Discussion Session 3 - soln

October 16, 2018

1 Discussion Session 3: Multi-Class Classification

1.0.1 Introduction
In this discussion session, you will implement one-vs-all logistic regression.

1.1 Multi-Class Classification
For this exercise, you will use logistic regression to recognize the category of each student based on their liveliness level and age.

1.1.1 Data Set
The data set was generated for you and you will train a model to classify each category independently. You are not allowed to look at your homework to complete this lab. You should be able to implement everything from scratch in groups. We have given you a ‘plotPoints’ function and loaded in the data for you.

In [5]: % generating the data
   plotPoints();
   [X , y] = loadData();
There are 300 training examples and three classes. Each class is labeled \{1,2,3\} and is represented by the green, red, or blue points. In this discussion session, you will fit three lines to the plot to separate each cluster from the other two clusters. This is known as the One vs All model. By the end of the session you will, you will be able to take a problem with several categories and classify each one accordingly. This is an easy example (i.e. the data is clearly separable) to help you get started building these types of models. This is a good strategy to help you classify data point into different categories.

1.2 Vectorizing Logistic Regression

1.2.1 Vectorizing the Cost function

\[
J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left[ -y^{(i)} \log(h_{\theta}(x^{(i)})) - (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) \right] + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2. \tag{4}
\]

Note that you should not be regularizing \(\theta_0\) which is used for the bias term.

Correspondingly, the partial derivative of regularized logistic regression cost for \(\theta_j\) is defined as

\[
\frac{\partial J(\theta)}{\partial \theta_0} = \frac{1}{m} \sum_{i=1}^{m} (h_{\theta}(x^{(i)}) - y^{(i)})x_j^{(i)} \quad \text{for } j = 0 \tag{5}
\]
\[
\frac{\partial J(\theta)}{\partial \theta_j} = \left( \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})x^{(i)}_j \right) + \frac{\lambda}{m} \theta_j \quad \text{for } j \geq 1
\]  

(6)

In [6]: % Write a function to get the sigmoid function.

```matlab
function g = sigmoid(z)
%SIGMOID Compute sigmoid function
% J = SIGMOID(z) computes the sigmoid of z.

g = 1.0 ./ (1.0 + exp(-z));
end
```

In [7]: % lrCostFunction(theta, X, y, lambda)

% computes the cost and the gradient. You should return a vector for grad and a scalar for the cost.

```matlab
function [J, grad] = lrCostFunction(theta, X, y, lambda)

m = length(y);  % number of training examples

h = sigmoid(X * theta);
J = (1/m) * sum(-y.*log(h) - (1-y).*log(1-h)) + ...
    (1/m) * 0.5 * lambda * sum(theta(2:end).^2);

grad = (1/m) .* X' .* ((sigmoid(X * theta)) - y);
grad(2:end) = grad(2:end) + (1/m) * lambda * theta(2:end);

grad = grad(:);
end
```

1.2.2 Vectorizing the gradient

Recall that the gradient of the (unregularized) logistic regression cost is a vector where the \( j^{th} \) element is defined as

\[
\frac{\partial J}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^{m} \left( (h_\theta(x^{(i)}) - y^{(i)})x^{(i)}_j \right).
\]  

(2)

**Implementation**

\( J = lrCostFunction(\theta, X, y, \lambda) \) computes the cost of using \( \theta \) as the parameter for regularized logistic regression and the gradient of the cost w.r.t. to the parameters.

2 One vs. All

[all_theta] = oneVsAll(X, y, num_labels, lambda) trains \( num\_labels \) logistic regression classifiers and returns each of these classifiers in a matrix \( all\_theta \), where the \( i^{th} \) row of \( all\_theta \) corresponds to the classifier for label \( i \).
You should complete the function below to train num_labels logistic regression classifiers with regularization parameter lambda.

- initial_theta = zeros(n + 1, 1); ==> (set initial theta)
- options = optimset('GradObj', 'on', 'MaxIter', 50); ==> (Set options for fminunc)

Run fmincg to obtain the optimal theta, this function will return theta and the cost:

- [theta] = fmincg (@(t)(lrCostFunction(t, X, (y == c), lambda)),initial_theta, options);

In [8]: % oneVsAll function oneVsAll(X, y, num_labels, lambda)
    function [all_theta] = oneVsAll(X, y, num_labels, lambda)
        m = size(X, 1); % Some useful variables
        n = size(X, 2);
        all_theta = zeros(num_labels, n + 1); % You need to return the final... 
        X = [ones(m, 1) X]; % Add ones to the X data
        options = optimset('GradObj', 'on', 'MaxIter', 50); % Set options for fminunc
        for c = 1:num_labels % Run fmincg to obtain the optimal theta
            % Set Initial theta
            initial_theta = randn(n + 1, 1);

            % Set options for fminunc
            %options = optimset('GradObj', 'on', 'MaxIter', 50);

            % Run fmincg to obtain the optimal theta
            % This function will return theta and the cost
            [theta] = ... fmincg (@(t)(lrCostFunction(t, X, (y == c), lambda)), ...
                            initial_theta, options);

            all_theta(c, :) = theta';
        end
    end

2.0.3 Plot and prediction

Predict the label for a trained one-vs-all classifier. To what label would you predict the following inputs:

[1, 0, 65] = ?
[1,12, 72] = ?
[1, 5, 80] = ?

To check that your theta works, use the function provided for you and pass in your all_theta variable. We will handle the plotting for you.
In [9]: % generating the data  
% blue is your first theta  
% green is your second theta  
% red is your third theta  
lambda = 1;  
num_labels = 3;  
[all_theta] = oneVsAll(X, y, num_labels, lambda);  
all_theta

Iteration 50 | Cost: 8.852257e-03  
Iteration 50 | Cost: 6.606947e-03  
Iteration 50 | Cost: 4.303838e-03  
all_theta =

19.19495 -1.33984 -0.20877  
7.70509 1.60125 -0.28319  
-111.59964 -0.26612 1.49705

In [10]: % plot the lines use the plotLines(all_theta) function provided  
plotLines(all_theta);
In [19]: % predict the outcome!
x_1 = sigmoid([1, 0, 65] * all_theta(1,:)'); % 1 is blue
x_2 = sigmoid([1, 12, 72] * all_theta(3,:)'); % 2 is green
x = sigmoid([1, 6, 80] * all_theta(3,:)'); % 3 is red
x = sigmoid([1, 4, 74] * all_theta(3,:')) % 3 is red
x = 0.13215

In [15]: % Use the plot lines function provided for you

2.1 Neural Networks - Feed forward

We will go over the Neural Network you implemented for homework, below. Remember the equations were:

\[
J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} \left[-y_k^{(i)} \log \left(h_\theta(x^{(i)})_k\right) - (1 - y_k^{(i)}) \log \left(1 - h_\theta(x^{(i)})_k\right) \right] + \\
\frac{\lambda}{2m} \left[ \sum_{j=1}^{25} \sum_{k=1}^{400} (\Theta^{(1)}_{j,k})^2 + \sum_{j=1}^{10} \sum_{k=1}^{25} (\Theta^{(2)}_{j,k})^2 \right]
\]

\[
g'(z) = \frac{d}{dz} g(z) = g(z)(1 - g(z))
\]

where

\[
sigmoid(z) = g(z) = \frac{1}{1 + e^{-z}}
\]

In this part of the lab, we will ask you to compute the feed forward part of a neural network with 3 layers. The weight are randomly initialized and the result does not mean anything, because we are doing a feed forward on fake data. However, you should think about the dimensions of the input. Your solution should work for m training examples where m is any number.

In [16]: % GRADED FUNCTION: predict

    function p = predict(Theta1, Theta2, Theta3, X)
    % X is m by n
    % Theta1 is y by n+1
    % Theta2 is z by y + 1
    % Theta2 is a by z + 1
    
    m = size(X, 1); % Useful values
    num_labels = size(Theta3, 1);
    p = zeros(size(X, 1), 1); % Return the following variable
% YOUR CODE HERE

h1 = sigmoid([ones(m, 1) X] * Theta1');
size(h1);
h2 = sigmoid([ones(m, 1) h1] * Theta2');
h3 = sigmoid([ones(m, 1) h2] * Theta3')
[dummy, p] = max(h3, [], 2);

% end

In [17]: rand('seed',0)
x1 = rand(([1, 40]))*100;
theta1 = rand(([30, 41]))*20;
theta2 = rand(([15, 31]));
theta3 = rand(([5, 16]));

In [18]: % testing our predict function
   p = predict(theta1, theta2, theta3, x1);

h3 =
   0.99950  0.99997  0.99994  0.99966  0.99897

In [19]: % function softmax
   function s = softmax(z)
      s = exp(z)/sum(exp(z))
   end

In [20]: softmax([1;2;3;4])

s =
   0.032059
   0.087144
   0.236883
   0.643914

ans =
   0.032059
   0.087144
   0.236883
   0.643914