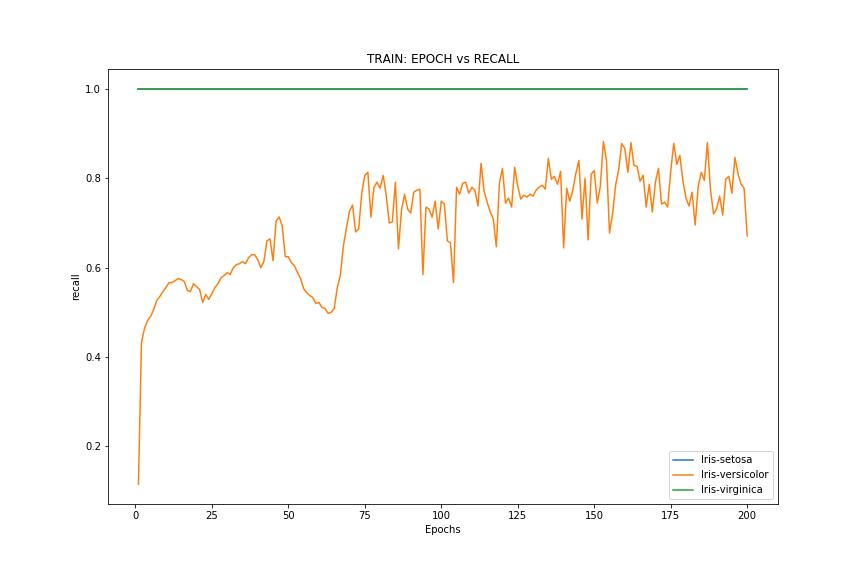
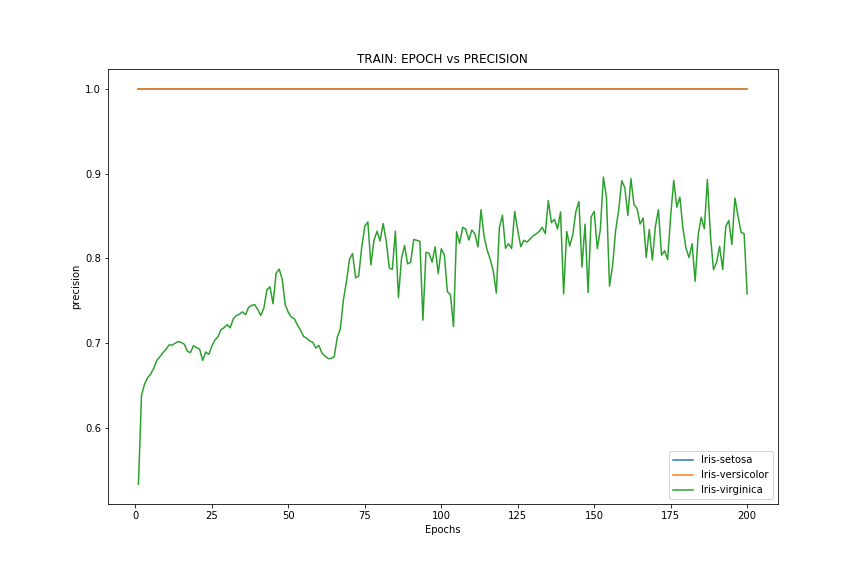
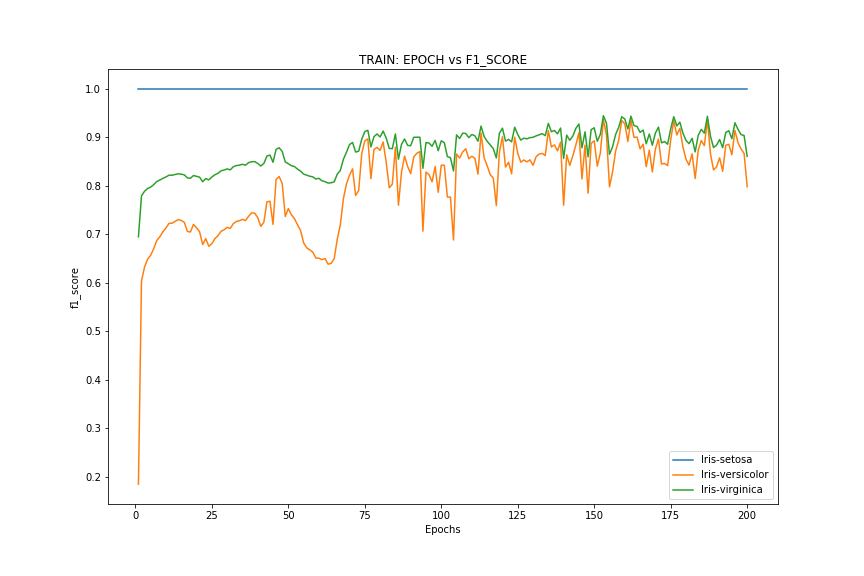
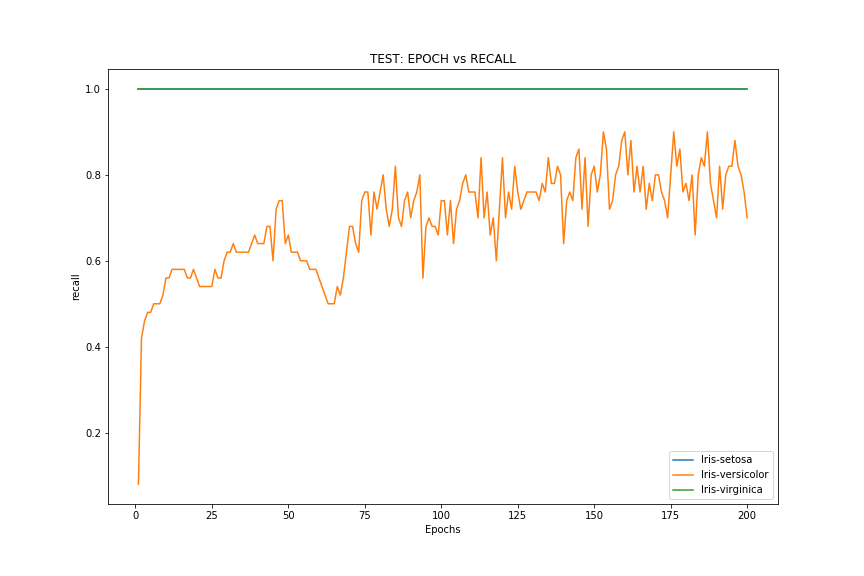
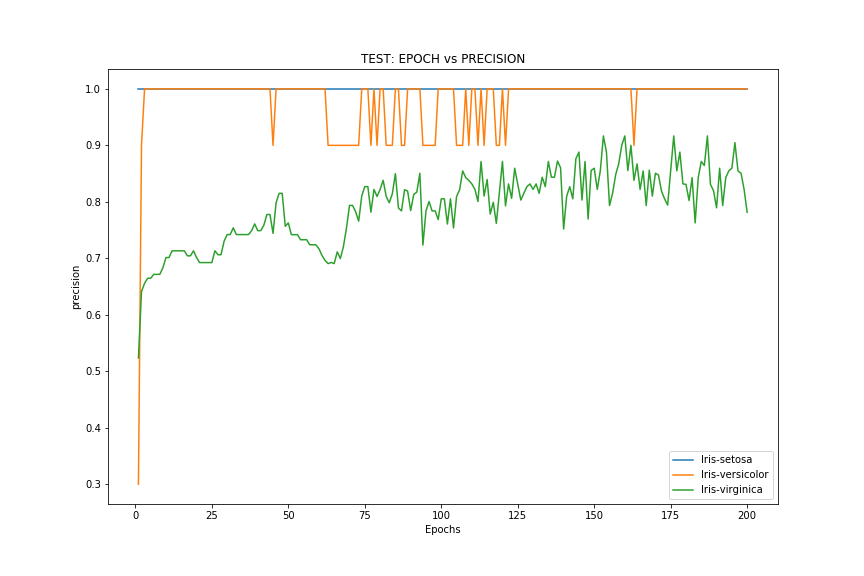
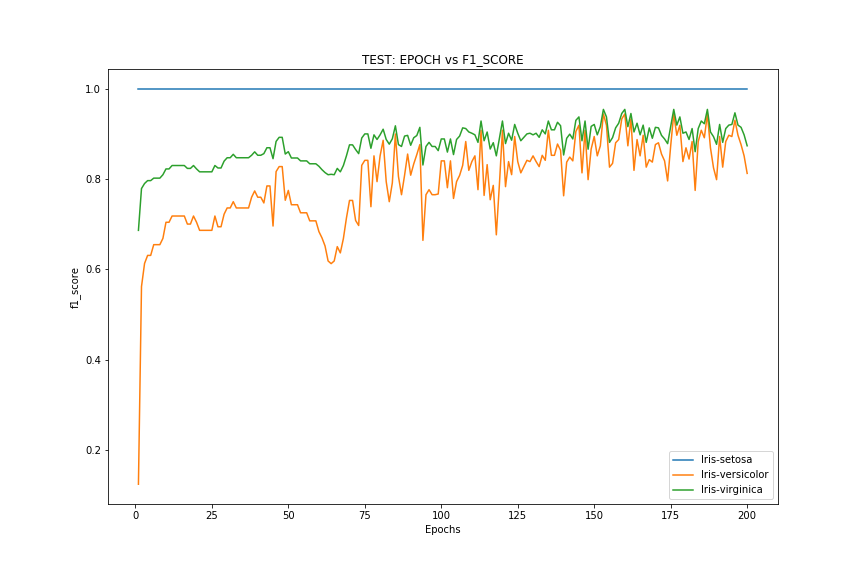
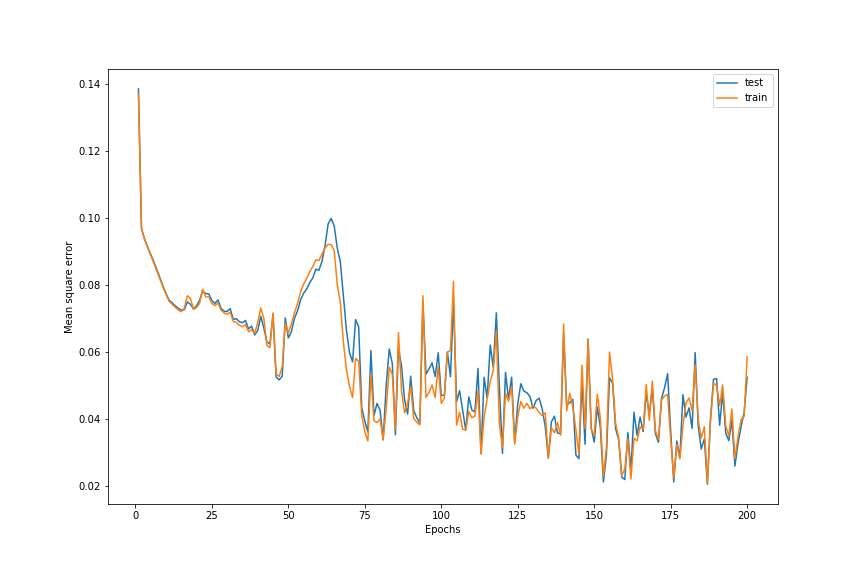
* **Design of 5 different ANN:**
  + ANN with 1 hidden layer:
    - **Layer of Nodes: 4, 15, 3**
  + ANN with 3 hidden layer:
  + ANN with 5 hidden layer:
  + ANN with 6 hidden layer:
  + ANN with 7 hidden layer:
* **Tables**
* **Plots**
  + Number of Hidden layer = 1****
  + Number of Hidden layer = 3
  + Number of Hidden layer = 5
  + Number of Hidden layer = 6
  + Number of Hidden layer = 7
* **Program Code:**
  + The code segment is divided into 7 functions.
    - evaluate\_ann\_model.m
    - cross\_validate.m
    - fit.m
    - predict.m
    - get\_error.m
    - get\_class\_label.m
    - get\_f1\_score.m

# evaluate\_ann\_model.m

***% function name: evaluate\_ann\_model***

***% input:***

***% n\_hidden\_layer = number of hidden layer***

***% epoch = number of iterations***

***% cv = cross validation fold number***

***% Output:***

***% report = Matrix containg output traing and test mean square error, precision, recall, f1 score.***

***function [ report ] = evaluate\_ann\_model( n\_hidden\_layer, epoch, cv)***

***%EVALUATE\_ANN\_MODEL Summary of this function goes here***

***% Detailed explanation goes here***

***FID = fopen('Data\_and\_Info/iris.data.txt');***

***C\_data0 = textscan(FID,'%f %f %f %f %s', 200, 'Delimiter',',');***

***X = cell2mat(C\_data0(:,1:4)); %ignores the last column of strings***

***[Nx,P]=size(X); % // Nx = # of sample in X, P= # of feature in X***

***target = C\_data0(:,5);***

***class\_values = target{1};***

***Y = ones(length(class\_values), 3);***

***[Ny, K]=size(Y); % // Ny = # of target output in Y, K= # of class for K classes when K>=3 otherwise, K=1 (for Binary case)***

***for i = 1: Ny***

***if strcmp(class\_values{i}, 'Iris-setosa')***

***Y(i, :) = [1 0 0];***

***end***

***if strcmp(class\_values{i}, 'Iris-versicolor')***

***Y(i, :) = [0 1 0];***

***end***

***if strcmp(class\_values{i}, 'Iris-virginica')***

***Y(i, :) = [0 0 1];***

***end***

***end***

***cv\_test\_err\_list = zeros(epoch, 1);***

***cv\_train\_err\_list = zeros(epoch, 1);***

***cv\_test\_precision\_list = zeros(epoch, K);***

***cv\_test\_recall\_list = zeros(epoch, K);***

***cv\_test\_f1\_score\_list = zeros(epoch, K);***

***cv\_train\_precision\_list = zeros(epoch, K);***

***cv\_train\_recall\_list = zeros(epoch, K);***

***cv\_train\_f1\_score\_list = zeros(epoch, K);***

***n\_layer = n\_hidden\_layer + 2;***

***L = zeros(n\_layer, 1); % initiate L, layer arrays of neural network***

***L(1) = P;***

***L(end) = K;***

***for i = 2: n\_layer - 1***

***L(i) = randi([2, 20]);***

***end***

***folds\_list = zeros(epoch, 1);***

***for i = 1: epoch***

***[ cv\_test\_err, cv\_train\_err, cv\_test\_precision, cv\_test\_recall, cv\_test\_f1\_score, cv\_train\_precision, cv\_train\_recall, cv\_train\_f1\_score] = cross\_validate( X, Y, cv, L);***

***cv\_test\_err\_list(i) = cv\_test\_err;***

***cv\_train\_err\_list(i) = cv\_train\_err;***

***cv\_test\_precision\_list(i, :) = cv\_test\_precision;***

***cv\_test\_recall\_list(i, :) = cv\_test\_recall;***

***cv\_test\_f1\_score\_list(i, :) = cv\_test\_f1\_score;***

***cv\_train\_precision\_list(i, :) = cv\_train\_precision;***

***cv\_train\_recall\_list(i, :) = cv\_train\_recall;***

***cv\_train\_f1\_score\_list(i, :) = cv\_train\_f1\_score;***

***folds\_list(i) = i;***

***end***

***report\_filename = strcat('report\_nh\_', int2str(n\_hidden\_layer), '.csv');***

***report = [folds\_list, cv\_test\_err\_list, cv\_train\_err\_list, cv\_test\_precision\_list, cv\_test\_recall\_list, cv\_test\_f1\_score\_list, cv\_train\_precision\_list, cv\_train\_recall\_list, cv\_train\_f1\_score\_list];***

***csvwrite(report\_filename,report);***

***layer\_filename = strcat('layer\_nh\_', int2str(n\_hidden\_layer), '.csv');***

***csvwrite(layer\_filename,L);***

***end***

# cross\_validate.m

***% function name: cross\_validate***

***% Input:***

***% X = feature file,***

***% Y = target,***

***% cv = cross validation fold number,***

***% L = Layer array***

***% Output:***

***% cv\_test\_err,***

***% cv\_train\_err,***

***% cv\_test\_precision,***

***% cv\_test\_recall,***

***% cv\_test\_f1\_score,***

***% cv\_train\_precision,***

***% cv\_train\_recall,***

***% cv\_train\_f1\_score***

***function [ cv\_test\_err, cv\_train\_err, cv\_test\_precision, cv\_test\_recall, cv\_test\_f1\_score, cv\_train\_precision, cv\_train\_recall, cv\_train\_f1\_score] = cross\_validate( X, Y, cv, L)***

***%CROSS\_VALIDATE Summary of this function goes here***

***% Detailed explanation goes here***

***alpha = 0.2; % //usually alpha < 0, ranging from 0.1 to 1***

***target\_mse=0.05 % // one of the exit condition***

***Max\_Epoch=2000 % // one of the exit condition***

***Min\_Error=Inf***

***Min\_Error\_Epoch=-1***

***[ny, n\_class] = size(Y);***

***y\_indices = (1: ny);***

***CVO = cvpartition(y\_indices,'k',cv);***

***test\_err = zeros(CVO.NumTestSets,1);***

***test\_precision = zeros(CVO.NumTestSets,n\_class);***

***test\_recall = zeros(CVO.NumTestSets,n\_class);***

***test\_f1\_score = zeros(CVO.NumTestSets,n\_class);***

***precision\_train\_list = zeros(CVO.NumTestSets,n\_class);***

***recall\_train\_list = zeros(CVO.NumTestSets,n\_class);***

***f1\_score\_train\_list = zeros(CVO.NumTestSets,n\_class);***

***train\_err = zeros(CVO.NumTestSets,1);***

***for i = 1:CVO.NumTestSets***

***trIdx = CVO.training(i);***

***teIdx = CVO.test(i);***

***X\_train = X(trIdx, :);***

***X\_test = X(teIdx, :);***

***Y\_train = Y(trIdx, :);***

***Y\_test = Y(teIdx, :);***

***[Err, Epo, B\_min\_error] = fit(L,alpha,target\_mse, Max\_Epoch, Min\_Error, Min\_Error\_Epoch, X\_train, Y\_train);***

***[ Y\_pred\_train ] = predict( L, X\_train, B\_min\_error);***

***[ Y\_pred ] = predict( L, X\_test, B\_min\_error);***

***[ avg\_mse\_error\_rate\_train, precision\_train, recall\_train, f1\_score\_train ] = get\_error( Y\_pred\_train, Y\_train);***

***[ avg\_mse\_error\_rate, precision, recall, f1\_score ] = get\_error( Y\_pred, Y\_test);***

***test\_err(i) = avg\_mse\_error\_rate;***

***train\_err(i) = avg\_mse\_error\_rate\_train;***

***test\_precision(i, :) = precision;***

***test\_recall(i, :) = recall;***

***test\_f1\_score(i, :) = f1\_score;***

***precision\_train\_list(i, :) = precision\_train;***

***recall\_train\_list(i, :) = recall\_train;***

***f1\_score\_train\_list(i, :) = f1\_score\_train;***

***end***

***cv\_test\_err = sum(test\_err)/CVO.NumTestSets;***

***cv\_train\_err = sum(train\_err)/CVO.NumTestSets;***

***cv\_test\_precision = sum(test\_precision)/CVO.NumTestSets;***

***cv\_test\_recall = sum(test\_recall)/CVO.NumTestSets;***

***cv\_test\_f1\_score = sum(test\_f1\_score)/CVO.NumTestSets;***

***cv\_train\_precision = sum(precision\_train\_list)/ CVO.NumTestSets;***

***cv\_train\_recall = sum(recall\_train\_list)/ CVO.NumTestSets;***

***cv\_train\_f1\_score = sum(f1\_score\_train\_list)/ CVO.NumTestSets;***

***end***

# fit.m

***% function name: fit***

***% Input:***

***% L = Layer array,***

***% alpha = learning rate,***

***% target\_mse = minimum mse threshold,***

***% Max\_Epoch,***

***% Min\_Error,***

***% Min\_Error\_Epoch,***

***% X = feature,***

***% Y = target***

***% Output:***

***% Err = Error***

***% Epo = Epoch***

***% B\_min\_error = Weight for minimum error.***

***function [Err, Epo, B\_min\_error] = fit(L,alpha,target\_mse, Max\_Epoch, Min\_Error, Min\_Error\_Epoch, X, Y)***

***% L=[4 4 4 3]; % // Defining the layers: Total of 4 layers, # of nodes are 2, 4, 4, 1 respectively from input to output layer***

***% alpha = 0.2; % //usually alpha < 0, ranging from 0.1 to 1***

***% target\_mse=0.05 % // one of the exit condition***

***% Max\_Epoch=10 % // one of the exit condition***

***% Min\_Error=Inf***

***% Min\_Error\_Epoch=-1***

***epoch=0; % // 1 epoch => One forward and backward sweep of the net for each training sample***

***mse =Inf; % // initializing the Mean Squared Error with a very large value.***

***Err=[];***

***Epo=[];***

***[Nx,P]=size(X); % // Nx = # of sample in X, P= # of feature in X***

***[Ny, K]=size(Y); % // Ny = # of target output in Y, K= # of class for K classes when K>=3 otherwise, K=1 (for Binary case)***

***% Optional: Since input and output are kept in different files, it is better to verify the loaded sample size/dimensions.***

***if Nx ~= Ny***

***error ('The input/output sample sizes do not match');***

***end***

***% Optional***

***if L(1) ~= P***

***error ('The number of input nodes must be equal to the size of the features')***

***end***

***% Optional***

***if L(end) ~= K***

***error ('The number of output node should be equal to K')***

***end***

***%Let us allocate places for Term, T***

***T=cell(length(L),1);***

***for i=1:length(L)***

***T{i} =ones (L(i),1);***

***end***

***%Let us allocate places for activation, i.e., Z***

***Z=cell(length(L),1);***

***Z{1} = zeros (L(1) + 1,1);***

***for i=2:length(L)-1***

***Z{i} =zeros (L(i),1); % it does not matter how do we initialize (with '0' or '1', or whatever,) this is fine!***

***end***

***Z{end} =zeros (L(end),1); % at the final layer there is no Bias unit***

***B=cell(length(L)-1,1); % forming the number of Beta/weight matrix needed in between the layers***

***for i=1:length(L)-1***

***B{i} =rand(L(i + 1), L(i) + 1)'; % populate initial weight***

***end***

***B\_min\_error = B;***

***%Let us allocate places for error term delta, d***

***d=cell(length(L),1);***

***for i=1:length(L)***

***d{i} =zeros(L(i),1);***

***end***

***while (mse > target\_mse) && (epoch < Max\_Epoch) % outer loop with exit conditions***

***CSqErr=0; % //Cumulative Sq Err of each Sample; we will take the average after computing Nx\_th sample (=> mse)***

***for j=1:Nx % // for loop #1***

***Z{1} = [X(j,:) 1]'; % // Load Inputs with bias=1***

***Yk = Y(j,:)'; % // Load Corresponding Desired or Target output***

***% forward propagation***

***% ----------------------***

***for i=1:length(L)-1***

***T{i+1} = B{i}' \* Z{i};***

***if (i+1)<length(L)***

***Z{i+1}=[(1./(1+exp(-T{i+1}))) ;1];***

***else***

***Z{i+1}=(1./(1+exp(-T{i+1})));***

***end***

***end % // end of forward propagation***

***CSqErr= CSqErr+sum((Yk-Z{end}).^2); % // collect sample wise Cumulative Sq Err***

***% // Compute error term delta 'd' for each of the node except the input unit***

***% -----------------------------------------------------------------------***

***d{end}=(Z{end}-Yk).\*Z{end}.\*(1-Z{end}); % // delta error term for the output layer***

***for i=length(L)-1:-1:2***

***d{i}=Z{i}(1:end-1).\*(1-Z{i}(1:end-1)).\*sum(d{i+1}'\*B{i}(1:end-1,:)'); % //compute the error term for all the hidden layer (and skip the input layer).***

***end***

***% Now we will update the parameters/weights***

***for i=1:length(L)-1***

***B{i}(1:end-1,:)=B{i}(1:end-1,:)-alpha.\*(Z{i}(1:end-1)\*d{i+1}');***

***B{i}(end,:)=B{i}(end,:)-alpha.\*d{i+1}'; % // update weight connected to the bias unit(or, intercept)***

***end***

***end % //end of for loop #1***

***CSqErr= (CSqErr) /(Nx); % //Average error of N sample after an epoch***

***mse=CSqErr***

***epoch = epoch+1***

***Err = [Err mse];***

***Epo = [Epo epoch];***

***if mse < Min\_Error***

***Min\_Error=mse;***

***Min\_Error\_Epoch=epoch;***

***B\_min\_error = B;***

***end***

***end % //while\_end***

***Min\_Error***

***Min\_Error\_Epoch***

***end %end of method fit***

#predict.m

***function [ Y\_pred ] = predict( L, X, B)***

***%PREDICT Summary of this function goes here***

***% Detailed explanation goes here***

***%***

***%***

***% Y\_pred = Predicted output***

***% X = feature***

***% B = Weights***

***%***

***% %// ================================================================================================================***

***% % ///////////////////////////////////////////////// Predict output based on learned weight ////////////////////////////////////////////////***

***% % // Here I will be using the last B computed to demo test data to classify but you should save and use best B.***

***% % // Feed forward part will actually be used, assume test points: 1.(0.5,0.3) and 2.(5,4)***

***% % // NOTE: For point (1) the output is expected to be close to zero***

***% % // For point (2) the output is expected to be close to one.***

***%***

***%***

***[Nx,P]=size(X); % // Nx = # of sample in X, P= # of feature in X***

***Y\_pred = zeros(Nx, L(end));***

***%Let us allocate places for Term, T***

***T=cell(length(L),1);***

***for i=1:length(L)***

***T{i} =ones (L(i),1);***

***end***

***%Let us allocate places for activation, i.e., Z***

***Z=cell(length(L),1);***

***Z{1} = zeros (L(1) + 1,1);***

***for i=2:length(L)-1***

***Z{i} =zeros (L(i),1); % it does not matter how do we initialize (with '0' or '1', or whatever,) this is fine!***

***end***

***Z{end} =zeros (L(end),1); % at the final layer there is no Bias unit***

***%// ====== Same (or similar) code as we used before for feed-forward part (see above)***

***for j=1:Nx % for loop #1***

***Z{1} = [X(j,:) 1]'; % Load Inputs with bias=1***

***%%% //(Note: desired output here) ..... Yk = Y(j,:)'; % Load Corresponding Desired or Target output***

***% // forward propagation***

***% //----------------------***

***for i=1:length(L)-1***

***T{i+1} = B{i}' \* Z{i};***

***if (i+1)<length(L)***

***Z{i+1}=[(1./(1+exp(-T{i+1}))) ;1];***

***else***

***Z{i+1}=(1./(1+exp(-T{i+1})));***

***end***

***end % //end of forward propagation***

***Y\_pred(j, :) = Z{end};***

***end***

***end***

# get\_error.m

***function [ avg\_mse\_error\_rate, precision, recall, f1\_score] = get\_error( Y\_pred, Y\_actual )***

***%GET\_ Summary of this function goes here***

***% Detailed explanation goes here***

***Y\_pred\_class = get\_class\_label( Y\_pred )***

***avg\_mse\_error\_rate = mean(mean((Y\_actual - Y\_pred).^2, 2))***

***[ precision, recall, f1\_score] = get\_f1\_score( Y\_pred\_class, Y\_actual );***

***end***

# get\_class\_label.m

***function [ Y\_class ] = get\_class\_label( Y )***

***%GET\_CLASS\_LABEL Summary of this function goes here***

***% Detailed explanation goes here***

***[ny, cn] = size(Y)***

***Y\_class = zeros(ny, cn)***

***for i = 1:ny***

***ind = find ( Y(i,:) == max(Y(i, :)))***

***Y\_class(i, ind) = 1***

***end***

***end***

# get\_f1\_score.m

***function [ precision, recall, f1\_score] = get\_f1\_score( Y\_pred, Y\_act )***

***%GET\_F1\_SCORE Summary of this function goes here***

***% Detailed explanation goes here***

***[ny, nclass] = size(Y\_pred);***

***confusion\_matrix = zeros(nclass, nclass);***

***for i = 1: ny***

***row = find(Y\_act(i, :), 1);***

***col = find(Y\_pred(i, :), 1);***

***confusion\_matrix(row, col) = confusion\_matrix(row, col) + 1;***

***end***

***precision = zeros(nclass, 1);***

***recall = zeros(nclass, 1);***

***f1\_score = zeros(nclass, 1);***

***for i = 1: nclass***

***if sum(confusion\_matrix(:, i)) == 0***

***precision(i) = 0;***

***else***

***precision(i) = confusion\_matrix(i, i) / sum(confusion\_matrix(:, i));***

***end***

***if sum(confusion\_matrix(i, :)) == 0***

***recall(i) = 0;***

***else***

***recall(i) = confusion\_matrix(i, i) / sum(confusion\_matrix(i, :));***

***end***

***if (precision(i) + recall(i)) == 0***

***f1\_score(i) = 0;***

***else***

***f1\_score(i) = 2 \* precision(i) \* recall(i) /(precision(i) + recall(i));***

***end***

***end***

***end***