EE609 Term Paper

Visualization of High Dimensional Data using Convex Optimization

Shreesh Ladha - 12679
Shreyash Pandey - 12683

Abstract

In today's era, data is being generated at a rapid pace. A variety of domains such as speech recognition, fMRI, Natural Language Processing work with data of high dimensionality. There is a need to reduce the dimension, both for better understanding of the problem as well as to address the issues that arise in higher dimensions(such as overfitting, higher computational complexity etc). In this regard, we have done a survey of four different techniques for dimensionality reduction. These techniques have been designed especially to extract non-linearity in the data using convex optimization methods.

1 Introduction

Dimensionality reduction has become a very popular tool in all forms of research. It allows the user to perform exploratory analysis and comment about hidden patterns, properties in the data and to better get an intuition of what is happening behind the scenes. Traditionally, people have been using Principal Component Analysis(PCA) and Multi-Dimensional Scaling(MDS) for dimension reduction purposes. However, with the growing complexity of data, such conventional linear dimensionality reduction techniques have become obsolete. Many non-linear dimension reduction techniques have, as a reason, come up in recent times. It has also been seen that most realworld data is likely to lie on a nonlinear manifold, and modelling any technique with this background is able to achieve very good results. In our project we have dealt with a specific class of such techniques which use manifolds to reduce the data to a lower dimensional space using convex optimization methods. We have divided our report into four sections excluding this section. In section II we discuss the algorithm of the techniques, in section III we report the results on some of our chosen datasets, in section IV we analyse and compare the discussed techniques followed by a conclusion as the last section.

2 Dimension Reduction Techniques

In this section we first talk about the intuition behind the techniques followed by their mathematical formulations. In all cases, $y \in R^p$ is considered to be the vector in high dimension, and $x \in R^d$ its lower dimesional embedding.

2.1 Isomap

In case of a non-linear manifold, a simple euclidean distance between two points would give a wrong impression of the actual distance between two points. Isomap tries to resolve this issue by structuring the manifold as a graph, the points as nodes and using the shortest path distance between two points as the estimated distance between the points in the non-linear manifold(also called geodesic distance) and formulates a geodesic distance matrix D^y. The graph is constructed using a k-NN approach i.e each node is connected to its k nearest neighbours(a free parameter). The final embedding in a lower dimension(which uses euclidean distance) is the one that most faithfully preserves the manifold's intrinsic geometry as estimated above. The problem reduces to classical Multi-Dimensional Scaling(MDS) where distances are converted to inner products. This is done because inner products uniquely characterize the geometry of data, and also support efficient optimization. The objective function is,

$$min_X ||XX^T - YY^T||_2^2 \tag{1}$$

This objective function is convex, and the closed form solution for this optimization problem can be found. Note that, here we have a distance matrix D^y of geodesic distances and we need to find a representation of YY^T in terms of D^y. An element of the matrix $G = YY^T$ if of the form $y_i^T y_j$. A centering assumption such as $\sum_{ij} y_{ij} = 0$ ensures that we

get a unique representation of G in terms of D^y.

$$
\begin{gathered}
d_{ij}^2 = (y_i - y_j)^T(y_i - y_j) = y_i^T y_i + y_j^T y_j - 2y_i^T y_j \text{ so,} \\
\frac{1}{n}\sum_{i=1}^{n} d_{ij}^2 = \frac{1}{n}\sum_{i=1}^{n} y_i^T y_i + y_j^T y_j \text{ and similarly, } \frac{1}{n}\sum_{j=1}^{n} d_{ij}^2 = \frac{1}{n}\sum_{j=1}^{n} y_j^T y_j + y_i^T y_i \\
\text{and } \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}^2 = \frac{2}{n}\sum_{i=1}^{n} y_i^T y_i \text{ which gives} \\
G_{ij} = y_i^T y_j = -\frac{1}{2}(d_{ij}^2 - \frac{1}{n}\sum_{i=1}^{n} d_{ij}^2 - \frac{1}{n}\sum_{j=1}^{n} d_{ij}^2 + \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}^2) = -\frac{1}{2}(a_{ij} - a_{i.} - a_{j,} + a_{..}) \\
\text{where } a_{i.} = \frac{1}{n}\sum_{j}^{n} a_{ij}, a_{j.} = \frac{1}{n}\sum_{i}^{n} a_{ij}, a_{..} = \frac{1}{n}\sum_{ij}^{n} a_{ij}
\end{gathered}
\tag{2}
$$

G can now be written and computed as $G = -\frac{1}{2}HSH$, where $H = I - \frac{1}{n}\mathbf{1}\mathbf{1}^T$ and $S_{ij} = a_{ij} = d_{ij}^2$. Since G is an inner product matrix, it is positive definite and can be spectrally decomposed. The new objective function becomes

$$
min_x ||U\Sigma U^T - XX^T||_2^2 \tag{3}
$$

This is minimized when $U\Sigma U^T = XX^T$. If the dimension of lower space is d, then rank(XX^T) = rank(X)= d. Therefore Σ will have only d non zero eigenvalues. X can thus be written in the form $X = U_d\Sigma_d^{\frac{1}{2}}$ which gives the closed form solution.

2.2 Locally Linear Embedding(LLE)

LLE is designed to preserve the local properties of data. It expects the neighbours of a data point to lie on a locally linear patch and learns weights to recover that point using its k nearest neighbours (k is a free parameter). Secondly, the weights are learnt such that they are rotation, scale and translation invariant. Now, the linear mapping used to map the higher dimensional points to a lower dimension essentially perform such transformations to which the weights are already invariant. This allows us to use the same weights again in the lower dimension to learn the embeddings. The optimization problem thus formulated is convex with affine constraints used to impose the properties that we discussed above.

$$
\begin{gathered}
min_W \sum_{i}^{n} ||y_i - \sum_{j=1}^{n} w_{ij} y_j||_2^2 \\
1) \sum_{j=1}^{n} w_{ij} = 1 \ \forall \ i \\
2) w_{ij} = 0 \text{ if } x_i \text{ is not a neighbour of} x_j
\end{gathered}
\tag{4}
$$

Using the above contraints the objective function reduces to :

$$
= \sum_{i}^{n} ||\sum_{j} w_{ij}(y_i - y_j)||_2^2 = \sum_{i}^{n} ||\sum_{j} w_{ij} z_j||_2^2 \tag{5}
$$

where $z_j = y_i - y_j$ and j varies from 1 to k. We define a matrix $z^{(i)}$ composed of these vectors, of size $k \times p$, and w_i as the vector from the weight matrix that we are calculating. The expression above then reduces to $\sum_i^n w_i^T z^{(i)}(z^{(i)})^T w_i$. Let $z^{(i)}(z^{(i)})^T$ be written as G_i. Here since the expressions in the summation can be treated independently, we move on to finding the minimum value of that expression, which can then be generalized. Note here, the nearest neighbour constraint has already been taken care of while defining $z^{(i)}$ and hence is not considered while defining the new optimization problem.

$$
\begin{gathered}
\min_{w_i} w_i^T G_i w_i \\
\text{subject to } \mathbf{1}^T w_i = 1
\end{gathered}
\tag{6}
$$

The minimum can be obtained by writing the Lagrangian and using KKT conditions:

$$
\begin{gathered}
\mathcal{L}(w_i, \lambda) = w_i^T G_i w_i - \lambda(\mathbf{1}^T w_i - 1) \\
\frac{\partial \mathcal{L}}{\partial w_i} = 2G_i w_i - \lambda \mathbf{1} = 0, \ \frac{\partial \mathcal{L}}{\partial \lambda} = 1^T w_i - \mathbf{1} = 0 \\
\text{or } w_i = \frac{\lambda}{2} G_i^{-1}\mathbf{1}, \ \mathbf{1}^T w_i = 1 \implies w_i = \frac{G_i^{-1}\mathbf{1}}{\mathbf{1}^T G_i \mathbf{1}}
\end{gathered}
\tag{7}
$$

A regularization term may be added if the matrix G_i is coming out to be singular. Once the weight matrix has been computed, data points x_i's in the lower dimension are computed by learning those embeddings which can be reconstructed using the weight matrix calculated above.

$$\begin{gathered} min_x \sum_i^n ||x_i - \sum_{j=1}^n w_{ij} x_j||_2^2 \\ 1) \sum_i x_i = 0 \ \forall \ i \\ 2) \frac{1}{N} \sum_i x_i x_i^T = I \end{gathered} \tag{8}$$

x_i can be translated without affecting the objective function. The first constraint makes the embedding invariant to such translations while the second constraint is used to obtain a unique solution .Let X be the matrix that contains x_i as its columns, the objective function above can then be written as, $||(I-W)X||_2^2 = X^T M X$ where $M = (I-W)^T(I-W)$. Similar to the above procedure, using lagrangian and setting derivatives equal to zero we obtain,

$$MY = \Gamma X \tag{9}$$

Here Γ is diagonal matrix. It is visible from the expressions that X is an eigenvector of M. Now eigenvectors with the minimum eigenvalues would be chosen to minimize the objective function. The first eigenvalue in this list is discarded since it is zero and enforces the first constraint.

2.3 Maximum Variance Unfolding(MVU)

Curves and surfaces such as beads on a necklace can be arranged in a line if we pull the necklace taut, which leads to dimensionality reduction from R^3 to R^1. Similarly, MVU tries to unfold the manifold by pulling the data points apart. It does so by maximizing the sum total of pairwise distances between all data points while keeping the distances between neighbours in both the dimensions same. It has only one free parameter, namely the number of nearest neighbours to consider for preserving the distance in the lower dimension. Specifically the optimization problem is,

$$\begin{gathered} \max \sum_{i,j} ||x_i - x_j||_2^2 \\ 1) ||x_i - x_j||_2^2 = ||y_i - \vec{y_j}||_2^2 \\ \forall \ (i,j) \ \text{ with } \ \eta_{ij} = 1 \\ 2) \sum_i y_i = 0 \end{gathered} \tag{10}$$

Here η_{ij} is 1 when i is a neighbour of j. Thus, the first constraint ensures that pairwise distances for neighbours are preserved. The second constrain centers the lower dimensional embedding around the origin and helps us get a unique solution (up to rotation). However, the above problem is not convex and the following transformations are done to convert it into the same,

$$\begin{gathered} \sum_{i,j} ||y_i - y_j||_2^2 = \sum_{i,j} (y_i^2 + y_j^2 - y_i.y_j - y_j.y_i) = \sum_{i,j} y_i^2 + \sum_{i,j} y_i^2 - 0 - 0 \\ (\text{since} \sum_i y_i = 0 \implies (\sum_i y_i)(\sum_j y_j) = 0 \implies \sum_{i,j} y_i.y_j = 0) \end{gathered} \tag{11}$$

Also,$||x_i - x_j||_2^2 = x_i^2 + x_j^2 - x_i.x_j - x_j.x_i = K_{ii} + K_{jj} - 2K_{ij}$ where K is the gram matrix for X

The new convex problem, formulated as an SDP, is :

$$\begin{gathered} \max \text{trace}(K) \\ 1) K_{ii} + K_{jj} - 2K_{ij} = ||\vec{y_i} - \vec{y_j}||_2^2 \ \forall \ (i,j) \ \text{ with } \ \eta_{ij} = 1 \\ 2) \sum_{i,j} K_{ij} = 0 \\ 3) K \succeq 0 \end{gathered} \tag{12}$$

The first two constraints follow from (10). The third constraint ensures that K is positive semidefinite, which is a property of all inner product matrices. Once we have K, it can be spectrally decomposed into $K = U\Sigma U^T = XX^T$, and the lower dimensional embeddings x are obtained from the top d eigen values and eigen vectors of K.

2.4 Fast MVU

Fast MVU is basically a scalable implementation of the MVU algorithm that we discussed above. It approximates the K matrix as QLQ^T and the optimization is now performed only over the smaller L matrix, leading to higher computational efficiency. The convex optimization problem is:

$$\begin{gathered}\max \operatorname{trace}(QLQ^T) \\ 1)(QLQ^T)_{ii} + (QLQ^T)_{jj} - 2(QLQ^T)_{ij} <= ||\vec{y_i} - \vec{y_j}||_2^2 \ \forall \ (i,j) \ \text{ with } \ \eta_{ij} = 1 \\ 2) \sum_{i,j} QLQ^T_{ij} = 0 \\ 3) L \succeq 0,\end{gathered} \tag{13}$$

Here η_{ij} is 1 when i is a neighbour of j. The constraints are very similar to MVU with a small relaxation in the first constraint from that of equality to an inequality. This is performed to preserve feasibility of the problem which may be affected due to the approximation. Here Q is precomputed using data and the lower dimensional embeddings are reconstructed using the Q and L

3 Simulations and Results

We experimented with the above methods on three types of datasets : Swiss Roll(Artificial), Teapot Dataset(natural) and Yale Faces Dataset(natural). Swiss roll is a 3 dimensional dataset used often for testing non-linear dimension reduction techniques. The teapot dataset is a collection of 400(=n) 23028 dimensional images of a teapot with all possible rotations in horizontal direction. The Yale Faces dataset is a collection of photos of 15 people in different situations and having different expressions, with a total of 11 images per person. Out of these 15 people, 4 are Asians. Each image is a data point in the 77760 dimensional space. Example images of all the datasets have been shown in Figure 1. Note the swiss roll in Figure 1 has been made using 5000 points, while only a subset of it has been considered while testing.

Figure 1: Images from Datasets used

For the teapot dataset, we selected a subset of 8 images for display, each rotated by a constant angle to obtain one full rotation. Since the teapot images were rotated by 360 degrees, their lower dimensional embeddings captured that property and were getting embedded within a ring in the correct order. Similarly, for the Swiss Roll dataset, we selected a subset of 1000 points for Isomap and LLE, while 500 for MVU(since it was taking a lot of time to converge). The embeddings obtained captured the non-linearity in the data with correct colours being embedded closer to each other. Results for Isomap and LLE look more clearer than those obtained using MVU.

For the Yale Faces dataset, we obtain different lower dimensional embeddings ($d = 3$) for each image using Isomap, LLE and MVU. The idea is to classify whether the lower-dimensional embedding results from a photo of an Asian person or not and compare the misclassification errors of these different dimensionality reduction approaches. For this purpose, an SVM classifier is trained on this lower dimensional embedding, and the misclassification errors are tabulated in Table 3. Out of 165 images, 120 are used for training the classifier, and 45 are used for testing. It is indeed remarkable that such 3 dimensional embeddings are able to capture the 77760 dimensional information faithfully, resulting in such low errors.

4 Comparison

We start off with comparing the time complexities of each of the algorithms. Isomap computes geodesic distances using the shortest path algorithm which results in an $\mathcal{O}(n^3)$ complexity, where n is the number of data points. LLE has to solve n optimization problems, each of which involves inverting G_i which is a $k \times k$ matrix (where k is the number of neighbours) leading to an overall complexity of $\mathcal{O}(nk^3)$. While for MVU, we have to solve an SDP problem, which has $\mathcal{O}(nk)$ number of constraints. Thus it has an overall complexity of $\mathcal{O}((nk)^3)$. It is evident from these complexities, that LLE is the fastest, followed by Isomap and MVU. The running times for each of these algorithms on the SwissRoll dataset with 500 data points have been tabulated in Table 1. We have also compared running times

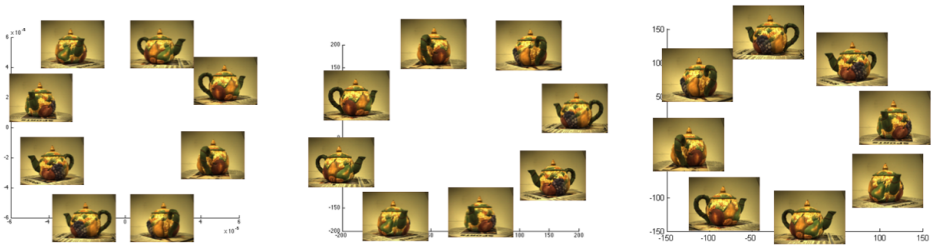

Figure 2: Embedding of 8 teapot images obtained with LLE, Isomap and MVU respectively with k=15

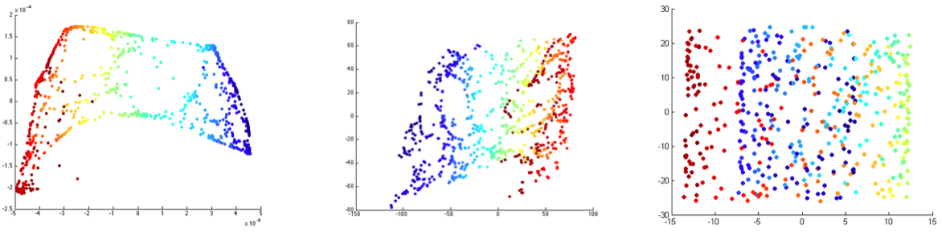

Figure 3: Embedding obtained of Swissroll with LLE, Isomap and MVU respectively with k=10

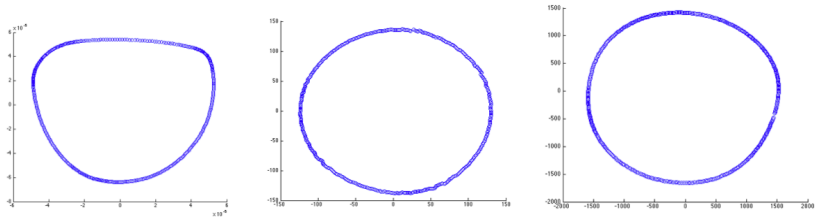

Figure 4: Embedding obtained over the entire teapot dataset with LLE, Isomap and MVU respectively

of MVU and FastMVU on the teapot dataset(Table 2). As expected FastMVU converged much faster than MVU, although the quality of results obtained weren't up to the mark.

For judging the quality of dimensionality reduction tehcniques, we evaluated the embeddings visually. For the teapot dataset, because of the cyclic nature of the images in the dataset, we would expect the perfect lower dimensional embedding to be a full circle. As is visible from Figure 4, MVU produced an embedding closest to it, followed by Isomap and then LLE, which is the least symmetric out of the three. By maximizing the sum total of their pairwise distances during the unfolding of data points in high dimensional space, MVU was able to capture the cyclic nature of the images. On the SwissRoll dataset, quality is measured by observing the visual separatedness of different colours, and results are shown in Figure 3. Because of the locally linear structure of the manifold in three dimensions, LLE was able to obtain the best unfolding, followed by Isomap and then MVU. However, since we were only able to test MVU on a subset of 500 images from the total dataset, it had lesser information to extract from its nearest neighbours, as opposed to Isomap and LLE where a subset of 1000 points were used, and hence the embeddings obtained were not as clearly separated as with Isomap or LLE.

Method	**Complexity**	**Runtime**
LLE	$\mathcal{O}(nk^3)$	0.24s.
Isomap	$\mathcal{O}(n^3)$	18.13s
MVU	$\mathcal{O}((nk)^3)$	358.41s

Table 1: Comparing complexities between methods

Method	**Runtime**
MVU	358.41s
FastMVU	77.49s

Table 2: Comparing runtime between MVU and Fastmvu

Method	Error Rate
LLE	4.45%
Isomap	8.89%
MVU	8.89%

Table 3: Error rates using different methods

The quality of dimensionality reduction is also evaluated by comparing the misclassification error rates for each of Isomap, MVU and LLE on the Yale Faces dataset. LLE performs the best and achieves the lowest error of 4.45%. Isomap and MVU also have low error rates but are clearly behind LLE.

5 Conclusion

In this project, we surveyed four techniques for visualizing high dimensional data. All these techniques obtain a lower dimensional embedding by formulating their approach as a convex optimization problem. For two of these techniques, namely Isomap and LLE, closed form solutions are obtained, while we are required to solve Semi-Definite Programs for both MVU and Fastmvu. There isn't any task-independent metric that could be used to compare dimension reduction techniques. For comparison, therefore, we tested these algorithms on three datasets, comparing their visualizations, computational complexities, and error rates in classifications. MVU was the slowest of all, but provided the best embedding for the teapot dataset. On the other hand, LLE being the fastest of all, wasn't able to capture the cyclic nature of images as well as the other two. On the other hand for the Swissroll dataset, Isomap and LLE obtained better embeddings as compared to MVU. And finally, in the classification task the face embeddings obtained using LLE gave the least error rate out of all three. Based on our testing LLE, with a faster runtime and good overall results, emerged as the better framework for dimension reduction tasks.

References

[1] Weinberger, Kilian Q., and Lawrence K. Saul. "An introduction to nonlinear dimensionality reduction by maximum variance unfolding." *AAAI*. Vol. 6. 2006.

[2] Weinberger, Kilian Q., Benjamin Packer, and Lawrence K. Saul. "Nonlinear Dimensionality Reduction by Semidefinite Programming and Kernel Matrix Factorization." *AISTATS*. 2005.

[3] Tenenbaum, Joshua B., Vin De Silva, and John C. Langford. "A global geometric framework for nonlinear dimensionality reduction." *Science 290.5500* (2000): 2319-2323.

[4] Roweis, Sam T., and Lawrence K. Saul. "Nonlinear dimensionality reduction by locally linear embedding." *Science 290.5500* (2000): 2323-2326.

[5] LJP, Postma EO, and Herik HJ Van Den. "Dimensionality Reduction: A Comparative Review." *Tech. Rrep* (2007).

[6] www.stat.cmu.edu/ cshalizi/350/lectures/14/lecture-14.pdf

[7] https://pdfs.semanticscholar.org/e3fa/abdc800d2400f072eb5b48e9ad6dc94d7625.pdf

[8] http://cseweb.ucsd.edu/ elkan/254spring05/Ettinger419.pdf

[9] http://faculty.ucmerced.edu/mhyang/course/eecs275/lectures/lecture23.pdf

[10] https://en.wikipedia.org/wiki/Semidefinite_embedding

6 CODES

6.1 ISOMAP

```
function [Y] = Isomap(I,k)
%tic
n=size(I,2);
Idx= knnsearch(I', I', 'k', k);

G= repmat(realmax,n,n);

for i=1:size(Idx,1)
    for j=1:size(Idx,2)
        G(i,Idx(i,j)) = norm(I(:,i) - I(:,Idx(i,j)))^2;
    end
end


for k=1:n
    for i=1:n
        for j=1:n
            G(i,j) = min(G(i,j), G(i,k) + G(k,j));
        end
    end
end

% Formulate tau(G) = -HSH/2

S= G.^2;
H = eye(n,n) + repmat(-1/n, n, n);
tau= -H*S*H/2 ;

[V,D] = eig(tau);
e=sort(diag(D), 'descend');

i=2; % Take top 2 for visualization
e= e(1:i);
evectors= []; % dominant eigen vectors
evalues = [];
for i=1:n
    if (ismember(D(i,i),e))
        evectors = [evectors V(:,i)];
        evalues  = [evalues D(i,i)];
    end
end

Y= evectors*sqrt(diag(evalues))';
%toc
end
```

6.2 LLE

```
function [Y] = lle(I,k)
tic
tol=1;
Idx= knnsearch(I', I', 'k', k);
n= size(I,2);
w = zeros(k,1);
W = zeros(n,n);
for i=1:n
G = I(:,i)*ones(1,k) - I(:, Idx(i,:));
```

```
Gi = G'*G;
invg = inv(Gi + eye(k,k)*200);
wi = invg*ones(k,1)/sum(sum(invg));
for j=1:k
W(i,Idx(i,j)) = wi(j);
end
end


M = (eye(n) - W)'*(eye(n) - W);
[V,D] = eig(M);
e=sort(diag(D));

i=3; % Take top 2 for visualization
e= e(1:i);
evectors= []; % dominant eigen vectors
evalues = [];
for i=1:n
    if (ismember(D(i,i),e))
        evectors = [evectors V(:,i)];
        evalues  = [evalues D(i,i)];
    end
end

Y= evectors*sqrt(diag(evalues))';
Y = Y(:,2:end)';
toc
end

% for plotting
% idx = [1,50,100,150,200,250,300,350];
% plot(Y(1,idx),Y(2,idx),'-r')
% hold on
% for ii = 1:length(idx)
% text(Y(1,idx(ii)),Y(2,idx(ii)),num2str(ii),'Color','r')
% end
```

6.3 MVU

```
function [Y] = MVU(I,k)

n= size(I,2);
Idx= knnsearch(I', I', 'k', k);

cvx_begin
variable K(n,n) semidefinite
maximize trace(K)
subject to
  for i=1:n
    for j=1:n
        if (ismember(j, Idx(i,:)))
            K(i,i)-2*K(i,j)+K(j,j) == norm(double(I(:,i)-I(:,j)))^2;
        end
    end
  end
  sum(sum(K)) == 0
cvx_end

[V,D] = eig(K);
e=sort(diag(D), 'descend');
```

```
i=2; % Take top 2 for visualization
e= e(1:i);
evectors= []; % dominant eigen vectors
evalues = [];
for i=1:n
    if (ismember(D(i,i),e))
        evectors = [evectors V(:,i)];
        evalues  = [evalues D(i,i)];
    end
end

Y= evectors*sqrt(diag(evalues));
end
```

6.4 FastMVU

```
function [Y] = fastmvu(I,k,r,m)
tic
n= size(I,2);
Idx= knnsearch(I', I', 'k', k);

W = zeros(n,n);
for i=1:n
G = I(:,i)*ones(1,k) - I(:, Idx(i,:));
Gi = G'*G;
invg = inv(Gi+eye(k,k)*150);
wi = invg*ones(k,1)/sum(sum(invg));
for j=1:k
W(i,Idx(i,j)) = wi(j);
end
end

phi = (eye(n) - W)'*(eye(n) - W);
Q = [eye(m);inv(phi(m+1:end,m+1:end))*phi(m+1:end,1:m)];
Idx2 = Idx(:,1:r);
idx = linspace(1,m,m);
left_index = linspace(m+1,n,n-m);
iterate = 1;
while(iterate==1)
cvx_begin
variable L(m,m) semidefinite
M = Q*L*Q';
maximize trace(M)
subject to
  for i=idx
    for j=idx
        if (ismember(j, Idx2(i,:)))
            M(i,i)-2*M(i,j)+M(j,j) <= norm(double(I(:,i)-I(:,j)))^2;
        end
    end
  end
  sum(sum(M)) == 0
cvx_end

curr_size = size(idx,2);
for i=left_index
    for j=left_index
     if (ismember(j, Idx2(i,:))) && ~(M(i,i)-2*M(i,j)+M(j,j) <= norm(double(I(:,i)-I(:,j)))^2);
     if i==j and ~(ismember(i, idx))
     idx = [idx i];
     left_index(left_index==i) = [];
```

```
     elseif ~(ismember(j, idx))
     idx = [idx j];
     left_index(left_index==j) = [];

     elseif ~(ismember(i, idx))
     idx = [idx i];
     left_index(left_index==i) = [];
     end
     iterate = 1;
     %break
     end
    end
end

if curr_size==size(idx,2)
break
end

[V,D] = eig(L);
e=sort(diag(D), 'descend');

i=2; % Take top 2 for visualization
e= e(1:i);
evectors= []; % dominant eigen vectors
evalues = [];
for i=1:m
    if (ismember(D(i,i),e))
        evectors = [evectors V(:,i)];
        evalues  = [evalues D(i,i)];
    end
end

Lnew= evectors*sqrt(diag(evalues));
Y = Q*Lnew;
Y = Y';
toc
end
```

6.5 Code for evaluation

Evaluation code:

```
ImgPath= 'D:\Convex\project\yalefaces';
I=[];
labels=zeros(165,1);
tempIndex=1;
for i=1:15
    if (i<10)
        subject=strcat(ImgPath,'\subject','0',int2str(i));
    else
        subject=strcat(ImgPath,'\subject',int2str(i));
    end
    if(i==1)
        img= imread(strcat(subject,'.gif'));
    else
        img= imread(strcat(subject,'.CENTERLIGHT'));
    end
    img=img';
    I=[I,img(:)];
    img= imread(strcat(subject,'.glasses'));
```

```
    img=img';
    I=[I,img(:)];
    img= imread(strcat(subject,'.happy'));
    img=img';
    I=[I,img(:)];
    img= imread(strcat(subject,'.leftlight'));
    img=img';
    I=[I,img(:)];
    img= imread(strcat(subject,'.noglasses'));
    img=img';
    I=[I,img(:)];
    img= imread(strcat(subject,'.normal'));
    img=img';
    I=[I,img(:)];
    img= imread(strcat(subject,'.rightlight'));
    img=img';
    I=[I,img(:)];
    img= imread(strcat(subject,'.sad'));
    img=img';
    I=[I,img(:)];
    img= imread(strcat(subject,'.sleepy'));
    img=img';
    I=[I,img(:)];
    img= imread(strcat(subject,'.surprised'));
    img=img';
    I=[I,img(:)];
    img= imread(strcat(subject,'.wink'));
    img=img';
    I=[I,img(:)];
    if(i==4|i==6|i==8|i==14)
        labels(tempIndex:tempIndex+11)=1;
    end
    tempIndex= tempIndex+11;
end

Y = MVU(double(I),10);
%Y = llee(double(I),10);
%Y = Isomap(double(I));
%class = classify(Y(121:165,:),Y(1:120,:),labels(1:120),'quadratic');
SVMModel = svmtrain(Y(1:120,:),labels(1:120));
class = svmclassify(SVMModel, Y(121:165,:));
error=  norm(class-labels(121:165),1);
%scatter(Y(121:165,1), Y(121:165,2),[],class+1,'filled');
scatter3(Y(121:165,1), Y(121:165,2), Y(121:165,3),[],class+1,'filled');
```