Training and Testing: Logistic Regression

LBYCP29 – Laboratory 6

Gervin Guevarra, Allen Koizumi, Nicholas Moreno, Charleston Uy

Department of Electronics and Communications Engineering

Gokongwei College of Engineering, De La Salle University

Manila, Philippines

*Abstract*—This laboratory report presents the implementation of an optimized logistic regression using the minFunc package. Unlike previous experiments involving the use of logistic regression, this lab experiment deals with a huge database consisting of handwritten symbols (for the purposes of this lab work, the symbols are only 1s and 0s). The goal is to teach the computer to distinguish these handwritten symbols whether it is a 1 or a 0 using the logistic regression.

Keywords—Logistic regression, training data, testing data

# Introduction

In the previous lab exercise, we observed the differences between training and testing of data using linear regression, but for this lab we will exhibit training and testing using logistic regression. As we have observed in the previous experiment a trained machine does not ensure a 100% accuracy on new data this is because each data set hat its own local minima that could affect the machine’s performance over it and as such it cannot guarantee a 100% accuracy even if it has been already trained.

In this lab exercise, we will use database collection of handwritten 0s and 1s and have the machine decide whether it looks like a 0 or a 1 to it, and we would try to achieve a 100% accuracy in both training and testing, since the data set is not that large to not be capable of producing 100% accuracy

# Objectives

The experiment aims to achieve the following objectives

* To obtain a 100% accuracy for both training and testing data sets
* To observe logistic regression over training and testing conditions and deduce its behavior.

# Data and Results

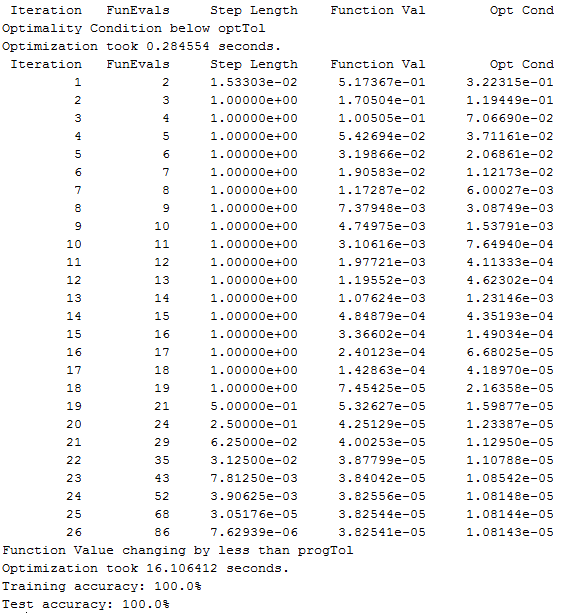


Figure 1. Screenshot of the results obtained after running the ex1b\_logreg.m

# Analysis and Conclusion

<insert analysis and conclusion here>

# Bibliography

|  |  |
| --- | --- |
| [1] | A. Ng, "Spervised Learning," n.d. [Online].  Available: http://cs229.stanford.edu/notes/cs229-notes1.pdf [Accessed 22 September 2015]. |
| [2] | A. Ng, "Machine Learning,"  n.d. [Online]. Available: https://class.coursera.org/ml-003/lecture.  [Accessed 22 September 2015]. |
| [3] | Institute for Digital Research and Edcation, "Stata Data Analysis and  Exampes", 2015 [Online]. Available: http://www.ats.ucla.edu/stat/stata/dae/logit.htm.  [Accessed 22 September 2015]. |

# Appendix

**ex1b\_logreg.m**

addpath ../common

addpath ../common/minFunc\_2012/minFunc

addpath ../common/minFunc\_2012/minFunc/compiled

% Load the MNIST data for this exercise.

% train.X and test.X will contain the training and testing images.

% Each matrix has size [n,m] where:

% m is the number of examples.

% n is the number of pixels in each image.

% train.y and test.y will contain the corresponding labels (0 or 1).

binary\_digits = true;

[train,test] = ex1\_load\_mnist(binary\_digits);

% Add row of 1s to the dataset to act as an intercept term.

train.X = [ones(1,size(train.X,2)); train.X];

test.X = [ones(1,size(test.X,2)); test.X];

% Training set dimensions

m=size(train.X,2);

n=size(train.X,1);

% Train logistic regression classifier using minFunc

options = struct('MaxIter', 100);

% First, we initialize theta to some small random values.

theta = rand(n,1)\*0.001;

% Call minFunc with the logistic\_regression.m file as the objective function.

%

% TODO: Implement batch logistic regression in the logistic\_regression.m file!

%

tic;

theta=minFunc(@logistic\_regression, theta, options, train.X, train.y);

fprintf('Optimization took %f seconds.\n', toc);

% Now, call minFunc again with logistic\_regression\_vec.m as objective.

%

% TODO: Implement batch logistic regression in logistic\_regression\_vec.m using

% MATLAB's vectorization features to speed up your code. Compare the running

% time for your logistic\_regression.m and logistic\_regression\_vec.m implementations.

%

% Uncomment the lines below to run your vectorized code.

theta = rand(n,1)\*0.001;

tic;

theta=minFunc(@logistic\_regression\_vec, theta, options, train.X, train.y);

fprintf('Optimization took %f seconds.\n', toc);

% Print out training accuracy.

tic;

accuracy = binary\_classifier\_accuracy(theta,train.X,train.y);

fprintf('Training accuracy: %2.1f%%\n', 100\*accuracy);

% Print out accuracy on the test set.

accuracy = binary\_classifier\_accuracy(theta,test.X,test.y);

fprintf('Test accuracy: %2.1f%%\n', 100\*accuracy);

**logistic\_regression\_vec.m**

function [f,g] = logistic\_regression\_vec(theta, X,y)

%

% Arguments:

% theta - A column vector containing the parameter values to optimize.

% X - The examples stored in a matrix.

% X(i,j) is the i'th coordinate of the j'th example.

% y - The label for each example. y(j) is the j'th example's label.

%

m=size(X,2);

% initialize objective value and gradient.

f = 0;

g = zeros(size(theta));

lambda=0;

%

% TODO: Compute the logistic regression objective function and gradient

% using vectorized code. (It will be just a few lines of code!)

% Store the objective function value in 'f', and the gradient in 'g'.

%

%%% YOUR CODE HERE %%%

h=1./(1+exp(-theta'\*X));

f = 1./m \* ( -y \* log(h)' - ( 1 - y ) \* log ( 1 - h)' );

g = 1./m\* X \* (h - y)';

end