EECE 5136/6036: Intelligent Systems

Report-2

Sanjana P Shetty

10/17/2017

Problem 1

Realizing KNN classifier and neighborhood based classifier.

Plots obtained were:

1. Hit rate value of KNN for k={1,3,5,7,9,11}
2. Hit rate value of neighborhood-based classifier R={1:2:224}

Results

1. *Hit Rate value of KNN*

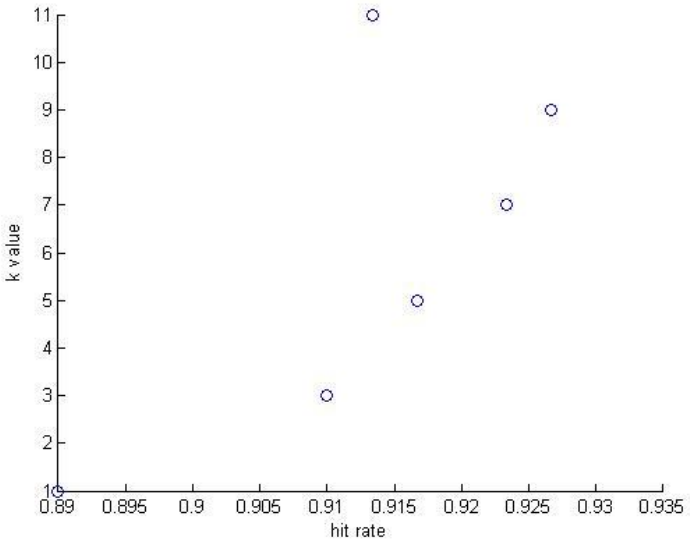


Figure 1 Hit rate vs k value

2. *Hit rate value of NC with radius*

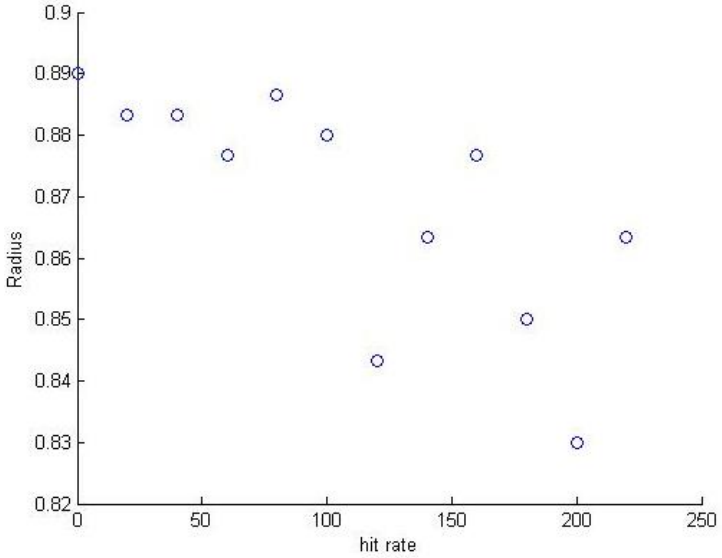


Figure 2 Hit rate vs radius

Discussion

The kNN algorithm is widely used in neural networks. In this the following steps are followed:

1. euclidian distances of individual points from every other point in the data set are calculated and stored.
2. Each point of the 300 points is considered; the distances corresponding to that point are sorted in ascending order.
3. The distances are assessed and the nearest k neighbors are considered;
4. The consequent D values are obtained and the number of positive neighbors is compared to the negative neighbors. The one with the higher value is assigned to the point under test
5. This process is repeated for 6 values of k and the k with highest efficiency is determined.

Figure 5 membrane potential vs time step I=15

The algorithm followed for nearest neighbor classifier:

Steps 1 and 2 are the same as followed in KNN

3. A radius is considered and all the neighbors within this radius are assessed.

4. The consequent D values are obtained and the number of positive neighbors is compared to the negative neighbors. The one with the higher value is assigned to the point under test
5. This process is repeated for an array of radius values and the radius with highest efficiency is determined.
6. Any radius with no neighbors enclosed is considered as positive.

Conclusion

The most efficient value of k is considered to be 9 which gives a 93% hit rate and the most efficient radius is considered as the 80 which provides an 88% hit rate.

Problem 2

The Perceptron model was implemented with the data split into a 80/20 set. The number of epochs was varied to find the optimum number.

Plots obtained were:

1. Error rate for training Set
2. Error rate for test Set

Results

1. Error rate for training set and test Set

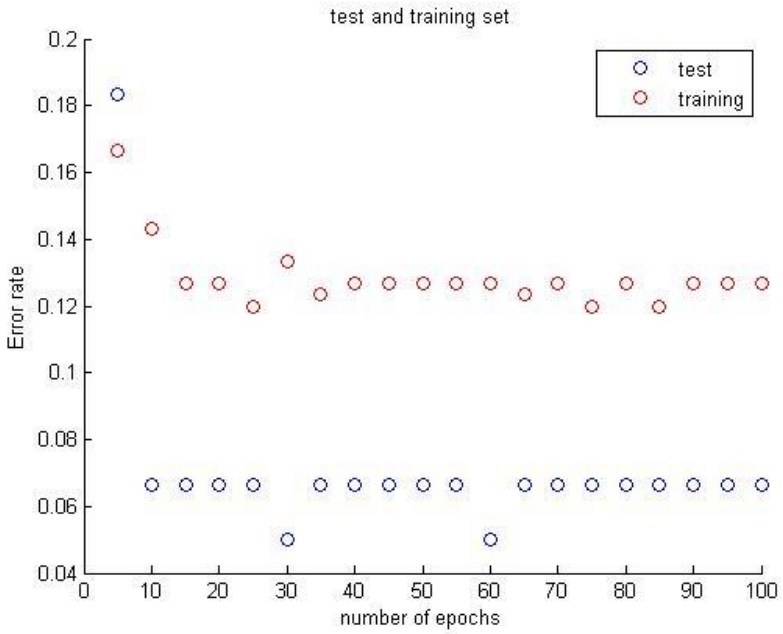


Figure 4 E-train of Test and training Set vs Epoch number

Discussion

The Perceptron was realized as follows:

1. The data is first normalized between 0 and 1 to suit the algorithm.
2. The data was divided into data set and training set (80/20)
3. Arbitrary weights and bias (w1=w2=b0) are selected for the values. Learning rate was initialized as 0.01
4. The training data is first evaluated for a fixed number of epochs.
5. The data set is manipulated by the weights and compared to a threshold value. According to this value, the data is sorted
6. The sorted data is simultaneously compared with the expected output; if a mismatch is detected, the weights are readjusted accordingly.
7. The weight is then fixed in the last epoch and the test set is evaluated using these weights.
8. A range of epoch values were evaluated and the best value was selected

Conclusion

A hit rate of as high as 93% was observed in the perceptron (error rate=6%) is obtained after the test set. The number of epochs. has been chosen as 10 after the initial trial runs because after 10 iterations, the error almost becomes constant.

Problem 3

The best parameters are selected from the previous problem and a random dataset was implemented on all three algorithms.

Various plots were obtained

Results

1. Perceptron features, before and after training

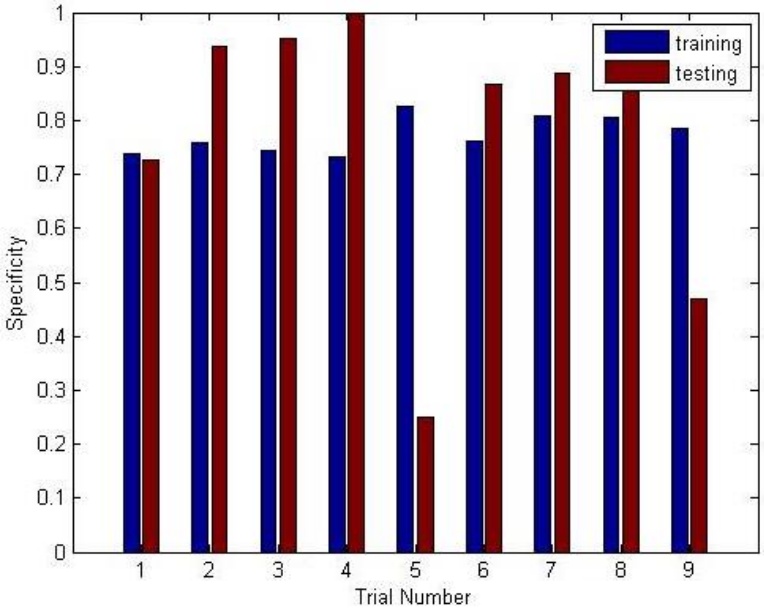


Figure 5 Specificity vs trial number

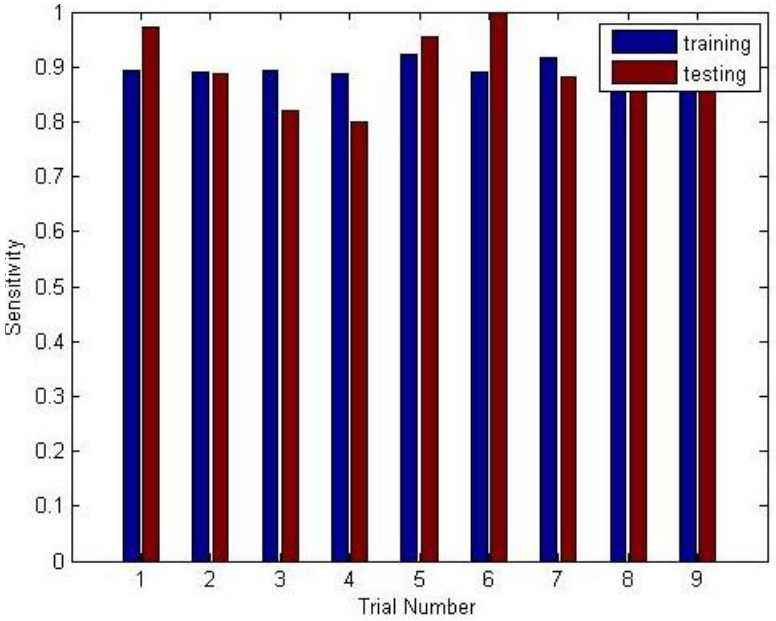


Figure 6 Sensitivity vs trial number

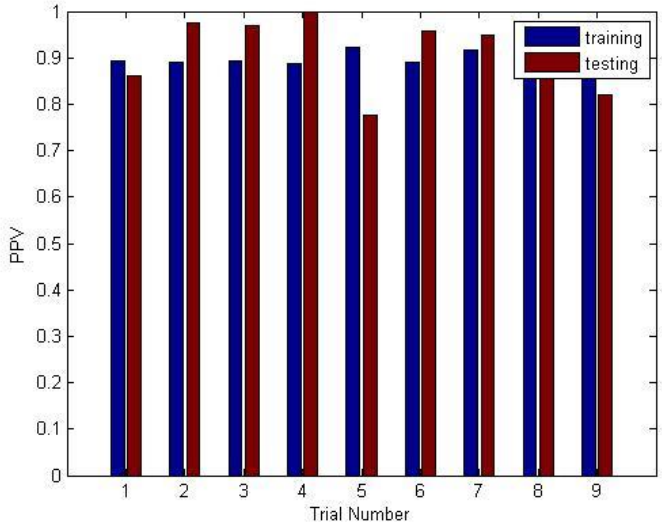


Figure 9 PPV vs trial number

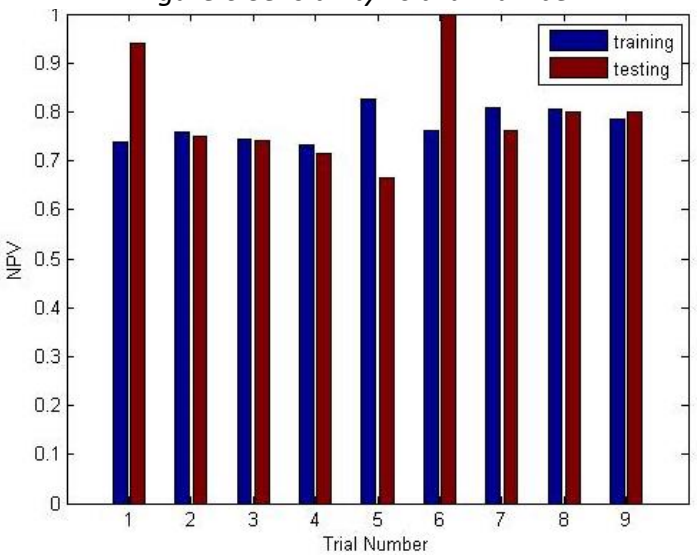


Figure 8 NPV vs trial number

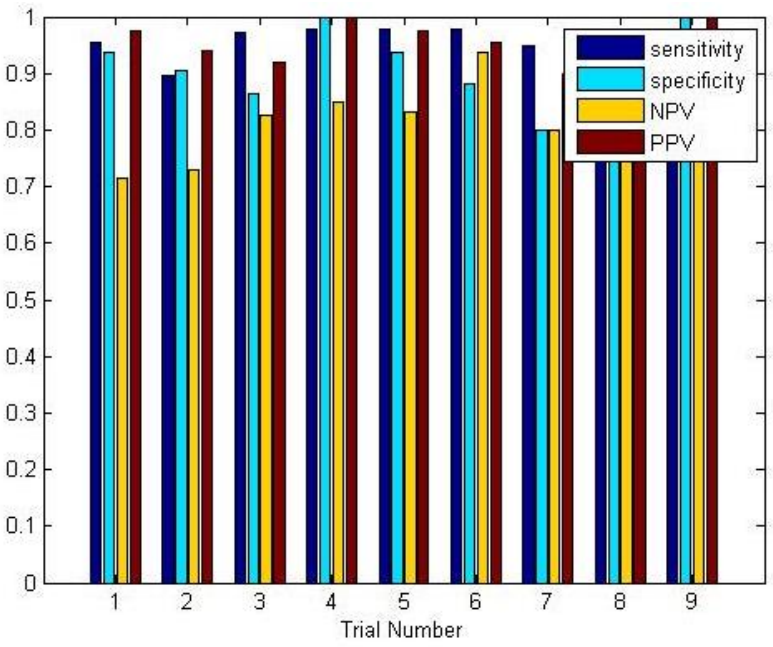


Figure 10 Feature set of nearest neighbor vs trial number

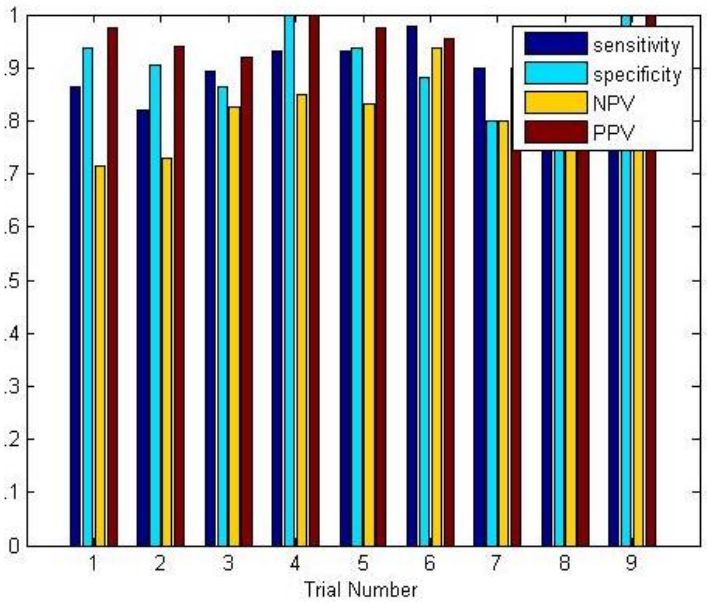
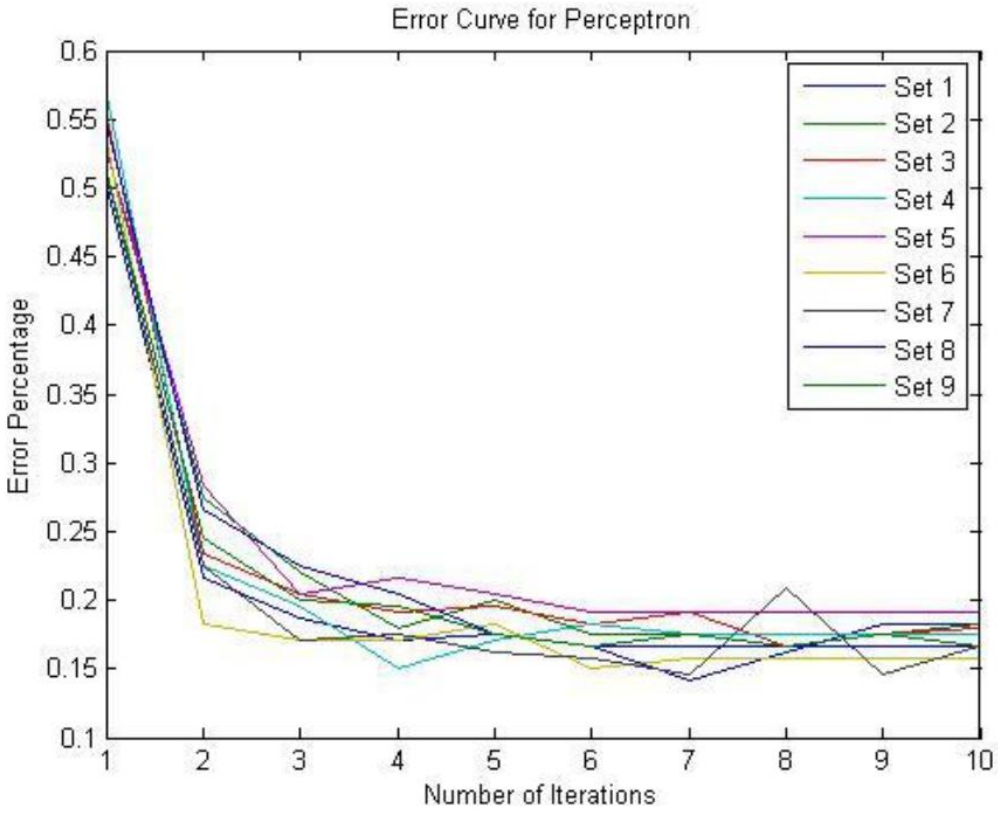


Figure 11 Feature set of KNN vs trial number

2.Table with standard deviation and mean of all feature values

	Mean					Standard deviation				
	NPV	Sensitivity	specificity	PPV	hit rate	NPV	Sensitivity	specificity	PPV	hit rate
KNN	0.118036	0.05425	0.0837	0.033901	0.04088	0.7964	0.8967	0.9086	0.9574	0.9
Nearest neighbor	0.082834	0.02718	0.0590	0.0220	0.02222	0.8639	0.9469	0.7729	0.9040	0.89259
Perceptron	0.115425	0.08047	0.189878	0.0615	0.03017	0.6968	0.7951	0.8788	0.9644	0.82592

2.Error Calculation of perceptron over 10 epochs



Analysis of Result (k-NN , Nearest Neighbor Perceptron):

Perceptron: The perceptron classifier improves with the number of iterations to an extent. After that, the error starts to increase. The low error rates for all the trials indicates that the algorithm is quite robust. The performance of individual trials for all of the trials did not vary too much.
Pros: Simple to understand, easy to implement, very logical **Cons:** With a single layer perceptron the scope of error is more
k-NN: The k-NN algorithm on the other hand does not seem to be very robust. The performance of individual trials shows quite a bit of variation. The error with respect to the weights decrease with increase in the number of iterations to an extent. After that error starts to increase.
Pros: Easy to understand, very straightforward, logical
Cons: Slightly more difficult to implement, selection of k is always a problem
Nearest Neighbor: The nearest neighbor algorithm was tougher to execute than the k-NN, but easier than the perceptron. The average hit rate from the beginning was not impressive and it is observed to be prone to error. The error further increases on randomization of dataset.
Pros: Easy to understand, covers a wide range and does not confine to numbers
Cons: Slightly more difficult to implement, radius variance plays a factor

Recommendation: I would recommend the Perceptron because it is easier to implement than the augmented k-NN and nearest neighbor algorithms. The k-NN and nearest neighbor algorithms are not as robust as the Perceptron due to the distance calculation problem. The number of k and radius to be taken into consideration is never clear. Concepts like cross-validation do help to an extent but not completely. The learning rate in the perceptron can be lowered and the number of iterations can be increased to attain greater accuracy.

Code

```
Clear all;
data = xlsread('C:\Users\sanjana\Documents\intelligent systems\HW2\HW2\data.xlsx');
data1(:,1)=data(:,1);
data1(:,2)=data(:,2);
data2(:,1)=data(:,3);


%%%%%%%%%%%%%%%%%%%%%% arrays needed to store hit rate and error%%%%%%%%
hit_rate_knn=[];
hit_rate_rad=[];
hit_rate_percep=[];
error_rate_knn=[];
erro_rate_rad=[];
erro_rate_percep=[];

%%%%%%%%%%%%%%%% sensi_tivity, NPV, PPV, specificity%%%%%%%%%%%%%%%
sensi_tivity=[];
specificity=[];
NPV=[];
PPV=[];



for trial=1:9

    TotalSet=zeros(300,3);
    TrainingSet=zeros(240,2);
    TestSet=zeros(60,2);
    count=1;
 %%%%%%%%%% generate array of random numbers%%%%%%%
 for i=1:300
test_1(i)=i;
end
for i=1:300
test_2=test_1(randperm(length(test_1)));
end

%%%%%%%%% assign values to training set and test set%%%%%%%

for i=1:300
    TotalSet_percep(i,1:2)= data1(test_2(i),:);
    TotalSet_percep(i,3) = data2(test_2(i),:);
end

for i=1:240
    TrainingSet(i,:)= data1(test_2(i),:);
    TrainingSet2(i,:)= data2(test_2(i),:);
end
for i=241:300
    TestSet(count,:)= data1(test_2(i),:);
    TestSet2(count,:)= data2(test_2(i),:);
    count=count+1;

end
%
 TotalSet(1:240,1:2)= TrainingSet(1:240,:);
```

```
  TotalSet(241:300,1:2)=  TestSet(1:60,:);
  TotalSet(1:240,3)= TrainingSet2(1:240,1);
  TotalSet(241:300,3)=  TestSet2(1:60,1);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% KNN%%%%%%%%%%%%%%%%%%%%%%%%%%%%


j=1; i=1;
eucdist_sq=[];
temp=zeros(2,300);
data3=[];
k=9;                          %%%%%%best value of k
negative=zeros(300,6);
positive=zeros(300,6);

negative_rad=zeros(300);
positive_rad=zeros(300);

final_result=[];

final_result_rad=[];


for j=1:60
for i=1:240
    if j~=i
        eucdist_sq(i,j)=((TrainingSet(i,1)-TestSet(j,1))^2)+((TrainingSet(i,2)-
TestSet(j,2))^2);

    else

        eucdist_sq(i,j)=0;
    end;
end;
end;

    for c=1:60          %positive and negatives for each value; c corresponds to the
value
         tes=k;
         j=1;
        TrainingSet2(:,2)=eucdist_sq(:,c);
        data3=sortrows(TrainingSet2,2);
    while(tes~=0)
    if(data3(j,2)~=0)
       if(data3(j,1)==1)
        positive(c)=positive(c)+1;
       else
           negative(c)=negative(c)+1;
       end
       tes=tes-1;
    end

    j=j+1;



    end
```

```
    end
   % hit_rate(i)= (sum(positive) +sum(negative))/



%%%%%%%%%%%%%%%%%% hit rate%%%%%%%%%%%%%%%%%%%%%%
    Q10=0;
    Q01=0;
    Q11=0;
    Q00=0;
    for j=1:60
        if positive(j)>=negative(j)
            final_result(j)=1;
             if TestSet2(j)==final_result(j)
                   Q11=Q11+1;
             else
                 Q01=Q01+1;
             end
        else
            final_result(j)=0;
            if TestSet2(j)==final_result(j)
                   Q00=Q00+1;
             else
                 Q10=Q10+1;
            end
        end

    end

    hit_rate_knn(trial)=(Q11+Q00)/(Q10+Q01+Q11+Q00);
    sensi_tivity(trial,1)= (Q11)/(Q11+ Q10);
    specificity(trial,1)= (Q00)/(Q01+ Q00);
    PPV(trial,1)= (Q11)/(Q11+ Q01);
    NPV(trial,1)= (Q00)/(Q00+Q10);
    error_rate_knn(trial)=1-hit_rate_knn(trial);

Y_mean=mean(mean(eucdist_sq));
Y_min=min(min(eucdist_sq(eucdist_sq>0)));
Y_max=max(max(eucdist_sq));
Y_std= std(std(eucdist_sq));

data2_neighbor=zeros();
data3=zeros();
i=1;

radius=0: 20 : Y_std;

i=1;
neighbor=zeros(300);
inc=100;                                  %%%%%%%%%%%%%%%%optimum radius%%%%%%%%%%%%

  i=i+1;
%end


   for c=1:60          %positive and negatives for each value; c corresponds to the value
         tes=k;
         j=1;
         neighbor=0;
```

```
        TrainingSet2(:,2)=eucdist_sq(:,c);
        data3=sortrows(TrainingSet2,2);
     while(data3(j,2)<=inc)
          if(data3(j,2)~=0)
              neighbor=neighbor+1;
              if(neighbor>300)
                  break;
              end
             if(data3(j,1)==1)
                  positive_rad(c)=positive_rad(c)+1;
             else
                  negative_rad(c)=negative_rad(c)+1;
             end
          end

    j=j+1;



    end
   if neighbor==0
       neighbor=1;
        positive_rad(c)=1;

        negative_rad(c)=0;
   end


end

    Q10=0;
    Q01=0;
    Q11=0;
    Q00=0;
    for j=1:60
        if positive_rad(j)>=negative_rad(j)
            final_result_rad(j)=1;
             if TestSet2(j)==final_result_rad(j)
                   Q11=Q11+1;
             else
                 Q01=Q01+1;
             end
        else
            final_result_rad(j)=0;
            if TestSet2(j)==final_result_rad(j)
                   Q00=Q00+1;
             else
                 Q10=Q10+1;
            end
        end
    end
           hit_rate_rad(trial)=(Q11+Q00)/(Q10+Q01+Q11+Q00);
           sensi_tivity(trial,2)= (Q11)/(Q11+ Q10);
            specificity(trial,2)= (Q00)/(Q01+ Q00);
            PPV(trial,2)= (Q11)/(Q11+ Q01);
            NPV(trial,2)= (Q00)/(Q00+Q10);
            error_rate_rad(trial)=1-hit_rate_rad(trial);


disp(hit_rate_knn);
disp(hit_rate_rad);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%perceptron%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

TrainingSet=zeros(240,2);
TestingSet=zeros(60,2);
TotalSet=zeros(300,3);


%%%%%%%%%%%%%%%%%%%%normalizing data%%%%%%%%%%%%%%%%%%%

normL=[];
normP=[];

 Q11=0;
 Q10=0;
 Q01=0;
 Q00=0;
 count=1;

%normalize data%
minL= min(TotalSet_percep(:,1));
maxL= max(TotalSet_percep(:,1));
minP= min(TotalSet_percep(:,2));
maxP= max(TotalSet_percep(:,2));

rangeL= maxL-minL;
rangeP= maxP-minP;

for i=1:300
    normL(i,1)= (TotalSet_percep(i,1)-minL)/rangeL;
    normP(i,1)= (TotalSet_percep(i,2)-minL)/rangeP;
end



%%%%%%%%%% assign values to training set and test set%%%%%%%

TrainingEpochError=[];
NoEpochChange=[];
%dividing data into training and testing set%
TrainingSet(:,1)=normL(1:240,1); %normL
TrainingSet(:,2)=normP(1:240,1); %normP

TestingSet(:,1)=normL(241:300,1);
TestingSet(:,2)=normP(241:300,1);


TestingError_1=[];
%TrainingOutput=zeros(1:240,1);
nep=10:10:100;
%initializing weights and bias for training%
w1=0;
w2=0;
b=0;
%initializing learning rate%
n=0.01;
```

```
%initialising inputs and outputs%
y=zeros();
TrainingError=0;
ExpectedOutput=TotalSet_percep(1:240,3);

%training the perceptron for 1:100 epochs%
TrainingEpochError=zeros();
for epoch=1:10 %limit

    x1= TrainingSet(:,1);
    x2= TrainingSet(:,2);
    TrainingOutput=zeros();

for i=1:240%length(TrainingSet)


     y(i)= x1(i)*w1 + x2(i)*w2 + b;

    if(y(i)>=1)
        TrainingOutput(i,1)= 1;
    else
        TrainingOutput(i,1)= 0;
    end;

  TrainingError=  ExpectedOutput(i,1)-TrainingOutput(i,1) ;

  w1= w1 + TrainingError*n*x1(i);
  w2= w2 + TrainingError*n*x2(i);
  b= b + n*(TrainingError);
 end

TrainingEpochError(epoch,trial) = (nnz(ExpectedOutput - TrainingOutput))/240;
end
for r=1:240
    if TrainingOutput(r)==1
        if  TrainingOutput(r)== ExpectedOutput(r)
            Q11=Q11+1;
        else
            Q01=Q01+1;
        end
    else
       if  TrainingOutput(r)== ExpectedOutput(r)
           Q00=Q00+1;
        else
            Q10=Q10+1;
       end
    end
end

   hit_rate_train_percep(trial)=(Q11+Q00)/(Q11+Q10+Q01+Q00);
   sensi_tivity(trial,3)= (Q11)/(Q11+ Q10);
    specificity(trial,3)= (Q00)/(Q01+ Q00);
    PPV(trial,3)= (Q11)/(Q11+ Q01);
    NPV(trial,3)= (Q00)/(Q00+Q10);
    error_rate_train_percep(trial)=1-hit_rate_train_percep(trial);
```

```
%%%%%%%%%testing the perceptron%%%%%%%%%%%

    x1= TestingSet(:,1);
    x2= TestingSet(:,2);
    TestingOutput=zeros(1:60,1);
    y=zeros();
    ExpectedOutput=TotalSet_percep(241:300,3);

for i=1:60%length(TrainingSet)


    y(i)= x1(i)*w1 + x2(i)*w2 + b;

    if(y(i)>=1)
        TestingOutput(i,1)= 1;
    else
        TestingOutput(i,1)= 0;
    end;

  TestingError_1(end+1)= ExpectedOutput(i)- TestingOutput(i,1);

end
TestingErrorPerc = (nnz(ExpectedOutput - TestingOutput))/60;

Q11=0;
 Q10=0;
 Q01=0;
 Q00=0;

for r=1:60
    if TestingOutput(r)==1
        if  TestingOutput(r)== ExpectedOutput(r)
            Q11=Q11+1;
        else
            Q01=Q01+1;
        end
    else
       if  TestingOutput(r)== ExpectedOutput(r)
           Q00=Q00+1;
        else
            Q10=Q10+1;
       end
    end
end

   hit_rate_percep(trial)=(Q11+Q00)/(Q11+Q10+Q01+Q00);
   sensi_tivity(trial,4)= (Q11)/(Q11+ Q10);
    specificity(trial,4)= (Q00)/(Q01+ Q00);
    PPV(trial,4)= (Q11)/(Q11+ Q01);
    NPV(trial,4)= (Q00)/(Q00+Q10);
    error_rate_percep(trial)=1-hit_rate_knn(trial);

end

disp(hit_rate_percep);


%%%%%%%%%%%%%%%%%PLOTS%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%perceptron%%%%%%%%%%%%%%%%%%%%%%
figure('name','Perceptron Sensitivity after training and after testing');
```

```
title('Bar Graph Sensitivity vs Trial Number Perceptron');
 bar([sensi_tivity(1:9,3),sensi_tivity(1:9,4)]);
 xlabel('Trial Number');
ylabel('Sensitivity');
legend('training', 'testing');

figure('name','Perceptron specificity after training and after testing');
title('Bar Graph Specificity vs Trial Number Perceptron');
 bar([specificity(1:9,3),specificity(1:9,4)]);
 xlabel('Trial Number');
ylabel('Specificity');
legend('training', 'testing');

figure('name','Perceptron NPV after training and after testing');
title('Bar Graph NPV vs Trial Number Perceptron');
 bar([NPV(1:9,3),NPV(1:9,4)]);
 xlabel('Trial Number');
ylabel('NPV');
legend('training', 'testing');

figure('name','Perceptron PPV after training and after testing');
title('Bar Graph PPV vs Trial Number Perceptron');
 bar([PPV(1:9,3),PPV(1:9,4)]);
 xlabel('Trial Number');
ylabel('PPV');
legend('training', 'testing');

%%%%%%%%%%%%%%%%%%Nearest neighbor%%%%%%%%%%%%%%%%%%%%%%%%%%
figure('name','Nearest neighbor Sensitivity ');
title('NEC vs Trial Number ');
 bar([sensi_tivity(1:9,2),specificity(1:9,1),NPV(1:9,1),PPV(1:9,1)]);
 xlabel('Trial Number');
legend('sensitivity','specificity','NPV','PPV');


%%%%%%%%%%%%%%%%%%KNN%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%5
figure('name','KNN Sensitivity ');
title('KNN vs Trial Number ');
 bar([sensi_tivity(1:9,1),specificity(1:9,1),NPV(1:9,1),PPV(1:9,1)]);
 xlabel('Trial Number');
ylabel('Sensitivity');
legend('sensitivity','specificity','NPV','PPV');

 %%%%%%%%%%%%%%%%%%Perceptron error calculation%%%%%%%%%%%
figure(fig)
plot(1:1:10,TrainingEpochError)
name = sprintf('Error Curve for Perceptron');
title(name)
xlabel('Number of Iterations')
ylabel('Error Percentage')
legend('Set 1', 'Set 2', 'Set 3', 'Set 4', 'Set 5', 'Set 6', 'Set 7', 'Set 8', 'Set 9');
fig = fig+1;

    %%%%%%%%%%KNN%%%%%%%%%%
mean_arr(1,1)=mean(NPV(:,1));
mean_arr(1,2)=mean(sensi_tivity(:,1));
mean_arr(1,3)=mean(specificity(:,1));
mean_arr(1,4)=mean(PPV(:,1));
mean_arr(1,5)=mean(hit_rate_knn);

sd_arr(1,1)=std(NPV(:,1));
```

```
sd_arr(1,2)=std(sensi_tivity(:,1));
sd_arr(1,3)=std(specificity(:,1));
sd_arr(1,4)=std(PPV(:,1));
sd_arr(1,5)=std(hit_rate_knn);

%%%%%%%%%%NEC%%%%%%%%%%%%%
mean_arr(2,1)=mean(NPV(:,2));
mean_arr(2,2)=mean(sensi_tivity(:,2));
mean_arr(2,3)=mean(specificity(:,2));
mean_arr(2,4)=mean(PPV(:,2));
mean_arr(2,5)=mean(hit_rate_rad);

sd_arr(2,1)=std(NPV(:,2));
sd_arr(2,2)=std(sensi_tivity(:,2));
sd_arr(2,3)=std(specificity(:,2));
sd_arr(2,4)=std(PPV(:,2));
sd_arr(2,5)=std(hit_rate_rad);

%%%%%%%%%%%percep%%%%%%%%%%
mean_arr(3,1)=mean(NPV(:,4));
mean_arr(3,2)=mean(sensi_tivity(:,4));
mean_arr(3,3)=mean(specificity(:,4));
mean_arr(3,4)=mean(PPV(:,4));
mean_arr(3,5)=mean(hit_rate_percep);

sd_arr(3,1)=std(NPV(:,4));
sd_arr(3,2)=std(sensi_tivity(:,4));
sd_arr(3,3)=std(specificity(:,4));
sd_arr(3,4)=std(PPV(:,4));
sd_arr(3,5)=std(hit_rate_percep);



disp(mean_arr);
disp(sd_arr);

xlswrite('tst.xls',mean_arr);
```