**Data Mining**

**HW 5**

**Atieh Kashani**

**Part 1: Anomaly Detection:**

The given data set in this project contains 1599 win samples and the goal is to detect the excellent or poor wines samples. The script main\_part\_1\_outlier.m includes the code related to outlier detection. It loads the samples including the feature matrix X and ratings Y. Following are three major steps in this project:

**Step 1**: Generate ground\_truth labels for the samples. The provided code considers a sample as an outlier if its rating is larger or equal 8 or smaller or equal 3. This results in 28 outliers.

**Step 2:** Use nearest neighbor algorithms to select outliers.

I have implemented the following approaches to select the outliers.For settings the value of p, D, n, k I have tried different combinations as follows:

The value of D based on the average distance between samples.

The value of p based on the average number of neighboring points within the distance D.

The value of n close to the number of ground-truth outliers (28).

Also, for the value of k I have tried the following values: 5, 7, 10, 17,20

Followings are the main parts of the code and explanations for implementing each approach:

1. Data points for which there are fewer than p neighboring points within a distance D.

% Array to store the determined outlier labels using approach A

O1=zeros(size(Y));

% Arrays for storing the number Of Neigbouring Points and average neigbouring

% points

numberOfNeigbouringPoints = zeros(numOfSamples,1);

avgNeigbouringPoints=numberOfNeigbouringPoints;

for sample=1:numOfSamples

euclideanDistance = sum((repmat(features(sample,:),numOfSamples,1)-features).^2,2);

[distance,position] = sort(euclideanDistance,'descend');

%Average Distance between samples

D=mean(euclideanDistance);

%Total number of neighboring points within distance D

numberOfNeigbouringPoints(sample)=sum(euclideanDistance-D<0);

%Average number of neighboring points within distance D

avgNeigbouringPoints(sample)=round(mean(numberOfNeigbouringPoints));

end

%

for sample=1:numOfSamples

euclideanDistance = sum((repmat(features(sample,:),numOfSamples,1)-features).^2,2);

[distance,position] = sort(euclideanDistance,'descend');

%Number of neighboring points for each sample within distance D

numberOfPoints(sample)=sum(euclideanDistance-D<0);

%Determine outliers if there are fewer than p neigbouring points within

%distance D that is the average distance between samples and assign

%the outlier label to those points

if numberOfPoints(sample)<avgNeigbouringPoints(sample)

O1(sample)=1;

end

end

1. The top n data points whose distance to the k-th nearest neighbor is greatest.

% Matrix for storing the neighrest neigbour and their distances

neighborIds = zeros(numOfSamples,5);

neighborDistances = neighborIds;

for sample=1:numOfSamples

% Find the k nearest samples and store the sample index and distance

euclideanDistance = sum((repmat(features(sample,:),numOfSamples,1)-features).^2,2);

[distance,position] = sort(euclideanDistance,'ascend');

neighborIds(sample,:) = position(1:5);

neighborDistances(sample,:) = distance(1:5);

end

% Find the top n ( n is 28 ) data points that the distance to k-nearest

% neigbour is gretaest

[V,I]=maxk(neighborDistances,28);

%Array to store the determined outlier labels using approach B

O2=zeros(size(Y));

% Assign the label of data points that the distance to k-nearest

% neigbour is gretaest as outliers

O2(neighborIds(I))=1;

1. The top n data points whose average distance to the k nearest neighbors is greatest.

neighborIds = zeros(numOfSamples,20);

neighborDistances = neighborIds;

for sample=1:numOfSamples

euclideanDistance = sum((repmat(features(sample,:),numOfSamples,1)-features).^2,2);

[distance,position] = sort(euclideanDistance,'ascend');

neighborIds(sample,:) = position(1:20);

neighborDistances(sample,:) = distance(1:20);

% Find the average distance to the k-nearest neigbour

meanDistance(sample,:)=mean(neighborDistances(sample,:));

end

% Find the top n ( n is 28 ) data points that the average distance to k-nearest neigbour is gretaest

[V2,I2]=maxk(meanDistance,28);

O3=zeros(size(Y));

O3(neighborIds(I2))=1;

**Step 3: Results and Evaluation**

Following are the results and evolution for each of these three approaches using confusion matrix , Accuracy, recall and precision (using the function func\_confusion\_matrix

script : func\_confusion\_matrix.m )

1. By using the first method, 1275 data points predicted as normal point correctly and 6 data points predicted as outliers correctly.

CM =

1275 296

22 6

acc =

0.8011

arrR =

0.8116 0.2143

arrP =

* 1. 0.2143

1. By using the second method for k=5, 1509 data points predicted as normal point correctly and 1 data points predicted as outliers correctly.

For K=5:

CM =

1509 62

27 1

acc =

0.9443

arrR =

0.9605 0.0357

arrP =

0.9824 0.0357

For k=10

CM =

1502 69

27 1

acc =

0.9400

arrR =

0.9561 0.0357

arrP =

0.9823 0.0357

For K=20:

CM =

1499 72

27 1

accuracy =

0.9381

acc =

0.9381

arrR =

0.9542 0.0357

arrP =

0.9823 0.0357

* According to the results for the second method (B) as the number of K nearest neigbour increases the accuracy of this method decreases

**C) Following is the results using the third approach:**

CM =

1546 25

28 0

accuracy =

0.9669

acc =

0.9669

arrR =

0.9841 0

arrP =

0.9822 0

According to the results of evaluation, for this particular dataset the third method (using the average distance to the k nearest neighbors ) works best in identifying the most number of normal points correctly. However, in determining the outliers using the first method (checking the number of neighboring points within a specific distance) can identify the most number of outliers correctly ( compare with the other two method ) .

**Part 2: Link Analysis:**

The goal of this project predicts the rank of 100 linked webpages. Using the PageRank algorithm and the provided adjacent matrix as inputs to predict the rank of these pages. The main\_partII\_pagerank.m includes the following major steps:

Step -1: load adjacent matrix A and normalize it

%% Step 1 load the adjacent matrix;

load('webpages.mat', 'A','U') ;

spy(A)

%% PLACEHOLDER-Start

%Create the Adjacency matrix

adjacent\_matrix=full(A);

% calculate out-degree, sum of each row

r = sum(adjacent\_matrix,2)

% calculate in-degree, sum of each column

c = sum(adjacent\_matrix,1)

%Normalize matrix

normal\_adjacent\_matrix = adjacent\_matrix\*diag(1./c)

%PLACEHOLDER-End

Step -2: Update Rank of the pages as iterative procedure

%% step 2: Compute the PageRank scores for the graph, |G|, using 200 iterations and

% a damping factor of |0.85|.

d=0.85;

pr=ones(length(U),1); % initial ranks

e=ones(100);

%% PLACEHOLDER-Start

% update ranks according to adjacent matrix;

for iter=1:200

pr = (1-d)\*e+d\*normal\_adjacent\_matrix'\*pr;

end

%PLACEHOLDER-End

Step -3: Visualization

H = subgraph(G,find(pr > 0.5\*max(pr(:))));

figure, plot(H,'NodeLabel',{},'Layout','force');

title('high-ranked Websites')

colorbar

