Queen Mary
University of London

School of Electronic Engineering and Computer Science

ECS708 Machine Learning

Assignment 1: Part 1 - Linear Regression

Aim

The aim of this part is to become familiar with linear regression.

1- Linear Regression with One Variable

In this exercise, we have one variable, X, and we building a model to predict one output, Y.

The main program will be run from *mllab1.m* . Your first task is to run the main file.

You will see a graph open displaying the changing gradient of your hypothesis. When gradient descent has finished press "enter" to load the graph displaying the cost obtained with the theta values used at each iteration. Both the hypothesis and the cost graph will be flat because we need to calculate our hypothesis. For one variable linear regression, the hypothesis function is:

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 \qquad (1.1)$$

where x_0 is the bias term and is set to 1 for all training examples.

Task 1 Modify the function *calculate_hypothesis.m* to return the predicted value for a single specified training example. Include in the report the corresponding lines from your code.

[X]

For example, if the first training example is [1,5] corresponding to [x_0, x_1], return,

$$\theta_0 1 + \theta_1 5$$

When this is completed, run *mllab1.m* again and you should see the gradient of the hypothesis better fit the model and the cost going down over time. Notice that the hypothesis function is not being used in the *gradient_descent* function. Modify it to use the *calculate_hypothesis* function. Include the corresponding lines of the code in your report.

[X]

Now modify the values for the learning rate, alpha in *mllab1.m*.

Observe what happens when you use a very high or very low learning rate. Document and comment on your findings in the report.

[X]

2. Linear Regression with Multiple Variables

Create a new folder and copy all the files in the new one and start running the programs according to the steps below. In this part, we will be using mllab2.m

We will now look at linear regression with two variables (three including the bias). The hypothesis function now looks

like this:

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 \qquad (2.1)$$

In this exercise, we are looking at house price data. x_0 corresponds to the area of the house in square feet and x_1 corresponds to the number of bedrooms. If you open ex1data2.txt you will see there is a large difference between the values of x_1 and x_2.

In order to bring the two features into a similar range, X is normalized using the *normalise_features* method.

Task 2 Modify the functions *calculate_hypothesis* and *gradient_descent* to support the new hypothesis function. This should be sufficiently general so that we can have any number of extra variables. Include the relevant lines of the code in your report.

[X]

Run mllab2.m and see how different values of alpha affect the convergence of the algorithm. Print the theta values found at the end of the optimization. Does anything surprise you? Include the values of theta and your observations in your report.

[X]

Finally, we would like to use our trained theta values to make a prediction. Add some lines of code in *mllab2.m* to make predictions of house prices.

How much does your algorithm predicts that a house with 1650 sq. ft. and 3 bedrooms cost?

How about 3000 sq. ft. and 4 bedrooms?

To make a prediction you will need to normalize the two variables using the saved values for mean and standard deviation.

Remember that these are different for the two variables. The formula used for normalization is:

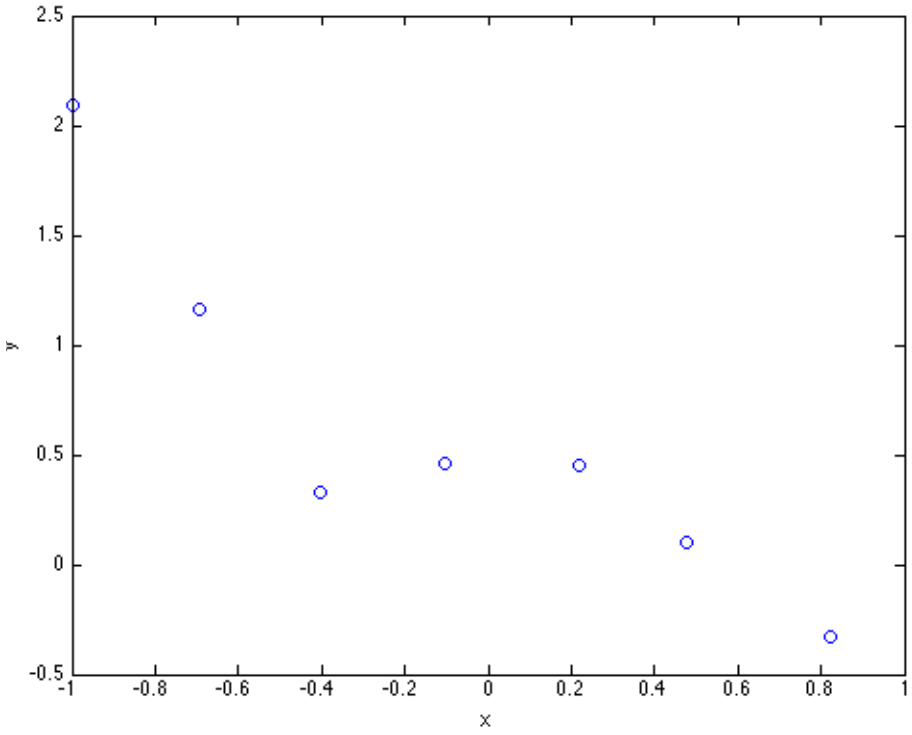
$$X_{norm} = \frac{X - mean}{standard\ deviation} \qquad (2.2)$$

Add the lines of the code that you wrote in your report. Include as well the predictions that you make for the prices of the houses above.

3. Regularized Linear Regression

Create a new folder and copy all the files in the new one and start running the programs according to the steps below. In this part, we will be using mllab3.m

In this exercise, we will be trying to create a model that fits data that is clearly not linear. We will be attempting to fit the data points seen in the graph below:



In order to fit this data we will create a new hypothesis function, which uses a fifth-order polynomial:

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^3 + \theta_4 x_1^4 + \theta_5 x_1^5 \qquad (3.1)$$

We will be fitting the following data:

X"	Y"
-0.99768	2.0885
-0.69574	1.1646
-0.40373	0.3287
-0.10236	0.46013
0.22024	0.44808
0.47742	0.10013
0.82229	-0.32952

As we are fitting a small number of points with a high order model, there is a danger of over fitting.

To attempt to avoid this we will use regularization. Our cost function becomes:

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=2\$}^{n} \theta_j^2\right] \qquad (3.2)$$

Task 3 Note that the punishment for having more terms is not applied to the bias. This cost function has been implemented already in the function *compute_cost_regularised*. Modify *gradient_descent* to use the compute_cost_regularised method instead of *compute_cost*. Include the relevant lines of the code in your report and a brief explanation.

[X]

Next, modify *gradient_descent* to incorporate the new cost function. Again, we do not want to punish the bias term. This means that we use a different update technique for the partial derivative of θ_0, and add the regularization to all of the others:

$$\theta_0 = \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_0^{(i)} \qquad \text{j=0}$$

$$\theta_j = \theta_j \left(1 - \alpha \frac{\lambda}{m}\right) - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_j^{(i)} \qquad \text{j>0}$$

Include the relevant lines of the code in your report.

[X]

After *gradient_descent* has been updated, run *mllab3.m*. This will plot the hypothesis function found at the end of the optimization.

First of all, find the best value of alpha to use in order to optimize best. Report the value of alpha that you found in your report.

[X]

Next, experiment with different values of λ and see how this affects the shape of the hypothesis. Note that *gradient_descent* will have to be modified to take an extra parameter, *l* (which represents λ). Include in your report the plots for a few different values of λ and comment.

[X]

Queen Mary
University of London

School of Electronic Engineering and Computer Science

Assignment 1 – Part 2: Logistic Regression and Neural Networks

Aim

The aim of this part is to become familiar with Logistic Regression and Neural Networks.

1. Logistic Regression

With logistic regression the values we want to predict are now discrete classes, not continuous variables. In other words, logistic regression is for classification tasks. In the binary classification problem we have classes 0 and 1, e.g. classifying email as spam or not spam based on words used in the email. The form of the hypothesis in logistic regression is a logistic/sigmoid function given by the formula below:

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

!

This is the form of the model. The cost function for logistic regression is

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

!

which when taking partial derivatives and putting these into the gradient descent update equation gives

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

In the *logistic_regression.m*, fill out the *sigmoid(x)* function by implementing the formula above. Now run *plot_sigmoid_function.m*. You should have plotted the sigmoid function.

Task 1: Include in your report the relevant lines of code and the result of the running the *plot_sigmoid_function.m*.

The dataset we use is an exam score dataset with a 2-dimensional input space (for easy visualization) and a binary classification of whether the student was admitted or not. The data can be found in: '**ex4x.dat'** and '**ex4y.dat'**. The data has been loaded into the variables X and y. Additionally, the bias variable 1 has been added to each data point. Run *plot_data.m* to plot the data.

To enable the data to be more easily optimized the features need to be normalized. To do this uncomment the line relevant to normalization.

Task 2. Plot the data again to see what it looks like in this new format. Enclose this in your report.

1.1. Cost function and gradient for logistic regression

Task 3. Modify the hypothesis function in *logistic_regession.m* so that for a given dataset, theta and training example it returns the hypothesis. For example, for the dataset:

X=[[1,10,20],[1,20,30] and for Theta = [0.5,0.6,0.7],

the call to the function *calculate_hypothesis(X,theta,0)* will return: *sigmoid*(1 * 0.5 + 10 * 0.6 + 20 * 0.7)

The function should be able to handle datasets of any size. Enclose in your report the relevant lines of code.

Task 4. Modify the line:
cost = 0.0 in *compute_cost(X,y,theta)* in **logistic_regression.m** so that we use our cost function:

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

To calculate a logarithm you can use log(x). Now run the file *lab2_lr_ex1.m*. What is the final cost found by the gradient descent algorithm? In your report include the modified code and the cost graph.

1.2. Draw the decision boundary

Task 5. Plot the decision boundary. This corresponds to the line where $\theta^T \mathbf{x} = 0$
For example, if $h_\theta = \theta_1 x_1 + \theta_2 x_2$ the boundary is where $\theta_1 x_1 + \theta_2 x_2 = 0$

Rearrange the equation in terms of x_2 and in the plot function in ***logistic_regression.m*** set *y1* equal to x_2 when x_1 is at the minimum in the data set and set y2 equal to x_2 when x_1 is at its maximum in the data set. Uncomment the relevant plot function in ***lab2_lr_ex1.m*** and include the graph in your report.

1.3. Non-linear features and overfitting

We don't always have access to the full dataset. In ***lab2_lr_ex2.m***, the dataset has been split into 10% training data and 90% testing data using the function, *return_test_set(X,y,training_set_size).* Gradient descent is run on the training data (this means that the parameters are learned using only the training set and not the test set). After theta has been calculated, *compute_cost()* is called on both datasets (training and test set) and the error is printed, as well as graphs of the data and boundaries shown.

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

Task 6. Run the code in ***lab2_lr_ex2.m*** several times.
What is the general difference between the training and test error? When does the training set generalize well? Demonstrate two splits with good and bad generalisation and put both graphs in your report.

In ***lab2_lr_ex3a.m***, instead of using just the 2D feature vector, incorporate the following non-linear features: $x_1 * x_2$ ${x_1}^2$ and ${x_2}^2$. This results in a 5D input vector per data point, and so you must use 6 parameters θ.

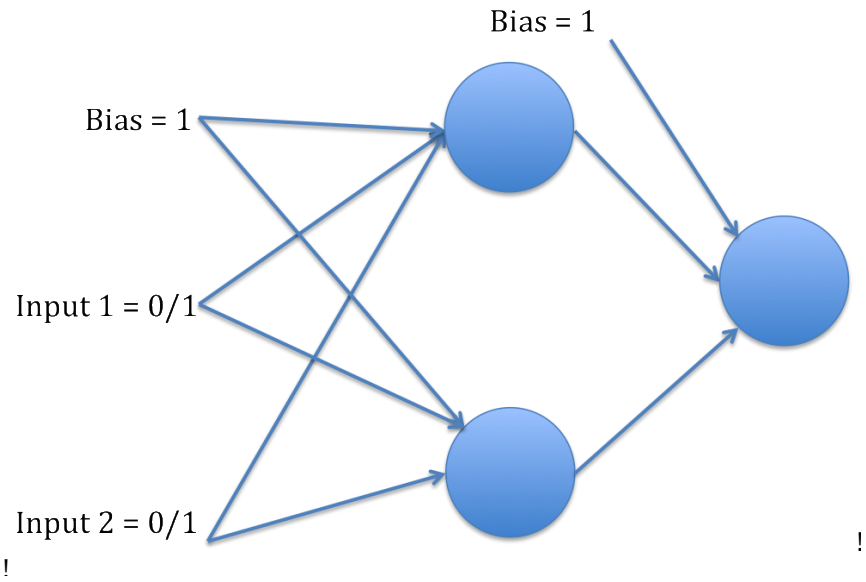
Task 7. Run logistic regression on this dataset. How does the error compare to using the original features (i.e. the error found in Task 4)? Include in your report the error and an explanation on what happens.

Task 8. In ***lab2_lr_ex3b.m*** the data is split into a test set and a training set. Add your new features from the question above. Modify the function *gradient_descent_training()* to store the current cost for the training set and testing set. Store the cost of the training set to *cost_array_training* and for the test set to *cost_array_test*

These arrays are passed to plotdata2(), which will show the cost function of the training (in blue) and test set (in red). Experiment with different sizes of training and test set (remember that the total data size is 80) and show the effect of using sets of different sizes by saving the graphs and putting them in your report. Add extra features (e.g. a third order polynomial) and analyse the effect. What happens when the cost function of the training set goes down but that of the test set goes up?
Task 9. With the aid of a diagram of the decision space, explain why a logistic regression unit cannot solve the XOR classification problem.

2. Neural Network

We will now perform backpropagation on a feedforward network to solve the XOR problem. The network has been set up 2 input neurons, 2 hidden neurons and one output neuron. There is also a bias on the hidden and output layer, e.g. with the following architecture:



Open the folder neural_networks and the file **xorExample.m** This is an example script that calls the model in ***NeuralNetwork.m***.

Your first task is to modify the function *sigmoid*(z) to use your sigmoid function completed in question 1 above.

The program works in the following way:
An array storing the input patterns and the desired outputs is stored in the pattern variable. This is passed to the *train*() function which also takes as arguments the number of hidden units desired (set to 2 for XOR), the number of iterations we should run the algorithm for, and the learning rate.

In *train*() the neural network data structure is created, and the main loop of the algorithm is executed. It works in the following way:
For each iteration:
    For each input_pattern:
        Feedforward()
        Backpropagate()

At for each input pattern, we activate the network by feeding the input signal through the neurons. After this is completed, we compare the actual output to the desired output and update the weights using gradient descent and the backpropagated error signal.

Task 10. Implement backpropagation. Although XOR only has one output, this should support outputs of any size. Do this following the steps below:

Step 1. For each output, k, calculate the error delta: $\delta_k = (y_k - t_k) * g'(x_k)$
where y_k is the response of the output unit and t_k is the desired output (target). This error, $(y_k - t_k)$, is multiplied by the derivative of the sigmoid function applied to the pre-sigmoided signal of the output neuron.

Use the function sigmoid_derivative(y) and see below. Store each error in the output_deltas[] list.

(The first derivative, g', of g(x) is, g' = g(x)*(1-g(x)). As we have already calculated a = g(x) on the forward pass, in the code g' is calculated using g' = a * (1-a))

Step 2. We now need to backpropagate this error to the hidden neurons. To accomplish this remember that,

$$\delta_j = g'(x_j) \sum_k w_{jk}\, \delta_j$$

where δ_j is the error on the *j*th hidden unit, x_j is activity of the hidden neuron (before it has been passed through the sigmoid function but see footnote1), g′ is the derivative of the sigmoid function, δ_k is the error from the output neuron that we have stored in output_deltas, and w_{kj} is the weight from the hidden neuron *j* to the output neuron k. Once this has been calculated add δ_j to the array, *hidden_deltas*.

Step 3. We now need to update the weights on the connections from the hidden neurons to the output neurons. This is accomplished using the formula:

$$w_{jk} = w_{jk} - \eta \delta_k a_j$$

where w_{jk} is the weight connecting the *j*th **hidden** neuron to the *k*th output neuron. a_j is the activity of the *j*th hidden **neuron** (after it has been transformed by the sigmoid function), δ_k is the error from the output neuron stored in output_deltas and α is the learning rate.

Step 4. Finally we need to update the weights on the connections from the hidden neurons to the input neurons. Here, again we use this equation

$$w_{jk} = w_{jk} - \eta \delta_k a_j$$

where w_{jk} is the weight connecting the *j*th input neuron to the *k*th hidden neuron. a_j is the activity of the *j*th input neuron (without any transform), δ_k is the backpropagated error from the *k*th hidden neuron and α is the learning rate.

2.1. Implement backpropagation on XOR

Your task is to implement backpropagation and then run the file with different learning rates (loading from **xorExample.m**).
What learning rate do you find best? Include a graph of the error function in your report. Note that the backpropagation can get stuck in local optima. What are the outputs and error when it gets stuck?

Task 11. Change the training data in ***xor.m*** to implement a different logical function, such as NOR or AND. Plot the error function of a successful trial.

2.2. Implement backpropagation on Iris

Now that you have implemented backpropagation we have built a powerful classifier. We will test this on the "Iris" dataset (http://en.wikipedia.org/wiki/Iris_flower_data_set), which is a benchmark problem for classifiers. It has four input features – sepal length, sepal width, petal length, and petal width – which are used to classify three different species of flower.

In ***irisExample.m***, we have taken this dataset and split it 50-50 into a training and a test set.

Task 12. The Iris data set contains three different classes of data that we need to discriminate between. How would accomplish this if we used a logistic regression unit? How is it different using a neural network?

Task 13. Run ***iris.m*** using the following number of hidden neurons: 1, 2, 3, 5, 7, 10. The program will plot the costs of the training set (blue) and test set (red) at each iteration. What are the differences for each number of hidden neurons? Which number do you think is the best to use? How well do you think that we have generalized?

Write a report about what you have done along with relevant graphs. Save the solution in a folder with your ID. Create and submit:

1) **a .zip that contains all of your code for both parts and**
2) **a copy of your report. The report should be in .pdf format named as Assignment1-StudentName-StudentNumber.pdf**