The University of British Columbia

CPSC 340

Machine Learning and Data Mining

Homework #6

Student name:

ALI SIAHKOOHI

Student ID:

84264167

Instructor:

Professor Schmidt

Date Submitted:

December 5th, 2016

Question #1

Part 1.1

I have normalized the PageRank such that it sums up to one. Figure (1 – 1) shows the bar graph for PageRank obtained.

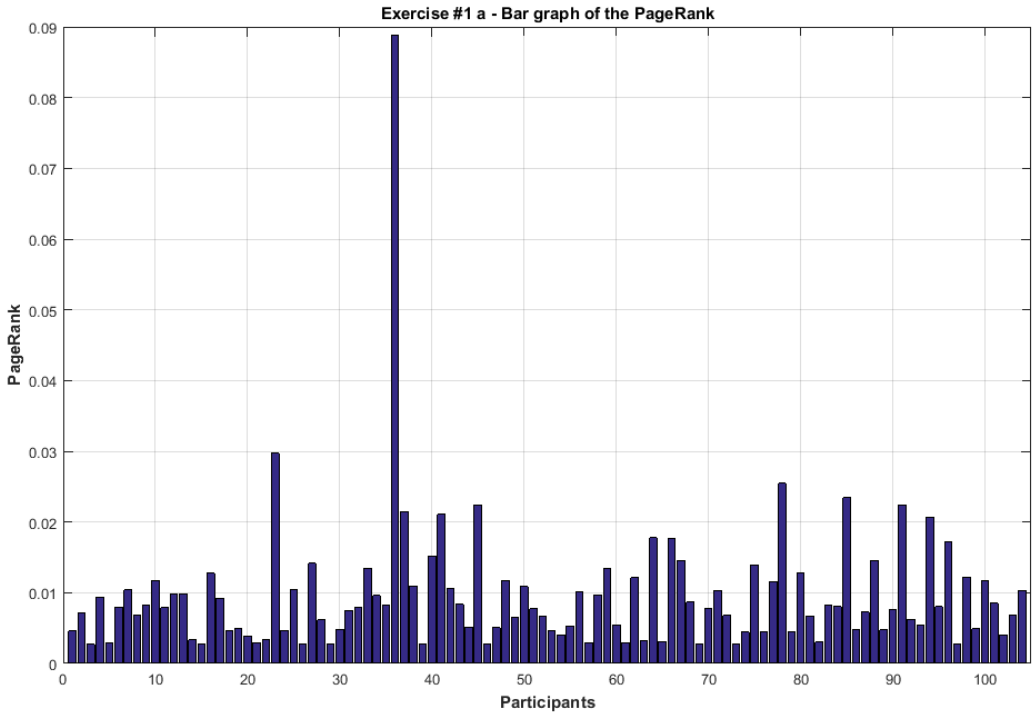

Figure (1 – 1) Exercise #1, the PageRank obtained

MATLAB code:

```
clear; clc; close all

load arrowhead.mat

n = length(names);
alpha = .1;

%% The matrix

H = sparse(X(:, 1), X(:, 2), 1, n, n);
H = full(H);


for i = 1:n
    for j = 1:n
        if H(i, j) ~= 0
            H(j, i) = H(i, j);
        end
```

```
    end
end


%% Random Walk
itr = 1e6;
PR = RandomWalk(H, itr);

%%
bar(PR)
xlim([0, 105])
title('Exercise #1 a - Bar graph of the PageRank')
xlabel('Participants', 'FontSize', 12, 'FontWeight','bold')
ylabel('PageRank', 'FontSize', 12, 'FontWeight','bold')
grid on
```

```
function PR = RandomWalk(H, itr)


n = size(H, 1);
k = 0;
v = ceil(n*rand);
PR = zeros(n, 1);

while (k <= itr)

    alpha = 0.1;

    if sum(H(v, :)) == 0
        alpha = 1;
    end

    if (rand <= alpha)
        v = ceil(n*rand);
    else
        ind = find(H(v, :));
        v = ind(ceil(length(ind)*rand));
    end

    k = k + 1;

    if k>100
        PR(v) = PR(v) + 1;
    end

end

PR = PR/sum(PR);

end
```

Part 1.2

```
[~, m] = max(PR)
m =
    36
>> names(36)
ans =
    'Golub'
```

Golub is the one with largest page rank.

Part 1.3

```
median(PR)
ans =
    0.0077
```

The median in this case is 0.0077.

Part 1.4

```
names(64)
ans =
    'Moler'
>> PR(64)
ans =
    0.0179
```

Moler’s page rank is 0.0179.

Question #2

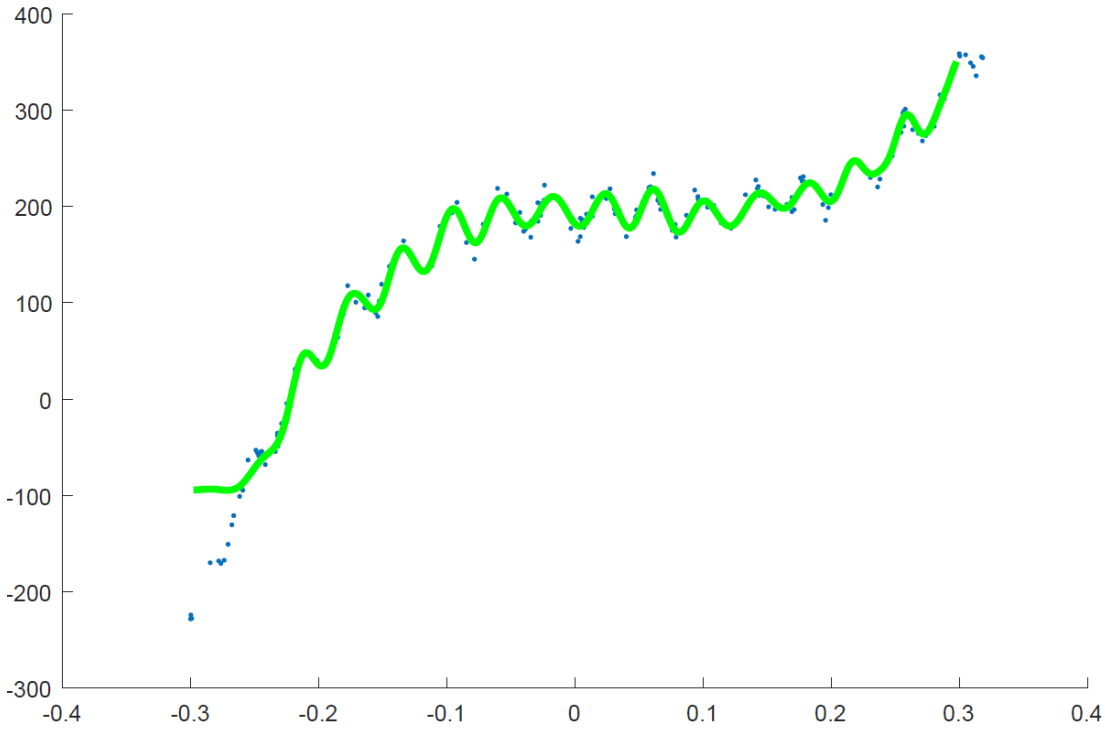

Figure (2 – 1) My result

The changes that I made:

1. Normalize the data:

```
X = (X - mean(X))/var(X);
X = [ones(length(X), 1), X];
```

2. Network structure:

```
nHidden = [50 50 50];
```

3. Initializatin:

```
w = 1e-2*randn(nParams,1);
```

4. Number of iterations:

```
maxIter = 500000;
```

5. Step size initial value, and added a if function to decrease the step size as below:

```
stepSize = 1e-6;

if mod(t-1,round(maxIter/10)) == 0
      stepSize = stepSize*0.85;
end
```

6. Activation function:

```
h = @ (z) log(1+exp(z)) + sin(5*z); % Activation function
dh = @(z) exp(z)./(1+exp(z)) + 5*cos(5*z); % Derivative of activiation
function
```

7. Regularization:

```
lamda = .001;
f = f + lamda*sum(Ww.^2)/2;

g = g + lamda*Ww;
```

MATLAB code:

```
clear; clc; close all
load basisData.mat

X = (X - mean(X))/var(X);
X = [ones(length(X), 1), X];

[n,d] = size(X);

% Choose network structure
nHidden = [50 50 50];

% Count number of parameters and initialize weights 'w'
nParams = d*nHidden(1);
for h = 2:length(nHidden)
    nParams = nParams+nHidden(h-1)*nHidden(h);
end
nParams = nParams+nHidden(end);
w = 1e-2*randn(nParams,1);

% Train with stochastic gradient
maxIter = 500000;
stepSize = 1e-6;
funObj = @(w,i)MLPregressionLoss(w,X(i,:),y(i),nHidden);
for t = 1:maxIter

    % The actual stochastic gradient algorithm:
    i = ceil(rand(2, 1)*n);
    [f,g] = funObj(w,i);
    w = w - stepSize*g;
```

```
    % Every few iterations, plot the data/model:
    if mod(t-1,round(maxIter/100)) == 0
        fprintf('Training iteration = %d\n',t-1);
        figure(1);clf;hold on
        Xhat = [-10:.05:10]';
        Xhat = (Xhat - mean(Xhat))/var(Xhat);
        Xhat = [ones(length(Xhat), 1), Xhat];
        yhat = MLPregressionPredict(w,Xhat,nHidden);
        plot(X(:, 2),y,'.');
        h=plot(Xhat(:, 2),yhat,'g-','LineWidth',3);
        drawnow;
    end
    if mod(t-1,round(maxIter/10)) == 0
        stepSize = stepSize*0.85;
    end
end
```

```
function [f,g] = MLPregressionLoss(Ww,X,y,nHidden)

[nInstances,nVars] = size(X);

% Form Weights
W1 = reshape(Ww(1:nVars*nHidden(1)),nVars,nHidden(1)); % Weight matrix W(1)
startIndex = nVars*nHidden(1);
for layer = 2:length(nHidden) % Weight matrix W(m) for m > 1
  Wm{layer-1} = reshape(Ww(startIndex+1:startIndex+nHidden(layer-
1)*nHidden(layer)),nHidden(layer-1),nHidden(layer));
  startIndex = startIndex+nHidden(layer-1)*nHidden(layer);
end
w = Ww(startIndex+1:startIndex+nHidden(end)); % Final weight vector 'w'


h = @ (z) log(1+exp(z)) + sin(5*z); % Activation function
dh = @(z) exp(z)./(1+exp(z)) + 5*cos(5*z); % Derivative of activiation
function

% h = @ (z) 1./(1+exp(-z)); % Activation function
% dh = @(z) exp(-z)./((1+exp(-z)).^2); % Derivative of activiation function

% Initialize gradient vector
f = 0;
if nargout > 1
    gradInput = zeros(size(W1));
    for layer = 2:length(nHidden)
       gradHidden{layer-1} = zeros(size(Wm{layer-1}));
    end
    gradOutput = zeros(size(w));
end

% Compute Output
for i = 1:nInstances
    innerProduct{1} = X(i,:)*W1;
    z{1} = h(innerProduct{1});
    for layer = 2:length(nHidden)
```

```
        innerProduct{layer} = z{layer-1}*Wm{layer-1};
        z{layer} = h(innerProduct{layer});
    end
    yhat = z{end}*w;

    r = yhat-y(i);
    f = f + r^2;

    if nargout > 1
        dr = 2*r;
        err = dr;

        % Output Weights
        gradOutput = gradOutput + err*z{end}';

        if length(nHidden) > 1
            % Last Layer of Hidden Weights
            backprop = err*(dh(innerProduct{end}).*w');
            gradHidden{end} = gradHidden{end} + z{end-1}'*backprop;

            % Other Hidden Layers
            for layer = length(nHidden)-2:-1:1
                backprop =
(backprop*Wm{layer+1}').*dh(innerProduct{layer+1});
                gradHidden{layer} = gradHidden{layer} + z{layer}'*backprop;
            end

            % Input Weights
            backprop = (backprop*Wm{1}').*dh(innerProduct{1});
            gradInput = gradInput + X(i,:)'*backprop;
        else
            % Input Weights
            gradInput = gradInput + err*X(i,:)'*(dh(innerProduct{end}).*w');
        end

    end

end

lamda = .001;
f = f + lamda*sum(Ww.^2)/2;

% Put Gradient into vector
if nargout > 1
    g = zeros(size(Ww));
    g(1:nVars*nHidden(1)) = gradInput(:);
    startIndex = nVars*nHidden(1);
    for layer = 2:length(nHidden)
        g(startIndex+1:startIndex+nHidden(layer-1)*nHidden(layer)) =
gradHidden{layer-1};
        startIndex = startIndex+nHidden(layer-1)*nHidden(layer);
    end
    g(startIndex+1:startIndex+nHidden(end)) = gradOutput;
end
g = g + lamda*Ww;
```

```
function [y] = MLPregressionPredict(Ww,X,nHidden)

[nInstances,nVars] = size(X);

% Form Weights
W1 = reshape(Ww(1:nVars*nHidden(1)),nVars,nHidden(1));
startIndex = nVars*nHidden(1);
for layer = 2:length(nHidden)
    Wm{layer-1} = reshape(Ww(startIndex+1:startIndex+nHidden(layer-
1)*nHidden(layer)),nHidden(layer-1),nHidden(layer));
    startIndex = startIndex+nHidden(layer-1)*nHidden(layer);
end
w = Ww(startIndex+1:startIndex+nHidden(end));

% h = @ (z) 1./(1+exp(-z)); % Activation function

h = @ (z) log(1+exp(z)) + sin(5*z); % Activation function

% Compute Output
y = zeros(nInstances,1);
for i = 1:nInstances
    innerProduct{1} = X(i,:)*W1;
    z{1} = h(innerProduct{1});
    for layer = 2:length(nHidden)
        innerProduct{layer} = z{layer-1}*Wm{layer-1};
        z{layer} = h(innerProduct{layer});
    end
    y(i) = z{end}*w;
end
```