GE 461 Introduction to Data Science
Project 4 Report
Oğuz Altan - 21600966

In this assignment, we are given 556 samples of motor actions along with respective action labels "F" or "NF". The 306 features from the sensors that reflect various properties including velocity, acceleration, temperature and pressure.

Part A)

In this part, we explore the data. We try to infer whether samples of sensor data form segregated clusters in the space spanned by the sensor features. Firstly, we do principal component analysis (PCA) to project the data from the originally 306 features to 2 features to be able to visualize data easily. We extract the top two principal components (PCs) and note how much variance they capture. Implementing PCA, the sum of percentage variances contributed by top two PCs is calculated as $\mathbf{83.8188}\%$. In other words, the variance contributed by the top two PCs is 83.8188% of all components. As high variance represents high and useful information about the data, we can say that while projecting data to 2D using the top two PCs, we do not compromise information provided by data and now, we are able to easily visualize and cluster data to infer whether samples of sensor data form segregated clusters in the space spanned by the sensor features. Therefore, applying PCA and using the top two PCs is a good choice. To implement PCA, we first extract the data from the dataset file and then apply mean normalization. The mean normalization method is subtracting the mean of data from the data. After getting normalized data, we perform PCA and get the plot of data in 2D as it follows:

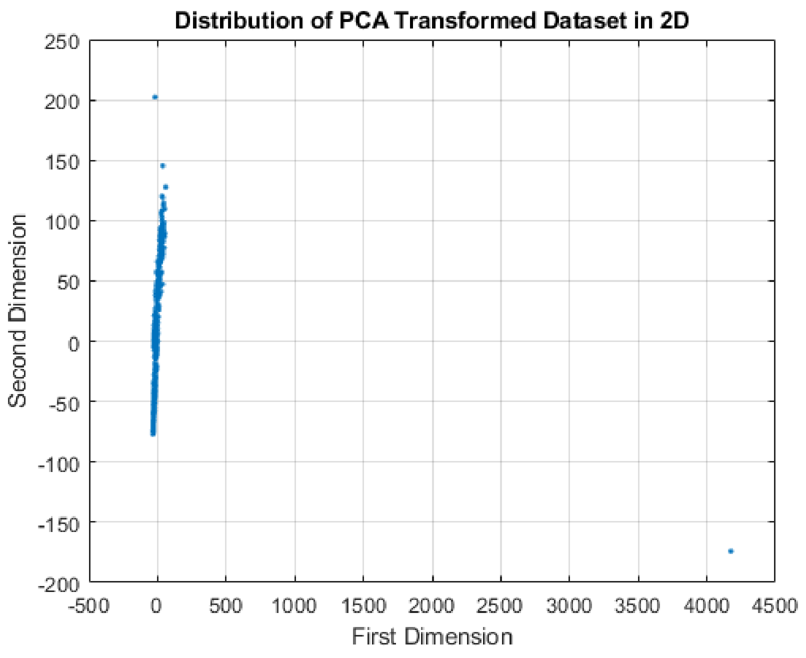

Figure 1: Scatter Plot of Data Reduced to 2D by PCA

Analyzing the distribution of samples, we can see that points are very close to each other, although there are several points far from the mass, formed by the majority of the points.

Now, we can use k-means clustering to separate data into clusters. To find the optimal number of clusters (N) that we can separate our data into, we use Davies–Bouldin index (DBI) and this method gives the optimal number of clusters as **N=2**. The plot of k-means clustering using 2 clusters is given below:

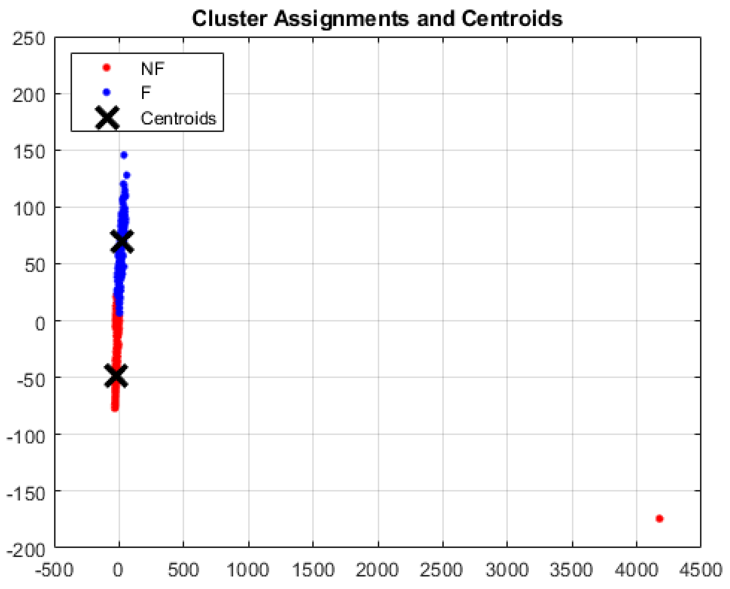

Figure 2: Clustering into 2 Clusters with k-means Algorithm

From the plot, we can see that our data can be separated into 2 clusters easily. To check the success of the k-means algorithm into 2 clusters, we look at the degree of percentage overlap/consistency between the cluster memberships and the action labels originally provided. The percentage overlap is calculated as **77.3852%.** Another way to check the success of clustering is by looking at the silhouette. Silhouette analysis can be used to study the separation distance between the resulting clusters. The silhouette plot displays a measure of how close each point in one cluster is to points in the neighboring clusters and thus provides a way to assess parameters like the number of clusters visually. Silhouette plot of the clusters is given as it follows:

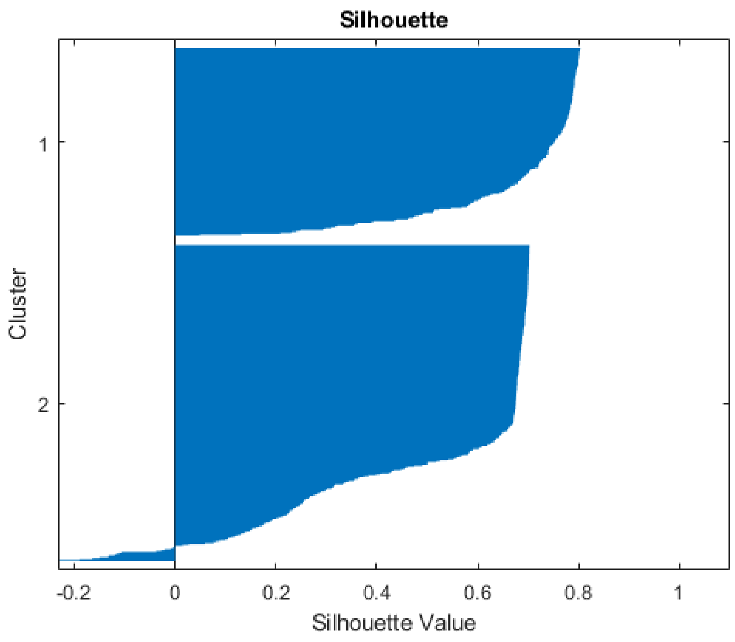

Figure 3: Silhouette Analysis of 2 Clusters

The silhouette values show how similar an object is to its cluster compared to other clusters. The silhouette ranges from −1 to +1, where a high value indicates that the object is well matched to its cluster and poorly matched to neighboring clusters. If most objects have a high value, then the clustering configuration is appropriate. If many points have a low or negative value, then the clustering configuration may have too many or too few clusters [1]. From the plot, analyzing silhouette values, we can see that this clustering is successful and convenient to apply, as the silhouette values are high and reasonable.

As a conclusion, based on these measurements, which are applying PCA and then using k-means method to separate data into two clusters, fall detection is possible, as data can be separated into two clusters and percentage overlap and Silhouette analysis show that this clustering is successful.

Part B)

As we have finished the exploration of the data, we can now proceed with the supervised learning stage. We aim to build a classifier that detects the action label (fall or non-fall) with high accuracy. We will use two different models for this purpose. For both methods, we implement a cross-validation procedure by splitting our data into three non-overlapping training/validation/testing sets, the percentages used for this aim is 70%, 15%, 15%, respectively. As a result, training, validation and testing data sets have 396, 85 and 85 samples, respectively, summing up to a total of 566 samples. As the dataset is shuffled randomly before separation into three groups, these three groups may not have the same samples on each run of the program. This supports the models to work robustly.

The models with various hyperparameters are fit on the training set, the parameter selection is performed based on the validation set, and the final classification accuracy is reported on the testing set.

The first model we will work on is Support Vector Machine (SVM) classifier. This classifier tries to find an optimal hyperplane. As we need to estimate class either “F” or “NF”, meaning we work on binary classification, SVM is to be considered. In this case, SVM tries to find an optimal line that separates data into two parts, where each class lays in each class lay on either side.

After working on hyperparameters, as well as training and testing the model using the cross-validation procedure, we get support vectors as given on the plot below:

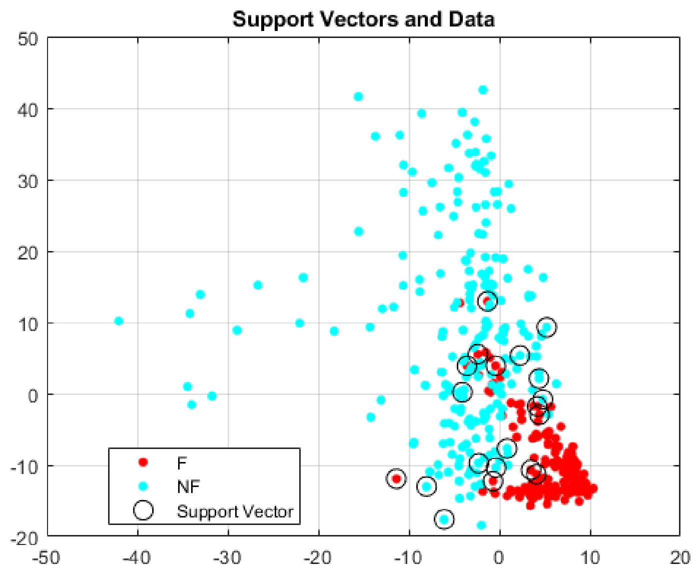

Figure 4: Support Vectors and Data

From the plot, we can see the support vectors and samples belonging to either class “F” or class “NF”. The accuracy of this SVM model is computed as **98.8235%,** with other words, the SVM model can perform classification with 98.8235% accuracy.

Now, as we have presented SVM, we proceed with multilayer perceptron (MLP) model for the same goal, binary classification. For this purpose, we use an artificial neural network (ANN) with a single hidden layer, with 100 hidden layer neurons. In this classifier ANN with a single hidden layer, the configuration found using cross-validation procedure results in successful estimations. Taking a deeper look at the configuration of our ANN model, initializing weights is done by choosing values randomly from Standard Normal Distribution multiplied with a standard deviation chosen as 0.01. The sigmoid activation function is used. Stopping is used to prevent unnecessary computation, as well as overfitting. The algorithm stops if the difference between two loss values for two consecutive iterations is smaller than a threshold called stop_diff having the value 0.00001. Normalization is used, this process increases the success of the network and affects the learning process for this application in a good way. Mean normalization is used. Losses are averaged over the number of data set instances. We expect the loss to be minimized over iterations. The loss function of this ANN is given as it follows:

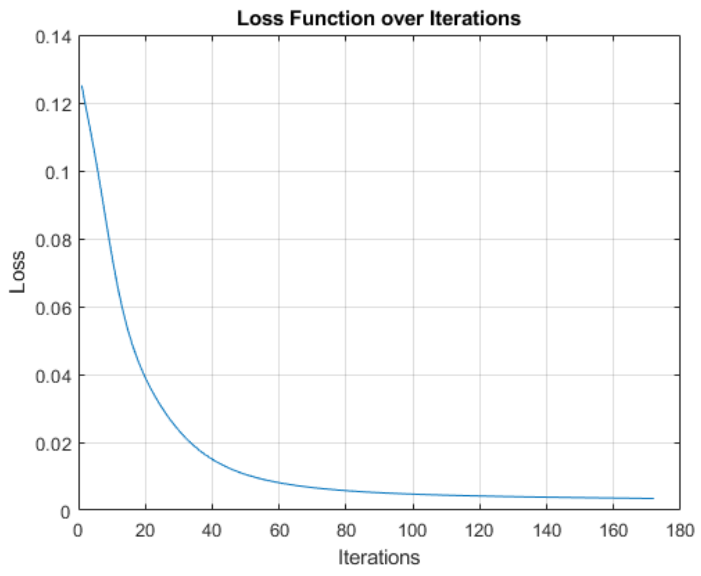

Figure 5: Loss Function Curve for MLP over Iterations

The testing procedure is done by comparing ground truth labels of data and output of the MLP. As the activation function used at the output of MLP is sigmoid, if the output is close to 0, it is estimated to belonging into class "NF", if the output is close to 1, it is estimated to belonging into class "F".

The accuracy of this MLP model is computed as **97.6471%,** in other words, the MLP model can perform classification with 97.6471% accuracy.

As we have represented and worked on SVM and MLP classifier models, we can comparatively evaluate them. Both models are used for binary classification, they both aim to classify samples belonging to either "F" or "NF". For MLP, an artificial neural network (ANN) with a single hidden layer is used. We can see that both models perform very well, their accuracies are very high and nearly perfect. Namely, they can classify the samples very accurately. Comparing SVM and ANN, we can say that they are both very successful and their success rates are nearly equal. As a side note, ANN has a high number of parameters and therefore tweaking these parameters may dramatically affect the performance. As the cross-validation procedure is used, parameter choices are done based on the validation set for both models. As a result, we can use both models for binary classification purposes, as we had in this project.

Commenting on the success of fall detection based on the wearable sensors, we can say that SVM and MLP models are very successful. These models can be used to classify "F" and "NF" samples. Also, by exploring dataset with PCA and k-mean clustering before proceeding with the supervised learning stage, we observed that dataset is prominent and convenient for clustering, therefore binary classification would be seemingly very successful just by looking at PCA and k-means clustering results. As a result, looking at the results of all methods and models we have used in this project, we can state that fall detection based on wearable sensors is very successful, fall and non-fall can be detected with a very high success rate. This example of machine learning in telehealth shows how machine learning is used in health care systems, by implementing a fall detection system that can detect an action of fall or non-fall is performed by the patient or not, using wearable sensor data.

Appendix

A) References

1. Wikipedia contributors. (2020, May 2). Silhouette (clustering). In *Wikipedia, The Free Encyclopedia*. Retrieved 21:06, May 10, 2020, from https://en.wikipedia.org/w/index.php?title=Silhouette_(clustering)&oldid=954469735

B) Source Code

proje4.m

```
clear all;
clc; close all;

%Loading Data
import_data = readtable('falldetection_dataset.csv');

%Preparing Data
dataset = table2array(import_data(:,3:end));
label_string = string(table2array(import_data(:,2)));
label_binary = label_string == "F";
label_binary = double(label_binary);

%Mean Normalization
dataset = dataset - mean(dataset);

%Splitting Data Randomly
rand_samples = randperm(566);
train_dataset = dataset(rand_samples(1:396),:);
label_train = label_binary(rand_samples(1:396),:);

valid_dataset = dataset(rand_samples(397:481),:);
label_valid = label_binary(rand_samples(397:481),:);

test_dataset = dataset(rand_samples(482:566),:);
label_test = label_binary(rand_samples(482:566),:);

%Principal Component Analysis (PCA)
[coeff, score, latent,~, explained, mu0] = pca(dataset,'NumComponents',2);
%applying PCA

correl_coef = corrcoef(score);

asd = dataset*coeff;

%Plotting PCA result
% figure;
% plot(asd(:,1),asd(:,2),'.');
% grid on;
% title("Distribution of PCA Transformed Dataset in 2D");
% xlabel("First Dimension");
% ylabel("Second Dimension");

figure;
% plot(score(:,1),score(:,2),'.');
scatter(asd(:,1),asd(:,2),'.');
```

```
grid on;
title("Distribution of PCA Transformed Dataset in 2D");
xlabel("First Dimension");
ylabel("Second Dimension");

sum_variance = sum(explained(1:2));
disp("Total variance percentage contributed by top 2 PCs is " +
sum_variance + "%");

cluster_dataset = asd;
cluster_dataset = score;

%K-means clustering
eva = evalclusters(score,'kmeans',"DaviesBouldin",'KList',[1:10]);
disp("Optimal number of clusters found by Davies-Bouldin index (DBI) is N =
" + eva.OptimalK);

opts = statset('Display','final');
[idx,C] = kmeans(cluster_dataset,2,'Distance','cityblock',...
    'Replicates',5,'Options',opts);
figure;
plot(cluster_dataset(idx==1,1),cluster_dataset(idx==1,2),'r.','MarkerSize',
12)
hold on
plot(cluster_dataset(idx==2,1),cluster_dataset(idx==2,2),'b.','MarkerSize',
12)
plot(C(:,1),C(:,2),'kx',...
    'MarkerSize',15,'LineWidth',3)
grid on
legend('NF','F','Centroids',...
    'Location','NW')
title 'Cluster Assignments and Centroids'
hold off

%Checking success of k-means accuracy
figure;
silhouette(cluster_dataset,idx,"Euclidean");
title("Silhouette");

%Percentage overlap between the cluster memberships and the action labels
overlap = 0;
idx_test = idx-1;

for i = 1:size(label_binary,1)
    if idx_test(i) == label_binary(i)
        overlap = overlap+1;
    end
end

overlay_perc = overlap*100/size(label_binary,1);

if (overlay_perc  < 50)
    overlay_perc = 100-overlay_perc;
end

disp("Percentage overlap between the cluster memberships and the action
labels is "...
    + overlay_perc + "%");
```

```
%Support Vector Machine (SVM)
SVMModel = fitcsvm(train_dataset,label_train);
classOrder = SVMModel.ClassNames;

sv = SVMModel.SupportVectors;
figure
gscatter(train_dataset(:,1),train_dataset(:,2),label_train)
hold on
plot(sv(:,1),sv(:,2),'ko','MarkerSize',10)
legend('F','NF','Support Vector')
title("Support Vectors and Data");
grid on
hold off

CompactSVMModel = compact(SVMModel);
% whos('SVMModel','CompactSVMModel');

%Testing SVM Model
labelz = predict(CompactSVMModel, test_dataset);
correct_test = 0;
for i = 1:85
    if(labelz(i) == label_test(i))
        correct_test = correct_test+1;
    end
end
correctness = correct_test*100/length(label_test);
disp("SVM model can predict with " + correctness + "% accuracy");

%Multilayer Perceptron (MLP)

%Network parameters
N_hidden = 100; %Number of hiddden layer neurons
std_hid = 0.01;
std_out = 0.01;
W_hid = std_hid*randn(N_hidden,(size(train_dataset,2)+1)); %Gaussian noise
weights
W_out = std_out*randn(1,N_hidden+1);
learning_rate = 0.0001;

loss_vec = [];
bias = ones(1,length(train_dataset));
max_epoch = 1000;
stop_diff = 0.00001; % Threshold difference between two consecutive loss

for i = 1:max_epoch
    if i > 5
        if (loss_vec(i-2)-loss_vec(i-1)) > stop_diff

            %Feed forward and loss calculation
            x_hid = train_dataset;
            x_hid = [x_hid bias'];
            v_hid = W_hid*x_hid';
            y_hid = sigmoid(v_hid);
            deriv_act = sigmoid(v_hid) .* (1 - sigmoid(v_hid));
            y_hid = [y_hid; bias];
            v_out = W_out*y_hid;
            y_out = sigmoid(v_out);
            error = (label_train-y_out');
            loss = 1/2*(sum(error.^2));
            loss = loss/length(train_dataset);

            %Gradient calculation
            delta_out = y_hid*error;
            er_hid = ((error*W_out)').*(y_hid.*(1-y_hid));

            delta_hidden = ((er_hid(1:end-1,:))*x_hid);

            %Weight update
            W_out = W_out + learning_rate*delta_out';
            W_hid = W_hid + learning_rate*delta_hidden;

            loss_vec = [loss_vec loss]; %Storing loss for each iteration
into a vector
        else
            break
        end
    else
        %Feed forward and loss calculation
        x_hid = train_dataset;
        x_hid = [x_hid bias'];
        v_hid = W_hid*x_hid';
        y_hid = sigmoid(v_hid);
        deriv_act = sigmoid(v_hid) .* (1 - sigmoid(v_hid));
        y_hid = [y_hid; bias];
        v_out = W_out*y_hid;
        y_out = sigmoid(v_out);

        error = (label_train-y_out');
        loss = 1/2*(sum(error.^2));
        loss = loss/length(train_dataset);

        %Gradient calculation
        delta_out = y_hid*error;
        er_hid = ((error*W_out)').*(y_hid.*(1-y_hid));

        delta_hidden = ((er_hid(1:end-1,:))*x_hid);

        %Weight update
        W_out = W_out + learning_rate*delta_out';
        W_hid = W_hid + learning_rate*delta_hidden;

        loss_vec = [loss_vec loss]; %Storing loss for each iteration into a
vector
    end
end

figure;
plot(1:length(loss_vec),loss_vec);
grid on;
title("Loss Function over Iterations");
xlabel("Iterations");
ylabel("Loss");

%Validation and Hyperparameter Selection
```

```
%Network parameters
N_hidden_valid = 100; %Number of hiddden layer neurons
std_hid_valid = 0.01;
std_out_valid = 0.01;
W_hid_valid =
std_hid_valid*randn(N_hidden_valid,(size(valid_dataset,2)+1)); %Gaussian
noise weights
W_out_valid = std_out_valid*randn(1,N_hidden_valid+1);
learning_rate_valid = 0.001;

loss_vec_valid = [];
bias_valid = ones(1,size(valid_dataset,1));
max_epoch_valid = 1000;
stop_diff_valid = 0.00001; % Threshold difference between two consecutive
loss

for i = 1:max_epoch_valid
    if i > 3
        if (loss_vec_valid(i-2)-loss_vec_valid(i-1)) > stop_diff_valid

            %Feed forward and loss calculation
            x_hid_valid = valid_dataset;
            x_hid_valid = [x_hid_valid bias_valid'];
            v_hid_valid = W_hid_valid*x_hid_valid';
            y_hid_valid = sigmoid(v_hid_valid);
            deriv_act_valid = sigmoid(v_hid_valid) .* (1 -
sigmoid(v_hid_valid));
            y_hid_valid = [y_hid_valid; bias_valid];
            v_out_valid = W_out_valid*y_hid_valid;
            y_out_valid = sigmoid(v_out_valid);

            error_valid = (label_test-y_out_valid');
            loss_valid = 1/2*(sum(error_valid.^2));
            loss_valid = loss_valid/length(valid_dataset);

            %Gradient calculation
            delta_out_valid = y_hid_valid*error_valid;
            er_hid_valid = ((error_valid*W_out_valid)').*(y_hid_valid.*(1-
y_hid_valid));

            delta_hidden_valid = ((er_hid_valid(1:end-1,:))*x_hid_valid);

            %Weight update
            W_out_valid = W_out_valid +
learning_rate_valid*delta_out_valid';
            W_hid_valid = W_hid_valid +
learning_rate_valid*delta_hidden_valid;

            loss_vec_valid = [loss_vec_valid loss_valid]; %Storing loss for
each iteration into a vector
        else
            break
        end
    else
        %Feed forward and loss calculation
        x_hid_valid = valid_dataset;
        x_hid_valid = [x_hid_valid bias_valid'];
        v_hid_valid = W_hid_valid*x_hid_valid';
        y_hid_valid = sigmoid(v_hid_valid);
        deriv_act_valid = sigmoid(v_hid_valid) .* (1 -
sigmoid(v_hid_valid));
        y_hid_valid = [y_hid_valid; bias_valid];
        v_out_valid = W_out_valid*y_hid_valid;
        y_out_valid = sigmoid(v_out_valid);

        error_valid = (label_test-y_out_valid');
        loss_valid = 1/2*(sum(error_valid.^2));
        loss_valid = loss_valid/length(valid_dataset);

        %Gradient calculation
        delta_out_valid = y_hid_valid*error_valid;
        er_hid_valid = ((error_valid*W_out_valid)').*(y_hid_valid.*(1-
y_hid_valid));

        delta_hidden_valid = ((er_hid_valid(1:end-1,:))*x_hid_valid);

        %Weight update
        W_out_valid = W_out_valid + learning_rate_valid*delta_out_valid';
        W_hid_valid = W_hid_valid + learning_rate_valid*delta_hidden_valid;

        loss_vec_valid = [loss_vec_valid loss_valid]; %Storing loss for
each iteration into a vector
    end
end

figure;
plot(1:length(loss_vec_valid),loss_vec_valid);
grid on;
title("Loss for MLP with Validation Data over Iterations");
xlabel("Iterations");
ylabel("Loss");

% Testing
W_hid_test = W_hid;
W_out_test = W_out;
accur = 0;

for i = 1:size(test_dataset,1)

    x_hid_test = test_dataset(i,:);
    x_hid_test = [x_hid_test 1];
    v_hid_test = W_hid_test*x_hid_test';
    y_hid_test = sigmoid(v_hid_test);
    y_hid_test = [y_hid_test; 1];
    v_out_test = W_out_test*y_hid_test;
    y_out_test = sigmoid(v_out_test);

    if y_out_test < 0.5
        if(label_test(i) == 0)
            accur = accur+1;
        end
    else
        if(label_test(i) == 1)
            accur = accur+1;
        end
    end
end

correctness = accur/size(test_dataset,1)*100;
disp("Multi Layer Perceptron model can predict with " + correctness + "%
accuracy");

function sigmoidOut = sigmoid(v)
sigmoidOut = 1./(1+exp(-v));
end
```