STAT 3710 Homework 2

Yuhui Ni

1. (a) since v and v' are two eigenvectors of A with corresponding eigenvalues $\lambda \neq \lambda'$, we have

$$\begin{cases} Av = \lambda v & ① \\ Av' = \lambda' v' & ② \end{cases}$$

② $\times v^T \Rightarrow$ $v^T Av' = \lambda' v^T v'$ ③

$(Av)^T = v^T A^T = v^T A = (\lambda v)^T = \lambda v^T$ ④

combine ② and ④, we have $\lambda v^T v' = \lambda' v^T v'$

$\Rightarrow$ $(\lambda - \lambda') v^T v' = 0$

since $\lambda \neq \lambda'$, we can get $v^T v' = 0$

i.e. v is orthogonal to v'

(b) suppose $\{w_1, w_2, \cdots, w_k\}$ are the original eigenvectors of A with eigenvalue λ, we can apply Gram-Schmidt process to $\{w_1, w_2, \cdots, w_k\}$, get mutually orthogonal vectors.

$$v_1 = \frac{w_1}{\|w_1\|}, \quad v_2 = w_2 - \langle w_2, v_1 \rangle \cdot v_1 \quad v_2 = \frac{w_2}{\|v_2\|},$$

$$v_3 = w_3 - \langle w_3, v_1 \rangle v_1 - \langle w_3, v_2 \rangle v_2, \quad v_3 = \frac{v_3}{\|v_3\|},$$

$$\vdots$$

$$v_k = w_k - \sum_{j=1}^{k-1} \langle w_k, v_j \rangle v_j, \qquad v_k = \frac{v_k}{\|v_k\|}.$$

since we have $A w_j = \lambda w_j$, $j = 1, 2, \cdots, k$.

$\Rightarrow$ $A v_1 = \lambda v_1$,

suppose, $A v_j = \lambda v_j$ is true, for $j = 1, 2, \cdots, m$, $m \leq k$,

then $A v_{m+1} = \frac{1}{c} A\left(w_{m+1} - \sum_{j=1}^{m} \langle w_{m+1}, v_j \rangle v_j\right) = \frac{\lambda}{c}\left(w_{m+1} - \sum_{j=1}^{m} \langle w_{m+1}, v_j \rangle v_j\right) = \lambda v_{m+1}$.

Hence, $A v_j = \lambda v_j$, for $j = 1, 2, \cdots, k$.

At the same time, $v_1, \cdots v_k$ are mutually orthogonal due to Gram-Schmidt process.

Therefore, we can find $V = \operatorname{span}\{v_1, \cdots v_k\}$, $v_1, \cdots v_k$ are mutually orthogonal and v_i, $i = 1, 2, \cdots, k$ are eigenvector A with λ.

(c) According to (a) and (b), we can construct an eigenvector matrix as a orthogonal matrix, i.e. $P = (v_1, v_2, \dots, v_d)$, $v_i^T v_i = 1$, $v_i^T v_j = 0 \ (i \neq j)$

$\therefore \quad P P^T = I. \quad P^T = P^{-1}$

According to theory of eigen decomposition

$$A = P \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_d \end{pmatrix} P^{-1} = P \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_d \end{pmatrix} P^T = (v_1, \dots v_d) \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_d \end{pmatrix} \begin{pmatrix} v_1^T \\ \vdots \\ v_d^T \end{pmatrix}$$

i.e. $A = \sum_{i=1}^{d} \lambda_i v_i v_i^T$

(d) if $w = \underset{w \in R^d / \{0\}}{\operatorname{argmin}} \frac{w^T A w}{\|w\|^2}$

then $\frac{(aw)^T A (aw)}{\|aw\|^2} = \frac{w^T A w}{\|w\|^2}$.

$\therefore$ we can solve equivalent problem find $\underset{w \in R^d / \{0\}}{\operatorname{argmin}} w^T A w$, $\|w\| = 1$.

$v_1, \dots v_d$ are orthogonal basis of R^d.

$\therefore$ w can be represented as $w = \sum_{j=1}^{d} a_j v_j$, $\sum_{j=1}^{d} a_j^2 = 1$.

$\therefore \quad w^T A w = \sum_{i=1}^{d} \lambda_i w^T v_i v_i^T w = \sum_{i=1}^{d} \lambda_i a_i^2$

$\because \sum_{j=1}^{d} a_j^2 = 1$

$\therefore \underset{w \in R^d / \{0\}}{\operatorname{argmin}} = \underset{}{\operatorname{argmin}} \sum_{i=1}^{d} \lambda_i a_i^2$, which is minimized when $a_1 = 1$, $a_2 = a_3 = \dots = a_d = 0$

$\therefore \quad w = v_1$

similarly, $\underset{w \in R^d / \{0\}}{\operatorname{argmax}} = \underset{}{\operatorname{argmax}} \sum_{i=1}^{d} \lambda_i a_i^2$, which is maximized when $a_1 = a_2 = \dots = a_{d-1} = 0$, $a_d = 1$

$\therefore \quad w = v_d$.

$\therefore \quad \underset{w \in R^d / \{0\}}{\operatorname{argmin}} \frac{w^T A w}{\|w\|^2} = v_1 \qquad \underset{w \in R^d / \{0\}}{\operatorname{argmax}} \frac{w^T A w}{\|w\|^2} = v_d$.

2. (a) Since $P_1 \cdots P_k$ are mutually orthogonal basis of V.

$\therefore$ y can be represented as $y = P\beta$.

Here $P = (P_1, P_2, \cdots P_k)$ $\beta = (\beta_1, \cdots \beta_k)^T$.

$\therefore$ $L = \|x-y\|^2 = \|x - P\beta\|^2 = (x-P\beta)^T(x-P\beta) = x^Tx - 2x^TP\beta + \beta^TP^TP\beta$

$\frac{\partial L}{\partial \beta} = 2P^TP\beta - 2P^Tx = 0$

$\Rightarrow$ $\hat{\beta} = (P^TP)^{-1}P^Tx = P^Tx$

$\therefore$ $x_V = P\hat{\beta} = PP^Tx = (P_1 \cdots P_k)\begin{pmatrix} P_1^T \\ \vdots \\ P_k^T \end{pmatrix}x = \sum_{i=1}^{k}(x \cdot P_i)\cdot P_i$

$\therefore$ x_V is given by the orthogonal projection of x to V.

(b) $E(v) = \frac{1}{n}\sum_{i=1}^{n}\|x_i - PP^Tx_i\|^2 = \frac{1}{n}\sum_{i=1}^{n}x_i^T(I-PP^T)\cdot x_i$

$= \frac{1}{n}\sum_{i=1}^{n}\|x_i\|^2 - \frac{1}{n}\sum_{i=1}^{n}x_i^TPP^Tx_i$

$\frac{1}{n}\sum_{i=1}^{n}x_i^TPP^Tx_i = tr\left(\frac{1}{n}\sum_{i=1}^{n}x_i^TPP^Tx_i\right) = tr\left(P^T\frac{\sum_{i=1}^{n}x_ix_i^T}{n}P\right) = \sum_{j=1}^{k}P_j^T\hat{\Sigma}P_j$, $\hat{\Sigma} = \frac{1}{n}\sum_{i=1}^{n}x_ix_i^T$

Hence, the goal is transformed to. max $\sum_{j=1}^{k}P_j^T\hat{\Sigma}P_j$,

subject to. $P_j^TP_j = 1$, $j = 1, \cdots k$, $P_i^TP_j = 0$

Let $L = \sum_{j=1}^{k}P_j^T\hat{\Sigma}P_j - \sum_{j=1}^{k}\lambda_j(P_j^TP_j - 1)$

$\frac{\partial L}{\partial \lambda_j} = P_j^TP_j - 1 = 0$. $\frac{\partial L}{\partial P_j} = 2\hat{\Sigma}P_j - 2\lambda_jP_j = 0$

This means λ_j is the eigenvalue of $\hat{\Sigma}$. Then $\hat{P_j}^T\Sigma P_j = \lambda_j$,

$\therefore$ $\sum_{j=1}^{k}P_j^T\hat{\Sigma}P_j = \sum_{j=1}^{k}\lambda_j$, This is maximized by choosing the k- leading eigenvalue.

At the same time, the condition $P_i^TP_j = 0$, $i \neq j$ is also achieved by choosing the k-leading eigenvectors of $\hat{\Sigma} = \frac{1}{n}\sum_{i=1}^{n}x_ix_i^T$

3. (a) write $X = (x_1, \cdots x_n)$

Then $k = X^T X$

$rank(k) = rank(X^T X) \le rank(X^T) + rank(X) - d \le d + d - d = d$.

i.e. $rank(k) \le d$.

(b) since k is positive semi-definite matrix (symmetric) of rank r.

$\Rightarrow k = P \Lambda P^T$ $\quad rank(\Lambda) = r$ $\quad \Lambda = \begin{pmatrix} \lambda_1 & & \\ & \lambda_r & \\ & & 0 \end{pmatrix}$

$\therefore k = P \Lambda^{\frac{1}{2}} (P \Lambda^{\frac{1}{2}})^T = U U^T$

Here $U = P \Lambda^{\frac{1}{2}} = P \begin{pmatrix} \sqrt{\lambda_1} & & \\ & \sqrt{\lambda_r} & \\ & & 0 \end{pmatrix} = (N_{n \times r} \vdots O_{n \times (d-r)})$

if we suppose $X^T = P \underbrace{\begin{pmatrix} \sqrt{\lambda_1} & & \\ & \sqrt{\lambda_r} & \\ & & 0 \end{pmatrix}}_{A}_{n \times d} = (N_{n \times r} \vdots O_{n \times (d-r)})$ X^T is a $n \times d$ matrix.

A is a $n \times d$ matrix.

Hence, we can get $k_{ij} = \sum_{l=1}^{r} N_{il} N_{jl} = \sum_{l=1}^{n} U_{il} U_{jl} = x_i^T x_j$

This means $k = X^T X$, X^T is a $n \times d$ matrix. X is a $d \times n$ matrix

$X = A^T P^T = \begin{pmatrix} \sqrt{\lambda_1} & & & \\ & \sqrt{\lambda_r} & & \\ & & 0 & \\ & & & 0 \end{pmatrix} P^T$, A^T is a $d \times n$ matrix.

$\therefore$ k is a gram matrix of column vectors of $X = A^T P^T$

4. (a) $P^2 = (I - \frac{1}{n} 1 1^T)^2 = I^2 - \frac{2}{n} 1 1^T + \frac{1}{n^2} 1^T 1 1^T = I^2 - \frac{2}{n} 1 1^T + \frac{1}{n^2} \times n \times 1 1^T$

$= I^2 - \frac{1}{n} 1 1^T$

$= P$

i.e. P is a projection matrix

(b) if $PV = 0$

① if $V = 0$, it's obviously true.

② if $V \neq 0$, $PV = 0 \Leftrightarrow (I - \frac{1}{n}11^T)V = 0$, $\Rightarrow V = 1(\frac{1}{n}1^TV) \Rightarrow V = \lambda 1$. $\lambda = \frac{1}{n}1^TV$.

i.e. $V \in U$

Hence, the kernel of P is the line $U = \{\lambda 1\}$.

(c) write $M = -\frac{1}{2}PDP$

$$PDP = (I - \frac{1}{n}11^T)D(I - \frac{1}{n}11^T) = D - \frac{1}{n}11^TD - \frac{1}{n}D11^T + \frac{1}{n^2}11^TD11^T$$

$$(PDP)_{ij} = \|x_i - x_j\|^2 - \frac{1}{n}\left(\sum_{l=1}^{n}\|x_l - x_j\|^2\right) - \frac{1}{n}\left(\sum_{k=1}^{n}\|x_i - x_k\|^2\right) + \frac{1}{n^2}\left(\sum_{l=1}^{n}\sum_{k=1}^{n}\|x_l - x_k\|^2\right)$$

$$= \|x_i\|^2 + \|x_j\|^2 - 2x_i^Tx_j - \frac{1}{n}\left(\sum_{l=1}^{n}\|x_l\|^2 - 2\sum_{l=1}^{n}x_l^Tx_j + n\|x_j\|^2\right) - \frac{1}{n}\left(n\|x_i\|^2 - 2\sum_{k=1}^{n}x_i^Tx_k + \sum_{k=1}^{n}\|x_k\|^2\right)$$

$$+ \frac{1}{n^2}\left(\left(\sum_{l=1}^{n}x_l\right)^T\left(\sum_{l=1}^{n}x_l\right) - 2\sum_{l=1}^{n}\sum_{k=1}^{n}x_l^Tx_k\right)$$

$$= -2x_i^Tx_j + \frac{2}{n}\sum_{l=1}^{n}x_l^Tx_j + \frac{2}{n}\sum_{k=1}^{n}x_i^Tx_k - \frac{2}{n^2}\left(\sum_{l=1}^{n}x_l\right)^T\left(\sum_{l=1}^{n}x_l\right)$$

$$\therefore m_{ij} = x_i^Tx_j - \frac{1}{n}\sum_{l=1}^{n}x_l^Tx_j - \frac{1}{n}\sum_{k=1}^{n}x_i^Tx_k + \frac{1}{n^2}\left(\sum_{l=1}^{n}x_l\right)^T\left(\sum_{l=1}^{n}x_l\right)$$

$$= \left(x_i - \frac{1}{n}\sum_{l=1}^{n}x_l\right)^T\left(x_j - \frac{1}{n}\sum_{k=1}^{n}x_k\right)$$

$$= (x_i - u)^T(x_j - u)$$

$$= \hat{G}_{ij}$$

$$\therefore \hat{G} = -\frac{1}{2}PDP$$

5. Here we denote $Y = (y_1, y_2, \ldots, y_n)$

Then $\Psi(y_1, \ldots, y_N) = \sum_{i=1}^{n}\|y_i - \sum_j w_{ij}y_j\|^2$

$$= \sum_{i=1}^{n}y_i^Ty_i - \sum_{i=1}^{n}y_i^T\sum_j w_{ij}y_j - \sum_{i=1}^{n}\sum_j w_{ij}y_j^Ty_i + \sum_{i=1}^{n}y_i\left(\sum_j w_{ij}y_j\right)^T\left(\sum_k w_{ik}y_k\right)$$

$$= \mathrm{tr}(Y^TY) - \mathrm{tr}(Y^TYW) - \mathrm{tr}(YW^TY^T) + \mathrm{tr}\left((YW)^TYW\right)$$

$$= tr\left(Y (I-W)^T (I-W) Y^T \right)$$

we rewrite $Y = \begin{pmatrix} z_1^T \\ z_2^T \\ \vdots \\ z_p^T \end{pmatrix}$, then we have

$$\Phi(y_1, \cdots y_n) = \Phi(z_1, \cdots z_p) = \sum_{j=1}^{p} z_j^T (I-W)^T (I-W) z_j$$

$$\sum_{i=1}^{n} y_i = \vec{0} \iff 1^T z_j = 0, \text{ for } j = 1, 2, \cdots, p$$

$$\frac{1}{n} \sum_{i=1}^{n} y_i y_i^T = I \iff \frac{1}{n} z_i^T z_i = 1, \quad z_i^T z_j = 0, \quad (i \neq j)$$

Hence, the problem becomes $\min \sum_{j=1}^{p} z_j^T (I-W)^T (I-W) z_j$

subject to $1^T z_j = 0$ for $j = 1, 2, \cdots, p$

$\frac{1}{n} z_i^T z_i = 1, \quad z_i^T z_j = 0 \ (i \neq j)$

Denote $(I-W)^T (I-W) = A$

Then Let $L = \sum_{j=1}^{p} z_j^T A z_j - \sum_{j=1}^{p} \lambda_j \left(\frac{1}{n} z_j^T z_j - 1 \right)$

$$\frac{\partial L}{\partial z_j} = 2 A z_j - \frac{2 \lambda_j}{n} z_j = 0 \quad \text{i.e.} \quad A z_j = \frac{\lambda_j}{n} z_j$$

Hence, z_j is the eigenvector of A

$$\sum_{j=1}^{p} z_j^T A z_j = \sum_{j=1}^{p} \lambda_j.$$

Since the smallest eigenvalue of A is 0, we need to discard it.

∴ we can choose the $(p+1)$ eigenvectors that corresponds to $(p+1)$ smallest eigen values, and discard the eigenvector 1. that corresponds to eigen value 0.

Now, we have $z_1, \cdots z_p$ eigenvectors of A. that match p smallest eigen values (exclude 0), they are mutually independent., so they suffice $z_i^T z_j = 0$, we can also normalize them such that $\frac{1}{n} z_i^T z_i = 1$,

At the same time, 1 is orthogonal to $z_1, \cdots z_p$, therefore, we have $1^T z_j = 0$ for $j = 1, 2 \cdots p$, Thus, these z_i's are the answer.

Homework2

April 19, 2018

1 Problem 6

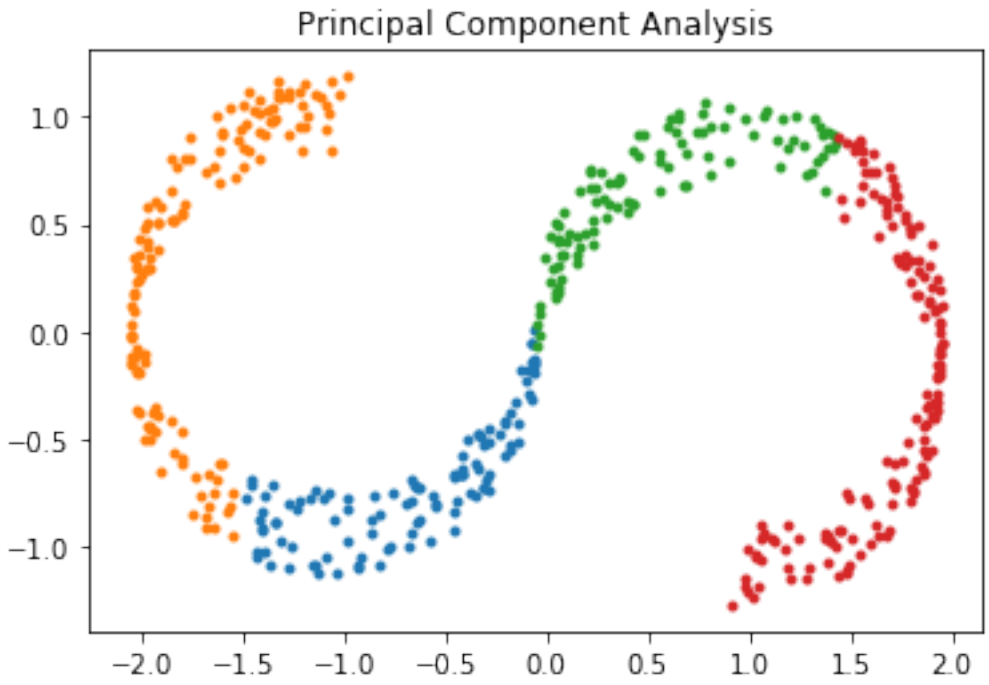
1.1 PCA

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import preprocessing
#read the txt file
data=np.loadtxt("3Ddata.txt")

X=data[:,0:3]
label=data[:,3]

#PCA algorithm
def PCA(X,k):
    nsample=X.shape[0]
    ndim=X.shape[1]

    #center the data
    X=X-np.mean(X,0)

    #compute the sample covariance matrix
    sigma=np.dot(X.T,X)/nsample

    #compute the eigen value and eigen vector
    w,v=np.linalg.eig(sigma)
    ind=np.argsort(w)
    ind=ind[::-1]
    eighv=v[:,ind[:k]]

    #compute the scores matrix
    scores=-np.dot(X,eighv)

    return scores
```

In [2]:
```
#plot the image
X_pca=PCA(X,2)
for i in range(1,5):
```

```
        plt.plot(X_pca[label==i,0],X_pca[label==i,1],'.')
    plt.title("Principal Component Analysis")
```

Out[2]: Text(0.5,1,'Principal Component Analysis')



1.2 Isomap

In [3]:
```
#Isomap analysis
inf=np.inf
#find k-nearest neighbour of given data
def k_nearest_neighbour(i,X,k):
    temp=X-X[i,:]
    distance=np.sqrt(np.sum(np.multiply(temp,temp),1))

    #sort the index and distance with asecending
    index=np.argsort(distance)[1:(k+1)]
    distance_knearest=distance[index]

    return index,distance_knearest

#construct graph distance matrix
def graph_distance(X,k):
    nsample=X.shape[0]
    graph_matrix=np.zeros((nsample,nsample))
    for i in range(nsample):
```

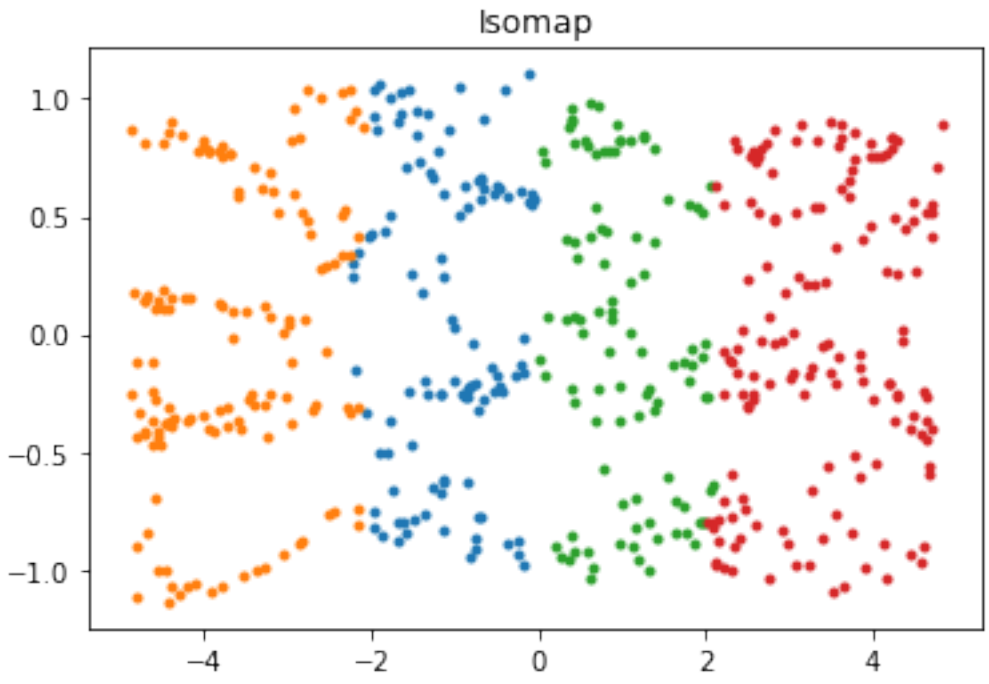
```
        index,distance=k_nearest_neighbour(i,X,k)
        index_remain=np.delete(np.arange(nsample),index)

        #assign k-neareast neighbours with distance
        graph_matrix[i,index]=distance

        #assign un-nearest points with inf
        graph_matrix[i,index_remain]=inf

        #assign itself with 0
        graph_matrix[i,i]=0

    return graph_matrix

#compute shortest path distance
def shorest_path(graph_matrix):
    #assign A with initial value of graph_matrix
    A=graph_matrix.copy()
    nsample=A.shape[0]
    #FloydWarshall algorithm
    for k in range(nsample):
        for i in range(nsample):
            for j in range(nsample):
                if A[i,j]>A[i,k]+A[k,j]:
                    A[i,j]=A[i,k]+A[k,j]
    return A

#compute Multidimensional scaling
def MDS(A,ndim):
    nsample=A.shape[0]

    #compute centered gram matrix
    P=np.eye(nsample)-np.ones((nsample,nsample))/nsample
    A=A**2
    gram_matrix=-0.5*np.dot(P,A).dot(P)

    #compute eigendecomposition of gram matrix
    w,v=np.linalg.eig(gram_matrix)
    ind=np.argsort(w)
    ind=ind[::-1]

    #extract diagnoal matrix and low-dimensional data
    lamb=np.concatenate((np.diag(np.sqrt(w[ind[:ndim]])),\
                         np.zeros((nsample-ndim,ndim))),axis=0)
    y=np.dot(v[:,ind],lamb)

    return np.real(y)
```

```
        #Isomap method
        def Isomap(X,k,ndim):

            #compute graph distance
            graph_matrix=graph_distance(X,k)

            #compute the shorest path distance
            A=shorest_path(graph_matrix)

            #compute Multidimensional scaling
            y=MDS(A,ndim)

            return y
```

In [4]:
```
k=10
ndim=2
y=Isomap(X,k,ndim)
```

In [5]:
```
for i in range(1,5):
    plt.plot(y[label==i,0],y[label==i,1],'.')
plt.title("Isomap")
```

Out[5]:
```
Text(0.5,1,'Isomap')
```



1.3 Locally Linear Embedding

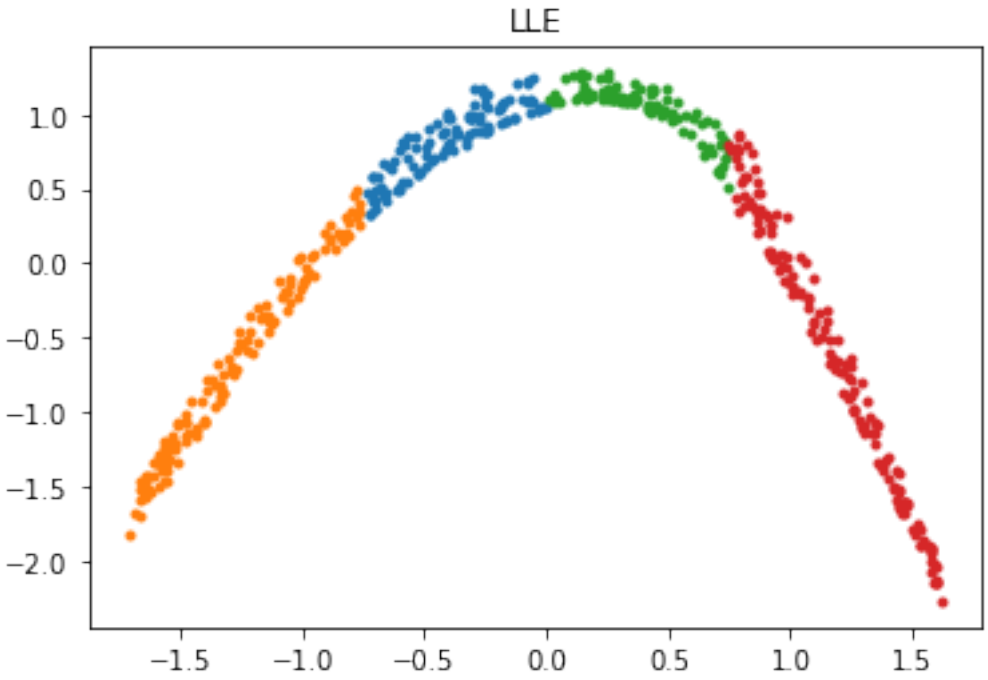
In [6]:
```
#find the weights
def weight_matrix(X,k,alpha):
    nsample=X.shape[0]
    weight=np.zeros((nsample,nsample))
    for i in range(nsample):
        #find k-nearest neighbours
        index,distance=k_nearest_neighbour(i,X,k)
        Z=X[index,:]-X[i,:]

        #compute gram matrix of neighbours
        gram_matrix=np.dot(Z,Z.T)

        #compute the weights of neighbours
        weight[i,index]=np.dot(np.linalg.inv(gram_matrix+np.eye(k)*alpha),np.ones((k,1))

        #normalize it to sum 1
        weight[i,index]=weight[i,index]/np.sum(weight[i,index])

    return weight

#find the coordinates
def new_coordinate(X,weight,ndim):
    nsample=X.shape[0]

    #compute matrix (I-W)'*(I-W)
    M=np.dot((np.eye(nsample)-weight).T,(np.eye(nsample)-weight))

    #compute eigen value and vector
    w,v=np.linalg.eig(M)
    ind=np.argsort(w)

    #extract bottom k+1 eigenvectors excluding 1 vector
    coords=v[:,ind[1:(ndim+1)]]*np.sqrt(nsample)

    return np.real(coords)

def LLE(X,k,ndim,alpha):

    #find the weights
    weight=weight_matrix(X,k,alpha)

    #find the coordinates
    coords=new_coordinate(X,weight,ndim)

    return coords
```

In [7]:
```
k=10
```

```
ndim=2
alpha=0.003
coords=LLE(X,k,ndim,alpha)
```

```
In [8]: for i in range(1,5):
            plt.plot(coords[label==i,0],coords[label==i,1],'.')
        plt.title("LLE")
```

```
Out[8]: Text(0.5,1,'LLE')
```



As we can see, these three methods have different reduction results. For PCA, its goal is to find a projection plane such that mean squared error of projection is minimized. The original 3-dimensional data is like a S shape. Therefore, the PCA reduction is a S shape. As for ISomap, it keeps the relationship between neighbors and flat the data in oringal space. So we can see a rectangle which is clearly splited into four parts. LLE method assumes each point should be approximately reconstructable as a linear combination of its neighbors and also keeps the relationships between neighbors. Since it's constructed by linear combination of its neighbors, it will be close to linear shape just as we see in the above figure.

2 Problem 7

```
In [9]: #loading data
        train=np.loadtxt('train35.digits')
        train=np.concatenate((train,np.ones((2000,1))),axis=1)
        train_label=np.loadtxt("train35.labels")
```

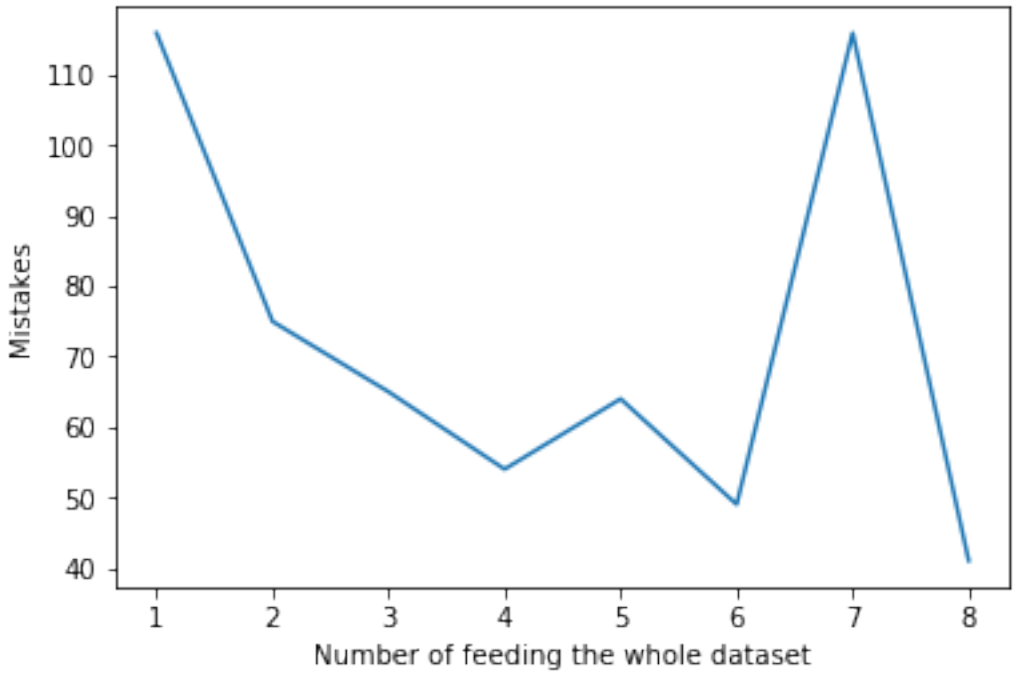
```
test=np.loadtxt("test35-1.digits")
test=np.concatenate((test,np.ones((200,1))),axis=1)

#normalize the data to unit norm
train=preprocessing.normalize(train, norm='l2')
test=preprocessing.normalize(test, norm='l2')
```

In [10]:
```
def batchperceptron(train,train_label,M):

    n=train.shape[0] #sample size
    ndim=train.shape[1] #sample dimension
    #initalize the weight
    w=np.zeros(ndim)
    error=np.zeros(M)
    k=0
    flag=1
    while(k<M and flag):
        flag=0
        for i in range(n):
            u=np.sum(np.multiply(w,train[i,:]))
            if train_label[i]*u<=0:
                w=w+train_label[i]*train[i,:]
                flag=1
        error[k]=np.sum(np.multiply(train_label,np.sum(np.multiply(train,w),1))<=0)
        k=k+1
    return w,error

#using K-folding cross validation to choose optimal M
def M_crossvalidation(train,train_label,K):
    nsample=train.shape[0]
    subsize=nsample/K
    m=np.arange(1,15)
    error=np.zeros(len(m))
    for M in m:
        for i in range(K):
            test_ind=np.arange(i*subsize,(i+1)*subsize,dtype="int")
            train_ind=np.delete(np.arange(nsample),test_ind)
            w,er=batchperceptron(train[train_ind,:],train_label[train_ind],M)
            error[M-1]+=np.sum(np.multiply(train_label[test_ind],\
                               np.sum(np.multiply(train[test_ind],w),1))<=0)

    M_best=np.argmin(error)+1

    return M_best
```

In [11]:
```
#choose best M
M_best=M_crossvalidation(train,train_label,5)
print(M_best)
```

8

```
In [12]: #train w using train data
         w,error=batchperceptron(train,train_label,M_best)

         #prediciton test label
         test_label=np.sum(np.multiply(test,w),1)
         test_label[test_label>0]=1
         test_label[test_label<=0]=-1

         #save test label
         np.savetxt("test35.predictions", test_label, fmt="%d", delimiter=",")

In [13]: plt.plot(np.arange(1,M_best+1),error)
         plt.xlabel('Number of feeding the whole dataset')
         plt.ylabel('Mistakes')

Out[13]: Text(0,0.5,'Mistakes')
```



According to above figure, we see that the number of mistakes decreases with increasing examples at first, but it increases after 5 times feeding and 7 times feeding. In other words, it will not always decrease with increasing examples, there exists some certain number that achieve the smallest mistake number.