ECE 805 – Machine Learning

Tutorial 3, Monday, March 30, 2020

Logistic Regression:

Sigmoidal combination of features: $f(x) = \frac{1}{1+e^{-(\theta^T x + \theta_0)}} = \frac{1}{1+e^{-\bar{\theta}^T \bar{x}}}$

$\theta = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^n$ $\quad$ $x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^n$

$\bar{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^n$ $\quad$ $\bar{x} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^n$ $\quad$ => parameters

$$\theta(k+1) = \theta(k) - \alpha\left(\frac{1}{1+e^{-\left(\theta^T(k)x(k)+\theta_0(k)\right)}} - y(k)\right)x(k)$$

$$\theta_0(k+1) = \theta_0(k) - \alpha\left(\frac{1}{1+e^{-\left(\theta^T(k)x(k)+\theta_0(k)\right)}} - y(k)\right), \textit{ where a is the learning rate.}$$

Cost function for Logistic Regression (with optimization):

$$Cost(h_\theta(x), y) = \begin{cases} -\log\left(h_\theta(x)\right), if\ y = 1 \\ -\log\left(1 - h_\theta(x)\right), if\ y = 0 \end{cases} = -y \cdot \log\left(h_\theta(x)\right) - (1-y) \cdot \log\left(1 - h_\theta(x)\right)$$

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

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

To minimize the cost function:

$$repeat\ until\ convergence \left\{ \quad \theta_j = \theta_j - a\frac{\partial J(\theta)}{\partial \theta_j} = \theta_j + \frac{a}{m}\left[\sum_{i=1}^{m}\left(y^{(i)} - h_\theta\left(x^{(i)}\right)\right){x_j}^{(i)}\right] \right\}$$

Multilayer Perceptron with Backpropagation Gradient Descent Algorithm:

Steps:

1. Initialize the weights and the biases of the neurons
2. Present the network with an epoch of training examples applying to each example steps 3-4.
3. **Forward Computation:** Compute the outputs of the neurons y_k starting from the neurons of hidden layer one. The neurons of the next layer are only computed after the output of all neurons of the particular layer have been computed.

   For the hidden layers: $a_j^{(1)} = \sum_i w_{ji}^{(1)} x_i \qquad z_j = h(a_j)$

   For the output: $a_k^{(2)} = \sum_j w_{kj}^{(2)} z_j \qquad y_k = g(a_k)$
4. **Backward Computation:** Compute the local gradients of the network starting from the output layer and going to the input layer using the following equations:

   Output error $=\delta_k = y_k - t_k$ (t-target) (k- # of outputs)

   Hidden neurons error = $\delta_j = h'(a_j) \cdot \sum_{i=0}^{j} w_{jk}\delta_k$ (j - # of hidden neurons)

   i - # of inputs

   Derivatives for hidden neurons: $\frac{\partial E_n}{\partial w_{kj}^{(2)}} = \delta_k z_j$ and the inputs: $\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \delta_j x_i$

   And update the weights of hidden neurons: $w_{ji}^* = w_{ji} - a \cdot \frac{\partial E_n}{\partial w_{ji}^{(1)}}$

   And the weights of output: $w_{kj}^* = w_{kj} - a \cdot \frac{\partial E_n}{\partial w_{kj}^{(2)}}$
5. **Convergence:** Iterate with new epochs for all learning examples until criterion is met.

*Error function: $E_n = \frac{1}{2}\sum_{k=1}^{K}(y_k - t_k)^2$

Confusion Matrix:

	Class 1 *Predicted*	Class 2 *Predicted*
Class 1 *Actual*	True Positive (TP)	False Negative (FN)
Class 2 *Actual*	False Positive (FP)	True Negative (TN)

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} \qquad Recall = Sensitivity = \frac{TP}{TP+FN}$$

$$\text{Precision} = \frac{TP}{TP+FP} \qquad Specificity = \frac{TN}{TN+FP}$$

Examples:

1. Logistic Regression Example 1: Perform logistic regression for a 2-class dataset. We have a dataset with 3 variables and the class. Split the dataset into 75% for training and 25% for testing and by using logistic regression predict the class of the test set. Set the threshold for classifying hypothesis output equal to 0.5. Compute the following performance values: accuracy on training and test set, sensitivity, specificity and the misclassification error on the test set.

```
% Perform logistic regression for a 2-class dataset. Learn regression params with built-in
  MATLAB/Octave unconstrained
% optimization routine, fminunc.
% Overview:
% Take in a csv file (without header), Split into a training and test set, Minimize cost function with
  fminunc, Compute performance metrics: prediction accuracy on training and test sets, confusion
  matrix, specificity, sensitivity, misclassification error
clc;clear all;close all;
%Input must contain feature columns followed by dependent variable column at end
data = load('class_function_01.txt');
%percentage of data to use for training
train_frac = .75;
%threshold for classifying hypothesis output
thresh = 0.5;
%extract columns to use
X = data(:,1:end-1);
y = data(:,end);
%split into training and test sets:
test_rows = round(size(X,1)*(1-train_frac)); %number of rows to use in test set
X_test = X(1:test_rows,:); y_test = y(1:test_rows,:);%this is the test set
X = X(test_rows+1:end,:); y = y(test_rows+1:end,:);%this is the training set
%Add intercept term to X
X = [ones(size(X,1), 1) X];
X_test = [ones(size(X_test,1), 1) X_test];
% Initialize fitting parameters
initial_theta = zeros(size(X,2), 1);
%  Set options for fminunc
options = optimset('GradObj', 'on', 'MaxIter', 400);
%  Run fminunc to obtain the optimal theta
%  fminunc finds a local minimum of a function of several variables.
[theta, cost] = fminunc(@(t)(costFunction(t, X, y)), initial_theta, options);
% Print theta to screen
fprintf('Cost at theta found by fminunc: %f\n', cost);
fprintf('theta: \n');
fprintf(' %f \n', theta);
fprintf('\n**********************\n');
%prediction accuracy
p_train = double(logsig(X*theta) >= thresh);
fprintf('Training set accuracy: %f\n', mean(double(p_train == y)) * 100);

p_test = double(logsig(X_test*theta) >= thresh);
```

```
fprintf('Test set accuracy: %f\n', mean(double(p_test == y_test)) * 100);
%confusion matrix, sensitivity, specificity, misclassification error
cm = confMatrix(y_test,p_test);
sens = cm(1,1) / (cm(1,1) + cm(1,2)); %ability to identify positive class
spec = cm(2,2) / (cm(2,2) + cm(2,1)); %ability to identify negative class
testError = misclassError(y_test,logsig(X_test*theta),thresh); %0/1 misclassification error on test
  set
fprintf('\n**********************\n');
fprintf('Confusion Matrix:\n');disp(cm)
fprintf('Sensitivity: %g\n',sens);
fprintf('Specificity: %g\n',spec);
fprintf('Misclassification error on test set: %g\n',testError);
```

```
function [J, grad] = costFunction(theta, X, y)
%COSTFUNCTION Compute cost and gradient for logistic regression
%   [J, grad] = COSTFUNCTION(theta, X, y) computes the cost of using theta as the
%   parameter for logistic regression and the gradient of the cost
%   w.r.t. to the parameters.
%
%   Code source: ml-class.org
m = length(y); % number of training examples
J = (1/m)*(-y'*log(logsig(X*theta)) - (1-y)'*log(1-logsig(X*theta)));
grad = ((1/m)*(logsig(X*theta) - y)'*X)';
end
function g = confMatrix(T,P)
%CONFMATRIX Compute confusion matrix for 2-class problem
% [cm] = CONFMATRIX(T,P) computes the 2x2 confusion matrix using true class
% (T) and predicted class (P) values
%
%T = true conversion value
%P = predicted conversion value
%T and P must be of same size, and can only take on values of 0 or 1
g = zeros(2);
for p = 1:length(T)
   if T(p) == 0 && P(p) == 0
      g(1,1) = g(1,1)+1;
   end
   if T(p) == 1 && P(p) == 1
      g(2,2) = g(2,2)+1;
   end
   if T(p) == 1 && P(p) == 0
      g(1,2) = g(1,2)+1;
   end
   if T(p) == 0 && P(p) == 1
      g(2,1) = g(2,1)+1;
   end
end
g = g';
```

```
end
```

```
function [testError] = misclassError(y,y_hat,thresh)
%MISCLASSERROR Compute 0/1 misclassification error for a 2-class problem
% misclassError [out] = MISCLASSERROR(in) computes the 0/1
% misclassification error given predicted and actual output.
%
% Input:
% y = actual class
% y_hat = predicted class
% thresh = probability threshold
%

m = size(y,1);

testError = (1/m) * sum(double(y_hat>=thresh & y==0 | y_hat<=thresh & y==1 ));

end
```

2. Logistic Regression Example 2: We are given a data file with students' grades from two courses to determine using logistic regression whether they pass or fail.

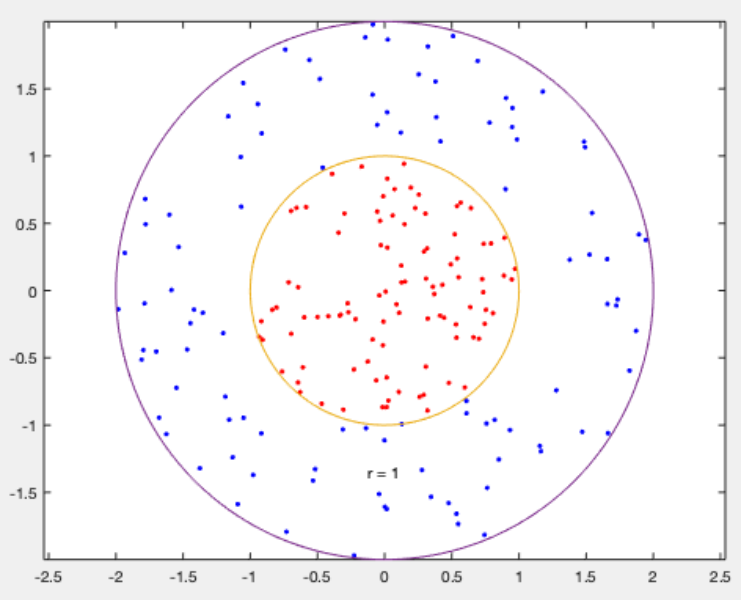
Mark 1	15	20	30	25	26	36	22	24	10	8	33	18	33
Mark 2	20	30	20	25	8	8	35	30	15	60	35	18	17
Class	0	1	1	0	0	0	1	1	0	1	1	0	1

```
clc; close all;
 clear all;
 % Data File for marks of students in two subject to determine whether a student is pass or fail
x=xlsread('marks.xlsx');
 ytrain=x(:,end); % Target variable
 xtrain=zscore(x(:,1:end-1));% Normalized Predictors
p=length(x);
xtrain=[ones(length(xtrain),1) xtrain]; % one is added for calculation of biases.
 xtest=xtrain;
 ytest=ytrain;
%compute cost and gradient
 iter=1000; % No. of iterations for weight updation
theta=zeros(size(xtrain,2),1); % Initial weights
alpha=0.1 % Learning parameter
[J grad h th]=cost(theta,xtrain,ytrain,alpha,iter) % Cost funtion
ypred=xtest*th; %target prediction
% probability calculation
 [hp]=logsig(ypred); % Hypothesis Function
 ypred(hp>=0.5)=1;
 ypred(hp<0.5)=0;
% Decision Boundary
 syms x1 x2 % Short-cut for constructing symbolic variables.
 fnn=th(1)+th(2)*x1+th(3)*x2-0.5
figure
 hold on
 scatter(xtest(ytest==1,2),xtest(ytest==1,3),'b+','linewidth',5.0)
 scatter(xtest(ytest==0,2),xtest(ytest==0,3),'r','linewidth',5.0)
 h1=ezplot(fnn)
 set(h1,'color','r')
 legend('+ class','- class','Decision boundary')
```

```
function [J grad h th] = cost(theta, xtrain,ytrain,alpha,iter)
%Cost Summary of this function goes here
%   Detailed explanation goes here
th=theta
m=size(xtrain,1); % length of training input data
for j=1:iter
h=logsig(xtrain*th); % sigmoid function
J=-(1/m)*sum(ytrain.*log(h)+(1-ytrain).*log(1-h)); % cost function
th=th+(alpha/m)*xtrain'*(ytrain-h) % update the weights
```

```
end
clear i k;
k=length(theta);
grad=zeros(k,1);

for i=1:k
grad(i)=(1/m)*sum((h-ytrain)'*xtrain(:,i)); % gradient calculation
end
end
```

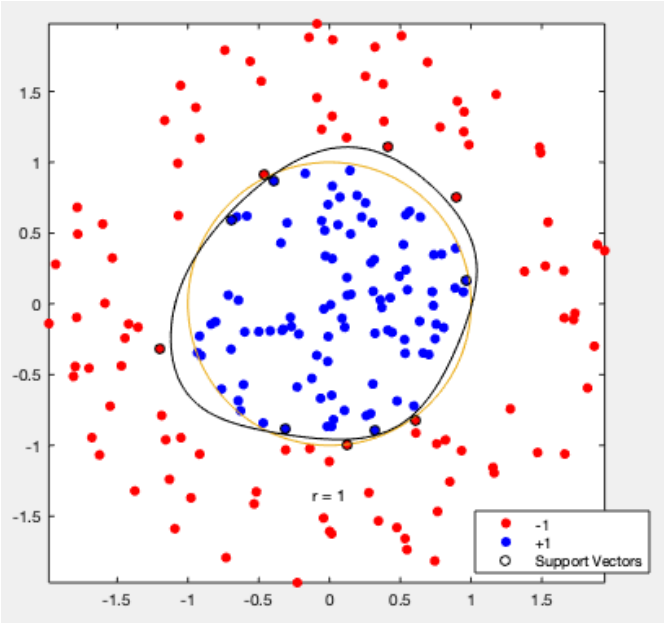
3. Support Vector Machines (SVMs):
   Generate a nonlinear classifier with Gaussian kernel function by using the datasvm.mat file provided. Set the box constraint parameter to Inf to make a strict classification.



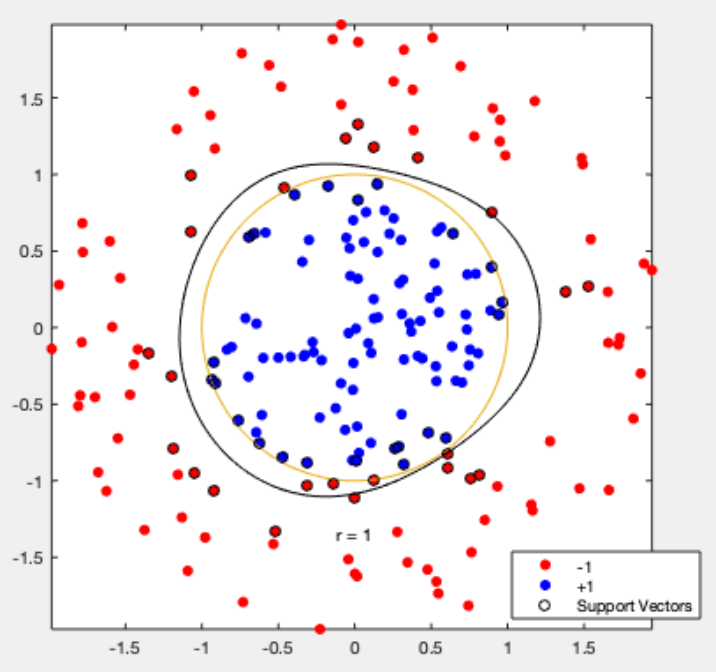

```
%% Train SVM Classifiers Using a Gaussian Kernel
% Plot the points, and plot circles of radii 1 and 2 for comparison.
load ('datasvm.mat')

figure;
plot(data1(:,1),data1(:,2),'r.')
hold on
plot(data2(:,1),data2(:,2),'b.')
axis equal
ezpolar(@(x)1);ezpolar(@(x)2);
hold off
%%
% Put the data in one matrix, and make a vector of classifications.

data3 = [data1;data2];
theclass = [class1;class2];
% theclass = ones(200,1);
% theclass(1:100) = -1;
%%
% Train an SVM classifier with |KernelFunction| set to |'rbf'| and |BoxConstraint|
% set to |Inf|. Plot the decision boundary and flag the support vectors.

%Train the SVM Classifier
cl = fitcsvm(data3,theclass,'KernelFunction','rbf','BoxConstraint',Inf,'ClassNames',[-1,1]);

% Predict scores over the grid
d = 0.02;
```

```
[x1Grid,x2Grid] =
meshgrid(min(data3(:,1)):d:max(data3(:,1)),min(data3(:,2)):d:max(data3(:,2)));
xGrid = [x1Grid(:),x2Grid(:)];
[~,scores] = predict(cl,xGrid);

% Plot the data and the decision boundary
figure;
h(1:2) = gscatter(data3(:,1),data3(:,2),theclass,'rb','.');
hold on
ezpolar(@(x)1);
h(3) = plot(data3(cl.IsSupportVector,1),data3(cl.IsSupportVector,2),'ko');
contour(x1Grid,x2Grid,reshape(scores(:,2),size(x1Grid)),[0 0],'k');
legend(h,{'-1','+1','Support Vectors'});
axis equal
hold off
%%
% |fitcsvm| generates a classifier that is close to a circle of radius 1. The
% difference is due to the random training data.
%%
% Training with the default parameters makes a more nearly circular classification
% boundary, but one that misclassifies some training data. Also, the default value
% of |BoxConstraint| is |1|, and, therefore, there are more support vectors.

cl2 = fitcsvm(data3,theclass,'KernelFunction','rbf');
[~,scores2] = predict(cl2,xGrid);

figure;
h(1:2) = gscatter(data3(:,1),data3(:,2),theclass,'rb','.');
hold on
ezpolar(@(x)1);
h(3) = plot(data3(cl2.IsSupportVector,1),data3(cl2.IsSupportVector,2),'ko');
contour(x1Grid,x2Grid,reshape(scores2(:,2),size(x1Grid)),[0 0],'k');
legend(h,{'-1','+1','Support Vectors'});
axis equal
hold off
```

Boxconstraint = Inf



Boxconstraint = 1



This example shows how to generate a nonlinear classifier with Gaussian kernel function. The default linear classifier is obviously unsuitable for this problem, since the model is circularly symmetric. Set the box constraint parameter to Inf to make a strict classification, meaning no misclassified training points. Other kernel functions might not work with this strict box constraint, since they might be unable to provide a strict classification. Even though the RBF classifier can separate the classes, the result can be overtrained.

4. Multilayer Perceptron with Backpropagation Gradient Descent Algorithm:

By using the Leukemia Cancer Database given in the file and by performing the Multilayer Perceptron with Backpropagation Gradient Descent Algorithm for prediction, give the training and testing accuracy. The dataset is divided into training and test set by default. At the beginning use 5 features from the 1000 provided in the dataset, and then try different values of features to see how this affects the training and testing accuracy. Choose some random values for the initial values of the weights.

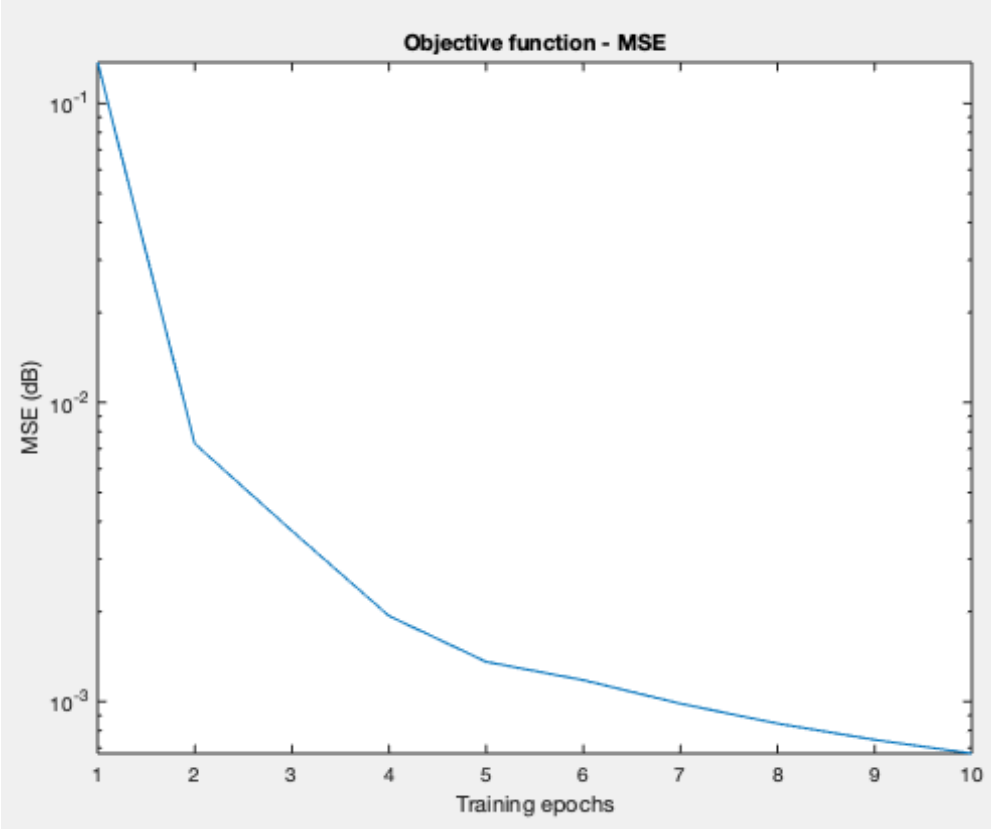
Learning rate = 0.5
Epochs = 10
Activation functions: 1^{st} layer: $h(a) = tansig(a) = \frac{2}{1+e^{-2n}} - 1$
2^{nd} layer: $g(a) = logsig(a) = sigmoid = \frac{1}{1+e^{-n}}$

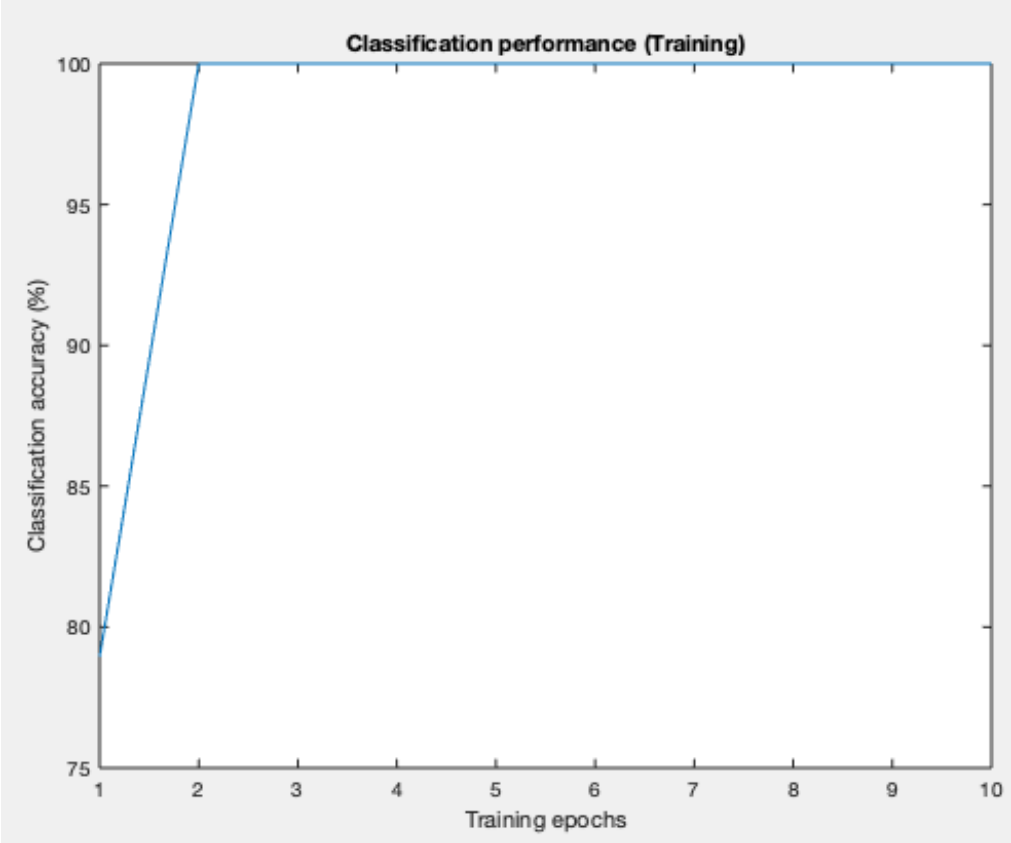
```
%% Backpropagation for MultiLayer Perceptron Neural Networks
% Description
% In this simulation I used a Golub et al(1999)'s Leukemia Cancer Database.
% The details of the dataset is available online at [1].  The leukemia db
% is a gene expression dataset contains 7128 genes, 2-classes (47-ALL &
% 25-AML), divided into two subsets training and test subsets. The training
% dataset contains 27-ALL, and 11-AML total 38 samples, and the test subset
% contains 20-ALL, and 14-AML total 34 samples.
%
% The genes are ranked using mRMR feature selection method [2] and the
% index of top 1000 genes is stored in 'feature_with_mRMr_d' vector.

% [1]
http://www.stats.uwo.ca/faculty/aim/2015/9850/microarrays/FitMArray/chm/Golub.html
% [2] https://kr.mathworks.com/matlabcentral/fileexchange/14608-mrmr-feature-selection--
using-mutual-information-computation-

%% Start
clc; clear all; close all;

%% Load Golub et al. Leukemia Cancer DB
load leukemia_dataset

feature_length=5;  % selecting top 5 most significant genes
train_data=train_data(feature_with_mRMr_d(1:feature_length),:)';
test_data=test_data(feature_with_mRMr_d(1:feature_length),:)';

%% Neural Network's parameters
% (size/structure of the MLP)
N1=20;              % Middle Layer Neurons
N2=1;               % Output Layer Neurons
N0=feature_length+1;    % Input Layer Neurons (feature length + bias)

% Training parameters
eta = 0.5; % Learning Rate 0-1   eg .01, .05, .005
```

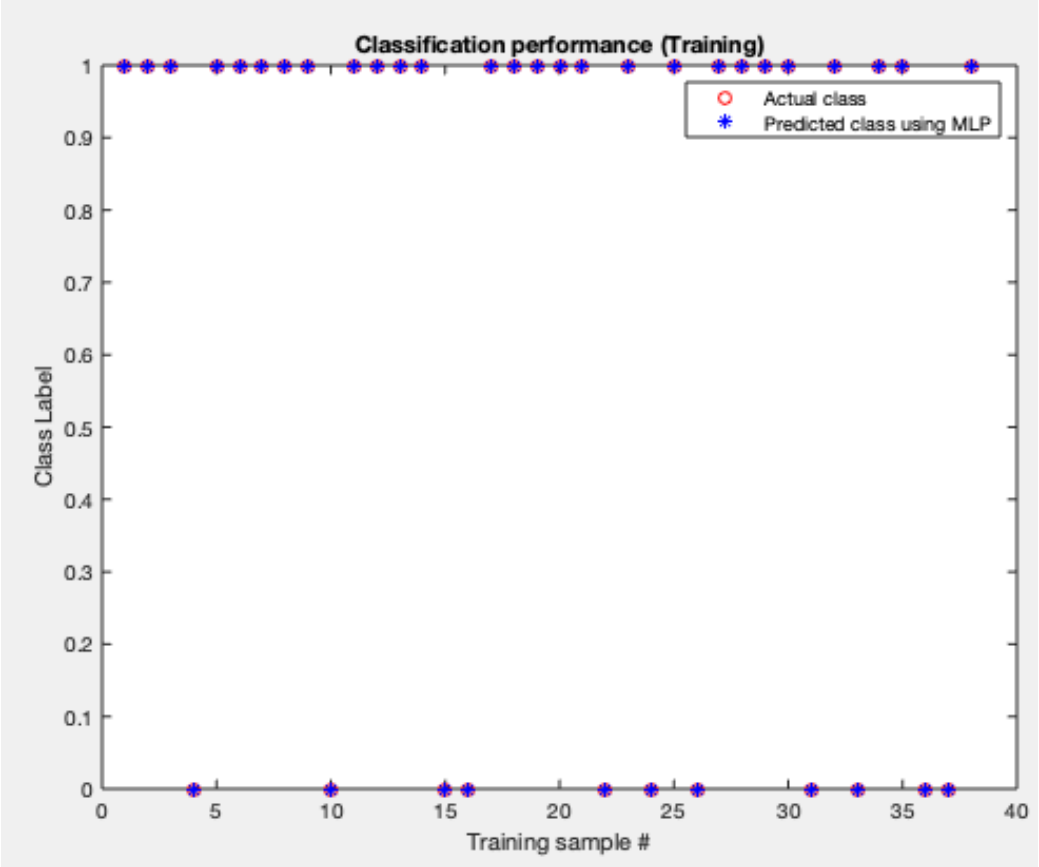
```
epoch=10;  % Training iterations

% Initialization of weights
w1=randn(N1,N0);    % Initial weights of input and middle layer connections
w2=randn(N2,N1);    % Initial weights of middle and output layer connections

for j=1:epoch
   % randomization of training data improves learning performance
   ind(j,:)=randperm(length(class_train));

   for k=1:size(train_data,1)

      Input=[1 train_data(ind(j,k),:)];  % {1} is for bias

      % Input layer
      n1 = w1*Input';
      a1=tansig(n1);

      % Hidden layer
      n2 = w2*a1;
      a2=logsig(n2);

      % output layer
      Output(k)=a2;
      e = class_train(ind(j,k)) - Output(k); % error

      % Training of NN using Backpropagation learning algorithm (gradient
      % descent-based learning rule)

      Y2 = 2*dlogsig(n2,a2)*e;  % local gradient of Output Layer
      Y1 = diag(dtansig(n1,a1),0)*w2'*Y2; % local gradient of Hidden Layer

      w1 = w1 + eta*Y1*Input;  % input layer neurons weight update
      w2 = w2 + eta*Y2*a1';    % hidden layer neurons weight update

      SE(j,k)= e*e';     % squared error
end
MSE(j)=mean(SE(j,:));      % objective function (mean squared error)

% Training classification error (classification accuracy in %)
TCE(j)=length(find((round(Output)-class_train(ind(j,:)))==0))*100/length(Output);
end

figure
semilogy(MSE)
xlabel('Training epochs')
ylabel('MSE (dB)')
title('Objective function - MSE')
```

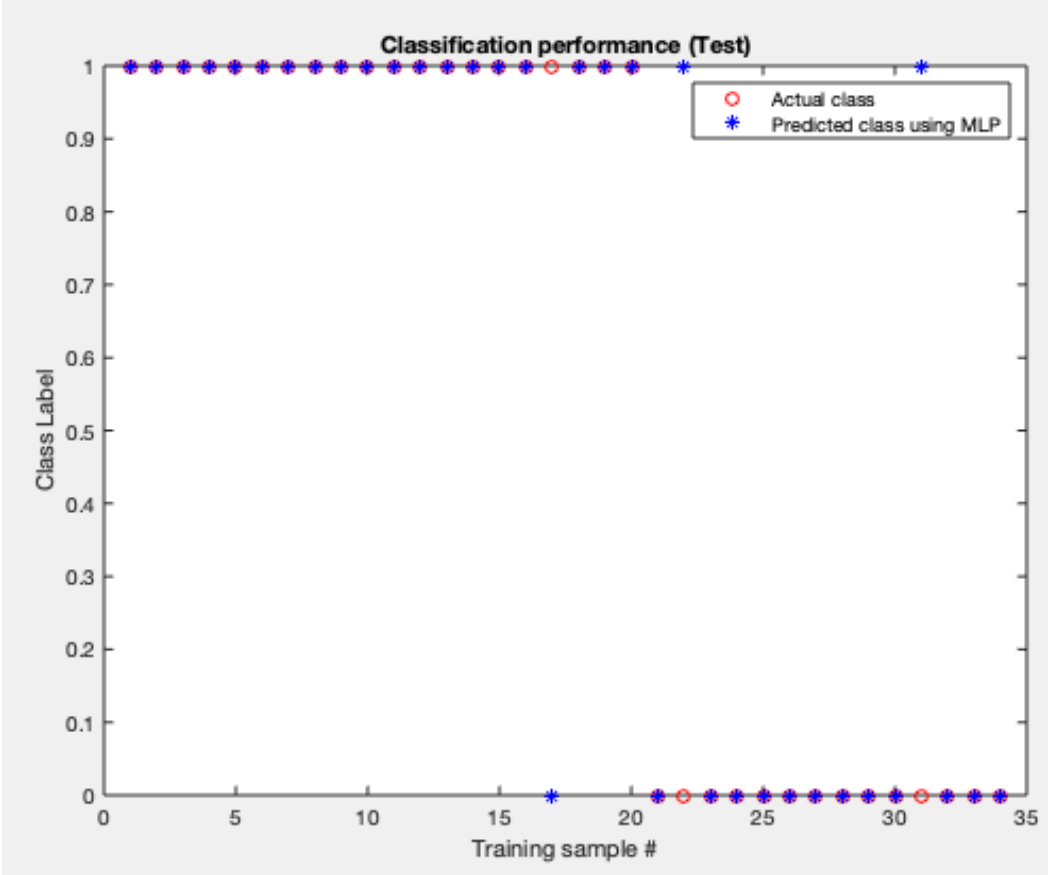
```
figure
plot(TCE)
xlabel('Training epochs')
ylabel('Classification accuracy (%)')
title('Classification performance (Training)')

figure
plot(class_train(ind(j,:)),'or')
hold on
plot(round(Output),'*b')

legend('Actual class','Predicted class using MLP')
xlabel('Training sample #')
ylabel('Class Label')
title('Classification performance (Training)')

Training_Accuracy=length(find((round(Output)-class_train(ind(j,:)))==0))*100/length(Output);

Output=[];

for k=1:size(test_data,1)

    Input=[1 test_data(k,:)];

    n1 = w1*Input';
    a1=tansig(n1);   %%%% Hidden Layer Activation Function  % tansig for [-1,+1] % logsig for
[0,1]
%
    n2 = w2*a1;
    a2=logsig(n2);   %%%% Output Layer Activation Function

    Output(k)=a2;
end

figure
plot(class_test,'or')
hold on
plot(round(Output),'*b')
legend('Actual class','Predicted class using MLP')
xlabel('Training sample #')
ylabel('Class Label')
title('Classification performance (Test)')

Testing_Accuracy=length(find((round(Output)-class_test)==0))*100/length(Output);

fprintf('\nTraining Accuracy: %.4f',Training_Accuracy);
fprintf('\nTesting Accuracy: %.4f',Testing_Accuracy);
```

Training Accuracy: 100.000000
Testing Accuracy: 91.176471






Classification performance (Training)
Actual class
Predicted class using MLP
Class Label
Training sample #
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
5
10
15
20
25
30
35
40


Classification performance (Test)
Actual class
Predicted class using MLP
Class Label
Training sample #
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
5
10
15
20
25
30
35