Homework 4

Nate Mankovich
Math Modeling

April 3, 2018

Part A: Theory and Foundations

Problem 1

Prove that the competitive learning update rule moves the winning center closer to the data point.

We just have to show that for any $n \in \mathbb{N}$ we have $d(c^{n+1}, x) \leq d(c^n, x)$ where x is a data point and c is a winning center. Notice this is true if and only if $d(c^{n+1}, x)^2 \leq d(c^n, x)^2$.

$$\begin{aligned} d(c^{n+1}, x)^2 &= \langle c^{n+1} - x, c^{n+1} - x \rangle \\ &= (c^{n+1} - x)(c^{n+1} - x)^T \text{ by definition of the inner product.} \\ &= (c^n + \epsilon(x - c^n) - x)(c^n + \epsilon(x - c^n) - x)^T \\ &= ((1-\epsilon)c^n - x)(1-\epsilon)c^n - x)^T \\ &= (1-\epsilon)^2(c^n - x)(c^n - x)^T \text{ since we choose } \epsilon \text{ small we can assume } \epsilon \leq 1 \\ &\leq (c^n - x)(c^n - x)^T \\ &= \langle c^n - x, c^n - x \rangle \\ &= d(c^n, x)^2 \end{aligned}$$

Problem 2

Show that competitive learning forgets when the learning rate is constant, i.e., $\varepsilon = \varepsilon_0$.

To show competitive learning forgets when the learning rate is constant we must show for large values of n that the nth center's dependence on the first center is extremely weak. If we can show

$$c^n = (1-\epsilon)^n c^0 + \epsilon(1-\epsilon)^{n-1} x^{(\mu_1)} + ... + \epsilon x^{(\mu_n)}$$

then after a large number of iterations the term (with n large) $(1-\epsilon)^n$ becomes small which means the value of c^0 has very little to affect c^n. For the above equation, the set $\{x^{(\mu_i)}\}_{i=1}^n$ are the data points and the set $\{c^i\}_{i=0}^{n-1}$ are the respective winning centers. We will prove this fact via mathematical induction. For our base case let $n = 1$. Then by our center updating algorithm

$$c^1 = \epsilon(x - c^0) + c^0 = \epsilon x^{(\mu_1)} - \epsilon c^0 + c^0 = (1-\epsilon)^1 c^0 + x^{(\mu_1)}$$

Suppose the statement we are trying to prove is true for all $n \leq k \in \mathbb{N}$ where k is fixed. Then

$$\begin{aligned} c^{n+1} &= \epsilon(x^{(\mu_{n+1})} - c^n) + c^n \\ &= \epsilon x^{(\mu_{n+1})} - \epsilon c^n + c^n \\ &= (1-\epsilon)c^n + \epsilon x^{(\mu_{n+1})} \\ &= (1-\epsilon)((1-\epsilon)^n c^0 + \epsilon(1-\epsilon)^{n-1} x^{(\mu_1)} + ... + \epsilon x^{(\mu_n)}) + \epsilon x^{(\mu_{n+1})} \text{ by the induction hypothesis} \\ &= (1-\epsilon)^{n+1} c^0 + \epsilon(1-\epsilon)^n x^{(\mu_1)} + ... + \epsilon(1-\epsilon) x^{(\mu_n)} + \epsilon x^{(\mu_{n+1})} \end{aligned}$$

So we have proved the induction step, therefore this statement is true for all $n \in \mathbb{N}$.

Problem 3

This problem concerns the training of a specific center in k-means. Use induction to prove that

$$c^k = \frac{1}{k}\sum_{i=1}^{k} x^{(\mu_i)}$$

Here we assume that the sequence of points $\{x^{(\mu_1)}, x^{(\mu_2)}, ..., x^{(\mu_k)}\}$ all have c as the nearest center. We use the superscripts m to denote the mth time that c is updated.

Recall we let $\epsilon_n = 1/n$ for all $n \in \mathbb{N}$ for the k-means algorithm. We will use the same assumptions about $x^{(\mu_i)}$ and c^i as we did in the previous problem.
First we will prove this statement is true for $k = 1$.

$$c^1 = \frac{1}{1}(x^{(\mu_{n+1})} - c^n) + c^n = x^{(\mu_{n+1})} - c^n + c^n = x^{(\mu_{n+1})} = \frac{1}{1}\sum_{i=1}^{1} x^{(\mu_i)}.$$

Now suppose this statement is true for all $n \leq k$ for a fixed $k \in \mathbb{N}$. Then

$$\begin{aligned}
c^{k+1} &= \frac{1}{k+1}(x^{(\mu_{n+1})} - c^k) + c^k \\
&= (1 - \frac{1}{k+1})c^k + \frac{1}{k+1}x^{(\mu_{n+1})} \\
&= \frac{k}{k+1}(\frac{1}{k}\sum_{i=1}^{k} x^{(\mu_i)}) + \frac{1}{k+1}x^{(\mu_{n+1})} \text{ by our induction hypothesis.} \\
&= \frac{1}{k+1}\sum_{i=1}^{k} x^{(\mu_i)} + \frac{1}{k+1}x^{(\mu_{n+1})} \\
&= \frac{1}{k+1}\sum_{i=1}^{k+1} x^{(\mu_i)}
\end{aligned}$$

Therefore we have proved this statement is true for all $k \in \mathbb{N}$ by way of mathematical induction on k.

Problem 4

In this problem we fill in the details of the proof that MDS provides the best approximate configuration to a non-Euclidean distance matrix. Let $D \in \mathbb{R}^{n\times n}$ be a non-euclidean distance matrix and $B = HAH$ with eigenvalues $\lambda_1 \geq \lambda_k > 0$, $\lambda_{k+1} = 0$ and $0 > \lambda_{k+2} \leq ... \leq \lambda_n$. (As usual $A = -D^2_{ij}/2$ and H is the usual centering matrix.) Let $\hat{B}$ be the best approximation to B in the sense that

$$\hat{B} = \text{arg min}_C \sum_{ij}(B_{ij} - C_{ij})^2$$

where C is taken from the set of symmetric positive semi-definite matrices.

a) Show that

$$\sum_{ij}(B_{ij} - C_{ij})^2 = \text{trace}((B-C)^2)$$

So consider the ith element of the diagonal of $(B-C)^2$.

$$((B-C)^2)_{ii} = ((B-C)(B-C))_{ii} = \sum_j (B-C)_{ij}(B-C)_{ij} \text{ by defintion of matrix multiplication.}$$

Then by the definition of trace

$$\text{trace}((B-C)^2) = \sum_i (B-C)^2 = \sum_i \sum_j (B-C)_{ij}(B-C)_{ij}) = \sum_{ij}(B-C)^2_{ij}$$

b) Using the spectral theorem we can write $S^TBS = \Lambda$ and $R^TCR = \hat{\Lambda}$. What can you say about the properties of the matrices R, S and the signs of the entries in the diagonal matrices Λ, $\hat{\Lambda}$?

Notice these two equations are just re-arranged eigenvector problems for B and C. Here is a list of facts about the equations:

i) The columns of S are orthonormal and so are the columns of R.
ii) The columns of S are the eigenvectors of B with the the elements of the diagonal of Λ as the associated eigenvalues.
iii) The columns of R are the eigenvectors of C with the the elements of the diagonal of $\hat{\Lambda}$ as the associated eigenvalues.
iv) Λ and $\hat{\Lambda}$ are both diagonal matrices.
v) The entries of the diagonal $\hat{\Lambda}$ are all positive
vi) The entries of the diagonal Λ follow $\lambda_1 \geq \lambda_k > 0$, $\lambda_{k+1} = 0$ and $0 > \lambda_{k+2} \leq ... \leq \lambda_n$.
vii) S and R are orthogonal matrices.

c) Show that $S^TCS = G\hat{\Lambda}G^T$ where G is orthogonal.

Since R is an orthogonal matrix $R^T = R^{-1}$

$$R^TCR = \hat{\Lambda} \implies C = R(R^TCR)R^T = R\hat{\Lambda}R^T$$

So we have

$$S^TCS = S^T(R\hat{\Lambda}R^T)S = (S^TR)\hat{\Lambda}(RS^T)^T$$

Let $G = S^TR$. G is orthogonal because

$$GG^T = (S^TR)(S^TR)^T = S^TRR^TS = S^TS = I.$$

d) Show that trace$((B-C)^2) = $ trace$((\Lambda - G\hat{\Lambda}G^T)^2)$

From part *b*) and *c*) we can write $\Lambda = S^TBS$ and $G\hat{\Lambda}G^T = S^TCS$. Therefore

$$\begin{aligned}
\text{trace}((\Lambda - G\hat{\Lambda}G^T)^2) &= \text{trace}((S^TBS - S^TCS)^2) \\
&= \text{trace}((S^T(B-C)S)^2) \\
&= \text{trace}((S^T(B-C)S)(S^T(B-C)S)) \\
&= \text{trace}(S^T(B-C)^2S) \text{ by the orthogonality of } S \\
&= \text{trace}(S(S^T(B-C)^2)) \text{ by the properties of trace} \\
&= \text{trace}((B-C)^2) \text{ by the orthogonality of } S.
\end{aligned}$$

e) Show that the quantity trace$((\Lambda - G\hat{\Lambda}G^T)^2)$ is a minimum when $G = I$.

$$\begin{aligned}
\text{tr}((\Lambda - G\hat{\Lambda}G^T)^2) &= \text{tr}(\Lambda^2 - G\hat{\Lambda}G^T\Lambda - \Lambda G\hat{\Lambda}G^T + G\hat{\Lambda}^2G^T) \\
&= \text{tr}(\Lambda^2) - \text{tr}(G\hat{\Lambda}G^T\Lambda) - \text{tr}(\Lambda G\hat{\Lambda}G^T) + \text{tr}(G\hat{\Lambda}^2G^T) \\
&= \sum_i \lambda_i - \sum_{ij} \hat{\lambda}_j\lambda_i g_{ij}^2 - \sum_{ij} \hat{\lambda}_j\lambda_i g_{ij}^2 + \sum_{ij} \hat{\lambda}_i^2 g_{ij}^2
\end{aligned}$$

Now we can take the derivative and set it equal to zero.

$$\begin{aligned}
\frac{d}{dG}\text{tr}((\Lambda - G\hat{\Lambda}G^T)^2) &= -4\sum_{ij} \hat{\lambda}_j\lambda_i g_{ij} + 2\sum_{ij} \hat{\lambda}_i^2 g_{ij} = 0 \\
0 &= -2\hat{\Lambda}G^T\Lambda + \hat{\Lambda}^2G^T \\
0 &= -2G^T\Lambda + \hat{\Lambda}G^T \\
0 &= -2R^TS\Lambda + \hat{\Lambda}R^TS \\
2R^TS\Lambda &= \hat{\Lambda}R^TS \\
2(S\Lambda S^T) &= R\hat{\Lambda}R^T \\
C &= 2B \\
\text{Since } R^TCR = \hat{\Lambda} &\implies R^T(2B)R = \hat{\Lambda} \\
\implies \hat{\Lambda} = S^TBS &= R^TBR = \hat{\Lambda}/2 \\
\implies 2\hat{\Lambda} &= \Lambda \\
0 &= -2G^T\Lambda + (2\Lambda)G^T \\
0 &= -G^T\Lambda + \Lambda G^T \\
G^T\Lambda &= \Lambda G^T \\
\Lambda &= G\Lambda G^T \\
\implies G &= I.
\end{aligned}$$

Problem 5

Show that zero is always an eigenvalue of the Laplacian eigenvector problem. What is the associated eigenvector?

Recall the Laplacian eigenvector problem is $Ly = \lambda W y$ where $L = D - W$. Let $\lambda = 0$ then $Ly = \lambda W y = 0$. Since $L = D - W$

$$(D - W)y = 0 \implies Dy = Wy$$

Where we define e as the column vector of ones. Recall D is a diagonal matrix with $D_{ii} = \sum_j W_{ij}$ so

$$De = D = \begin{bmatrix} \sum_j W_{1j} \\ \sum_j W_{2j} \\ . \\ . \\ . \end{bmatrix} = We$$

Part B: Computing

Problem 1

Use LBG clustering with 10 centers to quantize the color image provided on Canvas. Include the color quantized image in your write-up.

Below is the image generated via using LGB clustering from the color image provided on canvas. We calculated the error at each iteration to be the sum of the square distances from each data point and its respective winning center divided by the number of data points. We implemented a while loop that stopped the algorithm after the error dropped below 630. Below is the code used to generate the image.

```
data = imread('Penguins.jpg');

%unroll the data
X1 = reshape(data,[786432,3]);
X = double(X1');

%number of data points
p = 786432;
%data point dimension
N = 3;
%number of centers m
m = 10;

%initialize centers (randomize)
for i=1:m
    ind = floor(786432*rand);
    C(:,i) = X(:,ind);
end

%initialize error
E = 1000;

iteration=1

%TEST
%hold
```

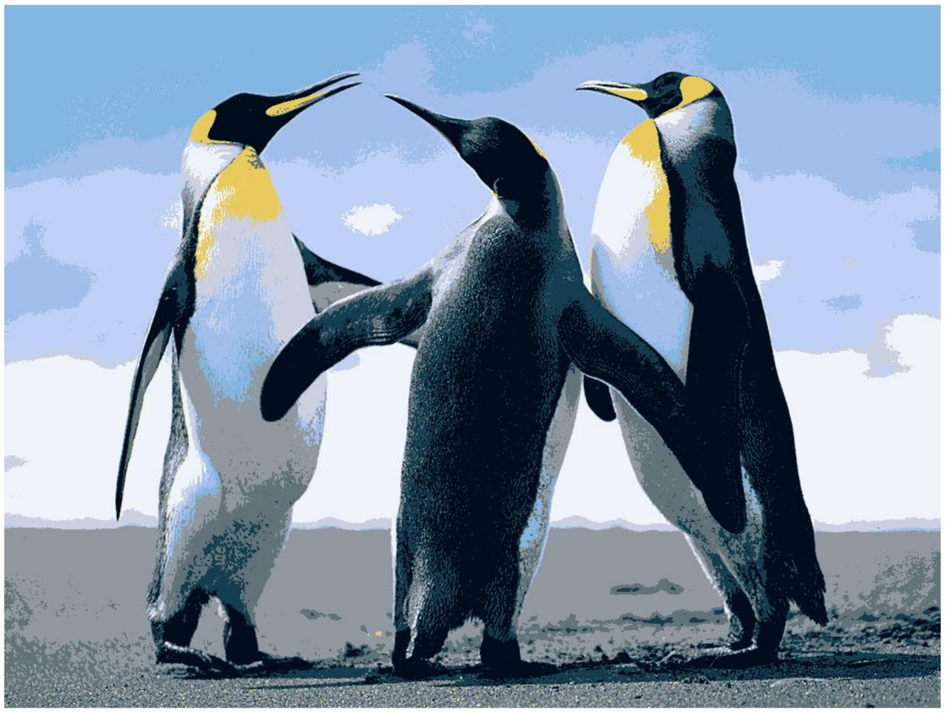

Figure 1: The color quantized image.

```
while E >=630
%for iteration=1:4

    %compute S_i

    %find a distance matrix
    for i=1:m
        for n=1:p
            dist(i,n) = norm(minus(X(:,n),C(:,i)));
        end
    end

    %calculate the minimum distance
    [M,index] = min(dist);

    %find the winning indices
    for i=1:m
        for j=1:p
            comp(i,j) = isequal(i,index(j));
        end
    end

    %Si is X(:,find(comp(i,:)));
    %update centers and calculate i for each S
    %ci = (1/abs(si))sum(x in Si of x)
    for i=1:m
        S = X(:,find(comp(i,:)));
        C(:,i) = (1/length(S(1,:)))*(sum(S,2));
    end

    E = (1/p)*sum(M.*M)
    Error(iteration) = E;
    iteration = iteration+1;
    %repeat until E(x,I) is less than desired tolerance

    %TEST
    %visualize the result
%     for i=1:m
%         k= find(comp(i,:));
%         for j=1:N
%             VisRaw(j,k) = C(j,i);
%         end
%     end
%     Vis = uint8(VisRaw);
%     scatter3(Vis(1,:),Vis(2,:),Vis(3,:))

end

%visualize the result
for i=1:m
    k= find(comp(i,:));
    for j=1:N
        VisRaw(j,k) = C(j,i);
    end
end

%visualize it!
Vis = uint8(reshape(VisRaw',[768,1024,3]));
imshow(Vis)
```

The commented code is used to test the result.

Problem 2

This is an SOM warm-up problem. Consider the data set consisting of the following 6 points:

μ	$x^{(\mu)}$
1	0.34
2	0.12
3	0.73
4	0.97
5	0.07
6	0.56

Run SOM on this data set using the index set $\{1, 2, 3, 4, 5, 6\}$ and the Euclidean and Clock metrics. Pick your initial centers randomly from the uniform distribution on the interval $[0, 1]$. Describe what SOM is doing in each case.

In general, the center update in the SOM algorithm will be larger using the Clock metric because the Clock distance matrix has smaller entries (on average) than the Euclidean distance matrix and the equation for the center update is inversely proportional to the distance matrix value due to the negative exponent. Note that the inverse proportionality is dependent on the entries in the distance matrix all being at least 1, which they all are (disregarding the diagonal).
For these results we took $\alpha = .9$ and took $\epsilon = .9(1 - n/T)$ where n is the iteration number and T is the total number of iterations.

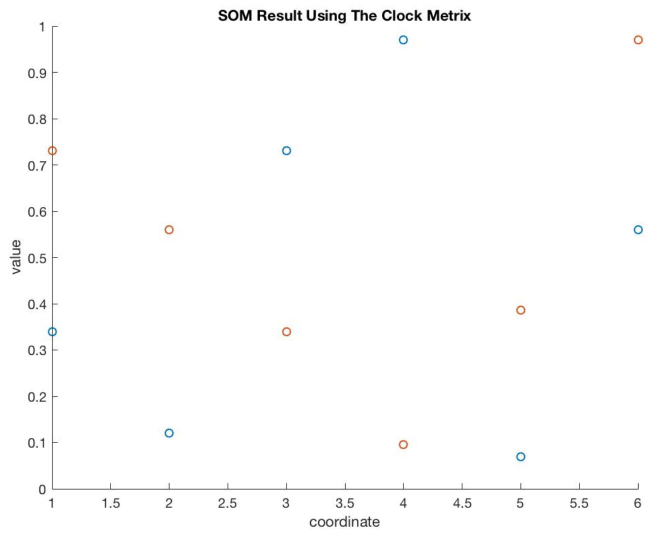

Figure 2: The result of 10000 iterations of SOM with the Clock metric. The blue circles are the data and the red circles are the centers at the final iteration.

Notice, in both cases, the algorithm missed having a center converging to the second smallest data value. Perhaps all we need to do is tune the alpha value and the equation for epsilon in order to achieve full convergence. We were able to achieve full convergence using the same number of iterations and α by altering one data point in the data set.

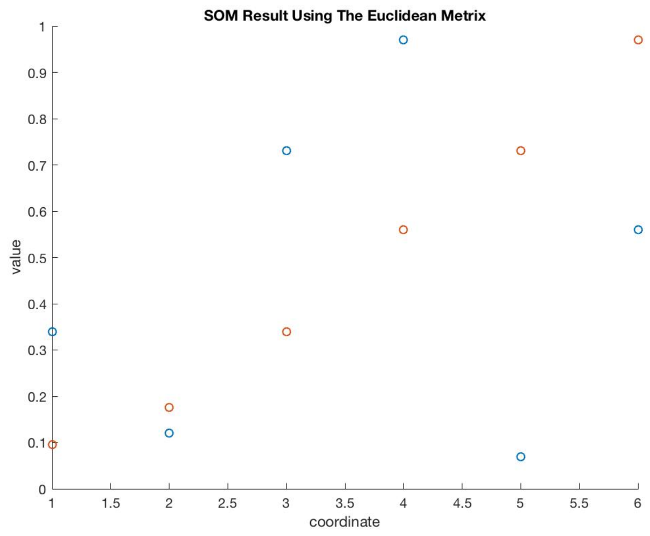

Figure 3: The result of 10000 iterations of SOM with the Euclidean metric. The blue circles are the data and the red circles are the centers at the final iteration.

Below is the code used to generate the images using the Euclidean distance.

```
%initialize data array
X = [0.34 0.12 0.73 0.97 0.07 0.56]; %switch .07 with .27

p = length(X);

%Initialize centers
nCenters = 6;
C = rand(1,nCenters);

%Initialize index set
A = [1;2;3;4;5;6];

%Initialize distance matrix

%Using euclidean metric
for i=1:nCenters
    for j=1:nCenters
            D(i,j) = abs(A(i)-A(j));
    end
end

T = 10000; %total number of times running algorithm
alpha = .9;

%initialize R (and epsilon = r)
for n = 1:T-1
    epsilon(n) = .9*(1-n/T);
end

for n = 1:T-1
```

```
    for mu=1:p

        %distances between centers and data point
        for k = 1:nCenters
            dist(k) = abs(C(k)-X(mu));
        end

        %Determine the winning center
        %by finding the winning indices
        [M,j] = min(dist);

        %update centers
        for i = 1:nCenters
            h= exp(-D(i,j)^2/((alpha*epsilon(n))^2));
            C(i) = C(i)+ epsilon(n)*h*(X(mu)-C(i));
        end

    end

end



%plot the final centers and the data
scatter(1:6,X)
hold
scatter(1:6,C)
```

If we are using the clock distance then we generate the distance matrix as follows:

```
%Using clock metric
% for i=1:nCenters
%     for j=1:nCenters
%             D(i,j) = min(mod(A(i)-A(j),6),mod(A(j)-A(i),6));
%     end
% end
```

Problem 3

Visualize the animal data in two dimensions using Self-Organizing Mappings.

Below is a table of the animals and they're respective numbers:

1	2	3	4	5	6	7	8
Dove	Hen	Duck	Goose	Owl	Hawk	Eagle	Fox

9	10	11	12	13	14	15	16
Dog	Wolf	Cat	Tiger	Lion	Horse	Zebra	Cow

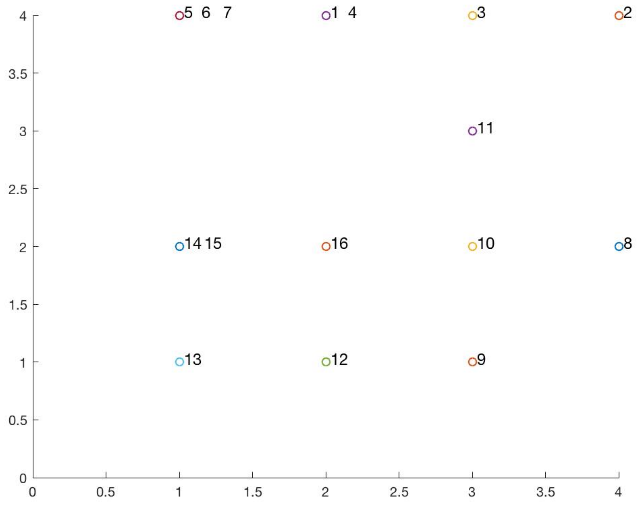

Figure 4: The result of the SOM algorithm run on the animal data.

It would seem this algorithm has been implemented correctly because similar animals have are closer together and some even share the same center index. For example the owl, hawk and eagle are sharing the same index. Note, the data set is assumed to be initialized as X.

```
p = length(X);

%Initialize centers
nCenters = 16;
C = rand(13,nCenters),0;

%Initialize index set
A = [1 1; 1 2; 1 3; 1 4; 2 1; 2 2; 2 3; 2 4; 3 1; 3 2; 3 3; 3 4; 4 1; 4 2; 4 3; 4 4;];
A = A';

%Using euclidean metric
for i=1:nCenters
    for j=1:nCenters
            D(i,j) = norm(A(:,i)-A(:,j));
    end
end

T = 1000; %total number of times running algorithm
alpha = .9;
```

```
%initialize R (and epsilon = r)
for n = 1:T-1
    R(n) = alpha*(1-n/T);
end

for n = 1:T-1
    for mu=1:p
        %distances between centers and data point
        for k = 1:nCenters
            dist(k) = norm(C(:,k)-X(:,mu));
        end

        %Determine the winning center
        %by finding the winning indices
        [M,j] = min(dist);

        %update centers
        for i = 1:nCenters
            h = exp(-D(i,j)^2/(R(n)^2));
            C(:,i) = C(:,i)+ R(n)*h*(X(:,mu)-C(:,i));
        end
    end
end

%visualize result (map pattern to center to index)
hold

for mu= 1:p

    %distances between centers and data point
    for k = 1:nCenters
        dist(k) = norm(C(:,k)-X(:,mu));
    end

    %Determine the winning center by finding the winning indices
    [M,j] = min(dist);

    scatter(A(1,j),A(2,j))
    xlim([0 4])
    ylim([0 4])
    if mu ==15
        text(A(1,j)+.17,A(2,j)+.03,num2str(mu),'FontSize', 12)
    elseif mu == 4
        text(A(1,j)+.15,A(2,j)+.03,num2str(mu),'FontSize', 12)
    elseif mu == 7
        text(A(1,j)+.15*2,A(2,j)+.03,num2str(mu),'FontSize', 12)
    elseif mu == 6
        text(A(1,j)+.15,A(2,j)+.03,num2str(mu),'FontSize', 12)
    else
        text(A(1,j)+.03,A(2,j)+.03,num2str(mu),'FontSize', 12)
    end
    drawnow
    %pause(3)
end
```

Problem 4

Visualize the animal data in two dimensions using Laplacian Eigenmaps.

Below is a plot of the output of the Laplacian Eigenmap algorithm used on the animal data. Notice the data is separated into two clusters based on whether the animals fly.

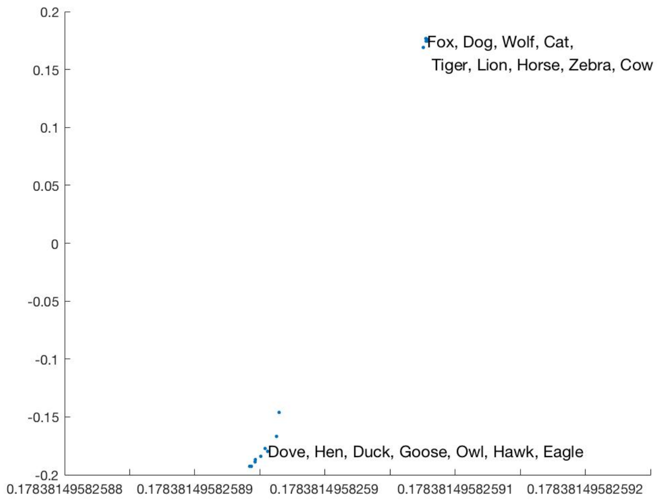

Figure 5: The result of the Laplacian Eigenmaps algorithm run on the animal data.

Below is the code used to produce this result. Note, the data set is assumed to be initialized as X.

```
p= length(X);

%set epsilon
epsilon = 3;

%set the heat kernel t
t=1


%generate distance matrix
% %using unweighted edges
% for i=1:p
%     for j=1:p
%         x = norm(X(:,i)-X(:,j))^2;
%         if x <= epsilon
%             W(i,j) = 1;
%         else
%             W(i,j) = 0;
%         end
%     end
% end
%using weighted edges
for i=1:p
    for j=1:p
        x = norm(X(:,i)-X(:,j));
        if x <= epsilon
            W(i,j) = exp(-x^2/t);
        else
            W(i,j) = 0;
        end
    end
end


%create D
D = zeros(p,p)
for i=1:p
    D(i,i) = sum(W(i,:));
end

%create L
L = D-W;

%generalized eigenvector problem
[evecs,evals] = eig(L,D);

%remove eigenvectors associated with zero eigenvalues
ii=1;
for i=1:p
    if evals(i,i) ~= 0
        evecsNoZero(:,ii) = evecs(:,i);
        ii=ii+1;
    end

end

%find y
Y = evecsNoZero';

scatter(Y(1,:),Y(2,:),'.')
text(Y(1,10),Y(2,10),'Dove, Hen, Duck, Goose, Owl, Hawk, Eagle','FontSize', 12);
text(Y(1,6),Y(2,6),'Fox, Dog, Wolf, Cat,','FontSize', 12);
text(Y(1,6),Y(2,6)-.02, ' Tiger, Lion, Horse, Zebra, Cow','FontSize', 12);
```