Computer Vision

SSI Pattern Recognition

Practical

Désiré Sidibé
dro-desire.sidibe@u-bourgogne.fr

Logistic Regression[1]

I. Introduction

In this exercise, we will implement logistic regression.
Note that you should have completed Lab1, and, in particular, understand the gradient descent algorithm.

II. Logistic regression

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.
The file `lab2data1.txt` contains historical data from previous applicants. The first two columns correspond to the applicant's scores on the two exams and the last column is the admission decision; a value equals to 0 means the applicant was not selected, while a value equals to 1 means that he/she was accepted.
Your task is to build a classifier that will estimate an applicant's probability of admission based on his scores in the two exams.

1. First, we need to visualize the data. Write a function `plotData(X,y)` that shows the data as in Figure 1.
   Note that you should first load the data and assign the exams scores to a matrix X and the decisions to a vector y.

2. Recall that logistic regression uses the following prediction function

$$y(\mathbf{x}) = \sigma(\mathbf{w}^T \phi(\mathbf{x})),$$

   where ϕ is a basis function, and σ is the sigmoid function defined by

$$\sigma(a) = \frac{1}{1 + e^{-a}}.$$

   (a) Write a function `sigmoid(x)` that computes the value of the sigmoid function for any input x. Note that your function should accept vectors or matrices as input!
   Check your results by drawing the curve of the function.

[1] Updated from Andrew Ng.

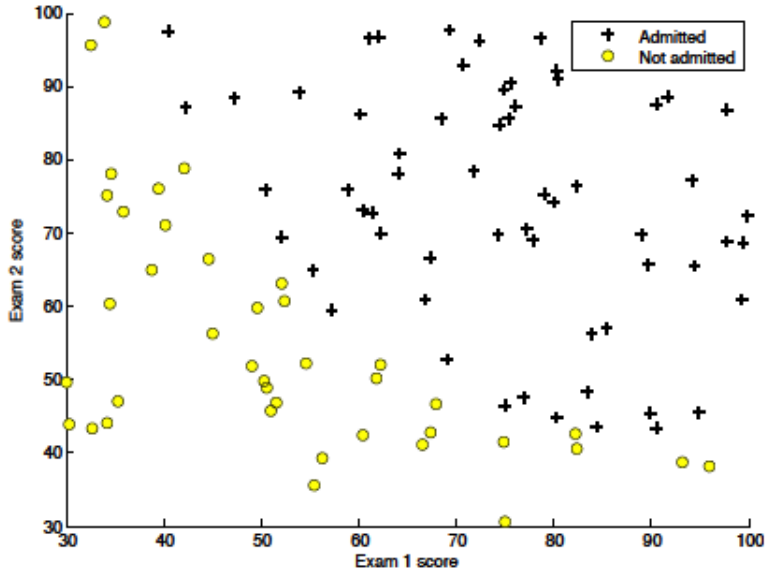

Figure 1: Scatter plot of training data using *plotData(X,y)*.

3. We learn the parameters $\mathbf{w}$ of the model using gradient descent. So we need to compute the cross-entropy function and its gradient. We have shown in class that the cross-entropy is given by

$$E(\mathbf{w}) = -\log p(\mathbf{y}|\mathbf{w}) = -\sum_{n=1}^{N}\{y_n \log(\phi_n) + (1-y_n)\log(1-\phi_n)\},$$

with $\phi_n = p(C_1|\phi(\mathbf{x}_n)) = y(\phi_n) = \sigma(\mathbf{w}^T\phi(\mathbf{x}_n))$.

And the gradient is given by

$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N}(y(\phi_n) - y_n)\phi_n.$$

Write a function `[E, grad] = costFunction(X, y, w)` that returns the cost and gradient given the training data and some parameters $\mathbf{w}$ as input.

Note that in this part we are using $\phi(\mathbf{x}) = \mathbf{x}$, so the function takes $\mathbf{X}$ as input. In the more general case, we should pass $\phi(\mathbf{X})$ as input to the function.

For an initial value of $\mathbf{w} = [0, \ldots, 0]$, your function should output a cost of 0.6931.

4. For finding the optimal value for $\mathbf{w}$, can use gradient descent and implement the following algorithm:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha\nabla E(\mathbf{w}).$$

So we would need to write a loop and set a learning rate α. However, we will use matlab built-in function called `fminunc`.

`fminuc` is an optimization solver that finds the minimum of an unconstrained function. It requires the following inputs:

- The initial values of the parameters $\mathbf{w}$ we want to optimize
- A function that, when given the training set and a particular $\mathbf{w}$, computes the cost function and the gradient w.r.t. $\mathbf{w}$ for the dataset (X, y).

So we will used our previous `costFunction` when we call `fminunc`.

An example of using `fminuc` is as follows:

```
[w, cost] = fminuc( @(t)(costFunction(X,y,t), initial_parameters, ...
                         'GradObj', 'on', 'MaxIter', 400);
```

where we specify that our function return both the cost and gradient (this is done by `GradObj` option set on), we specify the number of iterations (here 400), and we specify the actual function we rae minimizing using the 'short-hand' `@(t)(costFunction(X,y,t).` This creates a function, with argument t, which calls our function `costFunction`.

Note that, we do not need any loop, or set a learning rate for gradient descent. This all done by `fminunc`; we only need to provide afunction calculating the cost and gradient.

Note: this is a Matlab function. So if you are using Python, you might check out the equivalent `fmin_tnc` function.

5. After optimization, we find the final $\mathbf{w}$ that can be used to plot the decision boundary on the training data resulting in a figure similar to figure 2.

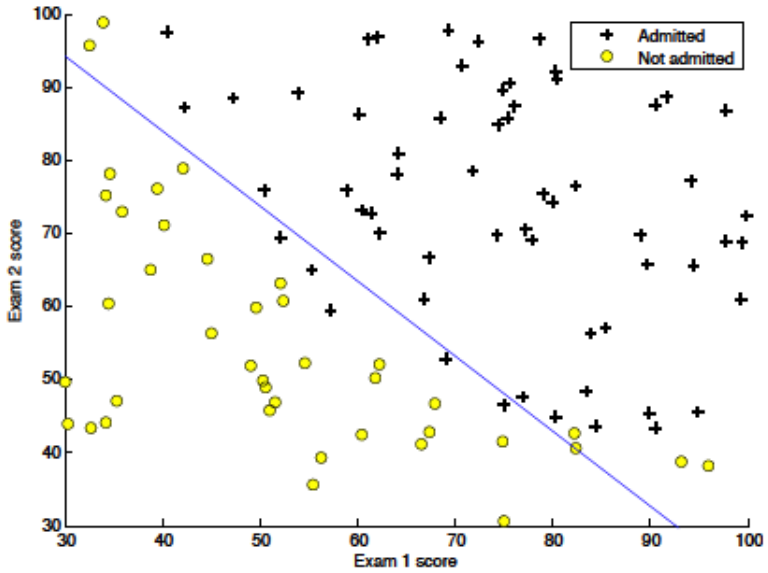

Figure 2: Decision boundary on training data.

6. You can now use your model to predict whether a candidate will be admited. For a student with an Exam 1 score of 45 and and Exam 2 score of 85, you should expect to see an admission probability of 0.776.

   Another way to evaluate the quality of the parameters is to see how well the learned model performed on the training set.

   Write a function `[y] = predict(X,w)` that produce the labels (1 or 0) given some input data and the parameters $\mathbf{w}$ of the model.

   Compute the training accuracy of your model (should be around 89%).

III. Logistic regression with regularization

In this part, we will see the full power of logistic regression. Suppose that you are the production manager of a factory that produces microchips. After production, each microchip goes through various tests to ensure it functions correctly.
The file `lab2data2.txt` contains test results for some microchips on two tests. The first two columns correspond to the tests results, and the last column indicates whether the microchip should be accepted or not; 0 means rejected while 1 means accepted.

Using your function `plotData(X,y)`, you can visualize the training data. As can be seen in Figure 3, this data is clearly not linearly separable, so a straight-forward application of logistic regression will not find a good decision boundary.

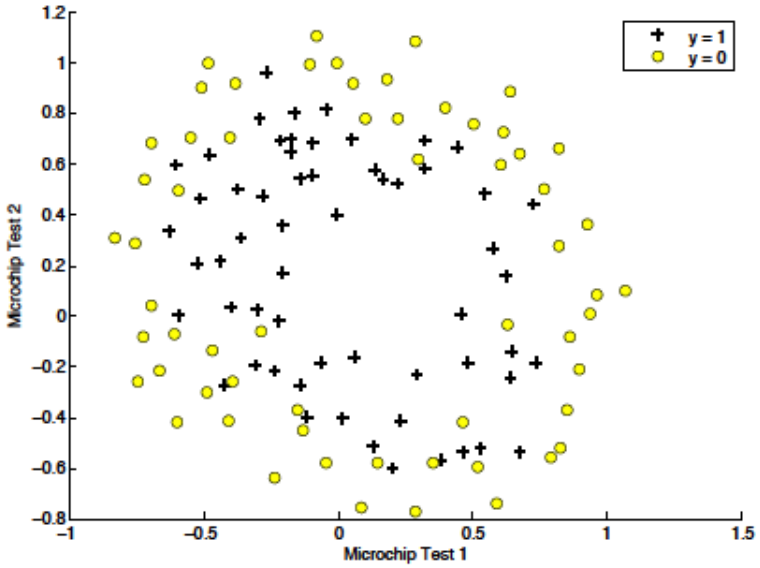

Figure 3: Scatter plot of training data using *plotData(X,y)*.

1. As discussed in class, we can apply a basis function ϕ to each input data to get more features. In this example, we will map the features into all polynomial terms of x_1 and x_2 up to the sixth power. As a result, the original two-dimensional feature vector $\mathbf{x}_i$ is transformed into a 28-dimensional vector $\phi(\mathbf{x}_i)$:

$$\phi(\mathbf{x}) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1^2 \\ x_1 x_2 \\ x_2^2 \\ x_1^3 \\ \vdots \\ x_1 x_2^5 \\ x_2^6 \end{bmatrix}.$$

   We hope that the data will be linearly separable in this higher dimensional space. So we apply logistic regression to the transformed data $\phi(\mathbf{X})$ and the linear decision boundary is projected back to the original 2-dimensional space; it will appear nonlinear.

Write a function `phi_X = transformFeatures(X)` that will map the input features as described above.

2. Note that while the feature mapping allows us to build a more expressive classifier, it is also more susceptible to overfitting. So we use regularization to ovoid overfitting.

   Recall that the regularized cost function for logistic regression is given by

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\{y_n \log(\phi_n) + (1-y_n)\log(1-\phi_n)\} + \frac{\lambda}{2}\sum_{k=1}^{M} w_k^2.$$

   Note that we do not regularize the bias term $\mathbf{w}_0$.

   The gradient of the cost fuction is a vector where the i^{th} element is defined as follows

   - if $i = 0$

$$\frac{\partial E(\mathbf{w})}{\partial w_0} = \sum_{n=1}^{N}(y(\phi_n) - y_n)\phi_n$$

   - else

$$\frac{\partial E(\mathbf{w})}{\partial w_i} = \sum_{n=1}^{N}(y(\phi_n) - y_n)\phi_n + \lambda w_i$$

   You should now modify your function `[E, grad] = costFunction(X, y, w, lambda)` to compute the regularized cost and the gradient. Note that you should pass $\phi(\mathbf{X})$ as input to the function.

   For an initial value of $\mathbf{w} = [0, \ldots, 0]$, your function should output a cost of about 0.69.

3. Similar the previous part, you can use `fminunc` to find the optimum values of $\mathbf{w}$.

   Once you know $\mathbf{w}$, you can plot the decision boundary is see something similar to figure 4.

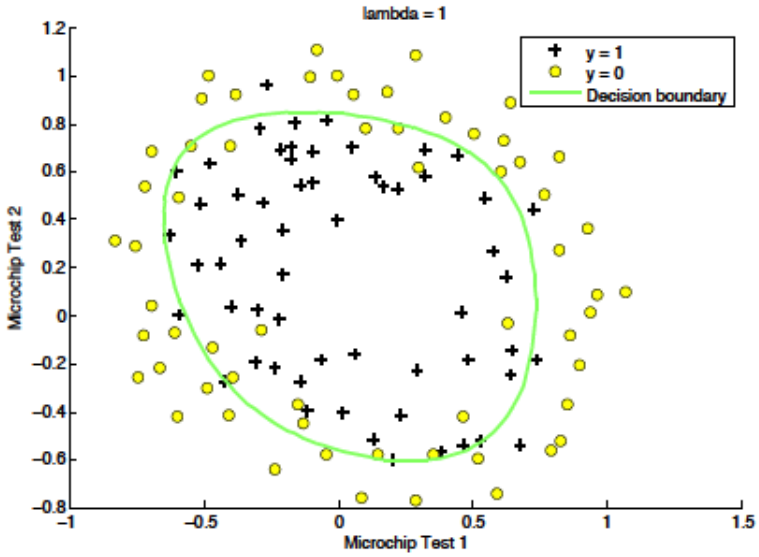

Figure 4: Decision boundary on training data.

4. Of course, you have to chose a value for the regularization parameter λ. Try different values and see the effect on the decision boundary.

   What values of λ help prevent from overfitting: small or large values?

5. **NOTE**: You can use the following code to plot the decision boundary, given the parameters $\mathbf{w}$ (only works in matlab!):

```
% Plot the data
plotData(X(:,2:3), y);
hold on

% Here is the grid range
u = linspace(-1, 1.5, 50);
v = linspace(-1, 1.5, 50);

z = zeros(length(u), length(v));
% Evaluate z = theta*x over the grid
for i = 1:length(u)
     for j = 1:length(v)
            z(i,j) = transformFeatures(u(i), v(j))*w;
      end
end
z = z'; % important to transpose z before calling contour

% you need to specify the range [0, 0]
contour(u, v, z, [0, 0], 'LineWidth', 2)

hold off
```