Práctica 3: Linear and Logistic regression

1 Files included

Linear regression files
ex1data2.txt - Dataset for linear regression with one variable
featureNormalize.m - Function to normalize data
computeCostMulti.m - Function to compute the cost of linear regression
gradientDescentMulti.m - Function to run gradient descent to learn theta
multivariable.m – Main file which calls the other functions

Logistic regression files
ex2.m - Main file which calls the other functions
ex2data1.txt - Training set for logistic regression
plotData.m - Function to plot 2D classification data
sigmoid.m - Sigmoid Function
costFunction.m - Logistic Regression Cost Function
plotDecisionBounday.m - Function to plot classifier's decision boundary

2 Linear regression with multiple variables

In this part, you will implement linear regression with multiple variables to predict the prices of houses. Suppose you are selling your house and you want to know what a good market price would be. One way to do this is to first collect information on recent houses sold and make a model of housing prices.

The file ex1data2.txt contains a training set of housing prices in Portland, Oregon. The first column is the size of the house (in square feet), the second column is the number of bedrooms, and the third column is the price of the house.

2.1 Feature Normalization

By looking at the values, note that house sizes are about 1000 times the number of bedrooms. When features differ by orders of magnitude, first performing feature scaling can make gradient descent converge much more quickly. Your task here is to complete the code in featureNormalize.m to

- Subtract the mean value of each feature from the dataset.
- After subtracting the mean, additionally scale (divide) the feature values by their respective standard deviations.

The standard deviation is a way of measuring how much variation there is in the range of values of a particular feature; this is an alternative to taking the range of values (max-min). In Matlab, you can use the “std" function to compute the standard deviation. For example, inside featureNormalize.m, the quantity X(:,1) contains all the values of x1 (house sizes) in the training set, so std(X(:,1)) computes the standard deviation of the house sizes. At the time that featureNormalize.m is called, the extra column of 1's corresponding to x0 = 1 has not yet been added to X (see multivariable.m).

Implementation Note: When normalizing the features, it is important to store the values used for normalization - the mean value and the standard deviation used for the computations. After learning the parameters from the model, we often want to predict the prices of houses we have not seen before. Given a new x value (living room area and number of bedrooms), we must first normalize x using the mean and standard deviation that we had previously computed from the training set.

2.2 Gradient Descent

Previously, you implemented gradient descent on a univariate regression problem. The only difference now is that there is one more feature in the matrix X. The hypothesis function and the batch gradient descent update rule remain unchanged.

You should complete the code in computeCostMulti.m and gradientDescentMulti.m to implement the cost function and gradient descent for linear regression with multiple variables. If your code in the previous part (single variable) already supports multiple variables, you can use it here too.

2.3 Selecting learning rates

In this part of the exercise, you will get to try out different learning rates for the dataset and find a learning rate that converges quickly. You can change the learning rate by modifying multivariable.m. The next phase in multivariable.m will call your gradientDescent.m function and run gradient descent at the chosen learning rate. The function should also return the history of $J(\theta)$ values in a vector J. After the last iteration, the multivariable.m script plots the J values against the number of the iterations.

If your value of $J(\theta)$ increases or even blows up, adjust your learning rate and try again. We recommend trying values of the learning rate on a log-scale, at multiplicative steps of about 3 times the previous value (i.e., 0.3, 0.1, 0.03, 0.01 and so on). You may also want to adjust the number of iterations you are running if that will help you see the overall trend in the curve.

Implementation Note: If your learning rate is too large, $J(\theta)$ can diverge and `blow up', resulting in values which are too large for computer calculations. In these situations, Matlab will tend to return NaNs. NaN stands for `not a number' and is often caused by undefined operations that involve $-\infty$ and $+\infty$.

To compare how different learning learning rates affect convergence, it's helpful to plot J for several learning rates on the same figure. In Matlab, this can be done by performing gradient descent multiple times with a `hold on' command between plots. Concretely, if you've tried three different values of alpha (you should probably try more values than this) and stored the costs in J1, J2 and J3, you can use the following commands to plot them on the same figure (The arguments `b', `r', and `k' specify different colors for the plots):

```
plot(1:50, J1(1:50), `b');
hold on;
plot(1:50, J2(1:50), `r');
plot(1:50, J3(1:50), `k');
```

Notice the changes in the convergence curves as the learning rate changes. With a small learning rate, you should find that gradient descent takes a very long time to converge to the optimal value. Conversely, with a large learning rate, gradient descent might not converge or might even diverge! Using the best learning rate that you found, run the multivariable.m script to run gradient descent until convergence to find the final values of θ. Next, use this value of θ to predict the price of a house with 1650 square feet and 3 bedrooms.

3 Logistic regression

In this part of the exercise, you will build a logistic regression model to predict whether a student gets admitted into a university. Suppose that you are the administrator of a university department and you want to determine each applicant's chance of admission based on their results on two exams. You have historical data from previous applicants that you can use as a training set for logistic regression. For each training example, you have the applicant's scores on two exams and the admissions decision. Your task is to build a classification model that estimates an applicant's probability of admission based the scores from those two exams.

3.1 Visualizing the data

Before starting to implement any learning algorithm, it is always good to visualize the data if possible. In the first part of ex2.m, the code will load the data and display it on a 2-dimensional plot by calling the function plotData. You will now complete the code in plotData so that it displays a figure where the axes are the two exam scores, and the positive and negative examples are shown with different markers. You may find usefull to use the function find (pos = find(y==1) finds the indices of the positive examples) in Matlab.

3.2 Sigmoid function

Before you start with the actual cost function, recall that the logistic regression hypothesis is defined as: $h\theta(x) = g(\theta^T x)$; where function g is the sigmoid function. The sigmoid function is defined as: $g(z) = 1/(1 + e^{-z})$

Your first step is to implement this function in sigmoid.m so it can be called by the rest of your program. When you are finished, try testing a few values by calling sigmoid(x) at the Matlab command line. For large positive values of x, the sigmoid should be close to 1, while for large negative values, the sigmoid should be close to 0. Evaluating sigmoid(0) should give you exactly 0.5.

3.3 Learning parameters using fminunc

In the previous assignment, you found the optimal parameters of a linear regression model by implementing gradient descent. You wrote a cost function and calculated its gradient, then took a gradient descent step accordingly. This time, instead of taking gradient descent steps, you will use an Matlab built-in function called fminunc. Matlab's fminunc is an optimization solver that _nds the minimum of an unconstrained function. For logistic regression, you want to optimize the cost function $J(\theta)$ with parameters θ.

Concretely, you are going to use fminunc to find the best parameters θ for the logistic regression cost function, given a fixed dataset (of X and y values). You will pass to fminunc the following inputs:

- The initial values of the parameters we are trying to optimize.
- A function that, when given the training set and a particular θ, computes the logistic regression cost and gradient with respect to θ for the dataset (X, y)

In ex2.m, we already have code written to call fminunc with the correct arguments.

```
% Set options for fminunc
options = optimset('GradObj', 'on', 'MaxIter', 400);
% Run fminunc to obtain the optimal theta
% This function will return theta and the cost
[theta, cost] = fminunc(@(t)(costFunction(t, X, y)), initial theta, options);
```

In this code snippet, we first defined the options to be used with fminunc. Specifically, we set the GradObj option to on, which tells fminunc that our function returns both the cost and the gradient. This allows fminunc to use the gradient when minimizing the function. Furthermore, we set the MaxIter option to 400, so that fminunc will run for at most 400 steps before it terminates.

To specify the actual function we are minimizing, we use @(t) (costFunction(t, X, y)). This creates a function, with argument t, which calls your costFunction. This allows us to wrap the costFunction for use with fminunc. If you have completed the costFunction correctly, fminunc will converge on the right optimization parameters and return the final values of the cost and θ. Notice that by using fminunc, you did not have to write any loops yourself, or set a learning rate like you did for gradient descent. This is all done by fminunc: you only needed to provide a function calculating the cost and the gradient.

Once fminunc completes, ex2.m will call your costFunction function using the optimal parameters of θ. You should see that the cost is about 0.203. This final θ value will then be used to plot the decision boundary on the training data.

3.4 Evaluating logistic regression

After learning the parameters, you can use the model to predict whether a particular student will be admitted. For a student with an Exam 1 score of 45 and an Exam 2 score of 85, what is the expected admission probability?

Submitting your answer

The práctica can be solved in teams of two people (1 submission per team). Submission is through the Aula Global. Submissions should contain the code of the files you modified and the plots your programs generated. Deadline is the beginning of the next práctica. Late submissions will be penalized.