Homework 5

Nate Mankovich
Math Modeling

April 24, 2018

Part A: Theory and Foundations

Problem 1

Show that the number of zero eigenvalues indicates the number of connected components in Laplacian eigenmaps problem.

We will begin this proof by making two observations.

1) Given a laplacian graph ($L = D - W$) with m components generated from the running the k-nearest-neighbors algorithm on a data set X, we can organize the L matrix into blocks as follows:

$$L = \begin{bmatrix} C_1 & 0 & ... & 0 \\ 0 & C_2 & ... & 0 \\ . & . & ... & . \\ . & . & ... & . \\ 0 & 0 & ... & C_m \end{bmatrix}$$

where the ith component corresponds to C_i. Since D and W are symmetric, L is also symmetric. Take any data point x that corresponds to a column of L in the in the ith component. Take any coordinate of x that does not correspond to another data point in the ith. By the construction of $L = D - W$, this coordinate must be zero because x cannot be connected to any data point outside its component. Therefore, if we organize L into components we will see this block structure.

2) Take some C_i of size k, a vector v in the null space of C_i is exactly the vector of ones of size k. This is true by the construction of L as $L = D - W$ and the fact

$$D_{ij} = \begin{cases} 0 & \text{if } i \neq j \\ \sum_j W_{ij} & \text{if } i = j \end{cases}$$

3) Now we can show that each C_i has exactly one eigenvector associated with zero. This is the same as showing the algebraic multiplicity of 0 is 1 which is the same as saying $(t-0)$ divides the characteristic polynomial of C_i exactly once. We will prove this by contrapositive. Suppose C_i does not represent a connected graph. Then we can break C_i into a block diagonal graph using part 1) with blocks $B_1, B_2, ..., B_k$. By 2) we know 0 is an eigenvalue of every block $t^l \Big| \det(tI - B_j)$ for all B_j and some $l \geq 1$. Via a result from linear algebra

$$\det(tI - C_i) = \prod_{j=1}^{k} \det(tI - B_j) = t^{kl} P(t) \text{ where } P(t) \text{ is a polynomial in } t.$$

$$\implies t^{kl} \Big| \det(tI - C_i) \text{ with } kl > 1.$$

So we necessarily have more than one eigenvector associated with 0. There is 1 eigenvalue associated with 0 then C_i is the laplacian for a connected graph.
Now we will prove if C_i is the Laplacian for a connected graph then the algebraic multiplicity of 0 is exactly 1. Suppose by way of contradiction the algebraic multiplicity of 0 is greater than 1. Then there exist two linearly independent eigenvectors, x, y, of C_i associated with the eigenvalue of 0. Then

$$x^T C_i x = 0 \text{ and } y^T C_i y = 0.$$

From the definition of the Laplacian as $D - W$, we have

$$\sum_E (x_i - x_j)^2 = 0 \text{ and } \sum_E (y_i - y_j)^2 = 0$$

where E is the set of indices of vertices in the graph of C_i that are connected by an edge.
Since these are positive finite series both summing to zero, we have $y_i = y_j$ and $x_i = x_j$ for all i, j. This tells us both x and y are both scalar multiples of the vector of all ones. Therefore x and y are linearly dependent which is a contradiction.

4) Now we bring our attention back to L. Since L is a block diagonal matrix we know

$$\det(tI - L) = \prod_{i=1}^{m} \det(tI - C_i) = (t-0)^m Q(t) \text{ where } Q(t) \text{ is a polynomial in } t.$$

Because by 3) we know each C_i has zero as an eigenvalue of algebraic multiplicity 1. Since $\det(tI - L)$ (the characteristic polynomial of L has a root of multiplicity m of factors at $t = 0$, L has exactly m eigenvectors associated with the eigenvalue of 0. Therefore the number of zero eigenvalues indicates the number of connected components in Laplacian eigenmaps problem.

Problem 2

Show that the eigenvalues in the Local Linear Embedding problem are all non-negative.

In Local Linear Embedding we are finding the eigenvectors and eigenvalues of the matrix $M = (I-W)^T(I-W)$. By definition M is positive semidefinite. Therefore all the eigenvalues of M are either positive of 0 (i.e. non-negative).

Problem 3

In this problem we explore the details of the LLE derivation were the dimension reduced data matrix is given by $Y = [y_1, ..., y_P] \in \mathbb{R}^{k\times P}$. You will derive, in a sequence of steps, the LLE eigenvector equation

$$MY^T = Y^T\Lambda$$

where $M = (I-W)^T(I-W)$.

a) For a single point x the weight vector w is the solution to the minimization problem

$$E(w) = ||x - \sum w_j x_j||^2$$

subject to $\sum_j w_j = 1$. Show that

$$E(w) = \sum_{jk} w_j w_k C_{jk}$$

where $C_{jk} = (x - x_j)^T(x - x_k)$.

$$\begin{aligned}
||x - \sum_j w_j x_j||^2 &= (x - \sum_j w_j x_j)^T(x - \sum_k w_k x_k) \\
&= (x^T - \sum_j w_j x_j^T)(x - \sum_k w_k x_k) \\
&= x^T x - x^T \sum_k w_k x_k - (\sum_j w_j x_j^T)x + (\sum_j w_j x_j^T)(\sum_k w_k x_k)
\end{aligned}$$

Using the constraint $\sum_j w_j = 1$ we have

$$\begin{aligned}
&= (\sum_j w_j)(\sum_k w_k)x^T x - (\sum_j w_j)x^T \sum_k w_k x_k - (\sum_k w_k)(\sum_j w_j x_j^T)x + (\sum_j w_j x_j^T)(\sum_k w_k x_k) \\
&= \sum_{jk} w_j w_k x^T x - \sum_{jk} w_j w_k x^T x_k - \sum_{jk} w_j w_k x_j^T x + \sum_{jk} w_j w_k x_j^T x_k \\
&= \sum_{jk} w_j w_k (x^T x - x^T x_k - x_j^T x + x_j^T x_k) \\
&= \sum_{jk} w_j w_k (x^T - x_j^T)(x - x_k) \\
&= \sum_{jk} w_j w_k (x - x_j)^T(x - x_k) \\
&= \sum_{jk} w_j w_k C_{jk}
\end{aligned}$$

b) Show that the optimization problem

$$\min \sum_{jk} w_j w_k c_{jk} \text{ where } c_{jk} = (x - x_k)^T(x - x_j)$$

results in

$$Cw_j^* = e$$

where e is a vector of ones; note that you have to rescale this solution such that the weights sum to one. This latter formula is the one you will use in the computation of LLE.

$$\mathcal{L}(w,\lambda) = \sum_j \sum_k w_j w_k c_{jk} - \lambda(\sum_j w_j - 1)$$

We will find the solution to $\nabla\mathcal{L}(w,\lambda)$ =0 which results in the system of equations

$$\text{For j=1,2,...}\ \frac{\partial}{\partial w_i}\mathcal{L} = w_1c_{1j} + w_2c_{2j} + ... + w_nc_{nj} +$$
$$w_1c_{j1} + w_2c_{j2} + ... + w_nc_{jn} - \lambda = 0 \text{ and } \sum_j w_j = 1$$
$$0 = \sum_k +w_kc_{jk} + \sum_k +w_kc_{kj} - \lambda \text{ and } \sum_j w_j = 1$$

Notice $c_{kj} = (x-x_j)^T(x-x_k)$ =$(x-x_k)^T(x-x_j) = c_{jk}$ so we have:

$$0 = 2\sum_k w_kc_{jk} - \lambda \text{ and } \sum_j w_j = 1$$
$$0 = 2Cw - \lambda e \text{ and } ||w||_1 = 1$$
$$C\frac{2w}{\lambda} = e \text{ and } ||w||_1 = 1$$
$$Cw^* = e \text{ where } w^* := \frac{2w}{\lambda} \text{ and we need to re-scale.}$$

c) Show that

$$M_{ij} = \delta_{ij} - W_{ij} - W_{ji} + \sum_k W_{ki}W_{kj}$$

By definition of M we have

$$\begin{aligned} M_{ij} &= ((I-W)^T(I-W))_{ij} \\ &= ((I^T - W^T)(I-W))_{ij} \\ &= (I^TI - I^TW - W^TI + W^TW)_{ij} \\ &= (I - W - W^T + W^TW)_{ij} \\ &= \delta_{ij} - W_{ij} - W_{ji} + \sum_k W_{ki}W_{kj} \end{aligned}$$

d) Now show that

$$\sum_i ||y_i - \sum_j W_{ij}y_j||^2 = \sum_{ij} M_{ij}y_i^Ty_j$$

$$\begin{aligned}
\sum_i ||y_i - \sum_j W_{ij} y_j||^2 &= \sum_i (y_i - \sum_j W_{ij} y_j)^T (y_i - \sum_k W_{ik} y_k) \\
&= \sum_i (y_i^T - \sum_j W_{ij} y_j^T)(y_i - \sum_k W_{ik} y_k) \\
&= \sum_i (y_i^T y_i - \sum_k W_{ik} y_i^T y_k - \sum_j W_{ij} y_j^T y_i - (\sum_j W_{ij} y_j^T)(\sum_k W_{ik} y_k)) \\
&= \sum_i y_i^T y_i - \sum_i \sum_k W_{ik} y_i^T y_k - \sum_i \sum_j W_{ij} y_j^T y_i - \sum_i \sum_j \sum_k W_{ij} y_j^T W_{ik} y_k
\end{aligned}$$

Now we do some re-indexing and use the Kronecker Delta function.

$$\begin{aligned}
&= \sum_{ij} \delta_{ij} y_i^T y_j - \sum_{ij} W_{ij} y_i^T y_j - \sum_{ij} W_{ij} y_i^T y_j - \sum_{ijk} W_{ki} W_{kj} y_i^T y_j \\
&= \sum_{ij} (\delta_{ij} y_i^T y_j - W_{ij} y_i^T y_j - W_{ij} y_i^T y_j - \sum_k W_{ki} W_{kj} y_i^T y_j) \\
&= \sum_{ij} (\delta_{ij} - W_{ij} - W_{ij} - \sum_k W_{ki} W_{kj}) y_i^T y_j \\
&= \sum_{ij} M_{ij} y_i^T y_j
\end{aligned}$$

e) Using this result show

$$\sum_i ||y_i - \sum_j W_{ij} y_j||^2 = \text{trace}(YMY^T)$$

$$\begin{aligned}
\sum_i ||y_i - \sum_j W_{ij} y_j||^2 &= \sum_{ij} M_{ij} y_i^T y_j \\
&= \sum_{ij} M_{ij} (Y^T Y)_{ij} \\
&= \text{trace}(MY^T Y) \text{ by definition of trace.} \\
&= \text{trace}(YMY^T) \text{ because trace is invariant under cyclic permutations.}
\end{aligned}$$

f) Lastly, show that the eigenvector problem comes from the optimization problem minimize trace(YMY^T) subject to $YY^T = I$.

We will do this by taking the gradient of the Lagrangian and setting it equal to zero. We will be using properties concerning the derivative of trace which we assume to be true.

$$\frac{d}{dY} \text{trace}(YMY^T) = Y(M + M^T)$$

So via the taking the gradient of the Lagrangian we have the equations

$$YM + YM^T = \Lambda 2Y \text{ and } YY^T = I$$

M is symmetric because

$$M^T = ((I - W)^T (I - W))^T = (I - W)^T (I - W)^{T^T} = (I - W)^T (I - W) = M$$

Now we will leverage the facts that M is symmetric and $YY^T = Y^TY = I$.

$$\begin{aligned} 2YM &= 2\Lambda Y \\ M &= Y^T\Lambda Y \\ MY^T &= Y^T\Lambda \text{ which is the desired eigenvector problem.} \end{aligned}$$

Part B: Computing

Problem 1

Write a code to implement the LLE algorithm and apply it to Kohonen's animal data set. Compare with your SOM and Laplacian Eigenmaps data reduction results.

LLE as implemented using epsilon balls to determine the nearest neighbors rather than using the k-nearest neighbors algorithm. Notice, for ϵ large enough, we should have the same result as using the k-nearest neighbors algorithm $k = p - 1 = 15$.

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

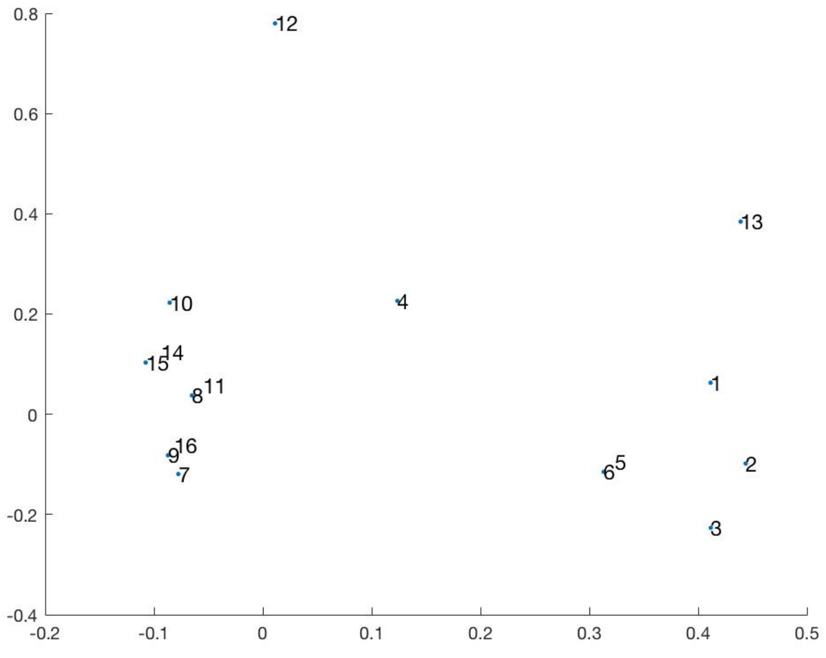

Figure 1: LLE using $\epsilon = 1$ to find the nearest neighbors.

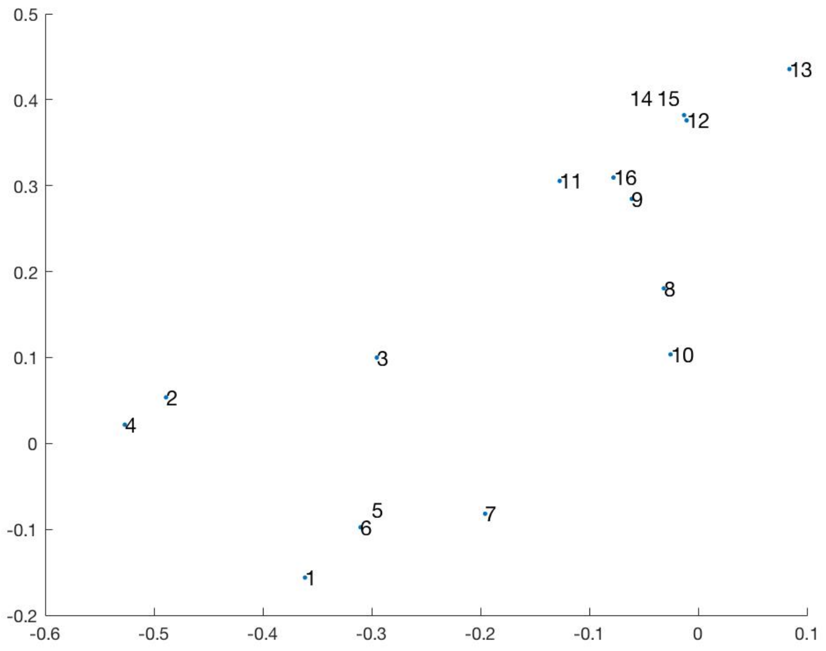

Figure 2: LLE using $\epsilon = 10$ to find the nearest neighbors.

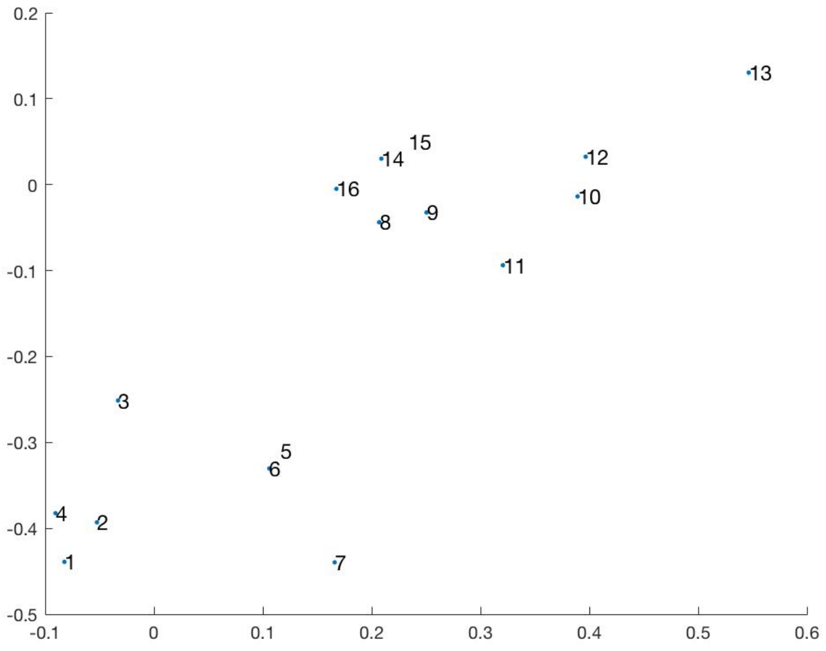

Figure 3: LLE using $\epsilon = 100$ to find the nearest neighbors.

SOM under-clusters data, Laplacian Eigenmaps over-clusters the data, while LLE finds a nice middle ground between over- and under-clustering. It also allows for a tunable parameter k in the k-nearest neighbors algorithm or ϵ in the ϵ ball algorithm (a portion of LGB clustering). Below is plots of the results of the three algorithms run on the Kohonen Animal data.

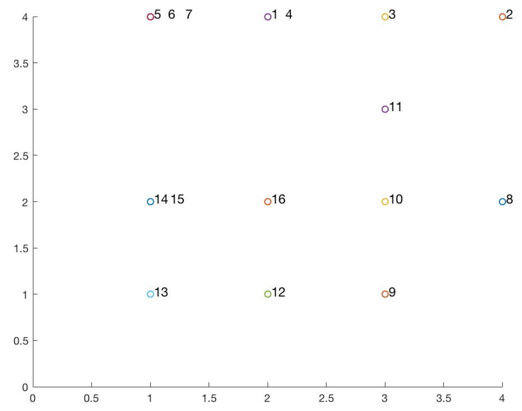

Figure 4: SOM result.

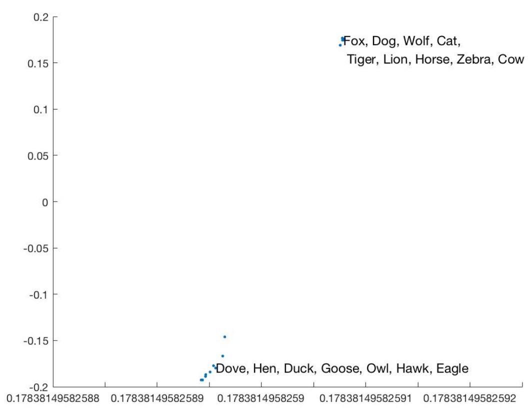

Figure 5: Laplacian Eigenmaps result..

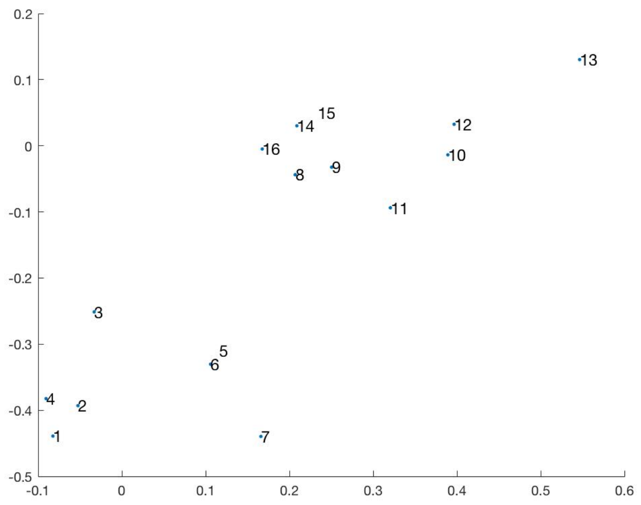

Figure 6: LLE with $\epsilon = 100$ ($k = 15$).

Problem 2

Write a code to implement the SVM algorithm using the quadratic programming subroutine in Matlab. Test your code using both separable and non-separable data sets in the plane. Plot the solution including the data (identifying the classes clearly) and the three SVM hyperplanes.

Below are the plots generated from an implementation of the SVM algorithm using the quadratic programming subroutine in Matlab on separable and non-separable data sets of 10 points in $\mathbb{R}^2$.

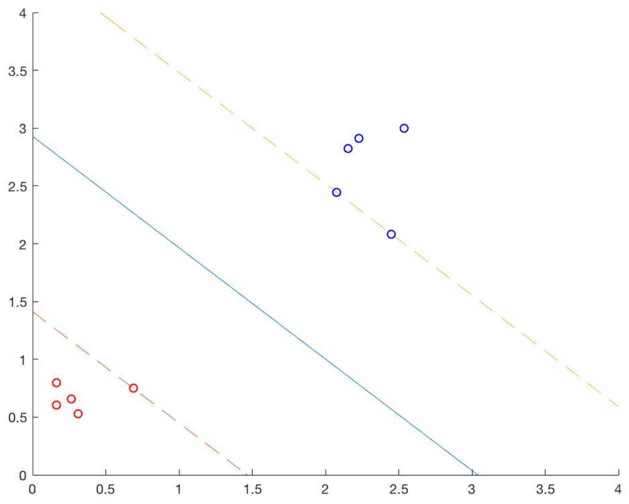

Figure 7: Separable data with $C = 1$.

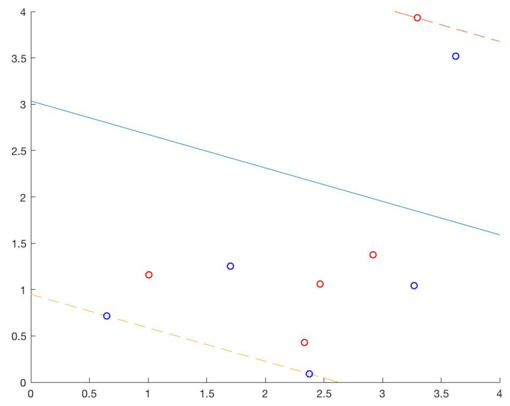

Figure 8: Non-separable data with $C = 1$.

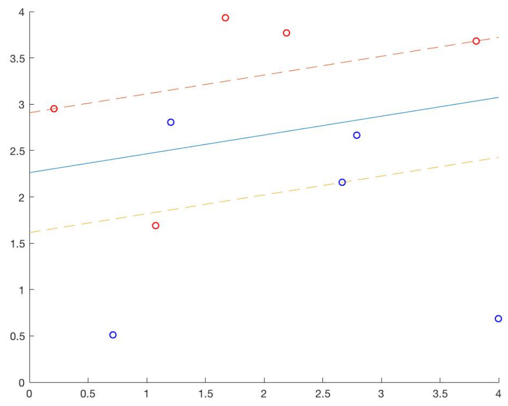

Figure 9: Non-separable data with $C = 1000$.

Problem 3

Write a code to implement the Sparse SVM algorithm using the linear programming sub- routine in Matlab. Test your code using both separable and non-separable data sets in the plane. Plot the solution including the same data as in problem 2 (again, identifying the classes clearly) and the three SSVM hyperplanes.

Below are the plots generated from an implementation of the SVM algorithm using the linear programming subroutine in Matlab on separable and non-separable data sets of 10 points in $\mathbb{R}^2$.

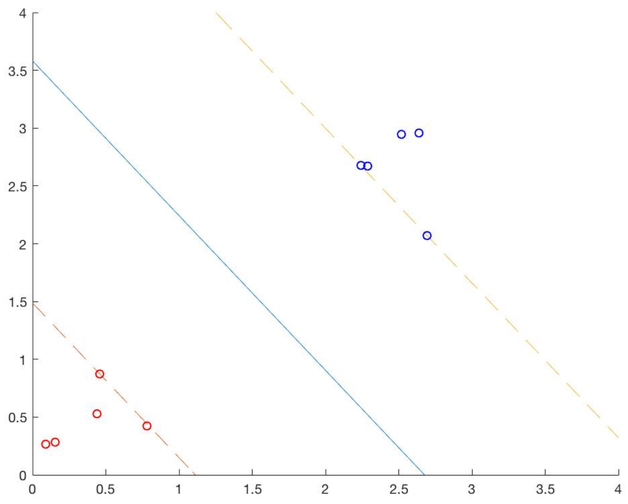

Figure 10: Separable data with $C = 1$.

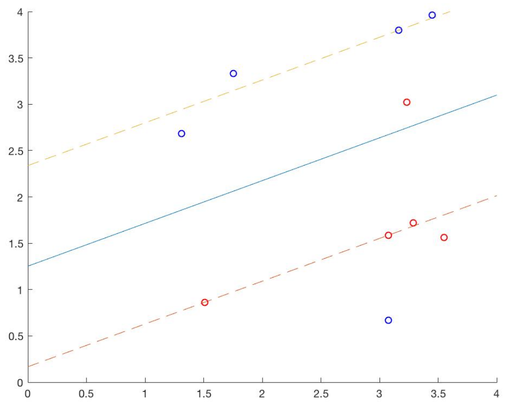

Figure 11: Non-separable data with $C = 1$.

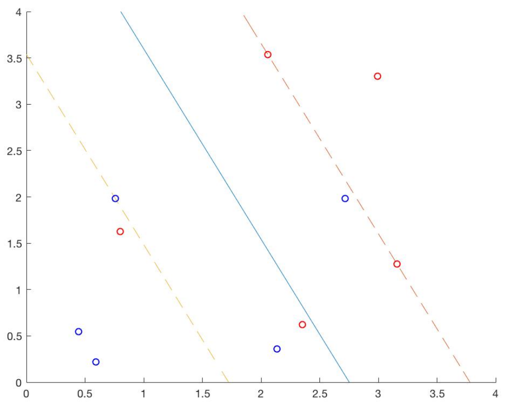

Figure 12: Non-separable data with $C = 1000$.

Problem 4

Using the two class hyperspectral data matrices X, Y (on canvas in the file HSIdata), build a classifier using the first 50 percent of each data set. Provide a confusion matrix for your predictions on the test data set consisting of the remaining 50 percent of the data.

The first half of the data was used for the training set and the second half of the data was used for the test set. The first table is found via an implementation of the SVM algorithm using the quadratic programming subroutine. The second table is found via an implementation of the SVM algorithm using the linear programming subroutine.

Using QuadProg	Predicted: in X	Predicted: in Y
Actual: in X	245	0
Actual: in Y	0	249

Using LinProg	Predicted: in X	Predicted: in Y
Actual: in X	245	0
Actual: in Y	1	248

Code

Below is the code for each of the algorithms:

Problem 1

```
 %%% LAPLACIAN EIGENMAPS %%%
 %description of the data: M(i,j) = attribute i, animal j
%i=1:13

%j=1:16
%Dove
%Hen
%Duck
%Goose
%Owl
%Hawk
%Eagle
%Fox
%Dog
%Wolf
%Cat
%Tiger
%Lion
%Horse
%Zebra
%Cow

%rows:
%small size
%medium size
%large size
%2 legs
%4 legs
%hair
%hooves
%mane
%feathers
%hunts
%runs
%flys
%swims

%the number one indicates that animal j possesses attribute i and zero indicates it does not.

X = [1  1  1  1  1  1  0  0  0  0  1  0  0  0  0  0;
0  0  0  0  0  0  1  1  1  1  0  0  0  0  0  0;
0  0  0  0  0  0  0  0  0  0  0  1  1  1  1  1;
1  1  1  1  1  1  1  0  0  0  0  0  0  0  0  0;
0  0  0  0  0  0  0  1  1  1  1  1  1  1  1  1;
0  0  0  0  0  0  0  1  1  1  1  1  1  1  1  1;
0  0  0  0  0  0  0  0  0  0  0  0  1  1  1;
0  0  0  0  0  0  0  0  0  1  0  0  1  1  1  0;
1  1  1  1  1  1  1  0  0  0  0  0  0  0  0  0;
0  0  0  0  1  1  1  1  0  1  1  1  1  0  0  0;
0  0  0  0  0  0  0  0  1  1  0  1  1  1  1  0;
1  0  0  1  1  1  1  0  0  0  0  0  0  0  0  0;
0  0  1  1  0  0  0  0  0  0  0  0  0  0  0  0];

p = length(X);

%define epsilon
epsilon = 100;



%create neighbor matrix
for i=1:p
    for j=1:p
        x = norm(X(:,i)-X(:,j))^2;
        if x <= epsilon
            Nbr(i,j) = 1;
        else
            Nbr(i,j) = 0;
        end
    end
end

%create c and ci (the inverses of the elements of c)
for i=1:p
    for j=1:p
        for k=1:p
            c(j,k,i) = ((X(:,i) - X(:,j))'*(X(:,i) - X(:,k)));
        end
    end
end

for i=1:p
    k = find(Nbr(i,:));
    WhtT = pinv(c(k,:,i))*ones(length(k),1);
    Wht = WhtT';
    for j=1:p
        if norm(Wht) ~= 0
            W(i,j) = Wht(j)/norm(Wht);
        else
            W(i,j) = 0;
        end
    end
    clear 'Wht' 'WhtT' 'k';
end



M = (eye(p)-W)'*(eye(p)-W);

[Yt,Lambda] = eig(M);


Y = Yt';
```

```
hold
scatter(Y(1,:),Y(2,:),'.');
for i=1:p
    if i ==15
        text(Y(1,i)+.025,Y(2,i)+.02,num2str(i),'FontSize', 12)
    elseif i ==5
        text(Y(1,i)+.01,Y(2,i)+.02,num2str(i),'FontSize', 12)
    else
        text(Y(1,i),Y(2,i),num2str(i),'FontSize', 12);
    end
end
```

Problem 2

```
 %%% SVM USING QUADPROG %%%

%%
%define n and p and c
p = 10;
n = 2;
c = 1;


%%
%Generate data and define initial y
% %create seperated 2D random data
%X=rand(n,p/2);
%X(1:n,p/2+1:p)=2+rand(n,p/2);
% create 2D random data
X=4*rand(n,p);

%define Y
Y = ones(5,1);
Y(6:10,1) = -1*ones(5,1);

%%
%Define the problem for quadprog
%define A
%make Y a diagonal matrix
Ydiag = zeros(p,p);
for i=1:p
    Ydiag(i,i) = Y(i);
end

A = [Ydiag*X' Y -1*eye(p,p);
zeros(p,n+1) -1*eye(p,p)];

%construct b
b = [-1*ones(p,1); zeros(p,1)];

%construct f
f = [zeros(n+1,1); c*ones(p,1)];
```

```
%construct H
H = zeros(n+p+1);
H(1:n,1:n) = eye(n);


%%
%use quadprog
x = quadprog(H,f,A,b)

%pull out variables from x
w = x(1:n);
b= x(n+1);
psi = x(n+2:n+1+p);

%%
%visualize result
hold
fplot(@(px) (-b-w(1)*px)/w(2))
fplot(@(px) (-1-b-w(1)*px)/w(2),'--')
fplot(@(px) (1-b-w(1)*px)/w(2),'--')
scatter(X(1,1:5),X(2,1:5),'r')
scatter(X(1,6:10),X(2,6:10),'b')
xlim([0 4])
ylim([0 4])
```

Problem 3

```
%%% SVM USING LINPROG %%%
%%
%define n and p and c
p = 10;
n = 2;
c = 1000;


%%
%Generate data and define initial y
% %create seperated 2D random data
%X=rand(n,p/2);
%X(1:n,p/2+1:p)=2+rand(n,p/2);
%create 2D random data
X=4*rand(n,p);

%define Y
Y = ones(5,1);
Y(6:10,1) = -1*ones(5,1);

%%
%Define the problem for linprog
%define A
%make Y a diagonal matrix
Ydiag = zeros(p,p);
for i=1:p
```

```
    Ydiag(i,i) = Y(i);
end

A = [eye(n) zeros(n,p+1) -1*eye(n,n);
-1*eye(n) zeros(n,p+1) -1*eye(n,n);
Ydiag*X' Y -1*eye(p,p) zeros(p,n);
zeros(p,n+1) -1*eye(p,p) zeros(p,n)];

%construct b
b = [zeros(2*n,1); -1*ones(p,1); zeros(p,1)];

%construct f
f = [zeros(n+1,1); c*ones(p,1); ones(n,1)];


%%
%use linprog
x = linprog(f,A,b);

%pull out variables from x
w = x(1:n);
b= x(n+1);
psi = x(n+2:n+1+p);
t = x(n+2+p:n*2+1+p);

%%
%visualize result
hold
fplot(@(px) (-b-w(1)*px)/w(2))
fplot(@(px) (-1-b-w(1)*px)/w(2),'--')
fplot(@(px) (1-b-w(1)*px)/w(2),'--')
scatter(X(1,1:5),X(2,1:5),'r')
scatter(X(1,6:10),X(2,6:10),'b')
xlim([0 4])
ylim([0 4])
```

Problem 4

```
 %%% Classifier SVM Implimentation %%%
%import data
data = load('IP2classes.mat');
dX = data.X;
dY = data.Y;

%define training data
halfX = floor(size(dX,2)/2);
halfY = floor(size(dY,2)/2);

trSet(:,1:halfX) = dX(:,1:halfX);
trSet(:,halfX+1 :halfX + halfY) = dY(:,1:halfY);

%define n and p and c
p = halfX + halfY;
```

```
n = size(trSet(:,1),1);
c =1;

%define Y
%X is positive Y is negative
Y = -1*ones(halfX,1);
Y((halfX + 1):(halfX + halfY),1) =  ones(halfY,1);

[w,b] = ssvm_l(trSet,Y,n,p,c); %switch l to q to use quadprog

%create confusion matrix store as Conf
Conf = zeros(2);
bonus = 0;
for i=halfX + 1: size(dX,2)
    if w'*dX(:,i)+b > 0
        Conf(2,2) = Conf(2,2)+1;
    elseif w'*dX(:,i)+b < 0
        Conf(2,1) = Conf(2,1)+1;
    else
        bonus = bonus +1;
    end
end

for i=halfY + 1: size(dY,2)
    if w'*dY(:,i)+b > 0
        Conf(1,2) = Conf(1,2)+1;
    elseif w'*dY(:,i)+b < 0
        Conf(1,1) = Conf(1,1)+1;
    else
        bonus = bonus +1;
    end
end



function [w,b] = ssvm_q(X,Y,n,p,c)
%Define the problem for quadprog
%define A
%make Y a diagonal matrix
Ydiag = zeros(p,p);
for i=1:p
    Ydiag(i,i) = Y(i);
end

A = [Ydiag*X' Y -1*eye(p,p);
zeros(p,n+1) -1*eye(p,p)];

%construct b
b = [-1*ones(p,1); zeros(p,1)];

%construct f
f = [zeros(n+1,1); c*ones(p,1)];

%construct H
```

```
H = zeros(n+p+1);
H(1:n,1:n) = eye(n);


%%
%use quadprog
x = quadprog(H,f,A,b)

%pull out variables from x
w = x(1:n);
b= x(n+1);
psi = x(n+2:n+1+p);
end



function [w,b] = ssvm_l(X,Y,n,p,c)
Ydiag = zeros(p,p);
for i=1:p
    Ydiag(i,i) = Y(i);
end

A = [eye(n) zeros(n,p+1) -1*eye(n,n);
-1*eye(n) zeros(n,p+1) -1*eye(n,n);
Ydiag*X' Y -1*eye(p,p) zeros(p,n);
zeros(p,n+1) -1*eye(p,p) zeros(p,n)];

%construct b
b = [zeros(2*n,1); -1*ones(p,1); zeros(p,1)];

%construct f
f = [zeros(n+1,1); c*ones(p,1); ones(n,1)];


%%
%use linprog
x = linprog(f,A,b);

%pull out variables from x
w = x(1:n);
b= x(n+1);
psi = x(n+2:n+1+p);
t = x(n+2+p:n*2+1+p);

end
```