Classification with Neural Networks and Support Vector Machine

Rick Alayza, Trincy Thomas Kozhikkadan, Krystofer Newman

***Abstract*—In progressing though the class in machine learning, there are now two ways to classify datasets. On way was through neural networks and the new method is to implement support vector machines. Both methods will be compared on the ability to classify a dataset of 928 points and 32 features. Each method has advantages over the other. The comparison was simplified down to accuracy and difficulty. Both methods had similar accuracy but the Neural Network method was much easier to use with Matlab.**

***Index Terms*—classification, comparison, machine learning, neural network, and support vector machine.**

I. Introduction

CLASSIFICATION and prediction are common applications for supervised machine learning. In previous projects, different models of neural networks were trained and tested to predict the next 30 values from a given dataset. This project will focus on classification portion as well as conduct a comparison between to different methods. The first method is the more classical Neural Network (NN) design. This design used nodes and hidden layers to gain an insight to the dataset and apply that training to generate an output. The second method is called Support Vector Machines (SVM). SVMs and NNs both utilize cost functions but are significantly different. This report will discuss those cost function differences along with initial intuitions about the assigned datasets. The coding and results for each method well also be discussed followed by a comparison.

A. Neural Network Classification

Neural networks, for the purpose of this application, have a training set, validation set, and test set. All three of these subsets were divided from the original assigned dataset for the project. The training set functions as its description. This set was used to train the nodes within the hidden layers to ultimately reach an output. Figure 1 is an example of how the nodes in the neural network can be connected. For this project, Layer 1 will have a different number of inputs or features. The number of nodes and hidden layers are up to the programmer. Know that more layers and nodes may result in more accurate conclusions but also might lead to overfitting along with additional time required to conduct the training. The mathematical representation of the neural network as also list below. This is important to indicate the differences between the two model structures. Due to limited space, the neural network Cost Function (Equation 4) was broken into small segments. Equations 1 and 2 are the logistic regression components of the neural network. Equation 3 is the regularization component commonly applied to neural networks. Due to the overall spread in values within the datasets, a regularization component was implemented. The main purpose of conducting the training on the neural network is to minimize the Cost Function (Equation 5) using methods such as gradient decent and back propagation. A minimized Cost Function is associated with a minimized error. A minimized error indicates the accuracy of the neural network's classifications.

$$R_{11} = y_k^{(i)} \log\left(h_\theta\left(x^{(i)}\right)\right)_k \tag{1}$$

$$R_{12} = \left(1 - y_k^{(i)}\right) \log\left(1 - \left(h_\theta\left(x^{(i)}\right)\right)_k\right) \tag{2}$$

$$R_{13} = \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_l+1} \left(\Theta_{ji}^{(l)}\right)^2 \tag{3}$$

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} \sum_{k=1}^{K} R_{11} + R_{12}\right] + R_{13} \tag{4}$$

$$\min_\theta J(\Theta) \tag{5}$$

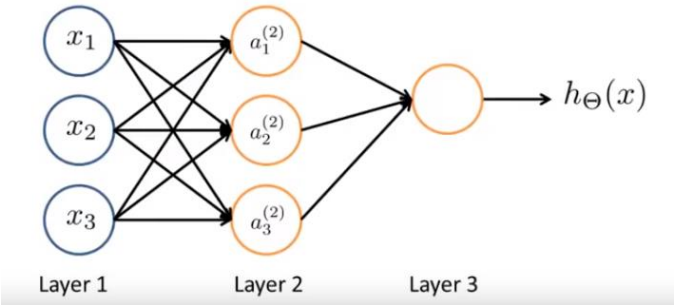

Figure 1: Neural Network Sample Diagram [3]

B. Support Vector Machine

Support Vector Machines use a hyperplane to find the separation between classes. Due to this hyperplane, the cost function associated with SVM is changed slightly when the logistic regression hypothesis is modified along with minor

naming conventions. Figure 2 is an example of a modified logistic regression hypothesis used in the video lectures by Andrew Ng. The blue line represents the logistic regression in Equation 4. The equation in the figure is for the magenta line that approximately mimics the original regression line. For the purpose of this introduction, a hypothesis with only two cost components (1 or 0) will be displayed. Equation 6 is the modified regression rewritten as a cost component. Equations 7 and 8 show the difference between the NN Cost Function and this one. Equation 10 is the complete Cost Function with the new naming convention for variable C. Equation 11 is similar to Equation 5 by stating the objective of the SVM by minimizing the Cost Function.

$$cost_0\left(\Theta^T x^{(i)}\right) = -\log\left(1 - \frac{1}{1+\exp(-z)}\right) \tag{6}$$

$$R_{21} = y^{(i)} cost_1\left(\Theta^T x^{(i)}\right) \tag{7}$$

$$R_{22} = \left(1 - y^{(i)}\right) cost_0\left(\Theta^T x^{(i)}\right) \tag{8}$$

$$R_{23} = \frac{1}{2}\sum_{i=1}^{n} \Theta_j^2 \tag{9}$$

$$J(\Theta) = C\sum_{i=1}^{m}[R_{21} + R_{22}] + R_{23}, C = \frac{1}{\lambda} \tag{10}$$

$$\min_{\theta} J(\Theta) \tag{11}$$

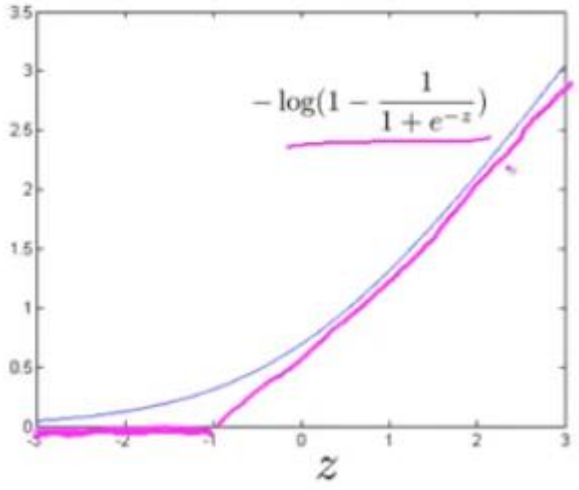

Figure 2: Modified Logistic Regression [3]

C. Datasets

The assigned datasets are equal in size but have interesting characteristics. Both datasets, PTO and DC, have 928 points with 32 features for each point. The points are either classified as 1 or 0. That means 464 points for each classification class. The number of features is a large departure from previous projects. The trend for previous projects was with time based or a singular feature. 32 features significantly change Layer 1 (Figure 1) of the NN.

II. Neural Network

As mentioned previously, both of the given datasets were structured with 32 features and 928 rows or a 928x32 matrix. Additionally, the category files given were constructed as a single 928x1 vector. As seen in Appendix section (insert first code reference here), some of the initial steps to processing the data and category files is to load them into Matlab and transpose them to match the input formats of MatLab's parameter requirements. For validation of the sizing in the two files, we output the size of the two input and target variables which are now 32x928 and 1x928 respectively. As an informative intermediate step, we then plot two rows of the input data as shown in Figure 3.

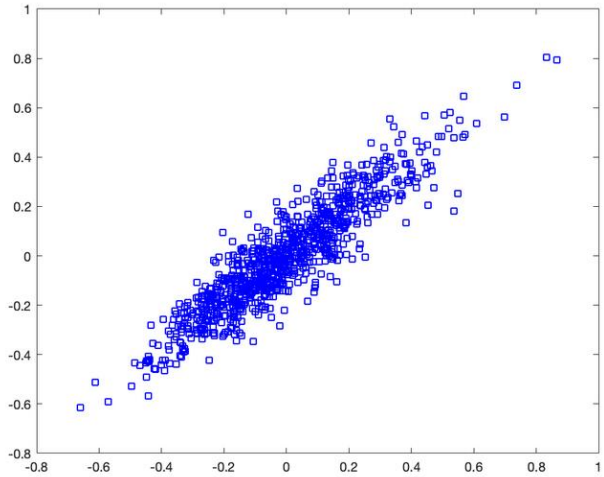

Figure 3: Dataset Sample Plot from DC Dataset

A. DC Dataset

1) Network/Training Configuration

Following the initial loading and formatting of the datasets, is the configuration of the neural nets. For the first dataset the configuration is set to a feedforward model with 5 hidden layers and the default training function 'trainlm' which is a definition of Levenberg-Marquardt algorithm. Figure 4 shows a visual representation of the neural network model used on the DC Dataset as well as the NN progress. According to the figure, this training was concluded after 8 epochs or iterations of the entire dataset.

Splitting the dataset into training, test, and validation sets using 70%, 15%, and 15% respectively. By utilizing the Levenberg-Marquardt algorithm, the validation subset was used to terminate the training. In the instance that the mean

square error increases in the validation performance, the train will stop.

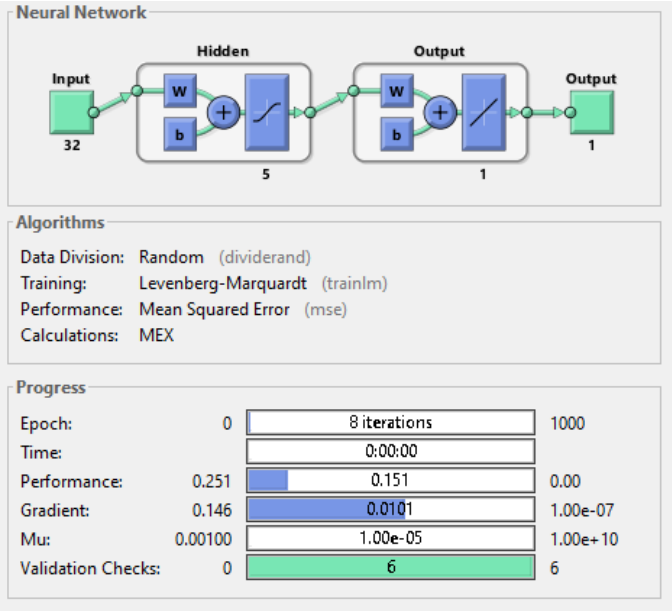

Figure 4: DC Neural Network Setup and Progress

2) DC Results

The results of the NN used on the DC Dataset is shown in Figure 5 and Figure 6. Figure 5 depicts the mean square error of the training, validation, and test outputs of the NN. A lower mean square error (MSE) value is preferred. According to the figure, this MSE is not ideal. This was expected with the spread of values within the dataset (Figure 6). At epoch 2, the lowest MSE in the validation progress is highlighted. Since this value occurred early in the iterations explains why the training finished 6 epochs later. The most important value from Figure 5 is the MSE for the testing portion. Using 139 samples, the MSE for the test was 0.198287. The MSE of 0.20693 in the title for Figure 5 is a reference the best MSE in the training process, not in testing the NN.

Figure 6 shows the fit line for each subset and the associated regression R value. The R value is the correlation between the NN output and the targets or known values. An R value of 1 indicates a close correlation while a value of 0 is interrupted as no relationship between targets and outputs. The R value is listed in the title of each plot. The important value in this figure is the R value for the testing which was .04187. This value hints to a partial relationship or correlation between the known values in the test and the NN output. Using the pattern recognition tool, this NN had an accuracy of approximately 66%.

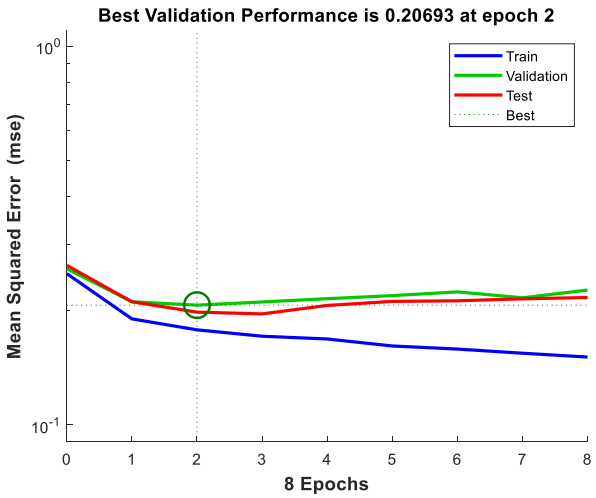

Figure 5: DC NN Performance Plot

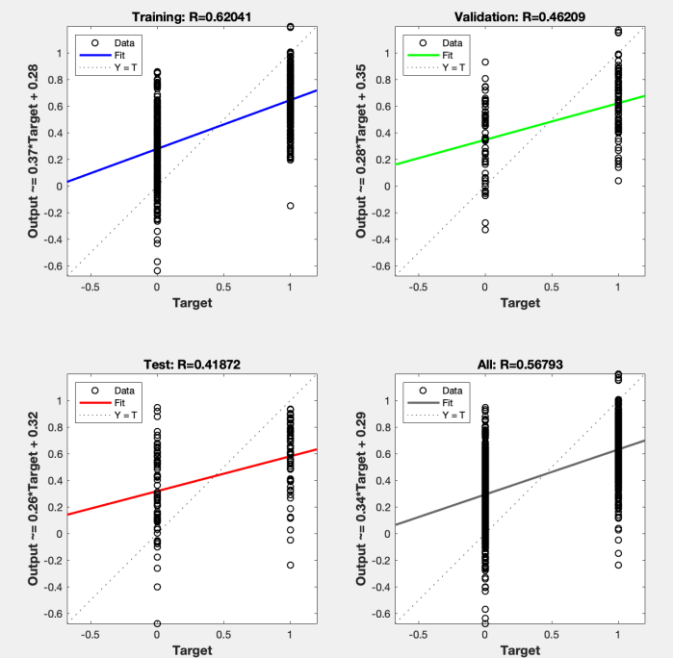

Figure 6: DC Neural Network Fit

B. PTO Dataset

This dataset was conducted using a different algorithm and different hidden network layers but used similar output analysis.

1) Network/Training Configuration

This configuration used the Bayesian Regularization with 8 hidden layers. This algorithm terminates the training when the regularization weights have reached a minimal value. Unlike the DC NN, this NN completed training in 255 epoch or dataset iterations (Figure 7).

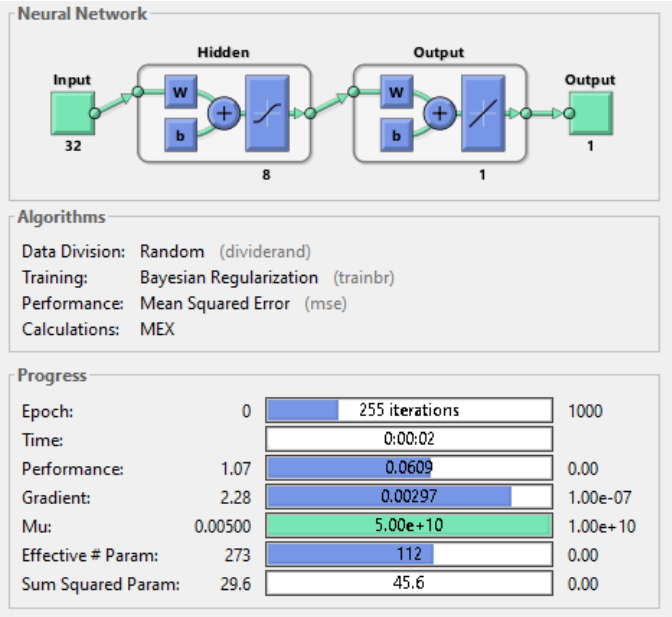

Figure 7: PTO Neural Network Setup and Progress

2) PTO Results

These test results will not include the validation subset since the algorithm does not depend on those results. Figure 8 shows the MSE for the test and train subsets. The MSE for the testing portion was 0.111408.

Figure 9 is the R values for each portion. The value of importance is for the testing portion. According to the figure, the R value for testing is 0.85216. This indicates a strong correlation between the neural network outputs and the testing targets. This NN had an approximate accuracy of 86% when implementing the pattern recognition tool.

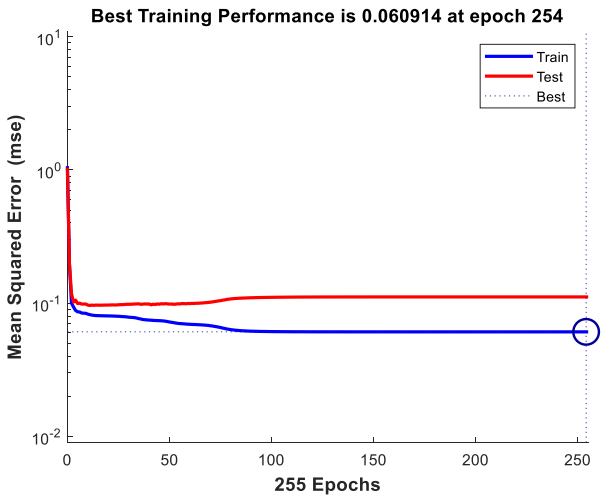

Figure 8: PTO NN Performance Results

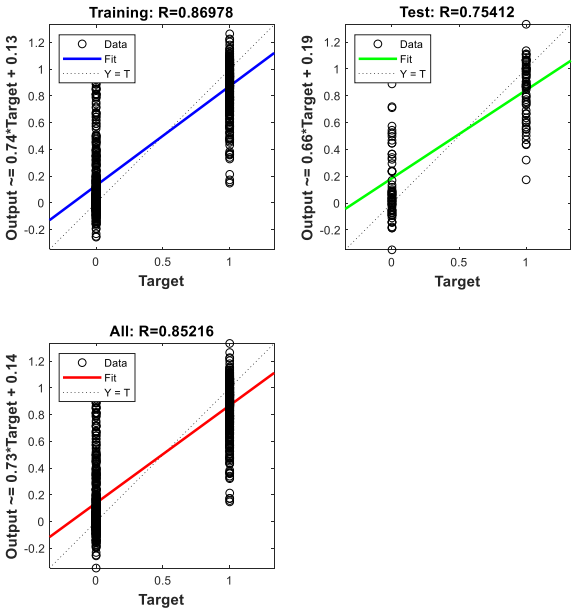

Figure 9: PTO Neural Network Fit Results

III. Support Vector Machines

The next step in the project was to apply SVM to the DC and PTO datasets. The dataset was assigned as a 928x32 size input called x and the labels category as a 928x1 size input y. A random permutation of the data is performed using randperm() function before it is split into test and training set where 80% of the samples are selected as training and 20% as test dataset. Out of the training data, 20% is selected as cross validation data set using the cvpartition (y_train,'k',148) command. The next step involves selecting the most useful features among the available features using the sequential feature selection operation in MATLAB where sequentialfs(fun,X,y) selects a subset of features from the data matrix X that best predict the data in y by sequentially selecting features until there is no improvement in prediction [1].The model is then trained using the fitcsvm operation. The training dataset, best features selected and Kernel function are passed as input to the fitcsvm. It generates a list of optimized BoxConstraint values and Kernel scaling values for the kernel based on the input training dataset [2]. The Kernel function used is the Gaussian (or radial basis function) kernel. OptimizeHyperparameters name-value pair argument of fitcsvm is used to find parameter values that minimize the cross-validation loss [2].

A. DC Dataset

The accuracy of the model with the optimized parameter values is calculated and verified with the test dataset and the decision plane is plotted. The SVM model generates an accuracy of about 60-65% for the DC dataset.

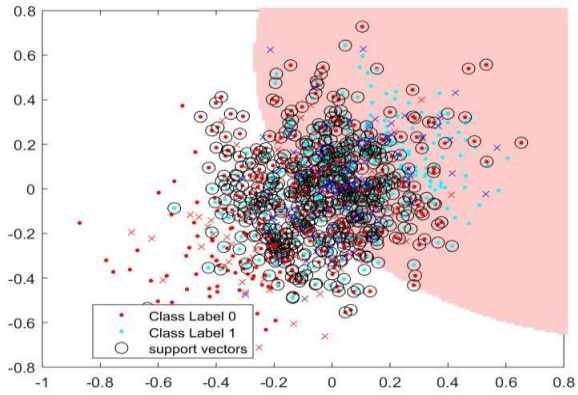

Figure 10: DC SVM Decision Boundary Plot

B. PTO Dataset

An accuracy of about 80-85% for the PTO dataset.

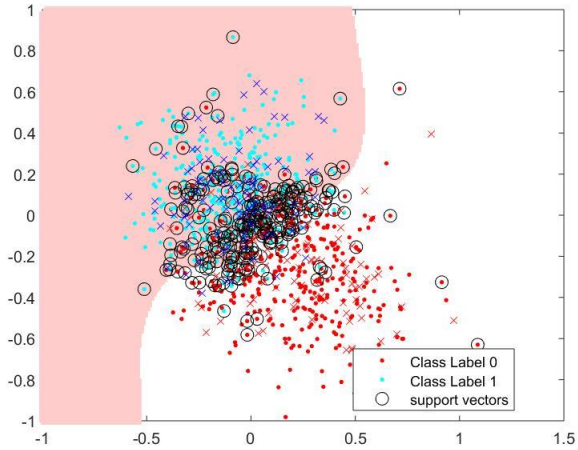

Figure 11: PTO SVM Decision Boundary Plot

IV. Comparison

The DC and PTO datasets had similar results when evaluating accuracy. For the DC dataset, the NN was approximately 66% accurate and the SVM was roughly 60%-65%. For the PTO dataset, both NN and SVM had accuracies around 86%. The implies that both methods are fully capable of classifying the datasets. The next comparison is ease of use. The NN method has a dedicated tool in Matlab with step by step instruction. The SVM method has a relatively short script to generate result but does not have a prepackaged graphical tool. Until further development in Matlab, the NN method is by far the simplest and effective way to classify the datasets for machine learning.

V. References

[1] https://www.mathworks.com/help/stats/sequentialfs.html
[2] https://www.mathworks.com/help/stats/support-vector-machines-for-binary-classification.html
[3] https://www.youtube.com/watch?v=0twSSFZN9Mc&index=50&list=PLLssT5z_DsK-h9vYZkQkYNWcItqhlRJLN

VI. Appendix

A. NN DC Code

```
%   DC - input data.
%   Category - target data.

x = DC';
t = Category';

hiddenLayerSize = 5;
net = patternnet(hiddenLayerSize);

% Setup Division of Data for Training, Validation,
Testing
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

% Train the Network
[net,tr] = train(net,x,t);

% Test the Network
y = net(x);
e = gsubtract(t,y);
performance = perform(net,t,y)
tind = vec2ind(t);
yind = vec2ind(y);
percentErrors = sum(tind ~= yind)/numel(tind);

% View the Network
view(net)

% Plots
%figure, plotperform(tr)
%figure, plottrainstate(tr)
%figure, ploterrhist(e)
%figure, plotconfusion(t,y)
%figure, plotroc(t,y)
```

B. NN PTO Code

```
%   PTO - input data.
%   Category - target data.

x = PTO';
t = Category';

trainFcn = 'trainbr';  % Bayesian Regularization
backpropagation.

% Create a Fitting Network
hiddenLayerSize = 8;
net = fitnet(hiddenLayerSize,trainFcn);

% Setup Division of Data for Training, Validation,
Testing
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

% Train the Network
[net,tr] = train(net,x,t);

% Test the Network
y = net(x);
e = gsubtract(t,y);
performance = perform(net,t,y)

% View the Network
view(net)
```

```
% Plots
%figure, plotperform(tr)
%figure, plottrainstate(tr)
%figure, ploterrhist(e)
%figure, plotregression(t,y)
%figure, plotfit(net,x,t)
```

C. SVM DC Code

```
clear; close all; clc;

%% preparing dataset

load ('Category DC.mat');
load ('dataset_DC.mat');
%Select x =data features , y= label category
x= [ DC(:,1) DC(:,2) DC(:,3) DC(:,4) DC(:,5) DC(:,6)
DC(:,7) DC(:,8) DC(:,9) DC(:,10) DC(:,11) DC(:,12)
DC(:,13) DC(:,14) DC(:,15) DC(:,16) DC(:,17)
DC(:,18) DC(:,19) DC(:,20) DC(:,21) DC(:,22)
DC(:,23) DC(:,24) DC(:,25) DC(:,26) DC(:,27)
DC(:,28) DC(:,29) DC(:,30) DC(:,31) DC(:,32) ];
y= [Category(:,1)];

rand_num = randperm(size(x,1));      %permutation
reordering  and randomly sorting  the data samples
X_train =
x(rand_num(1:round(0.8*length(rand_num))),:);
%select 80% from the randomly sorted data samples as
training dataset
y_train =
y(rand_num(1:round(0.8*length(rand_num))),:);

%select 20% of the randomly sorted data as test
dataset
X_test =
x(rand_num(round(0.8*length(rand_num))+1:end),:);
y_test =
y(rand_num(round(0.8*length(rand_num))+1:end),:);
%% CV partition
c = cvpartition(y_train,'k',148);  %select 20% from
the training dataset as cross validation data set

%% feature selection

opts = statset('display','iter');
classf = @(train_data, train_labels, test_data,
test_labels)...
    sum(predict(fitcsvm(train_data,
train_labels,'KernelFunction','rbf'), test_data) ~=
test_labels);
%select the  most useful features
[fs, history] = sequentialfs(classf, X_train,
y_train, 'cv', c, 'options', opts,'nfeatures',2);
%% Best hyperparameter
X_train_w_best_feature = X_train(:,fs);
%select most optimized params, using Gaussian rbf
kernel
Md1 =
fitcsvm(X_train_w_best_feature,y_train,'KernelFunctio
n','rbf','OptimizeHyperparameters','auto',...

'HyperparameterOptimizationOptions',struct('Acquisit
ionFunctionName',...
      'expected-improvement-
plus','ShowPlots',true)); % Bayes' Optimization

  %% Final test with test set
X_test_w_best_feature = X_test(:,fs);
%checking accuracy of the model
test_accuracy_for_iter =
sum((predict(Md1,X_test_w_best_feature) ==
y_test))/length(y_test)*100
%% Plotting results hyperplane
figure;
hgscatter =
gscatter(X_train_w_best_feature(:,1),X_train_w_best_
feature(:,2),y_train);
hold on;
h_sv=plot(Md1.SupportVectors(:,1),Md1.SupportVectors
(:,2),'ko','markersize',8);


% test set data

gscatter(X_test_w_best_feature(:,1),X_test_w_best_fe
ature(:,2),y_test,'rb','xx')

% decision plane
XLIMs = get(gca,'xlim');
YLIMs = get(gca,'ylim');
[xi,yi] =
meshgrid([XLIMs(1):0.01:XLIMs(2)],[YLIMs(1):0.01:YLI
Ms(2)]);
dd = [xi(:), yi(:)];
pred_mesh = predict(Md1, dd);
redcolor = [1, 0.8, 0.8];
bluecolor = [0.8, 0.8, 1];
pos = find(pred_mesh == 1);
h1 = plot(dd(pos,1),
dd(pos,2),'s','color',redcolor,'Markersize',5,'Marke
rEdgeColor',redcolor,'MarkerFaceColor',redcolor);
pos = find(pred_mesh == 2);
h2 = plot(dd(pos,1),
dd(pos,2),'s','color',bluecolor,'Markersize',5,'Mark
erEdgeColor',bluecolor,'MarkerFaceColor',bluecolor);
uistack(h1,'bottom');
uistack(h2,'bottom');
legend([hgscatter;h_sv],{'Class Label 0','Class
Label 1','support vectors'})
```

D. SVM PTO Code

```
clear; close all; clc;

%% preparing dataset

load ('Category PTO.mat');
load ('dataset_PTO.mat');

%Select x =data features , y= label category
x= [ PTO(:,1) PTO(:,2) PTO(:,3) PTO(:,4) PTO(:,5)
PTO(:,6) PTO(:,7) PTO(:,8) PTO(:,9) PTO(:,10)
PTO(:,11) PTO(:,12) PTO(:,13) PTO(:,14) PTO(:,15)
PTO(:,16) PTO(:,17) PTO(:,18) PTO(:,19) PTO(:,20)
PTO(:,21) PTO(:,22) PTO(:,23) PTO(:,24) PTO(:,25)
PTO(:,26) PTO(:,27) PTO(:,28) PTO(:,29) PTO(:,30)
PTO(:,31) PTO(:,32) ];
y= [Category(:,1)];

rand_num = randperm(size(x,1));      %permutation
reordering  and randomly sorting  the data samples
X_train =
x(rand_num(1:round(0.8*length(rand_num))),:);
%select 80% from the randomly sorted data samples as
training dataset
y_train =
y(rand_num(1:round(0.8*length(rand_num))),:);

%select 20% of the randomly sorted data as test
dataset
X_test =
x(rand_num(round(0.8*length(rand_num))+1:end),:);
y_test =
y(rand_num(round(0.8*length(rand_num))+1:end),:);
%% CV partition
c = cvpartition(y_train,'k',148);  %select 20% from
the training dataset as cross validation data set
```

```
%% feature selection

opts = statset('display','iter');
classf = @(train_data, train_labels, test_data,
test_labels)...
    sum(predict(fitcsvm(train_data,
train_labels,'KernelFunction','rbf'), test_data) ~=
test_labels);
%select the  most useful features
[fs, history] = sequentialfs(classf, X_train,
y_train, 'cv', c, 'options', opts,'nfeatures',2);
%% Best hyperparameter
X_train_w_best_feature = X_train(:,fs);
%select most optimized params, using Gaussian rbf
kernel
Md1 =
fitcsvm(X_train_w_best_feature,y_train,'KernelFuncti
on','rbf','OptimizeHyperparameters','auto',...

'HyperparameterOptimizationOptions',struct('Acquisit
ionFunctionName',...
      'expected-improvement-
plus','ShowPlots',true)); % Bayes' Optimization

  %% Final test with test set
X_test_w_best_feature = X_test(:,fs);
%checking accuracy of the model
test_accuracy_for_iter =
sum((predict(Md1,X_test_w_best_feature) ==
y_test))/length(y_test)*100
%% Plotting results hyperplane

figure;
hgscatter =
gscatter(X_train_w_best_feature(:,1),X_train_w_best_
feature(:,2),y_train);
hold on;
h_sv=plot(Md1.SupportVectors(:,1),Md1.SupportVectors
(:,2),'ko','markersize',8);


% test set data

gscatter(X_test_w_best_feature(:,1),X_test_w_best_fe
ature(:,2),y_test,'rb','xx')

% decision plane
XLIMs = get(gca,'xlim');
YLIMs = get(gca,'ylim');
[xi,yi] =
meshgrid([XLIMs(1):0.01:XLIMs(2)],[YLIMs(1):0.01:YLI
Ms(2)]);
dd = [xi(:), yi(:)];
pred_mesh = predict(Md1, dd);
redcolor = [1, 0.8, 0.8];
bluecolor = [0.8, 0.8, 1];
pos = find(pred_mesh == 1);
h1 = plot(dd(pos,1),
dd(pos,2),'s','color',redcolor,'Markersize',5,'Marke
rEdgeColor',redcolor,'MarkerFaceColor',redcolor);
pos = find(pred_mesh == 2);
h2 = plot(dd(pos,1),
dd(pos,2),'s','color',bluecolor,'Markersize',5,'Mark
erEdgeColor',bluecolor,'MarkerFaceColor',bluecolor);
uistack(h1,'bottom');
uistack(h2,'bottom');
legend([hgscatter;h_sv],{'Class Label 0','Class
Label 1','support vectors'})
```