You can also do logical operations. So one equals two. This evaluates to false. The percent command here means a comment. So, one equals two, evaluates to false. Which is represents by zero. One not equals to two. This is true. So that returns one. Note that a not equal sign is this tilde equals symbol. And not bang equals. Which is what some other programming languages use. Lets see logical operations one and zero use a double ampersand sign to the logical AND. And that evaluates false. One or zero is the OR operation. And that evaluates to true. And I can XOR one and zero, and that evaluates to one. This thing over on the left, this Octave 324.x equals 11, this is the default Octave prompt. It shows the, what, the version in Octave and so on. If you don't want that prompt, there's a somewhat cryptic command PF quote, greater than, greater than and so on, that you can use to change the prompt. And I guess this quote a string in the middle. Your quote, greater than, greater than, space. That's what I prefer my Octave prompt to look like. So if I hit enter. Oops, excuse me. Like so. PS1 like so. Now my Octave prompt has changed to the greater than, greater than sign.Which, you know, looks quite a bit better. Next let's talk about Octave variables. I can take the variable A and assign it to 3. And hit enter. And now A is equal to 3. You want to assign a variable, but you don't want to print out the result. If you put a semicolon, the semicolon suppresses the print output. So to do that, enter, it doesn't print anything. Whereas A equals 3. mix it, print it out, where A equals, 3 semicolon doesn't print anything. I can do string assignment. B equals hi Now if I just enter B it prints out the variable B. So B is the string hi C equals 3 greater than colon 1. So, now C evaluates the true. If you want to print out or display a variable, here's how you go about it. Let me set A equals Pi. And if I want to print A I can just type A like so, and it will print it out. For more complex printing there is also the DISP command which stands for Display. Display A just prints out A like so. You can also display strings so: DISP, sprintf, two decimals, percent 0.2, F, comma, A. Like so. And this will print out the string. Two decimals, colon, 3.14. This is kind of an old style C syntax. For those of you that have programmed C before, this is essentially the syntax you use to print screen. So the Sprintf generates a string that is less than the 2 decimals, 3.1 plus string. This percent 0.2 F means substitute A into here, showing the two digits after the decimal points. And DISP takes the string DISP generates it by the Sprintf command. Sprintf. The Sprintf command. And DISP actually displays the string. And to show you another example, Sprintf six decimals percent 0.6 F comma A. And, this should print Pi with six decimal places. Finally, I was saying, a like so, looks like this. There are useful shortcuts that type type formats long. It causes strings by default. Be displayed to a lot more decimal places. And format short is a command that restores the default of just printing a small number of digits. Okay, that's how you work with variables. Now let's look at vectors and matrices. Let's say I want to assign MAT A to the matrix. Let me show you an example: 1, 2, semicolon, 3, 4, semicolon, 5, 6. This generates a three by two matrix A whose first row is 1, 2. Second row 3, 4. Third row is 5, 6. What the semicolon does is essentially say, go to the next row of the matrix. There are other ways to type this in. Type A 1, 2 semicolon 3, 4, semicolon, 5, 6, like so. And that's another equivalent way of assigning A to be the values of this three by two matrix. Similarly you can assign vectors. So V equals 1, 2, 3. This is actually a row vector. Or this is a 3 by 1 vector. Where that is a fat Y vector, excuse me, not, this is a 1 by 3 matrix, right. Not 3 by 1. If I want to assign this to a column vector, what I would do instead is do v 1;2;3. And this will give me a 3 by 1. There's a 1 by 3 vector. So this will be a column vector. Here's some more useful notation. V equals 1: 0.1: 2. What this does is it sets V to the bunch of elements that start from 1. And increments and steps of 0.1 until you get up to 2. So if I do this, V is going to be this, you know, row vector. This is what one by eleven matrix really. That's 1, 1.1, 1.2, 1.3 and so on until we get up to two. Now, and I can also set V equals one colon six, and that sets V to be these numbers. 1 through 6, okay. Now here are some other ways to generate matrices. Ones 2.3 is a command that generates a matrix that is a two by three matrix that is the matrix of all ones. So if I set that c2 times ones two by three this generates a two by three matrix that is all two's. You can think of this as a shorter way of writing this and c2,2,2's and you can call them 2,2,2, which would also give you the same result. Let's say W equals one's, one by three, so this is going to be a row vector or a row of three one's and similarly you can also say w equals zeroes, one by three, and this generates a matrix. A one by three matrix of all zeros. Just a couple more ways to generate matrices . If I do W equals Rand one by three, this gives me a one by three matrix of all random numbers. If I do Rand three by three. This gives me a three by three matrix of all random numbers drawn from the uniform distribution between zero and one. So every time I do this, I get a different set of random numbers drawn uniformly between zero and one. For those of you that know what a Gaussian random variable is or for those of you that know what a normal random variable is, you can also set W equals Rand N, one by three. And so these are going to be three values drawn from a Gaussian distribution with mean zero and variance or standard deviation equal to one. And you can set more complex things like W equals minus six, plus the square root ten, times, lets say Rand N, one by ten thousand. And I'm going to put a semicolon at the end because I don't really want this printed out. This is going to be a what? Well, it's going to be a vector of, with a hundred thousand, excuse me, ten thousand elements. So, well, actually, you know what? Let's print it out. So this will generate a matrix like this. Right? With 10,000 elements. So that's what W is. And if I now plot a histogram of W with a hist command, I can now. And Octave's print hist command, you know, takes a couple seconds to bring this up, but this is a histogram of my random variable for W. There was minus 6 plus zero ten times this Gaussian random variable. And I can plot a histogram with more buckets, with more bins, with say, 50 bins. And this is my histogram of a Gaussian with mean minus 6. Because I have a minus 6 there plus square root 10 times this. So the variance of this Gaussian random variable is 10 on the standard deviation is square root of 10, which is about what? Three point one. Finally, one special command for generator matrix, which is the I command. So I stands for this is maybe a pun on the word identity. It's server set eye 4. This is the 4 by 4 identity matrix. So I equals eye 4. This gives me a 4 by 4 identity matrix. And I equals eye 5, eye 6. That gives me a 6 by 6 identity matrix, i3 is the 3 by 3 identity matrix. Lastly, to wrap up this video, there's one more useful command. Which is the help command. So you can type help i and this brings up the help function for the identity matrix. Hit Q to quit. And you can also type help rand. Brings up documentation for the rand or the random number generation function. Or even help help, which shows you, you know help on the help function. So, those are the basic operations in Octave. And with this you should be able to generate a few matrices, multiply, add things. And use the basic operations in Octave.



```
GNU Octave, version 3.2.4
Copyright (C) 2009 John W. Eaton and others.
This is free software; see the source code for copying conditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE.  For details, type `warranty'.

Octave was configured for "i686-pc-mingw32".

Additional information about Octave is available at http://www.octave.org.

Please contribute if you find this software useful.
For more information, visit http://www.octave.org/help-wanted.html

Report bugs to <bug@octave.org> (but first, please read
http://www.octave.org/bugs.html to learn how to write a helpful report).

For information about changes from previous versions, type `news'.

octave-3.2.4.exe:1> 5+6
ans =  11
octave-3.2.4.exe:2> 3-2
ans =  1
octave-3.2.4.exe:3> 5*8
```



```
ans =

   0.082306
   0.218295
   0.728084

>> w = randn(1
w =

  -1.44264   -1

>> w = randn(1
w =

  -0.33517   1

>>
>>
>>
>> w = -6 + sqrt(10)*(randn(1,10000))
>> hist(w)
>> hist(w,50)
>>
```

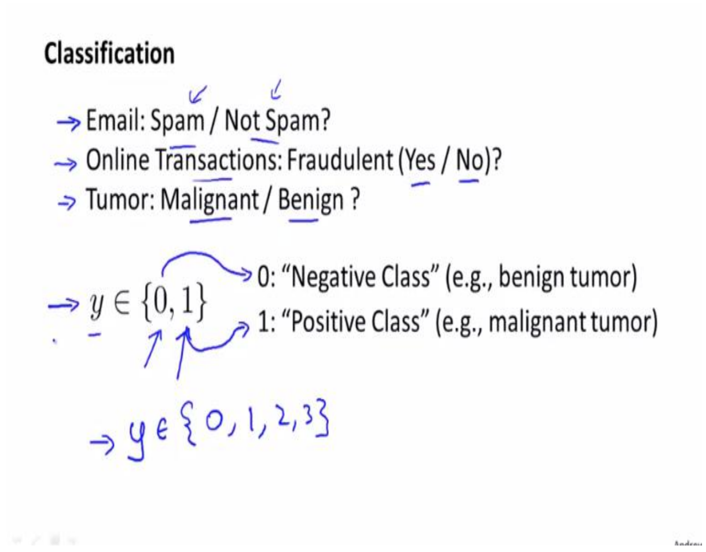


```
   0   0   1   0   0   0
   0   0   0   1   0   0
   0   0   0   0   1   0
   0   0   0   0   0   1

>> eye(3)
ans =

Diagonal Matrix

   1   0   0
   0   1   0
   0   0   1

>>
>>
>>
>>
>>
>> help eye
>> help rand
>> help help
>>
```

Lecture 6: Logistic Regression:

We'll develop an algorithm called logistic regression, which is one of the most popular and most widely used learning algorithms today. Here are some examples of classification problems.

Earlier we talked about email spam classification as an example of a classification problem. Another example would be classifying online transactions. So if you have a website that sells stuff and if you want to know if a particular transaction is fraudulent or not, whether someone is using a stolen credit card or has stolen the user's password. There's another classification problem. And earlier we also talked about the example of classifying tumors as cancerous, malignant or as benign tumors. In all of these problems the variable that we're trying to predict is a variable y that we can think of as taking on two values either zero or one, either spam or not spam, fraudulent or not fraudulent, related malignant or benign. Another name for the class that we denote with zero is the negative class, and another name for the class that we denote with one is the positive class. So zero we denote as the benign tumor, and one, positive class we denote a malignant tumor. The assignment of the two classes, spam not spam and so on. The assignment of the two classes to positive and negative to zero and one is somewhat arbitrary and it doesn't really matter but often there is this intuition that a negative class is conveying the absence of something like the absence of a malignant tumor. Whereas one the positive class is conveying the presence of something that we may be looking for, but the definition of which is negative and which is positive is somewhat arbitrary and it doesn't matter that much. For now we're going to start with classification problems with just two classes zero and one. Later one we'll talk about multi class problems as well where therefore y may take on four values zero, one, two, and three. This is called a multiclass classification problem. But for the next few videos, let's start with the two class or the binary classification problem and we'll worry about the multiclass setting later. So how do we develop a classification algorithm? Here's an example of a training set for a classification task for classifying a tumor as malignant or benign. And notice that malignancy takes on only two values, zero or no, one or yes. So one thing we could do given this training set is to apply the algorithm that we already know. Linear regression to this data set and just try to fit the straight line to the data. So if you take this training set and fill a straight line to it, maybe you get a hypothesis that looks like that, right. So that's my hypothesis. H(x) equals theta transpose x. If you want to make predictions one thing you could try doing is then threshold the classifier outputs at 0.5 that is at a vertical axis value 0.5 and if the hypothesis outputs a value that is greater than equal to 0.5 you can take $y = 1$. If it's less than 0.5 you can take $y=0$. Let's see what happens if we do that. So 0.5 and so that's where the threshold is and that's using linear regression this way. Everything to the right of this point we will end up predicting as the positive cross. Because the output values is greater than 0.5 on the vertical axis and everything to the left of that point we will end up predicting as a negative value. In this particular example, it looks like linear regression is actually doing something reasonable. Even though this is a classification toss we're interested in. But now let's try changing the problem a bit. Let me extend out the horizontal access a little bit and let's say we got one more training example way out there on the right. Notice that that additional training example, this one out here, it doesn't actually change anything, right. Looking at the training set it's pretty clear what a good hypothesis is. Is that well everything to the right of somewhere around here, to the right of this we should predict this positive. Everything to the left we should probably predict as negative because from this training set, it looks like all the tumors larger than a certain value around here are malignant, and all the tumors smaller than that are not malignant, at least for this training set. But once we've added that extra example over here, if you now run linear regression, you instead get a straight line fit to the data. That might maybe look like this. And if you know threshold hypothesis at 0.5, you end up with a threshold that's around here, so that everything to the right of this point you predict as positive and everything to the left of that point you predict as negative. And this seems a pretty bad thing for linear regression to have done, right, because you know these are our positive examples, these are our negative examples. It's pretty clear we really should be separating the two somewhere around there, but somehow by adding one example way out here to the right, this example really isn't giving us any new information. I mean, there should be no surprise to the learning algorithm. That the example way out here turns out to be malignant. But somehow having that example out there caused linear regression to change its straight-line fit to the data from this magenta line out here to this blue line over here, and caused it to give us a worse hypothesis. So, applying linear regression to a classification problem often isn't a great idea. In the first example, before I added this extra training example, previously linear regression was just getting lucky and it got us a hypothesis that worked well for that particular example, but usually applying linear regression to a data set, you might get lucky but often it isn't a good idea. So I wouldn't use linear regression for classification problems. Here's one other funny thing about what would happen if we were to use linear regression for a classification problem. For classification we know that y is either zero or one. But if you are using linear regression where the hypothesis can output values that are much larger than one or less than zero, even if all of your training examples have labels y equals zero or one. And it seems kind of strange that even though we know that the labels should be zero, one it seems kind of strange if the algorithm can output values much larger than one or much smaller than zero. So what we'll do in the next few videos is develop an algorithm called logistic regression, which has the property that the output, the predictions of logistic regression are always between zero and one, and doesn't become bigger than one or become less than zero. And by the way, logistic regression is, and we will use it as a classification algorithm, is some, maybe sometimes confusing that the term regression appears in this name even though logistic regression is actually a classification algorithm. But that's just a name it was given for historical reasons. So don't be confused by that logistic regression is actually a classification algorithm that we apply to settings where the label y is discrete value, when it's either zero or one. So hopefully you now know why, if you have a classification problem, using linear regression isn't a good idea.

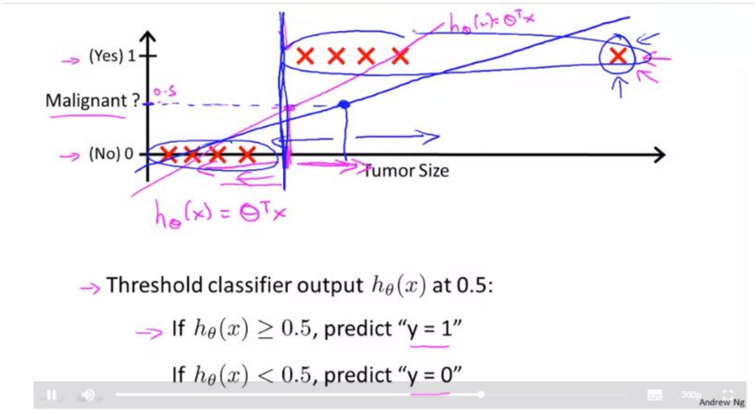


Classification

- Email: Spam / Not Spam?
- Online Transactions: Fraudulent (Yes / No)?
- Tumor: Malignant / Benign ?

$y \in \{0, 1\}$ 0: "Negative Class" (e.g., benign tumor)
1: "Positive Class" (e.g., malignant tumor)

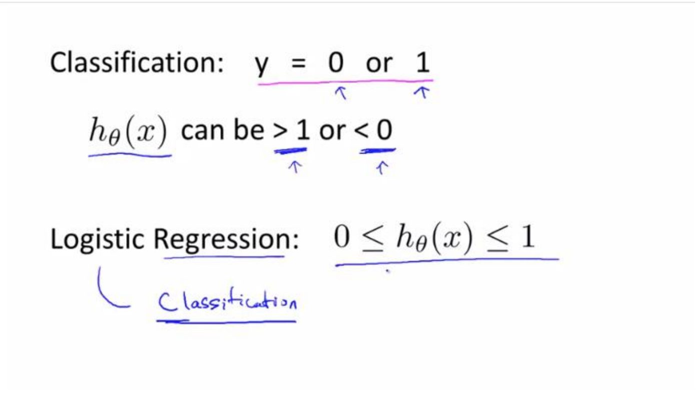
$y \in \{0,1,2,3\}$



Threshold classifier output $h_\theta(x)$ at 0.5:

If $h_\theta(x) \geq 0.5$, predict "y = 1"

If $h_\theta(x) < 0.5$, predict "y = 0"



Classification: y = 0 or 1

$h_\theta(x)$ can be > 1 or < 0

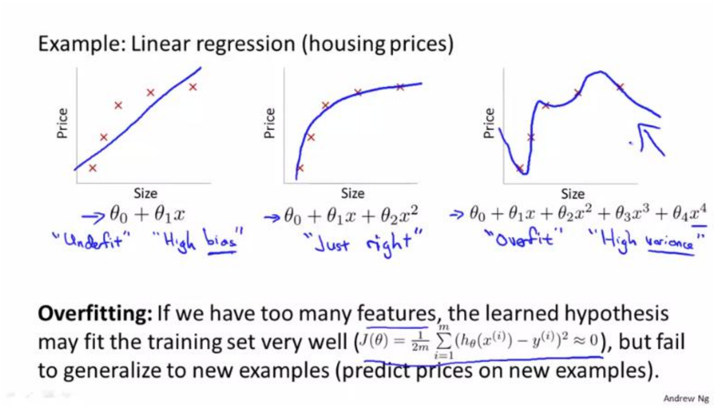
Logistic Regression: $0 \leq h_\theta(x) \leq 1$

Classification

Lecture 7: Regularization (The Problem of Over fitting):

So what is overfitting? Let's keep using our running example of predicting housing prices with linear regression where we want to predict the price as a function of the size of the house. One thing we could do is fit a linear function to this data, and if we do that, maybe we get that sort of straight line fit to the data. But this isn't a very good model. Looking at the data, it seems pretty clear that as the size of the housing increases, the housing prices plateau, or kind of flattens out as we move to the right and so this algorithm does not fit the training and we call this problem underfitting, and another term for this is that this algorithm has high bias. Both of these roughly mean that it's just not even fitting the training data very well. The term is kind of a historical or technical one, but the idea is that if a fitting a straight line to the data, then, it's as if the algorithm has a very strong preconception, or a very strong bias that housing prices are going to vary linearly with their size and despite the data to the contrary. Despite the evidence of the contrary is preconceptions still are bias, still closes it to fit a straight line and this ends up being a poor fit to the data. Now, in the middle, we could fit a quadratic functions enter and, with this data set, we fit the quadratic function, maybe, we get that kind of curve and, that works pretty well. And, at the other extreme, would be if we were to fit, say a fourth other polynomial to the data. So, here we have five parameters, theta zero through theta four, and, with that, we can actually fill a curve that process through all five of our training examples. You might get a curve that looks like this. That, on the one hand, seems to do a very good job fitting the training set and, that is processed through all of my data, at least. But, this is still a very wiggly curve, right? So, it's going up and down all over the place, and, we don't actually think that's such a good model for predicting housing prices. So, this problem we call overfitting, and, another term for this is that this algorithm has high variance.. The term high variance is another historical or technical one. But, the intuition is that, if we're fitting such a high order polynomial, then, the hypothesis can fit, you know, it's almost as if it can fit almost any function and this face of possible hypothesis is just too large, it's too variable. And we don't have enough data to constrain it to give us a good hypothesis so that's called overfitting. And in the middle, there isn't really a name but I'm just going to write, you know, just right. Where a second degree polynomial, quadratic function seems to be just right for fitting this data. To recap a bit the problem of over fitting comes when if we have too many features, then to learn hypothesis may fit the training side very well. So, your cost function may actually be very close to zero or may be even zero exactly, but you may then end up with a curve like this that, you know tries too hard to fit the training set, so that it even fails to generalize to new examples and fails to predict prices on new examples as well, and here the term generalized refers to how well a hypothesis applies even to new examples. That is to data to houses that it has not seen in the training set. On this slide, we looked at over fitting for the case of linear regression. A similar thing can apply to logistic regression as well. Here is a logistic regression example with two features X1 and x2. One thing we could do, is fit logistic regression with just a simple hypothesis like this, where, as usual, G is my sigmoid function. And if you do that, you end up with a hypothesis, trying to use, maybe, just a straight line to separate the positive and the negative examples. And this doesn't look like a very good fit to the hypothesis. So, once again, this is an example of underfitting or of the hypothesis having high bias. In contrast, if you were to add to your features these quadratic terms, then, you could get a decision boundary that might look more like this. And, you know, that's a pretty good fit to the data. Probably, about as good as we could get, on this training set. And, finally, at the other extreme, if you were to fit a very high-order polynomial, if you were to generate lots of high-order polynomial terms of speeches, then, logistical regression may contort itself, may try really hard to find a decision boundary that fits your training data or go to great lengths to contort itself, to fit every single training example well. And, you know, if the features X1 and X2 offer predicting, maybe, the cancer to the, you know, cancer is a malignant, benign breast tumors. This doesn't, this really doesn't look like a very good hypothesis, for making predictions. And so, once again, this is an instance of overfitting and, of a hypothesis having high variance and not really, and, being unlikely to generalize well to new examples. Later, in this course, when we talk about debugging and diagnosing things that can go wrong with learning algorithms, we'll give you specific tools to recognize when overfitting and, also, when underfitting may be occurring. But, for now, lets talk about the problem of, if we think overfitting is occurring, what can we do to address it? In the previous examples, we had one or two dimensional data so, we could just plot the hypothesis and see what was going on and select the appropriate degree polynomial. So, earlier for the housing prices example, we could just plot the hypothesis and, you know, maybe see that it was fitting the sort of very wiggly function that goes all over the place to predict housing prices. And we could then use figures like these to select an appropriate degree polynomial. So plotting the hypothesis, could be one way to try to decide what degree polynomial to use. But that doesn't always work. And, in fact more often we may have learning problems that where we just have a lot of features. And there is not just a matter of selecting what degree polynomial. And, in fact, when we have so many features, it also becomes much harder to plot the data and it becomes much harder to visualize it, to decide what features to keep or not. So concretely, if we're trying predict housing prices sometimes we can just have a lot of different features. And all of these features seem, you know, maybe they seem kind of useful. But, if we have a lot of features, and, very little training data, then, over fitting can become a problem. In order to address over fitting, there are two main options for things that we can do. The first option is, to try to reduce the number of features. Concretely, one thing we could do is manually look through the list of features, and, use that to try to decide which are the more important features, and, therefore, which are the features we should keep, and, which are the features we should throw out. Later in this course, where also talk about model selection algorithms. Which are algorithms for automatically deciding which features to keep and, which features to throw out. This idea of reducing the number of features can work well, and, can reduce over fitting. And, when we talk about model selection, we'll go into this in much greater depth. But, the disadvantage is that, by throwing away some of the features, is also throwing away some of the information you have about the problem. For example, maybe, all of those features are actually useful for predicting the price of a house, so, maybe, we don't actually want to throw some of our information or throw some of our features away. The second option, which we'll talk about in the next few videos, is regularization. Here, we're going to keep all the features, but we're going to reduce the magnitude or the values of the parameters theta J. And, this method works well, we'll see, when we have a lot of features, each of which contributes a little bit to predicting the value of Y, like we saw in the housing price prediction example. Where we could have a lot of features, each of which are, you know, somewhat useful, so, maybe, we don't want to throw them away. So, this subscribes the idea of regularization at a very high level. And, I realize that, all of these details probably don't make sense to you yet.

The Problem of Overfitting



Example: Linear regression (housing prices)

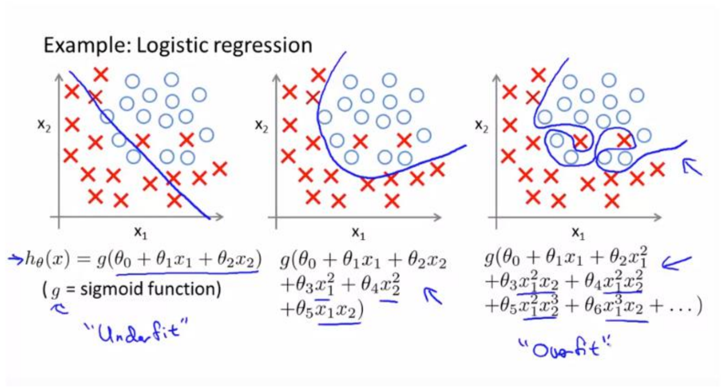
$\rightarrow \theta_0 + \theta_1 x$ "Underfit" "High $\underline{\text{bias}}$"

$\rightarrow \theta_0 + \theta_1 x + \theta_2 x^2$ "Just right"

$\rightarrow \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$ "Overfit" "High $\underline{\text{variance}}$"

Overfitting: If we have too many features, the learned hypothesis may fit the training set very well ($J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 \approx 0$), but fail to generalize to new examples (predict prices on new examples).

The Problem of Overfitting



Example: Logistic regression

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

(g = sigmoid function)

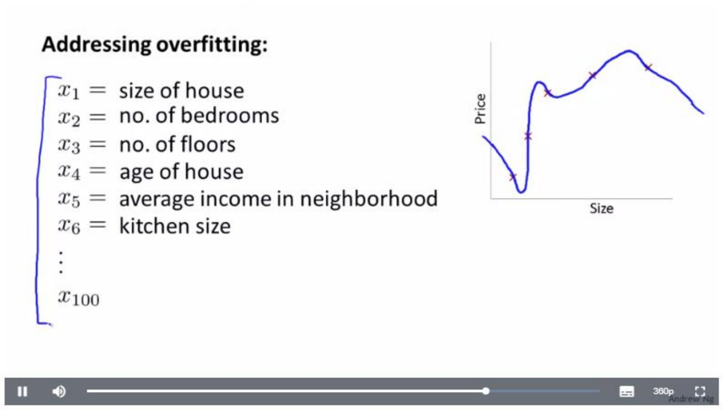
"Underfit"

$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2)$

$g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \theta_6 x_1^3 x_2 + \ldots)$

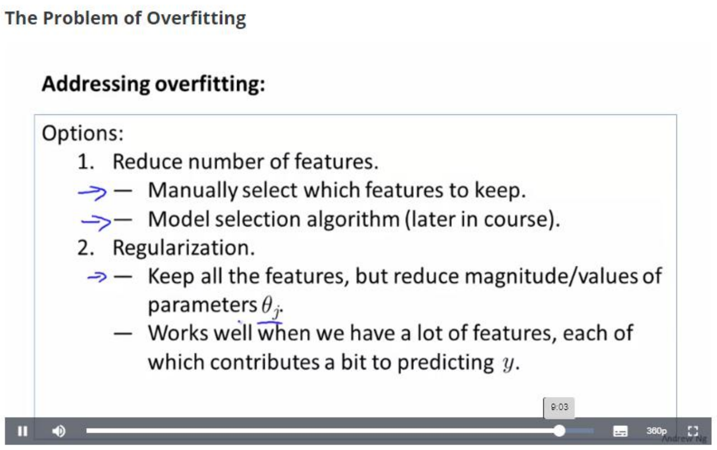
"Overfit"

The Problem of Overfitting



Addressing overfitting:

$x_1 =$ size of house
$x_2 =$ no. of bedrooms
$x_3 =$ no. of floors
$x_4 =$ age of house
$x_5 =$ average income in neighborhood
$x_6 =$ kitchen size
$\vdots$
x_{100}



The Problem of Overfitting

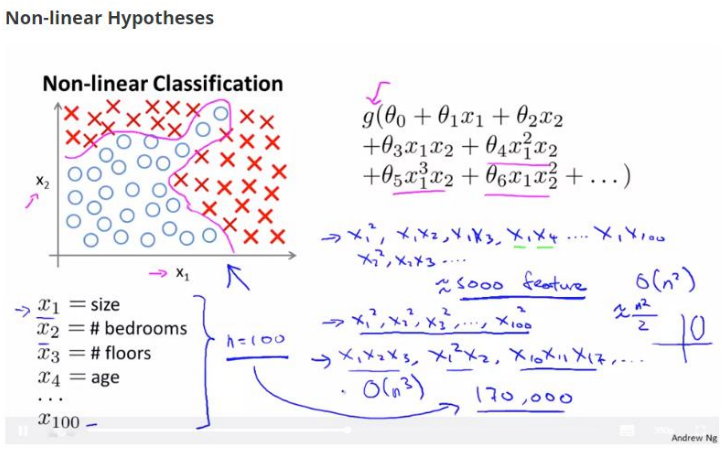
Addressing overfitting:

Options:

1. Reduce number of features.
   - Manually select which features to keep.
   - Model selection algorithm (later in course).
2. Regularization.
   - Keep all the features, but reduce magnitude/values of parameters θ_j.
   - Works well when we have a lot of features, each of which contributes a bit to predicting y.

Lecture 8: Neural Networks Representation:

Neutral networks is actually a pretty old idea, but had fallen out of favor for a while. But today, it is the state of the art technique for many different machine learning problems. So why do we need yet another learning algorithm? We already have linear regression and we have logistic regression, so why do we need, you know, neural networks? In order to motivate the discussion of neural networks, let me start by showing you a few examples of machine learning problems where we need to learn complex non-linear hypotheses. Consider a supervised learning classification problem where you have a training set like this. If you want to apply logistic regression to this problem, one thing you could do is apply logistic regression with a lot of nonlinear features like that. So here, g as usual is the sigmoid function, and we can include lots of polynomial terms like these. And, if you include enough polynomial terms then, you know, maybe you can get a hypotheses that separates the positive and negative examples. This particular method works well when you have only, say, two features - x1 and x2 - because you can then include all those polynomial terms of x1 and x2. But for many interesting machine learning problems would have a lot more features than just two. We've been talking for a while about housing prediction, and suppose you have a housing classification problem rather than a regression problem, like maybe if you have different features of a house, and you want to predict what are the odds that your house will be sold within the next six months, so that will be a classification problem. And as we saw we can come up with quite a lot of features, maybe a hundred different features of different houses. For a problem like this, if you were to include all the quadratic terms, all of these, even all of the quadratic that is the second or the polynomial terms, there would be a lot of them. There would be terms like x1 squared, x1x2, x1x3, you know, x1x4 up to x1x100 and then you have x2 squared, x2x3 and so on. And if you include just the second order terms, that is, the terms that are a product of, you know, two of these terms, x1 times x1 and so on, then, for the case of n equals 100, you end up with about five thousand features. And, asymptotically, the number of quadratic features grows roughly as order n squared, where n is the number of the original features, like x1 through x100 that we had. And its actually closer to n squared over two. So including all the quadratic features doesn't seem like it's maybe a good idea, because that is a lot of features and you might up overfitting the training set, and it can also be computationally expensive, you know, to be working with that many features. One thing you could do is include only a subset of these, so if you include only the features x1 squared, x2 squared, x3 squared, up to maybe x100 squared, then the number of features is much smaller. Here you have only 100 such quadratic features, but this is not enough features and certainly won't let you fit the data set like that on the upper left. In fact, if you include only these quadratic features together with the original x1, and so on, up to x100 features, then you can actually fit very interesting hypotheses. So, you can fit things like, you know, access a line of the ellipses like these, but you certainly cannot fit a more complex data set like that shown here. So 5000 features seems like a lot, if you were to include the cubic, or third order known of each others, the x1, x2, x3. You know, x1 squared, x2, x10 and x11, x17 and so on. You can imagine there are gonna be a lot of these features. In fact, they are going to be order and cube such features and if any is 100 you can compute that, you end up with on the order of about 170,000 such cubic features and so including these higher auto-polynomial features when your original feature set end is large this really dramatically blows up your feature space and this doesn't seem like a good way to come up with additional features with which to build none many classifiers when n is large. For many machine learning problems, n will be pretty large. Here's an example. Let's consider the problem of computer vision. And suppose you want to use machine learning to train a classifier to examine an image and tell us whether or not the image is a car. Many people wonder why computer vision could be difficult. I mean when you and I look at this picture it is so obvious what this is. You wonder how is it that a learning algorithm could possibly fail to know what this picture is. To understand why computer vision is hard let's zoom into a small part of the image like that area where the little red rectangle is. It turns out that where you and I see a car, the computer sees that. What it sees is this matrix, or this grid, of pixel intensity values that tells us the brightness of each pixel in the image. So the computer vision problem is to look at this matrix of pixel intensity values, and tell us that these numbers represent the door handle of a car. Concretely, when we use machine learning to build a car detector, what we do is we come up with a label training set, with, let's say, a few label examples of cars and a few label examples of things that are not cars, then we give our training set to the learning algorithm trained a classifier and then, you know, we may test it and show the new image and ask, "What is this new thing?". And hopefully it will recognize that that is a car. To understand why we need nonlinear hypotheses, let's take a look at some of the images of cars and maybe non-cars that we might feed to our learning algorithm. Let's pick a couple of pixel locations in our images, so that's pixel one location and pixel two location, and let's plot this car, you know, at the location, at a certain point, depending on the intensities of pixel one and pixel two. And let's do this with a few other images. So let's take a different example of the car and you know, look at the same two pixel locations and that image has a different intensity for pixel one and a different intensity for pixel two. So, it ends up at a different location on the figure. And then let's plot some negative examples as well. That's a non-car, that's a non-car . And if we do this for more and more examples using the pluses to denote cars and minuses to denote non-cars, what we'll find is that the cars and non-cars end up lying in different regions of the space, and what we need therefore is some sort of non-linear hypotheses to try to separate out the two classes. What is the dimension of the feature space? Suppose we were to use just 50 by 50 pixel images. Now that suppose our images were pretty small ones, just 50 pixels on the side. Then we would have 2500 pixels, and so the dimension of our feature size will be N equals 2500 where our feature vector x is a list of all the pixel testings, you know, the pixel brightness of pixel one, the brightness of pixel two, and so on down to the pixel brightness of the last pixel where, you know, in a typical computer representation, each of these may be values between say 0 to 255 if it gives us the grayscale value. So we have n equals 2500, and that's if we were using grayscale images. If we were using RGB images with separate red, green and blue values, we would have n equals 7500. So, if we were to try to learn a nonlinear hypothesis by including all the quadratic features, that is all the terms of the form, you know, Xi times Xj, while with the 2500 pixels we would end up with a total of three million features. And that's just too large to be reasonable; the computation would be very expensive to find and to represent all of these three million features per training example. So, simple logistic regression together with adding in maybe the quadratic or the cubic features - that's just not a good way to learn complex nonlinear hypotheses when n is large because you just end up with too many features.


Non-linear Hypotheses

Non-linear Classification

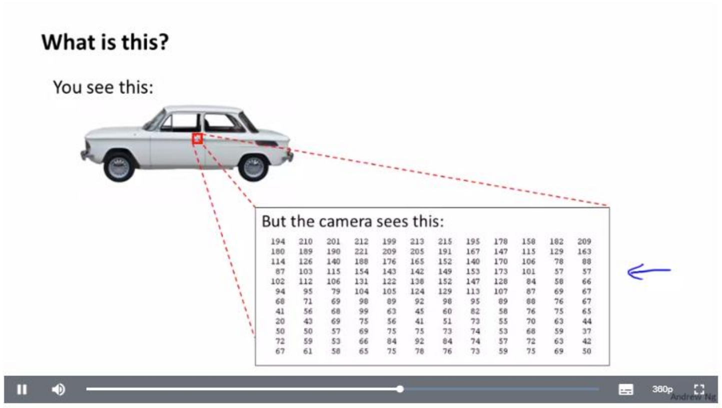
$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^3 x_2 + \theta_6 x_1 x_2^2 + \dots)$$

$x_1 =$ size
$x_2 =$ # bedrooms
$x_3 =$ # floors
$x_4 =$ age
...
x_{100}

$n = 100$

$\rightarrow x_1^2, x_1x_2, x_1x_3, x_1x_4 \dots x_1x_{100}$
$x_2^2, x_2x_3 \dots$
≈ 5000 feature $\quad O(n^2) \quad \approx \frac{n^2}{2}$
$\rightarrow x_1^2, x_2^2, x_3^2, \dots, x_{100}^2$
$\rightarrow x_1x_2x_3, x_1^2x_2, x_{10}x_{11}x_{17}, \dots$
$O(n^3)$ $\quad$ 170,000

Non-linear Hypotheses



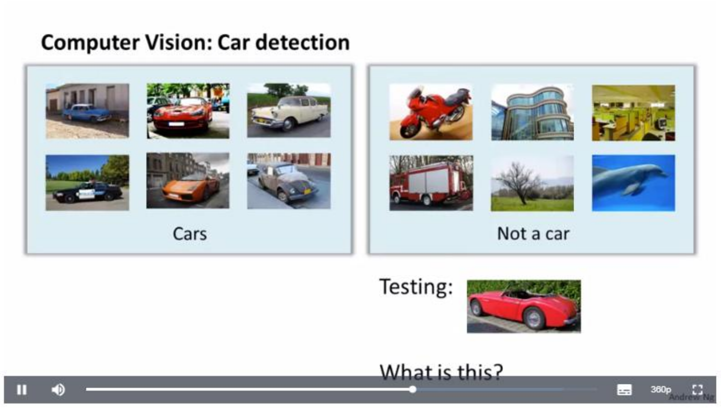
What is this?

You see this:

But the camera sees this:

194	210	201	212	199	213	215	195	178	158	182	209
180	189	190	221	209	205	191	167	147	115	129	163
114	126	140	188	176	165	152	140	170	106	78	88
87	103	115	154	143	142	149	153	173	101	57	57
102	112	106	131	122	138	152	147	128	84	58	66
94	95	79	104	105	124	129	113	107	87	69	67
68	71	69	98	89	92	98	95	89	88	76	67
41	56	68	99	63	45	60	82	58	76	75	65
20	43	69	75	56	41	51	73	55	70	63	44
50	50	57	69	75	75	73	74	53	68	59	37
72	59	53	66	84	92	84	74	57	72	63	42
67	61	58	65	75	78	76	73	59	75	69	50

Non-linear Hypotheses

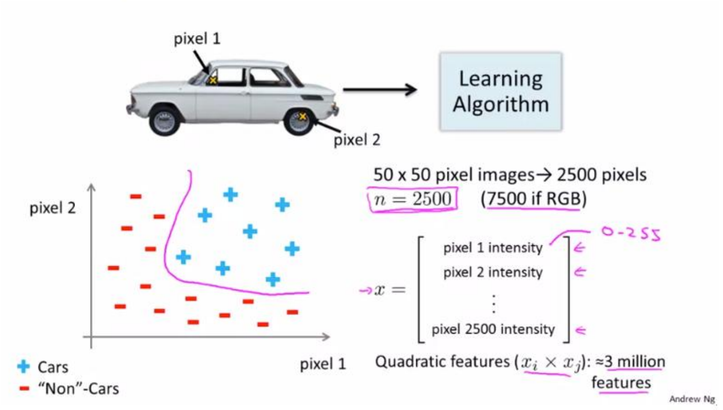


Computer Vision: Car detection

Testing:

What is this?

Non-linear Hypotheses



50 x 50 pixel images→ 2500 pixels

$\boxed{n = 2500}$ (7500 if RGB)

$$\rightarrow x = \begin{bmatrix} \text{pixel 1 intensity} \\ \text{pixel 2 intensity} \\ \vdots \\ \text{pixel 2500 intensity} \end{bmatrix} \begin{matrix} \leftarrow \; 0\text{-}255 \\ \leftarrow \\ \\ \leftarrow \end{matrix}$$

Quadratic features ($x_i \times x_j$): ≈3 million features

Lecture 9: Neurons and the Brain

Neural Networks are a pretty old algorithm that was originally motivated by the goal of having machines that can mimic the brain. Now in this class, of course I'm teaching Neural Networks to you because they work really well for different machine learning problems and not, certainly not because they're logically motivated. In this video, I'd like to give you some of the background on Neural Networks. So that we can get a sense of what we can expect them to do. Both in the sense of applying them to modern day machinery problems, as well as for those of you that might be interested in maybe the big AI dream of someday building truly intelligent machines. Also, how Neural Networks might pertain to that. The origins of Neural Networks was as algorithms that try to mimic the brain and those a sense that if we want to build learning systems while why not mimic perhaps the most amazing learning machine we know about, which is perhaps the brain. Neural Networks came to be very widely used throughout the 1980's and 1990's and for various reasons as popularity diminished in the late 90's. But more recently, Neural Networks have had a major recent resurgence. One of the reasons for this resurgence is that Neural Networks are computationally some what more expensive algorithm and so, it was only, you know, maybe somewhat more recently that computers became fast enough to really run large scale Neural Networks and because of that as well as a few other technical reasons which we'll talk about later, modern Neural Networks today are the state of the art technique for many applications. So, when you think about mimicking the brain while one of the human brain does tell me same things, right? The brain can learn to see process images than to hear, learn to process our sense of touch. We can, you know, learn to do math, learn to do calculus, and the brain does so many different and amazing things. It seems like if you want to mimic the brain it seems like you have to write lots of different pieces of software to mimic all of these different fascinating, amazing things that the brain tell us, but does is this fascinating hypothesis that the way the brain does all of these different things is not worth like a thousand different programs, but instead, the way the brain does it is worth just a single learning algorithm. This is just a hypothesis but let me share with you some of the evidence for this. This part of the brain, that little red part of the brain, is your auditory cortex and the way you're understanding my voice now is your ear is taking the sound signal and routing the sound signal to your auditory cortex and that's what's allowing you to understand my words. Neuroscientists have done the following fascinating experiments where you cut the wire from the ears to the auditory cortex and you re-wire, in this case an animal's brain, so that the signal from the eyes to the optic nerve eventually gets routed to the auditory cortex. If you do this it turns out, the auditory cortex will learn to see. And this is in every single sense of the word see as we know it. So, if you do this to the animals, the animals can perform visual discrimination task and as they can look at images and make appropriate decisions based on the images and they're doing it with that piece of brain tissue. Here's another example. That red piece of brain tissue is your somatosensory cortex. That's how you process your sense of touch. If you do a similar re-wiring process then the somatosensory cortex will learn to see. Because of this and other similar experiments, these are called neuro-rewiring experiments. There's this sense that if the same piece of physical brain tissue can process sight or sound or touch then maybe there is one learning algorithm that can process sight or sound or touch. And instead of needing to implement a thousand different programs or a thousand different algorithms to do, you know, the thousand wonderful things that the brain does, maybe what we need to do is figure out some approximation or to whatever the brain's learning algorithm is and implement that and that the brain learned by itself how to process these different types of data. To a surprisingly large extent, it seems as if we can plug in almost any sensor to almost any part of the brain and so, within the reason, the brain will learn to deal with it. Here are a few more examples. On the upper left is an example of learning to see with your tongue. The way it works is--this is actually a system called BrainPort undergoing FDA trials now to help blind people see--but the way it works is, you strap a grayscale camera to your forehead, facing forward, that takes the low resolution grayscale image of what's in front of you and you then run a wire to an array of electrodes that you place on your tongue so that each pixel gets mapped to a location on your tongue where maybe a high voltage corresponds to a dark pixel and a low voltage corresponds to a bright pixel and, even as it does today, with this sort of system you and I will be able to learn to see, you know, in tens of minutes with our tongues. Here's a second example of human echo location or human sonar. So there are two ways you can do this. You can either snap your fingers, or click your tongue. I can't do that very well. But there are blind people today that are actually being trained in schools to do this and learn to interpret the pattern of sounds bouncing off your environment - that's sonar. So, if after you search on YouTube, there are actually videos of this amazing kid who tragically because of cancer had his eyeballs removed, so this is a kid with no eyeballs. But by snapping his fingers, he can walk around and never hit anything. He can ride a skateboard. He can shoot a basketball into a hoop and this is a kid with no eyeballs. Third example is the Haptic Belt where if you have a strap around your waist, ring up buzzers and always have the northmost one buzzing. You can give a human a direction sense similar to maybe how birds can, you know, sense where north is. And, some of the bizarre example, but if you plug a third eye into a frog, the frog will learn to use that eye as well. So, it's pretty amazing to what extent is as if you can plug in almost any sensor to the brain and the brain's learning algorithm will just figure out how to learn from that data and deal with that data. And there's a sense that if we can figure out what the brain's learning algorithm is, and, you know, implement it or implement some approximation to that algorithm on a computer, maybe that would be our best shot at, you know, making real progress towards the AI, the artificial intelligence dream of someday building truly intelligent machines. Now, of course, I'm not teaching Neural Networks, you know, just because they might give us a window into this far-off AI dream, even though I'm personally, that's one of the things that I personally work on in my research life. But the main reason I'm teaching Neural Networks in this class is because it's actually a very effective state of the art technique for modern day machine learning applications.

1.