

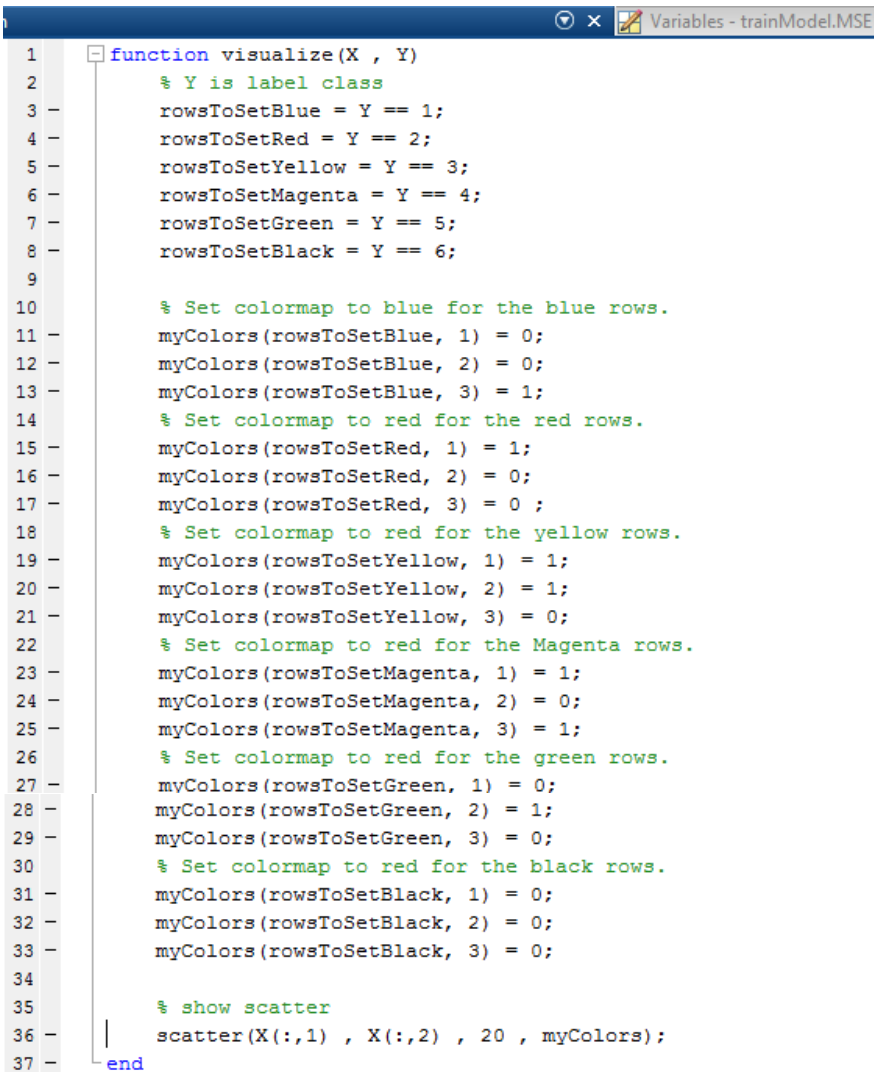
MULTI LAYER PERCEPTRON

1. Load and Visualize the Data

- MLPMain.m

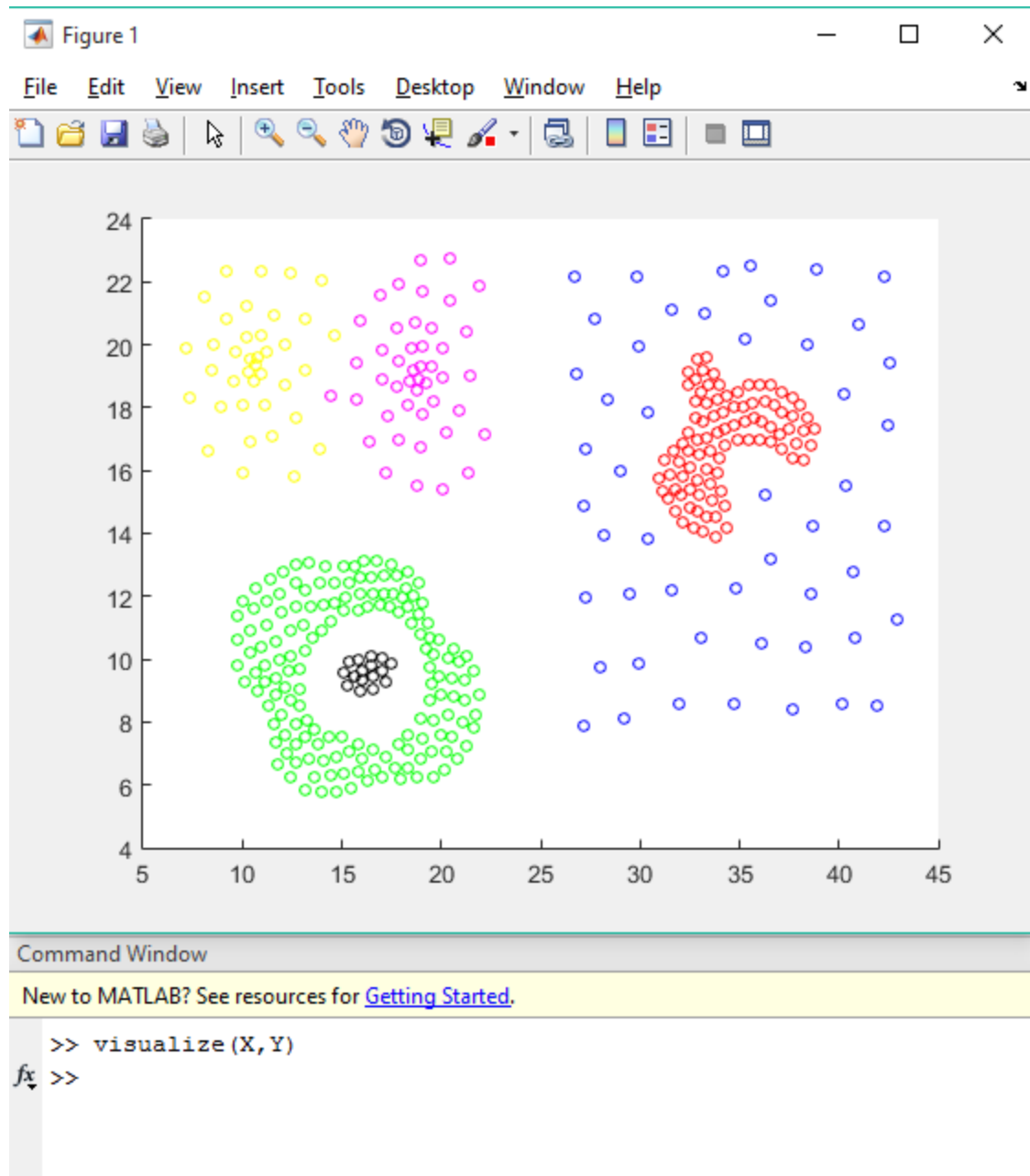
```
1 - Compound = csvread('Classification_datasets\Compound.csv');  
2 - X = Compound(:,1:2);  
3 - Y = Compound(:,3);  
4  
5 - visualize(X,Y);
```

- visualize.m



```
function visualize(X , Y)  
    % Y is label class  
    rowsToSetBlue = Y == 1;  
    rowsToSetRed = Y == 2;  
    rowsToSetYellow = Y == 3;  
    rowsToSetMagenta = Y == 4;  
    rowsToSetGreen = Y == 5;  
    rowsToSetBlack = Y == 6;  
  
    % Set colormap to blue for the blue rows.  
    myColors(rowsToSetBlue, 1) = 0;  
    myColors(rowsToSetBlue, 2) = 0;  
    myColors(rowsToSetBlue, 3) = 1;  
    % Set colormap to red for the red rows.  
    myColors(rowsToSetRed, 1) = 1;  
    myColors(rowsToSetRed, 2) = 0;  
    myColors(rowsToSetRed, 3) = 0 ;  
    % Set colormap to red for the yellow rows.  
    myColors(rowsToSetYellow, 1) = 1;  
    myColors(rowsToSetYellow, 2) = 1;  
    myColors(rowsToSetYellow, 3) = 0;  
    % Set colormap to red for the Magenta rows.  
    myColors(rowsToSetMagenta, 1) = 1;  
    myColors(rowsToSetMagenta, 2) = 0;  
    myColors(rowsToSetMagenta, 3) = 1;  
    % Set colormap to red for the green rows.  
    myColors(rowsToSetGreen, 1) = 0;  
    myColors(rowsToSetGreen, 2) = 1;  
    myColors(rowsToSetGreen, 3) = 0;  
    % Set colormap to red for the black rows.  
    myColors(rowsToSetBlack, 1) = 0;  
    myColors(rowsToSetBlack, 2) = 0;  
    myColors(rowsToSetBlack, 3) = 0;  
  
    % show scatter  
    scatter(X(:,1) , X(:,2) , 20 , myColors);  
end
```

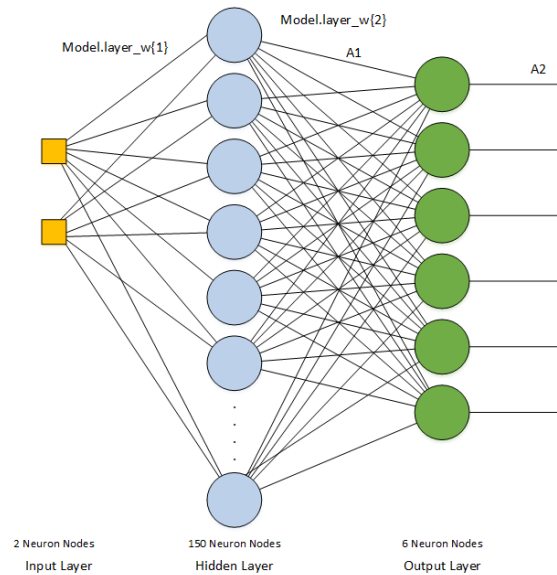
Output :



Analisis Program :

Pada tugas ini saya menggunakan data set dengan nama file Compound.csv, kemudian meload data tersebut kedalam variable X untuk features dan Y untuk labelnya. Dan menjalankan fungsi `visualize(X,Y)`, untuk memvisualisasikan setiap data point. Pada fungsi `visualize(x,y)` masing-masing data label class disesuaikan dengan data warna RGB. Kemudian memanggil `scatter(param1,param2,param3,param4)` dengan param1 dan param2 adalah kolom pertama dari data train dan kolom kedua dari data train, kemudian param3 adalah size dari ukuran plot, dan param4 adalah data color.

2. MLP Architecture

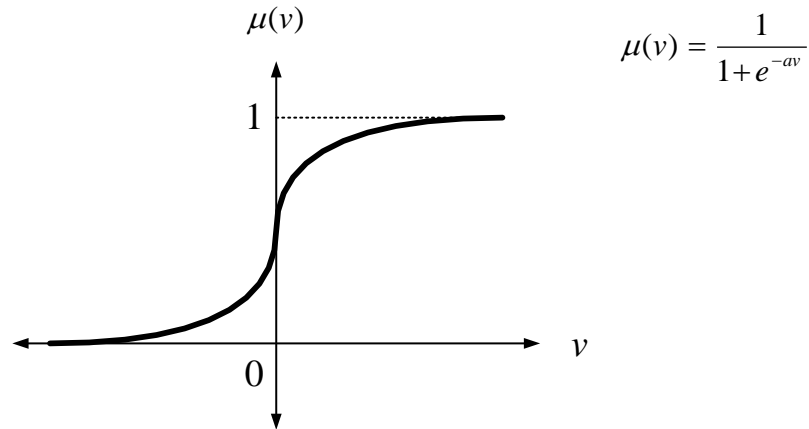


Jenis JST yang saya terapkan adalah multi layer perceptron dengan 1 Hidden Layer dan 1 Output layer. Dengan hidden node sebanyak 150 buah dan Output node sebanyak 6 buah. Pada Matlab dibuat sebuah model dari JST tersebut dengan variable objek `model.layer_w{1}` antara input layer dan hidden layer dan `model.layer_w{2}` antara hidden layer dengan output layer. Dan signal antara hidden layer adalah `A1` dan Output layer adalah `A2`. Kemudian untuk target menerapkan format sebagai berikut :

Label	Format Target					
1	1	0	0	0	0	0
2	0	1	0	0	0	0
3	0	0	1	0	0	0
4	0	0	0	1	0	0
5	0	0	0	0	1	0
6	0	0	0	0	0	1

Dengan output sebanyak 6 buah sesuai dengan banyaknya class diharapkan performansi untuk JST meningkat. Untuk learning rate saya set dengan 0.00001 dan inputan tanpa bias. Epoch sebanyak 10000 dengan waktu selesai sekitar 4 jam

menggunakan model ini. Activation function yang dipakai adalah sigmoid. Memetakan A1 dan A2 saat feedforward dengan nilai antara hampir 1 dan hampir 0. Dan untuk back propagation adalah turunan pertama dari sigmoid.



Berikut code MATLAB untuk activation code dan activation code derivative :

- sigmoid.m

```
Editor - C:\Users\Zero-Inside\Documents\Machine Learning\Assignment 3.2 MLP\sigmoid.m
+7 visualize.m x MLPMMain.m x MLP.m x makeTarget.m x decbound2D.m x classifyMLPMod.m x sigmoid.m x
1 function g = sigmoid(x)
2     % returns sigmoid evaluated elementwise in X
3     g = 1 ./ (1+exp(-x));
4 end
```

- sigmoidGrad.m

```
Editor - sigmoidGrad.m*
+5 trainMLP.m x backPropagation.m x classifyMLPModel.m x forwardFeed.m x sigmoidGrad.m* x visualize.m x
1 function g = sigmoidGrad(z)
2     % this function is to calculate the derivation of sigmoid function
3     g = (z).*(1-(z));
4 end
5
```

Berikut code MATLAB untuk membuat model JSP diatas.

- makeTarget.m

```
Editor - makeTarget.m
+5 Variables - epoch
1 function binaryLabel = makeTarget(label)
2     % convert the class label to formatted label
3     % for example if 1 then return 1 0 0 0 0 0
4     binaryLabel = zeros(1,6); % 0 0 0 0 0 0
5     binaryLabel(label) = 1;
6 end
```

- MLP.m

```
1 function obj = MLP(learning_Rate, num_layers, hidden_nodes, feature_length, epoch)
2 % this function is to create model of JST from the variable that we need
3 % learning_Rate is the move of weight
4 % num_layers is number of implement layers eg; 2 : 1 Hidden - 1 Output
5 % hidden_nodes is number of hidden nodes in hidden layers
6 % feature_length is number of coloumn in data
7 % epoch is maximum iteration of learning
8 if num_layers < 2
9     error('Number of layers must be at least 2');
10 end
11 obj.num_layers      = num_layers;
12 obj.hidden_nodes    = hidden_nodes;
13 obj.feature_length  = feature_length;
14 obj.output_nodes    = 6;
15 obj.layer_w         = cell(1, num_layers);
16 obj.layer_output    = cell(1, num_layers);
17 obj.learning_Rate   = learning_Rate;
18 obj.epoch           = epoch;
19
20 % give random value weight to w1
21 obj.layer_w{1} = randn(feature_length , hidden_nodes).*0.1;
22
23 % this for more than 2 layers
24 for layer = 2 : num_layers - 1
25     obj.layer_w{layer} = randn(hidden_nodes,hidden_nodes) .* 0.1;
26 end
27
28 % this weight is last weight layer before output layer
29 obj.layer_w{num_layers} = randn(hidden_nodes,obj.output_nodes) .* 0.1;
30
31 end
```

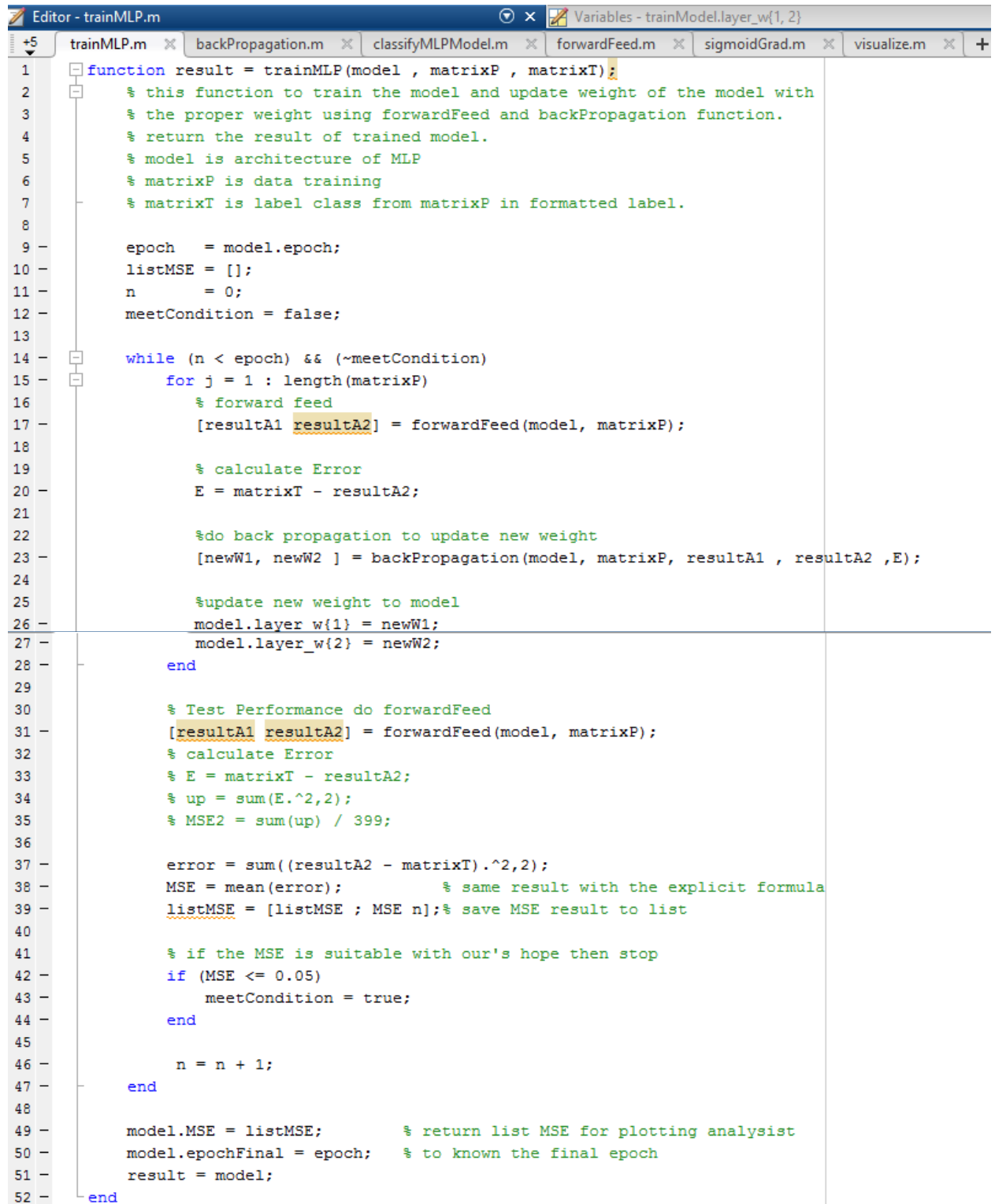
- MLPMain.m

```
6 %% MLP Model
7 %variable to make MLP Architecture
8 learningRate = 0.0001;
9 numLayers    = 2 ; %H-O
10 numHiddenNodes = 150;
11 numFeatures   = size(X,2);
12 epoch        = 10000;
13 %instance the MLP Architecture
14 model = MLP(learningRate, numLayers, numHiddenNodes, numFeatures ,epoch);
15
16 % Make Target Matrix from Y label
17 yTarget = [];
18 for i = 1:length(Y)
19     yTarget = [yTarget ; makeTarget(Y(i))];
20 end
21
```

3. Apply Multi-Layer Perceptron

a. Function for Learning

- trainMLP.m

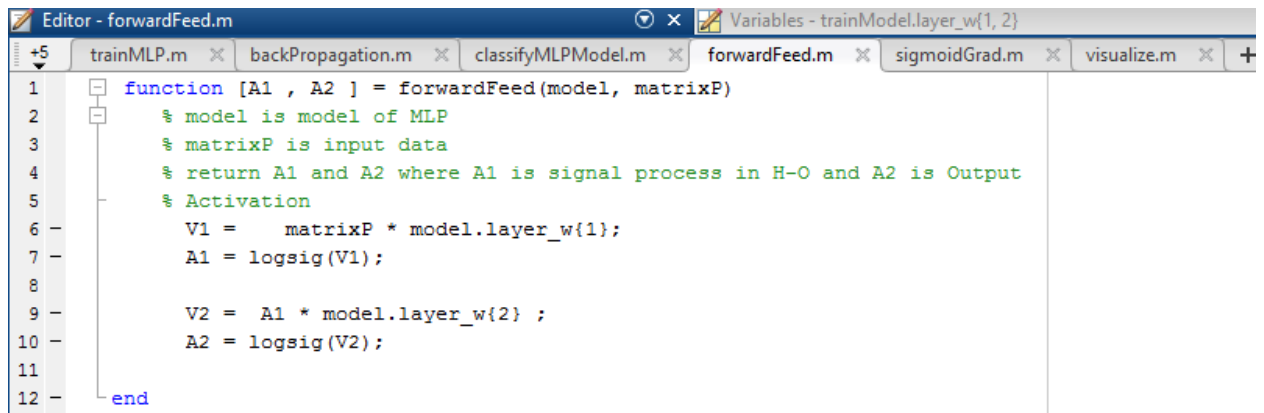


```
Editor - trainMLP.m
Variables - trainModel.layer_w{1, 2}

trainMLP.m  backPropagation.m  classifyMLPModel.m  forwardFeed.m  sigmoidGrad.m  visualize.m

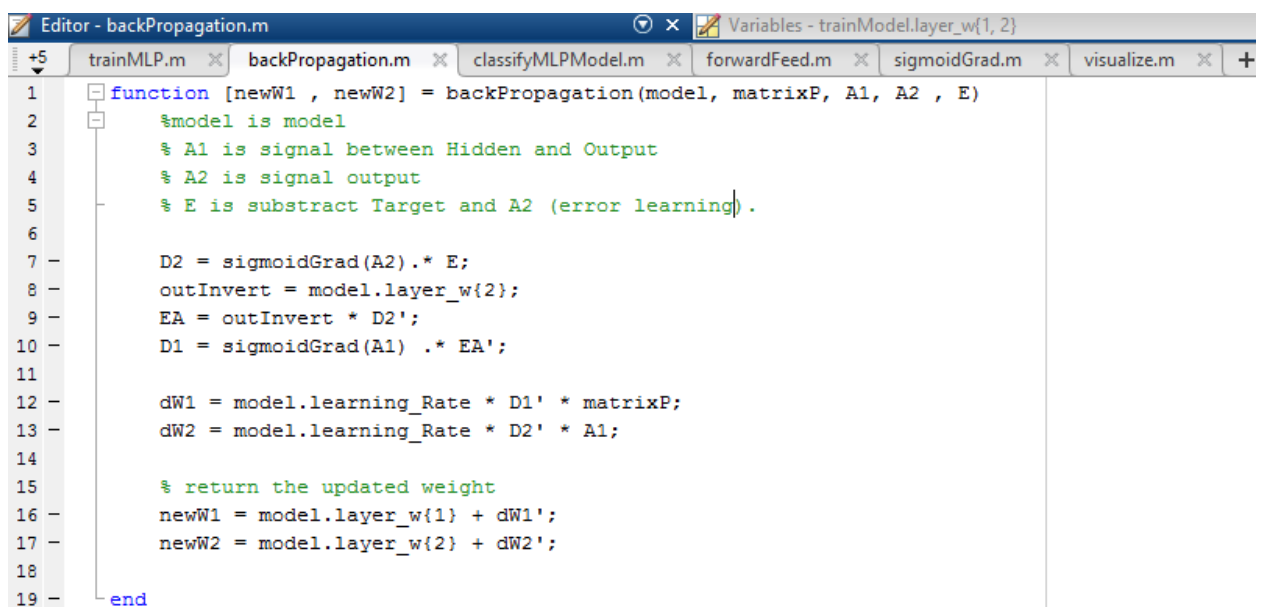
1 function result = trainMLP(model , matrixP , matrixT);
2 % this function to train the model and update weight of the model with
3 % the proper weight using forwardFeed and backPropagation function.
4 % return the result of trained model.
5 % model is architecture of MLP
6 % matrixP is data training
7 % matrixT is label class from matrixP in formatted label.
8
9 epoch = model.epoch;
10 listMSE = [];
11 n = 0;
12 meetCondition = false;
13
14 while (n < epoch) && (~meetCondition)
15     for j = 1 : length(matrixP)
16         % forward feed
17         [resultA1 resultA2] = forwardFeed(model, matrixP);
18
19         % calculate Error
20         E = matrixT - resultA2;
21
22         %do back propagation to update new weight
23         [newW1, newW2 ] = backPropagation(model, matrixP, resultA1 , resultA2 ,E);
24
25         %update new weight to model
26         model.layer_w{1} = newW1;
27         model.layer_w{2} = newW2;
28     end
29
30     % Test Performance do forwardFeed
31     [resultA1 resultA2] = forwardFeed(model, matrixP);
32     % calculate Error
33     % E = matrixT - resultA2;
34     % up = sum(E.^2,2);
35     % MSE2 = sum(up) / 399;
36
37     error = sum((resultA2 - matrixT).^2,2);
38     MSE = mean(error); % same result with the explicit formula
39     listMSE = [listMSE ; MSE n]; % save MSE result to list
40
41     % if the MSE is suitable with our's hope then stop
42     if (MSE <= 0.05)
43         meetCondition = true;
44     end
45
46     n = n + 1;
47 end
48
49 model.MSE = listMSE; % return list MSE for plotting analysist
50 model.epochFinal = epoch; % to known the final epoch
51 result = model;
52 end
```

- forwardFeed.m



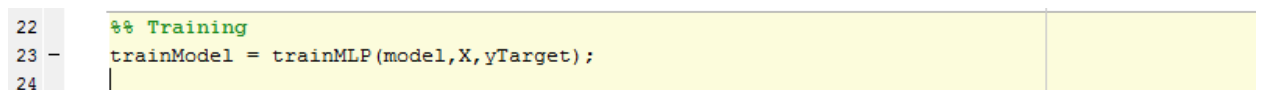
```
1 function [A1 , A2 ] = forwardFeed(model, matrixP)
2     % model is model of MLP
3     % matrixP is input data
4     % return A1 and A2 where A1 is signal process in H-O and A2 is Output
5     % Activation
6     V1 = matrixP * model.layer_w{1};
7     A1 = logsig(V1);
8
9     V2 = A1 * model.layer_w{2} ;
10    A2 = logsig(V2);
11
12    end
```

- backPropagation.m



```
1 function [newW1 , newW2] = backPropagation(model, matrixP, A1, A2 , E)
2     %model is model
3     % A1 is signal between Hidden and Output
4     % A2 is signal output
5     % E is substract Target and A2 (error learning) .
6
7     D2 = sigmoidGrad(A2) .* E;
8     outInvert = model.layer_w{2};
9     EA = outInvert * D2';
10    D1 = sigmoidGrad(A1) .* EA';
11
12    dW1 = model.learning_Rate * D1' * matrixP;
13    dW2 = model.learning_Rate * D2' * A1;
14
15    % return the updated weight
16    newW1 = model.layer_w{1} + dW1';
17    newW2 = model.layer_w{2} + dW2';
18
19    end
```

- MLPMain.m



```
22 %% Training
23 trainModel = trainMLP(model,X,yTarget);
24
```

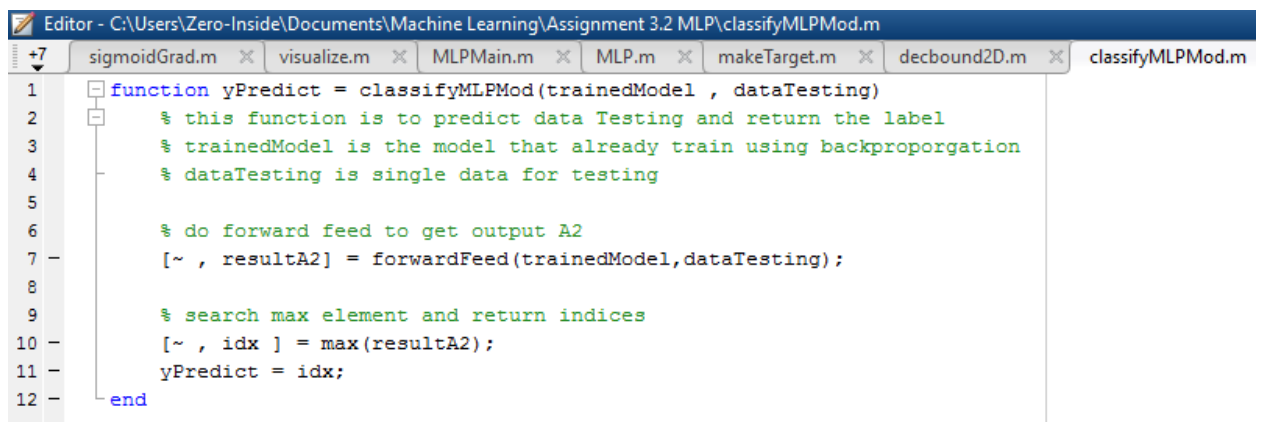
Analisis Program :

Untuk melatih model dengan menerapkan algoritma back propagation, menjalankan trainMLP(param1,param2,param3) dengan param1 adalah model JST, MatrixP adalah data training, dan param3 adalah data label masing-masing data training yang sudah disesuaikan dengan format. Proses pelatihan dengan mengulangi literasi 1 siklus pelatihan sebanyak epoch. Terlebih dahulu model JST yang sudah dibentuk dilakukan forward feed atau pelatihan maju dengan

menjalankan fungsi `forwardFeed(param1,param2)` dengan `param1` adalah model, dan `param2` adalah data training dihasilkan signal A1 dan signal A2, kemudian untuk mendapatkan error dengan mengurangkan matrix Target dan A2 lalu dikuadratkan dan dijumlah kemudian dirata-rata atau disebut MSE. Hasil A1,A2 dan error akan digunakan untuk menjalankan fungsi `backPropagation` untuk mengupdate bobot yang ada pada model. Untuk analisis error, dilakukan record MSE pada setiap epoch. Setelah sampai batas epoch atau nilai MSE yang diinginkan dalam hal ini saya mengeset error 0.05 maka didapat model baru pelatihan dengan bobot yang siap untuk ditesting.

b. Function for Predicting/Classifying Data

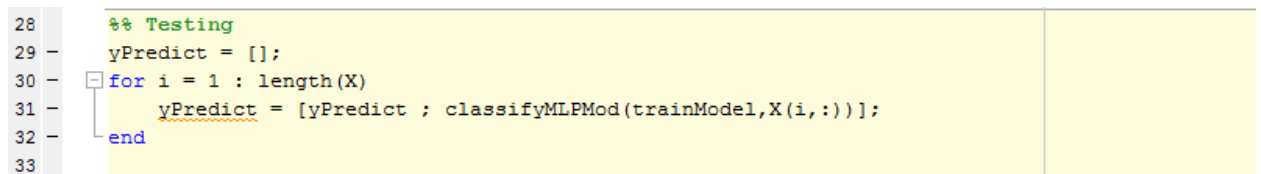
- `classifyMLPMod.m`



```
Editor - C:\Users\Zero-Inside\Documents\Machine Learning\Assignment 3.2 MLP\classifyMLPMod.m
sigmoidGrad.m visualize.m MLPMain.m MLP.m makeTarget.m decbound2D.m classifyMLPMod.m

1 function yPredict = classifyMLPMod(trainedModel , dataTesting)
2     % this function is to predict data Testing and return the label
3     % trainedModel is the model that already train using backproporgation
4     % dataTesting is single data for testing
5
6     % do forward feed to get output A2
7     [~, resultA2] = forwardFeed(trainedModel,dataTesting);
8
9     % search max element and return indices
10    [~, idx ] = max(resultA2);
11    yPredict = idx;
12 end
```

- `MLPMain.m`



```
28 %% Testing
29 yPredict = [];
30 for i = 1 : length(X)
31     yPredict = [yPredict ; classifyMLPMod(trainModel,X(i,:))];
32 end
33
```

Analisis Program :

Fungsi `classifyMLPMod(param1,param2)` akan memprediksi data testing dan mengembalikan nilai classnya. Dengan `param1` adalah model yang sudah dilatih dan `param2` adalah satu data testing yang akan diuji. Untuk memprediksi atau mengklasifikasikan data testing dilakukan pelatihan maju terlebih dahulu dengan menjalankan fungsi `forwardFeed` dan didapat output neuron atau `resultA2`.

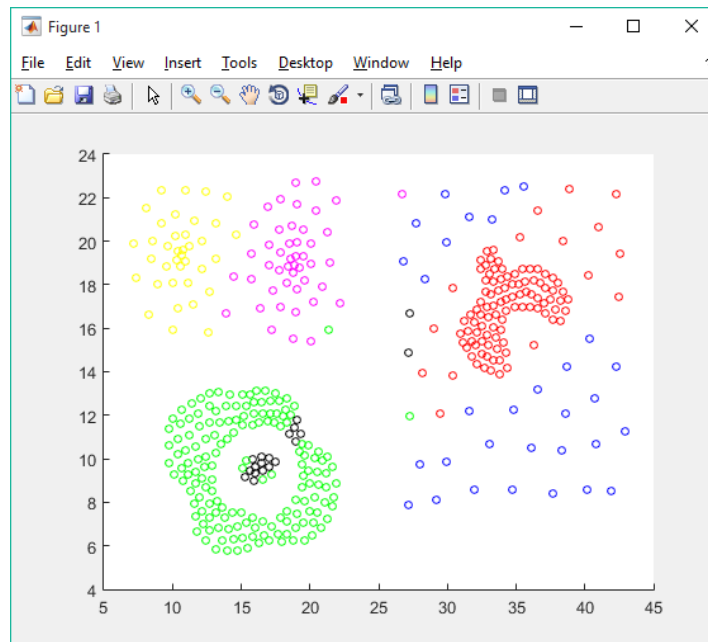
Kemudian mencari nilai index yang ada pada element maximal pada resultA2. Index ini menjadi hasil label yang diprediksi pada data tersebut. Untuk mencari label pada seluruh data testing dilakukan iterasi, bisa dilihat pada file MLPMain.m untuk seluruh data testing.

c. Visualize the predicting data

- MLPMain.m

```
34 %% Visualize
35 - visualize(X,yPredict);
36
```

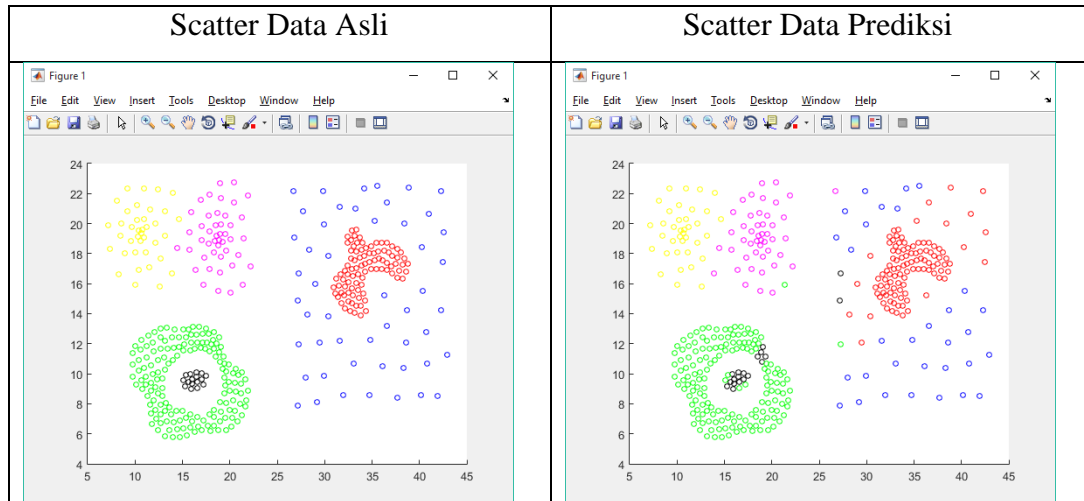
- Output



Analisis Program :

Dengan memanggil visualize(x,yPredict2), dimana x adalah train data dan yPredict adalah class yang dihasilkan melalui testing maka ditampilkan scatter yang memplot posisi dari data train.

d. Difference between testing and training



Analisis :

Terlihat ada warna plot yang berbeda dari yang hasil scatter data asli. Tetapi secara keseluruhan JST yang ditrain dapat mengelompokkan data tersebut sesuai kelasnya dalam visual ini adalah warna.

4. Performance Analysis

- ConfusionMatrix.m

```
Editor - ConfusionMatrix.m
makeTarget.m  decbound2D.m  classifyMLPMod.m  sigmoid.m  calculatePerformance.m  ConfusionMatrix.m

1  function cfsMatrix = ConfusionMatrix( Ytrain , Ypredict)
2  -     listClass = unique(Ytrain);
3  -     numberClass = length(listClass);
4
5  -     cfsMatrix = zeros(numberClass);
6
7  -     for i = 1 : length(Ytrain)
8  -         cfsMatrix(Ytrain(i), Ypredict(i)) = cfsMatrix(Ytrain(i),Ypredict(i)) + 1 ;
9  -     end
10 - end
```

- Output

```
Command Window

>> cfsMatrix = ConfusionMatrix(Y,yPredict)

cfsMatrix =

    31    15     0     1     1     2
     0    92     0     0     0     0
     0     0    37     1     0     0
     0     0     0    44     1     0
     0     0     0     0   153     5
     0     0     0     0     4    12

>> calculatePerformance(cfsMatrix,1)

ans =

    0.9248

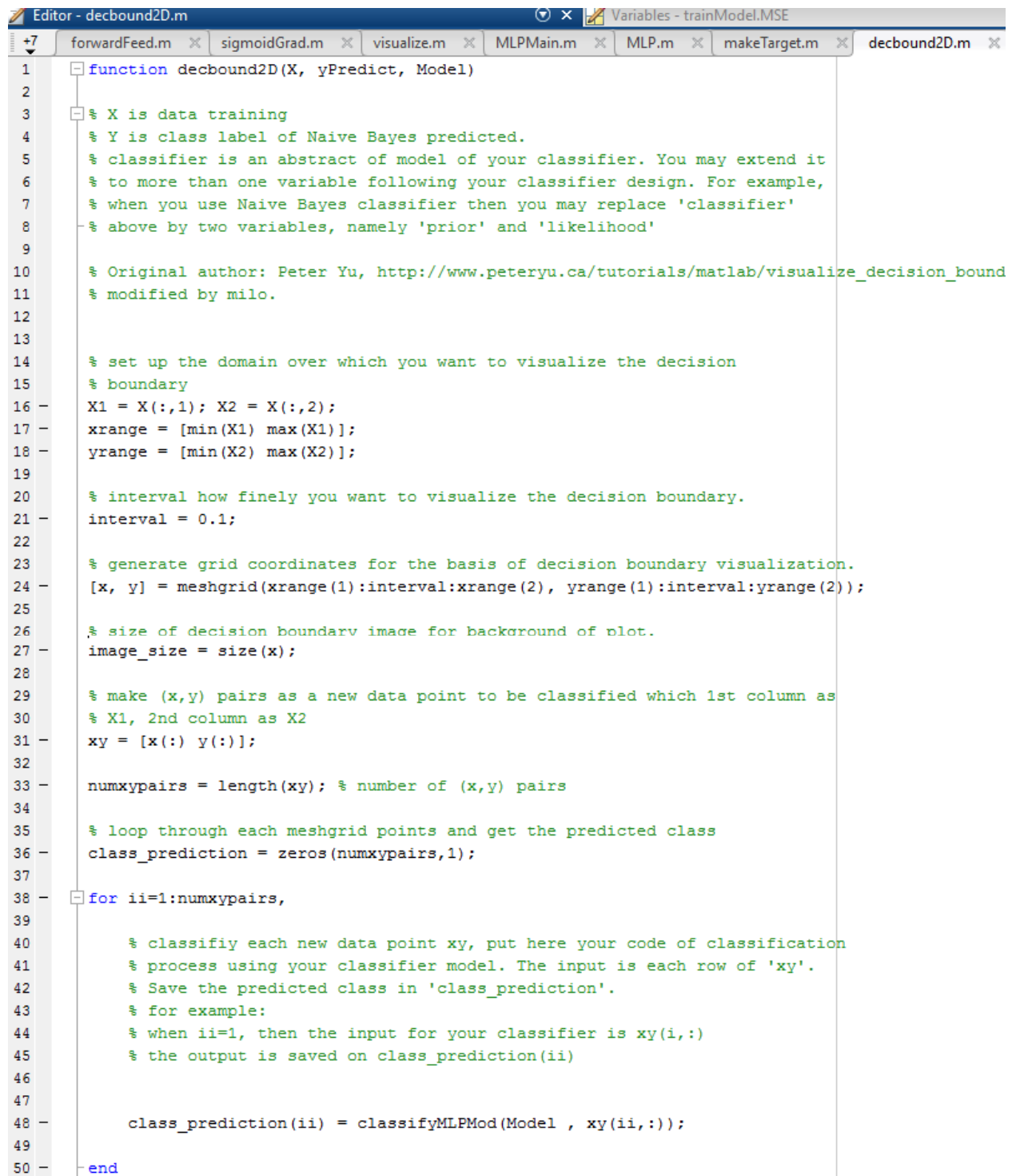
fx >>
```

Analisis Program :

Untuk menghitung performance terlebih dahulu membuat confusion matrix berdasarkan label data train dan label data testing. Kemudian untuk menghitung performasi memanggil fungsi `calculatePerformance(param1, param2)`. Dengan param1 adalah confusion matrix dan param2 adalah mode atau jenis performasi, 1 untuk F1 Micro 2 untuk F2 Macro dan 3 untuk Accuracy. Berdasarkan hasil perhitungan didapat hasil klasifikasi naïve bayes adalah 0.9248. atau dalam persen adalah 92.48%.

5. Decision Boundaries

- Decbound2D.m

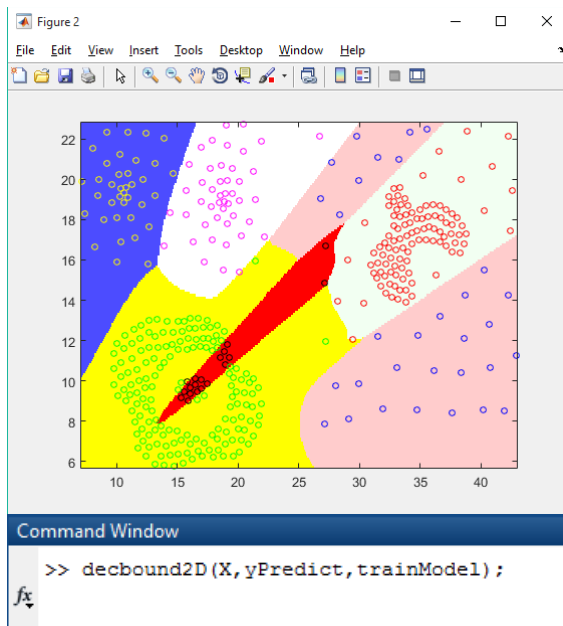


The image shows a MATLAB Editor window with the file 'decbound2D.m' open. The window has a title bar 'Editor - decbound2D.m' and a toolbar. Below the toolbar, there are several tabs: 'forwardFeed.m', 'sigmoidGrad.m', 'visualize.m', 'MLPMain.m', 'MLP.m', 'makeTarget.m', and 'decbound2D.m'. The 'decbound2D.m' tab is active, showing the following code:

```
1 function decbound2D(X, yPredict, Model)
2
3 % X is data training
4 % Y is class label of Naive Bayes predicted.
5 % classifier is an abstract of model of your classifier. You may extend it
6 % to more than one variable following your classifier design. For example,
7 % when you use Naive Bayes classifier then you may replace 'classifier'
8 % above by two variables, namely 'prior' and 'likelihood'
9
10 % Original author: Peter Yu, http://www.peteryu.ca/tutorials/matlab/visualize\_decision\_bound
11 % modified by milo.
12
13
14 % set up the domain over which you want to visualize the decision
15 % boundary
16 X1 = X(:,1); X2 = X(:,2);
17 xrange = [min(X1) max(X1)];
18 yrange = [min(X2) max(X2)];
19
20 % interval how finely you want to visualize the decision boundary.
21 interval = 0.1;
22
23 % generate grid coordinates for the basis of decision boundary visualization.
24 [x, y] = meshgrid(xrange(1):interval:xrange(2), yrange(1):interval:yrange(2));
25
26 % size of decision boundary image for background of plot.
27 image_size = size(x);
28
29 % make (x,y) pairs as a new data point to be classified which 1st column as
30 % X1, 2nd column as X2
31 xy = [x(:) y(:)];
32
33 numxypairs = length(xy); % number of (x,y) pairs
34
35 % loop through each meshgrid points and get the predicted class
36 class_prediction = zeros(numxypairs,1);
37
38 for ii=1:numxypairs,
39
40     % classifiy each new data point xy, put here your code of classification
41     % process using your classifier model. The input is each row of 'xy'.
42     % Save the predicted class in 'class_prediction'.
43     % for example:
44     % when ii=1, then the input for your classifier is xy(i,:)
45     % the output is saved on class_prediction(ii)
46
47
48     class_prediction(ii) = classifyMLPMod(Model , xy(ii,:));
49
50 end
```

```
51  
52 % reshape the idx (which contains the class label) into an image.  
53 decisionmap = reshape(class_prediction, image_size);  
54  
55 figure;  
56  
57 %show the image  
58 imagesc(xrange,yrange,decisionmap);  
59 hold on;  
60 set(gca,'ydir','normal');  
61  
62 % set RGB color for colormap for the classes:  
63 cmap = [1 0.8 0.8; % class 1 = light red  
64         0.95 1 0.95; % class 2 = light green  
65         0.3 0.3 1; % class 3 = blue  
66         1 1 1; % class 4 = white  
67         1 1 0; % class 5 = yellow  
68         1 0 0; % class 6 = black  
69         ];  
70 colormap(cmap);  
71  
72 visualize(X,yPredict);  
73 hold off;  
74 end
```

- Output



Analisis Program :

Fungsi `decbound2d(param1,param2,param3)` akan menampilkan decision boundaries dengan param1 adalah data testing , param2 adalah class label hasil dari model JST yang sudah di latih, dan param3 adalah model yang akan digunakan untuk memberi label kelas pada new data point yang dipetakan pada imagesec.