Homework 3
Student Name: Weining Hu
Student Number: 45606134

1 Non-Parametric Clustering
1.1 Effect of Parameters on DBSCAN
a.Find and report values for the two parameters of the density-based clustering algorithm such that it nds the correct 4 clusters and assigns all points to their appropriate cluster.

eps: 3.5
minPts: 3

b.Find and report values for the two parameters such that the top two clusters are merged into one cluster

eps: 15
minPts: 3

1.2 K-Means vs. DBSCAN Clustering

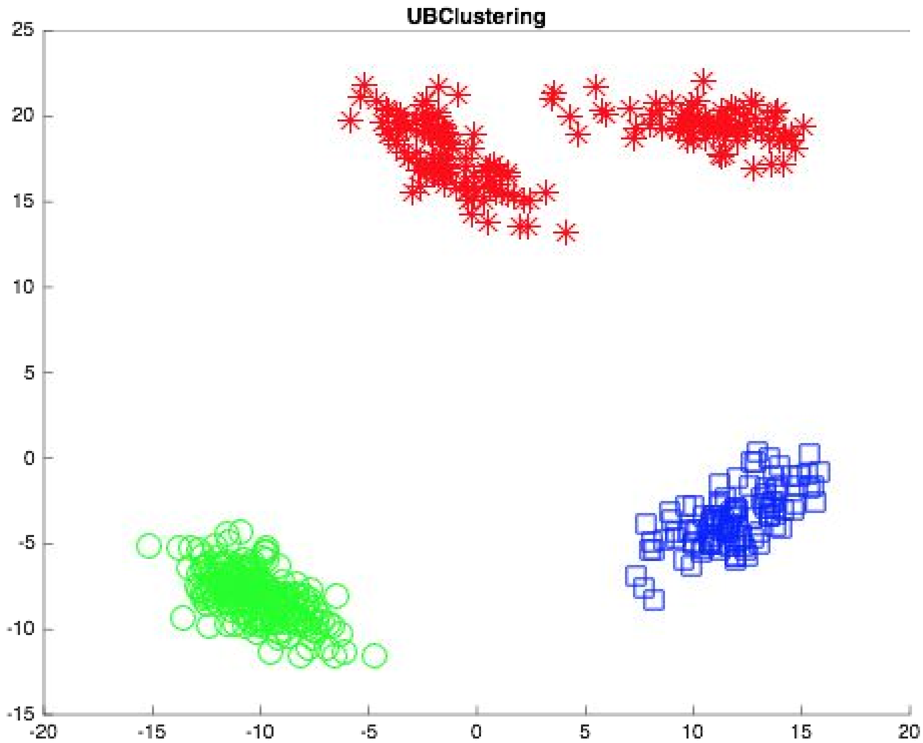
eps: 16
Cluster 0: grizzly+bear killer+whale beaver bat otter giant+panda polar+bear raccoon
Cluster 1: antelope horse hippopotamus moose elephant ox sheep rhinoceros giraffe buffalo zebra deer pig cow
Cluster 2: dalmatian persian+cat german+shepherd siamese+cat skunk mole tiger leopard fox hamster squirrel rabbit wolf chihuahua rat weasel bobcat lion mouse collie
Cluster 3: blue+whale humpback+whale seal walrus dolphin
Cluster 4: spider+monkey gorilla chimpanzee

1.3 UBClustering

```
function [model] = clusterUBClustering(X,K,nModels)

[N,D] = size(X);

for m = 1:nModels
   model.subModel{m} = clusterKmeans(X,K);
end

for m = 1:nModels
   clusters(:,m) = model.subModel{m}.predict(model.subModel{m},X);
end
%clusters = mode(clusters,2);


% times when two samples appears in the same cluster, record the time
T = zeros(N,N);
for i = 1:N
   for j = 1:N
      T(i,j) = sum(clusters(i,:) == clusters(j,:))/nModels;
   end

end

cluster = zeros(N,1); % store the label for each sample
visited = zeros(N,1); %keep track of what samples have been visited

K = 0;
eps = 0.5;
minPts = 4;

for i = 1:N
   if (~visited(i))
      visited(i) = 1;
      clear neighbors;
      % We consider points that appear in the same cluster more than half
      % of the time in the same cluster
      neighbors = find(T(:,i) > eps);
      if (length(neighbors) >=minPts)
         K = K+1;
         [visited,cluster] = expand(X,i,neighbors,K,eps,minPts,T,visited,cluster);
      end
   end
end

model.clusters = clusters;
```

```
% If we only have two features, make a colored scatterplot
if D == 2
   clf;hold on;
   colors = getColorsRGB;
   for k = 1:K

plot(X(clusters==k,1),X(clusters==k,2),'o','Color',.75*colors(k,:),'MarkerSize',5,'MarkerFaceC
olor',.75*colors(k,:));
   end
end
end

function [visited,cluster] = expand(X,i,neighbors,K,eps,minPts,D,visited,cluster)
cluster(i) = K;
ind = 0;
while 1
   ind = ind+1;
   if ind > length(neighbors)
      break;
   end
   n = neighbors(ind);
   cluster(n) = K;

   if ~visited(n)
      visited(n) = 1;
      neighbors2 = find(D(:,n) > eps);
      if length(neighbors2) >= minPts
         neighbors = [neighbors;setdiff(neighbors2,neighbors)];
      end
   end

   if size(X,2) == 2
      % Make plot
      clf;hold on;
      colors = getColorsRGB;
      symbols = getSymbols;
      h = plot(X(cluster==0,1),X(cluster==0,2),'.');
      set(h,'Color',[0 0 0]);
      for k = 1:K
         h = plot(X(cluster==k,1),X(cluster==k,2),'.');
         set(h,'Color',colors(k,:),'Marker',symbols{k},'MarkerSize',12);
      end
      pause(.01);
```

```
    end

end
end
```



2 Item Recommendation

2.1 Amazon Recommendation Algorithm

code:

```
function [model] = recommender(X,K)
X = double(X);
[N,D] = size(X);
% create a matrix cos to store all cosine similarity between i and j
cos = zeros(D,D);
for i = 1:D
    for j = 1:D
        n = (X(:,i))'*X(:,j);
        d = (norm(X(:,i))*norm(X(:,j)));
        cos(i,j) = n/d;
    end
end
model.predict = @predict;
model.K = K;
model.cos = cos;
end

function [wordNumbers] = predict(model,j)
cos = model.cos;
K = model.K;
all = cos(j,:);
[sortedValues,sortIndex] = sort(all,'descend');
wordNumbers = sortIndex(1:(K+1));
end
```

For each of the rst 5 words, list the recommended words when K = 5

word1: 'aids'
recommend: 'health' 'disease' 'patients' 'cancer' 'food'

word2:'baseball'
recommend: 'players' 'games' 'league' 'season' 'team'

word3:'bible'
recommend: 'god' 'jesus' 'christian' 'religion' 'fact'

word4:'bmw'
recommend: 'car' 'engine' 'honda' 'oil' 'drive'

word5: 'cancer'
recommend: 'patients' 'disease' 'medicine' 'vitamin' 'health'

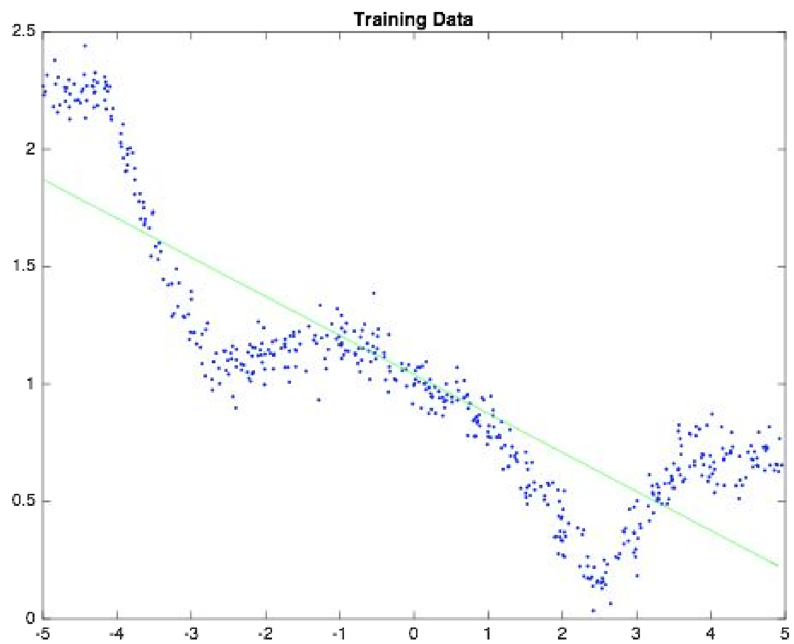
2.2 Fast Recommendation with Sparse Data

Let’s say the current item I want to make recommendation is A. We are given at most P items that have non-zero similarity and sorted list of user ID’s who also bought item A. We pick the users that have at least bought to avoid zero norm. And we sort the user ID for easier calculation. The format will be (userID, product, 1 or 0).

When making recommendation, we first calculate the cosine similarity between item A and the rest P items. For a single item, the cost will be O(u), this includes norm and inner product. And we have P items. So in total is O(up).

3 Linear Regression and Change of Basis
3.1

```
function [model] = simpleLeastSquares(X,y)

% Solve least squares problem
% add extra column with ones
[row,col] = size(X);
b = ones(row,1);
X = [X b];
w = (X'*X)\X'*y;
model.w = w;
model.predict = @predict;
end

function [yhat] = predict(model,Xtest)
[r,c] = size(Xtest);
b0 = ones(r,1);
Xtest = [Xtest b0];
w = model.w;
yhat = Xtest*w;
end
```



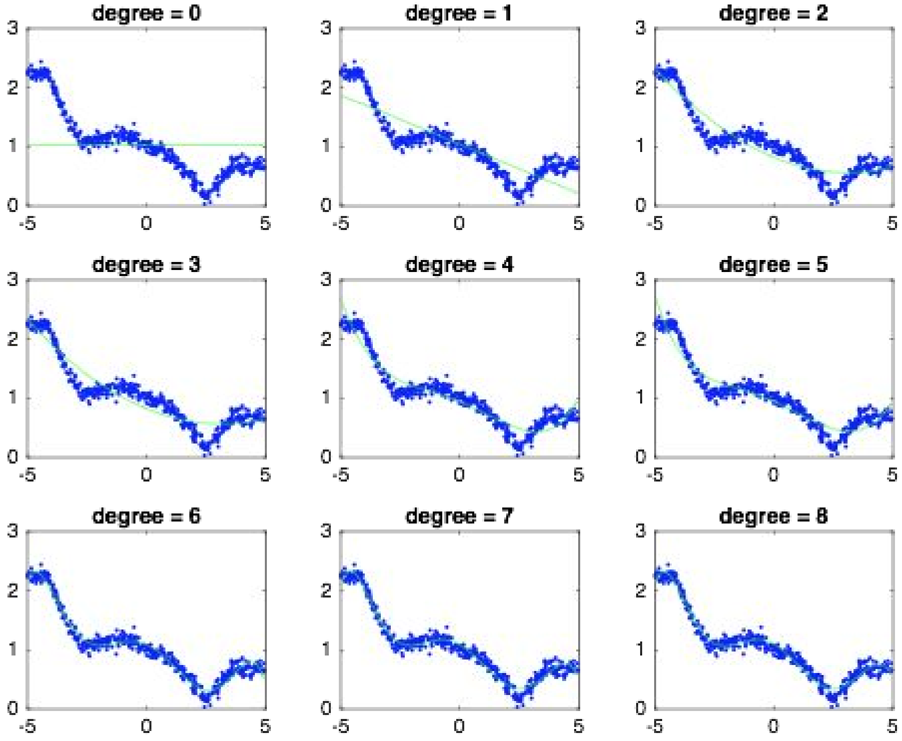
3.2 Polynomial Basis

code for leastSquaresBasis(x,y,degree):

```
function [model] = leastSquaresBasis(x,y,degree)
[N,D] = size(x);
xpoly = zeros(N,(degree+1));
for i = 0:degree
    xpoly(:,(i+1)) = x.^(i);
end
w = (xpoly'*xpoly)\xpoly'*y;
model.w = w;
model.predict = @predict;
model.degree = degree;
end


function [yhat] = predict(model,Xtest)
degree = model.degree;
[r,c] = size(Xtest);
xtestpoly = zeros(r,(degree+1));
for i = 0:degree
    xtestpoly(:,(i+1)) = Xtest.^(i);
end
w = model.w;
yhat = xtestpoly*w;
end
```

code for plot:

```
load basisData.mat
% Plot data
figure;
i = 0;
for i = 0:8
    subplot(3,3,i+1)
    plot(X,y,'b.')
    title(sprintf('degree = %d',i))
    hold on

    % Fit least-squares estimator
    model = leastSquaresBasis(X,y,i);

    % Draw model prediction
    Xsample = [min(X):.1:max(X)]';
    yHat = model.predict(model,Xsample);
    plot(Xsample,yHat,'g-');
end
```



3.3 Choosing the Basis

At degree = 0,training error = 0.30080,validation error = 0.31136, average error = 0.30608

At degree = 1,training error = 0.07832,validation error = 0.08051, average error = 0.07941

At degree = 2,training error = 0.03654,validation error = 0.04399, average error = 0.04027

At degree = 3,training error = 0.03653,validation error = 0.04397, average error = 0.04025

At degree = 4,training error = 0.02533,validation error = 0.03230, average error = 0.02881

At degree = 5,training error = 0.02441,validation error = 0.03371, average error = 0.02906

At degree = 6,training error = 0.00958,validation error = 0.01012, average error = 0.00985

At degree = 7,training error = 0.00825,validation error = 0.01098, average error = 0.00961

At degree = 8,training error = 0.00821,validation error = 0.01055, average error = 0.00938

At degree = 9,training error = 0.00779,validation error = 0.01181, average error = 0.00980
At degree = 10,training error = 0.00618,validation error = 0.00684, average error = 0.00651
At degree = 11,training error = 0.00553,validation error = 0.00763, average error = 0.00658
At degree = 12,training error = 0.00513,validation error = 0.00581, average error = 0.00547
At degree = 13,training error = 0.00492,validation error = 0.00654, average error = 0.00573
At degree = 14,training error = 0.00492,validation error = 0.00663, average error = 0.00577
At degree = 15,training error = 0.00487,validation error = 0.00548, average error = 0.00518
At degree = 16,training error = 0.00487,validation error = 0.00543, average error = 0.00515
At degree = 17,training error = 0.00487,validation error = 0.00573, average error = 0.00530
At degree = 18,training error = 0.00482,validation error = 0.00647, average error = 0.00564
At degree = 19,training error = 0.00465,validation error = 0.03200, average error = 0.01833
At degree = 20,training error = 0.00463,validation error = 0.05713, average error = 0.03088

Comment: We could see that when the degree increases, the training error gets smaller because the model is more complex.But as the curvature of the function increases, the higher the possibility that function overfits. Therefore, according to the fundamental trade-off in machine learning validation error will be worse approximation of the testing error. And as we could see in our example, when the degree is at 20, the training error and testing error is not at a same scale.

4 Radial Basis Functions and Regularization
4.1 Regularization
Test error before modification:
Test error with sigma = 8.000000 is 0.142595
Test error with sigma = 4.000000 is 0.081494
Test error with sigma = 2.000000 is 0.066216
Test error with sigma = 1.000000 is 0.060944
Test error with sigma = 0.500000 is 0.205424
Test error with sigma = 0.250000 is 4.480127
Test error with sigma = 0.125000 is 0.921877
Test error with sigma = 0.062500 is 0.434101

Code :

```
function [model] = leastSquaresRBF(X,y,sigma,lambda)

[N,D] = size(X);

Xrbf = rbfBasis(X,X,sigma);

% Solve least squares problem, the place I made modification
w = (Xrbf'*Xrbf + lambda*eye(D))\Xrbf'*y;

model.X = X;
model.w = w;
model.sigma = sigma;
model.predict = @predict;

end

function [yhat] = predict(model,Xtest)
Xrbf = rbfBasis(Xtest,model.X,model.sigma);
yhat = Xrbf*model.w;
end

function [Xrbf] = rbfBasis(X1,X2,sigma)
N1 = size(X1,1);
N2 = size(X2,1);
D = size(X1,2);
Z = 1/sqrt(2*pi*sigma^2);
D = X1.^2*ones(D,N2) + ones(N1,D)*(X2').^2 - 2*X1*X2';
Xrbf = Z*exp(-D/(2*sigma^2));
end
```

I explored through several lambda, I found that when lambda = 3, I could have smaller test error .

4.2 Proper Training/Validation/Testing

By searching through the combinations of different value of sigma and lambda, I found the combination with minimum error for validation data is :
Test error with sigma = 1.000000, lambda = 0.001953 is 0.054111
Final result:
Test error with sigma = 1.000000, lambda = 0.001953 is 0.063192

Code:

```
% Load data
warning off all
close all
clear all
```

```
load nonLinearData.mat
[n,d] = size(X);

% Plotting Code
plot(X,y,'b.');hold on
plot(Xtest,ytest,'g.');
xl = xlim;
yl = ylim;
Xvals = [xl(1):.1:xl(2)]';
pause(.1)

% Display result of fitting with RBF kernel
% Train on the first half of the
% training data and test on the second half of the training data (the `validation' set).
X_train = X(1:50,:);
y_train = y(1:50,:);
X_validate = X(51:100,:);
y_validate = y(51:100,:);
error = 1;

for sigma = 2.^[3:-1:-4]
    for lambda = 2.^[2:-1:-12]
        %% Train on X, test on Xtest
        model = leastSquaresRBF(X_train,y_train,sigma,lambda);
        yhat = model.predict(model,X_validate);
        fprintf('Test error with sigma = %f, lambda = %f is
%f\n',sigma,lambda,mean(abs(yhat-y_validate)));
        % find the parameter with minimum error
        if (mean(abs(yhat-y_validate)) < error)
            s = sigma;
            l = lambda;
            error = mean(abs(yhat-y_validate));
        end
        %% Plotting Code
        figure(1);clf;
        plot(X,y,'b.');hold on
        plot(Xtest,ytest,'g.');
        yvals = model.predict(model,Xvals);
        plot(Xvals,yvals,'r-');
        legend({'Train','Test'});
        ylim(yl);
        title(sprintf('RBF Basis (sigma = %f)',sigma));
        pause(.25)
    end
end
```

```
fprintf('Test error with sigma = %f, lambda = %f is %f\n',s,l,error);

% After finding the optimal sigma and lambda,
% sigma = 1.000000, lambda = 0.001953
% We train on the full training set and test on the test set

model = leastSquaresRBF(X,y,1,0.001953);
yhat = model.predict(model,Xtest);
fprintf('Test error with sigma = %f, lambda = %f is %f\n',1,0.001953,mean(abs(yhat-ytest)));
```

5 Least Squares with Outliers

5.1 Weighted Least Squares in One Dimension

5.3 Smooth Approximation to the L1-Norm

5.1

$$\underset{w \in R^d}{\operatorname{argmin}} \overbrace{\frac{1}{2}\sum_{i=1}^{n} z_i (y_i - w^T x_i)^2}^{M}$$

$$\frac{\partial M}{\partial w} = \frac{1}{2}\cdot 2 \left[\sum_{i=1}^{n} z_i (y_i - w^T x_i)\right](-x_i)$$

$$= \sum_{i=1}^{n} z_i (w^T x_i - y_i) x_i$$

$$= \sum_{i=1}^{n} z_i x_i^T x_i w^T - \sum_{i=1}^{n} z_i x_i^T y_i$$

$$= \sum_{i=1}^{n} z_i X^T X w^T - \sum_{i=1}^{n} z_i X^T y = 0$$

$$W = (X^T Z X)^{-1} X^T Z y$$

5.3 $f(w) = \sum_{i=1}^{n} \sqrt{(y_i - w^T x_i)^2 + \varepsilon}$

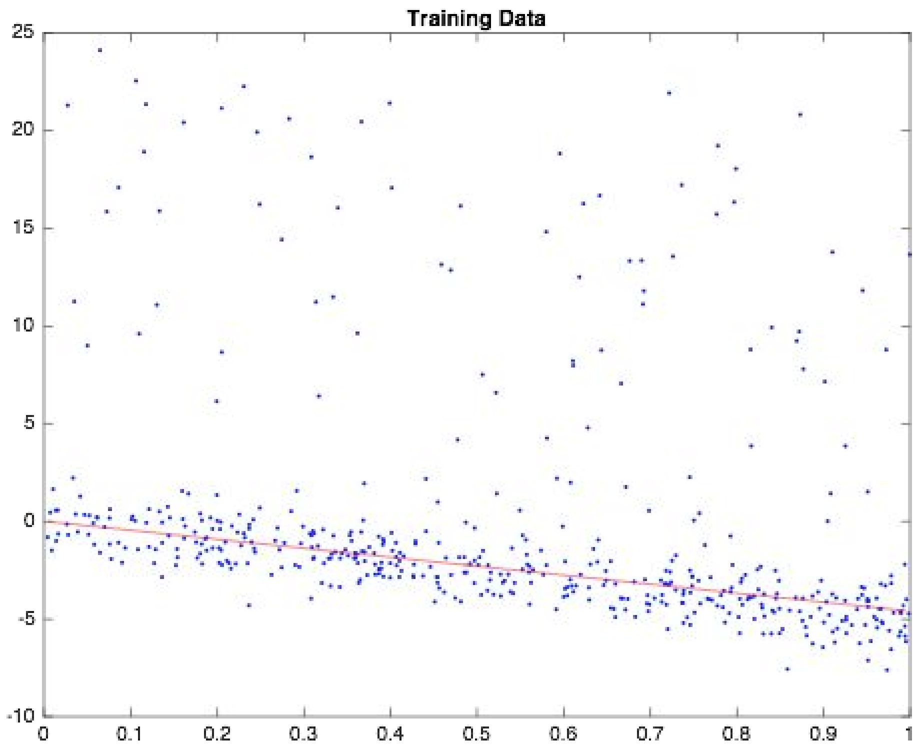
At each x_i, we take derivative, then we will have gradient function

$$g = \nabla f(w) = \begin{bmatrix} \frac{(y_i - w^T X)(-x_1)}{\sum_{i=1}^{n} \sqrt{(y_i - w^T x_i)^2 + \varepsilon}} \\ \frac{(y_i - w^T X)(-x_2)}{\sum_{i=1}^{n} \sqrt{(y_i - w^T x_i)^2 + \varepsilon}} \\ \vdots \\ \frac{(y_i - w^T X)(-x_n)}{\sum_{i=1}^{n} \sqrt{(y_i - w^T x_i)^2 + \varepsilon}} \end{bmatrix}$$

5.2 Weighted Least Squares Fitting

```
function [model] = weightedLeastSquares(x,y,z)
% Solve least squares problem
% in our question, the z is a diagonal matrix where
%z = ones(1,500);
%z(1:400) = 1;
%z(401:500) = 0.1;
%z = diag(z);
```

```
w = ((x'*z*x))\(x'*z*y);
model.w = w;
model.predict = @predict;
end

function [yhat] = predict(model,Xtest)
w = model.w;
yhat = Xtest*w;
end
```



5.4 Robust Regression

```
function [model] = robustRegressionGradient(X,y,epsilon)

[n,d] = size(X);

% Initial guess
w0 = zeros(d,1);

% This is how you compute the function and gradient:
[f,g] = funObj(w0,X,y,epsilon);

% Derivative check that the gradient code is correct:
[f2,g2] = autoGrad(w0,@funObj,X,y,epsilon);

if max(abs(g-g2) > 1e-4)
   fprintf('User and numerical derivatives differ:\n');
   [g g2]
else
   fprintf('User and numerical derivatives agree.\n');
end

% Solve robust regression problem
w = findMin(@funObj,w0,100,X,y,epsilon);

model.w = w;
model.predict = @predict;

end

function [yhat] = predict(model,Xtest)
w = model.w;
yhat = Xtest*w;
end

function [f,g] = funObj(w,X,y,epsilon)
   f = sum(sqrt((X*w-y).^2 + epsilon));
   [N,D] = size(X);
   g = zeros(1,D);
   for i = 1:D
   g(i) = sum(((X*w-y).^2+epsilon).^(-1/2).*(X*w-y).*X(:,i));
   end
end
```