COMSW 4772 - Advanced Machine Learning

Assignment 1

Arthur Argenson - aa3572

February 25, 2015

1 Part 1

In all this part, we use a linear kernel (dot product) for both PCA and MVE computation. We run MVE with k=2..4 nearest-neighbors and linear PCA. For each case we compute the fidelity, i.e.:

$$f = \frac{\lambda_1 + \lambda_2}{\sum_{i=1}^{N} \lambda_i} \tag{1}$$

N is the dimension of the covariance matrix and $\lambda_i, i = 1..N$ its eigenvalues such that $\lambda_1 > \lambda_2 > .. > \lambda_N$.

1.1 PCA visualization

The PCA gives a benchmark for fidelity results. We find that fidelity after PCA is $f = 0.2697$
The 2d-visualization after PCA gives:

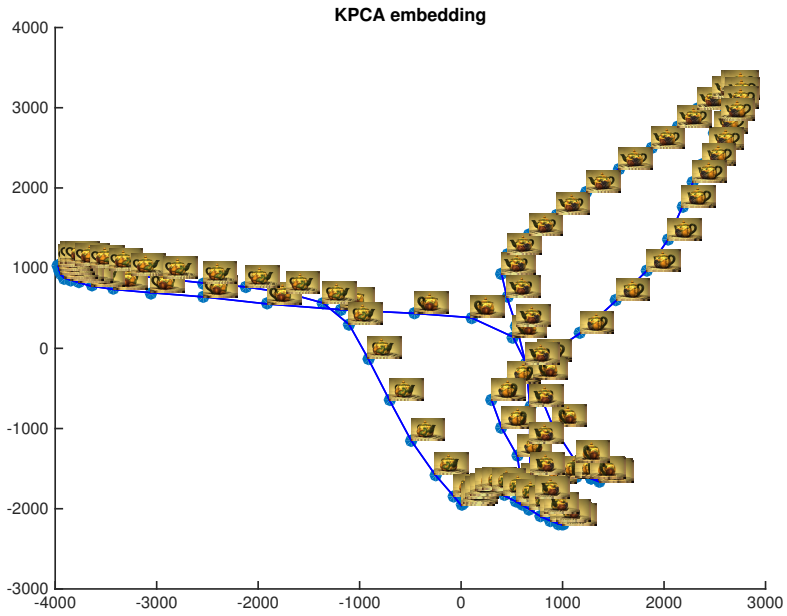
KPCA embedding
4000
3000
2000
1000
0
-1000
-2000
-3000
-4000
-3000
-2000
-1000
0
1000
2000
3000

Figure 1: PCA

1.2 MVE with k=2 nearest-neighbors

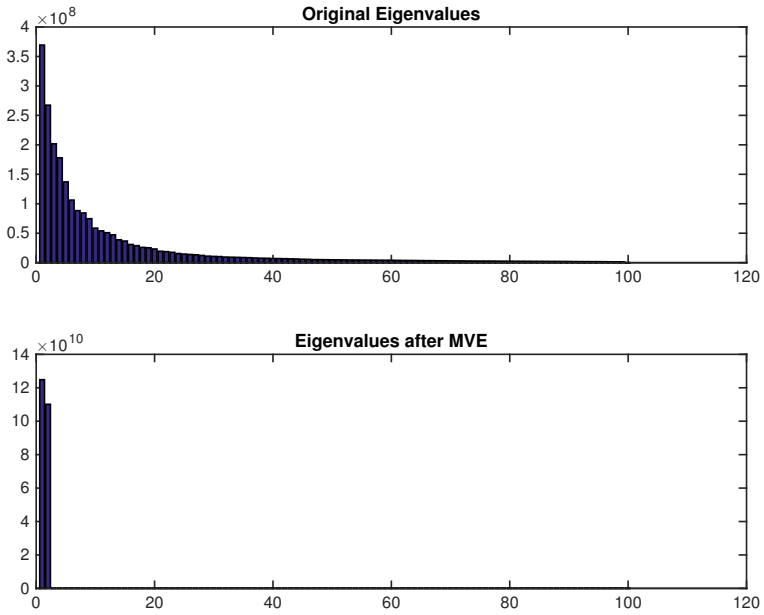

Figure 2: Eigenvalues for linear-kernel MVE and PCA, k=2

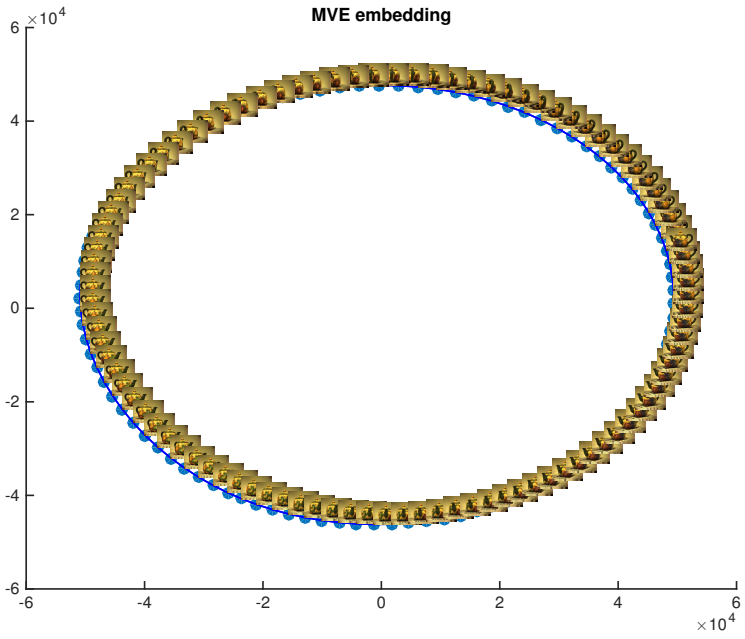

Figure 3: linear-kernel MVE, k=2

The MVE-fidelity is $f = 1$.

1.3 MVE with k=3 nearest-neighbors

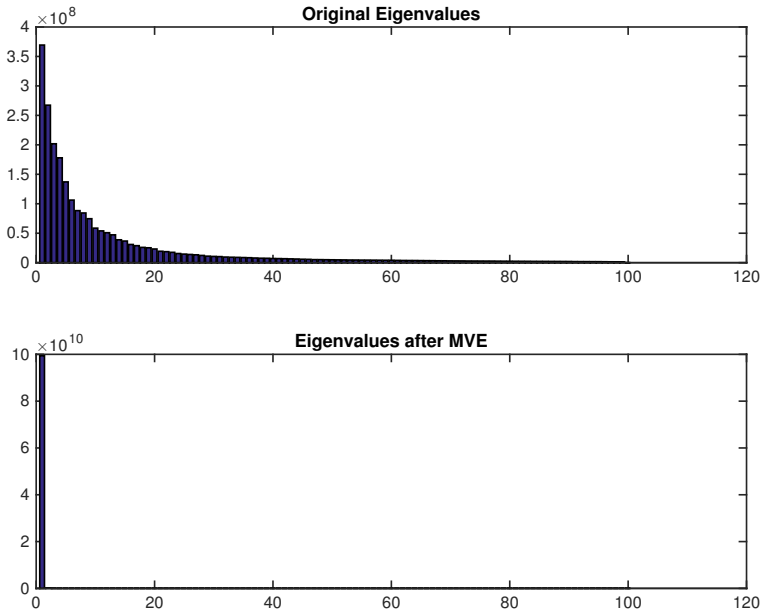

Figure 4: Eigenvalues for linear-kernel MVE and PCA, k=3

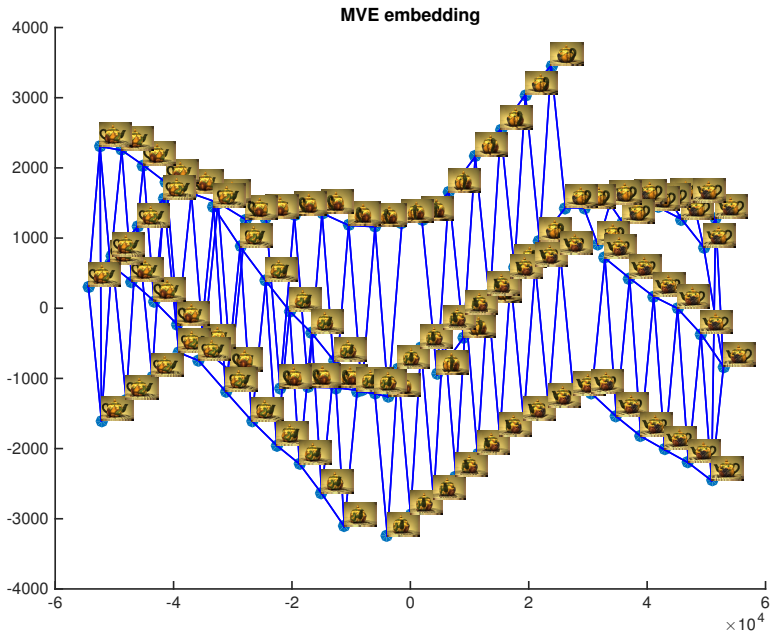

Figure 5: linear-kernel MVE, k=3

The MVE-fidelity is $f = 0.9998$.

1.4 MVE with k=4 nearest-neighbors

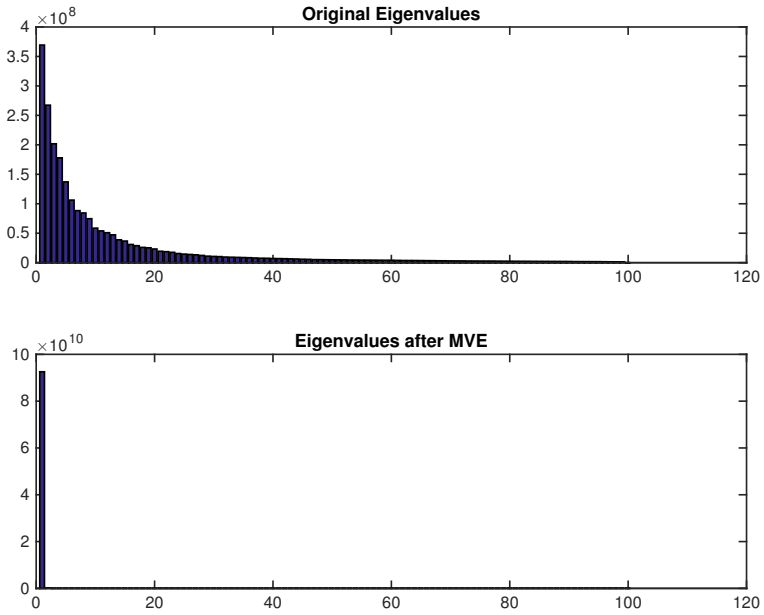

Figure 6: Eigenvalues for linear-kernel MVE and PCA, k=4

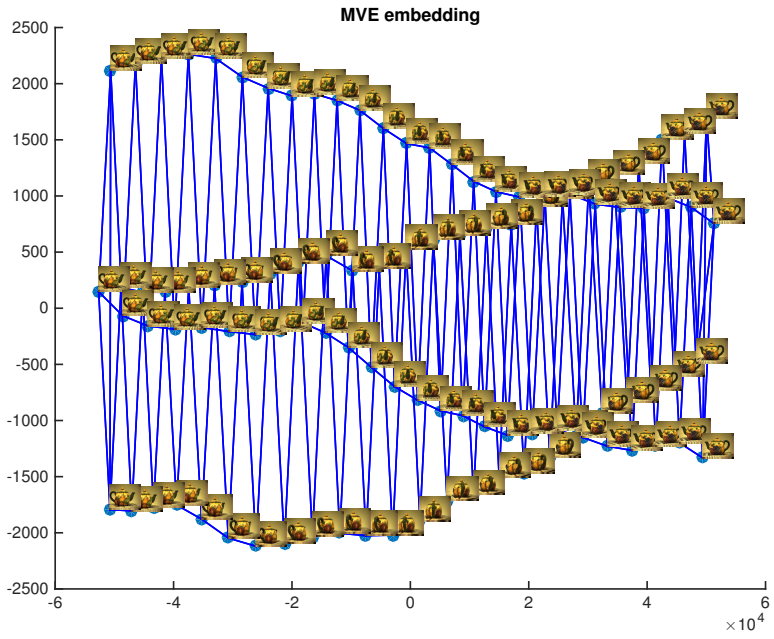

Figure 7: linear-kernel MVE, k=4

The MVE-fidelity is $f = 0.9999$.

1.5 Results

k-neighbors	fidelity
PCA	0.2697
k=2	1.0
k=3	0.9998
k=4	0.9999

The k=2 nearest neighbors perform better than the two other. In this case it is not surprising. Indeed, there is a genuine 2-neighbors relationship in the data-set. Each teapot two-closest-neighbors are the previous and next teapots in the time-sequence.
The k=2 nearest-neighbors MVE exactly reflects this relationship by displaying a 2d-circle for the data structure, with two main eigenvalues that are almost the same, $\lambda_1 \approx \lambda_2 \gg \lambda_3..\lambda_N$.
With MVE we only need 2 dimensions to fully represent the data.

1.6 NB: axis range

In all the graphics we voluntarily chose non-equals axis to improve visualization. Yet we have to keep in mind that in most MVE-Visualization, the range of x-axis is much greater than the y-axis one because $\lambda_1 \gg \lambda_2$.

2 Part 2

2.1 Pre-processing of data

The current data is an arbitrary list of point-coordinates describing a cloud of points. It need to be pre-processed in order to run dimension-reduction algorithms.

We apply the following mapping for each digit (each row of the digits9.mat): $\Phi : digit \mapsto img$. Where img is a 27×27-matrix. The mapping algorithm is as follows:

$$w = 27, \text{ (width of the picture)}$$

$$digit \in N^{2 \cdot 70}, \text{ (one of the 155 lists of points from digits9.mat)}$$

$$img = zeros(w, w)$$

$$\forall (i,j) \in [1,w]^2, img(i,j) = \sum_{n=1}^{70} e^{-\frac{\|(i,j)-p_n\|}{2a^2}} \tag{2}$$

$$\text{where } p_n = (digit[2n-1], digit[2n])$$

The matrix img can be equivalently converted to a vector v such that

$$\forall (i,j) \in [1,w]^2, img(i,j) = v((i-1) \cdot w + j \tag{3}$$

Below is the Matlab code corresponding to this mapping:

```
function [p, img]=phi(digit, a)
    w=27;
    m=zeros(w^2,1);
    img=zeros(w,w);

    for i = 1:w;
        for j = 1:w;
            for k = 1:70;
                c=[digit(2*k-1), digit(2*k)];
                m(j+(i-1)*w,1) = m(j+(i-1)*w,1)+exp(-0.5*(norm(c-[i,j])^2)/a^2);
                img(i,j) = img(i,j)+exp(-0.5*(norm(c-[i,j])^2)/a^2);
            end
        end
    end
```

p=m;
end

This mapping is equivalent to applying the kernel $K : (x, y) \mapsto \Phi(x)^T \cdot \Phi(y)$, where Φ is the function described above. It is also equivalent to converting the list of dots to an image with a blur-parameter, a. The higher a, the closer the images from each other.

For all the computations below, we previously apply this mapping to the data. We then use various kernels.

2.2 Linear kernel

We apply MVE and PCA with a linear-kernel, for k=2..4 nearest-neighbors.

2.2.1 PCA

As a benchmark, the fidelity after PCA is $f = 0.3524$.

Visualization after PCA:

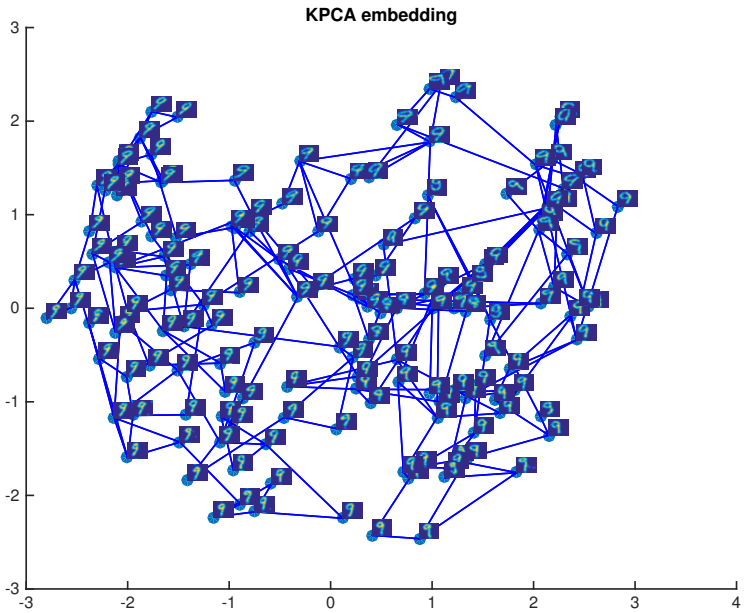

Figure 8: After PCA 2d-visualization

2.2.2 Linear-kernel MVE

Fidelity results:

k-neighbors	MVE-f	PCA-f
2	0.9994	0.3524
3	0.9859	0.3524
4	0.9693	0.3524

2.2.3 Visualization

The best result for visualization is given by k=2:
$f = 0.9994$

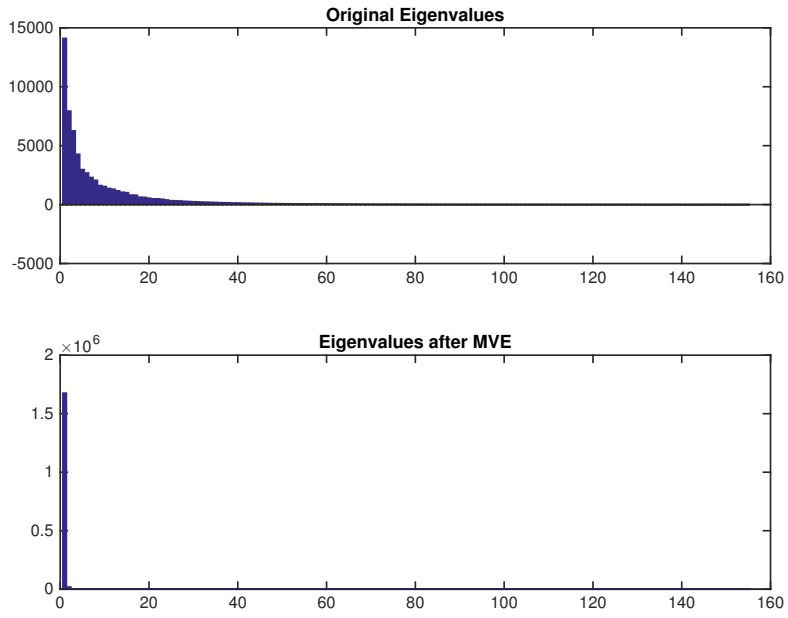

Figure 9: eigenvalues for linear-kernel-MVE, k=2

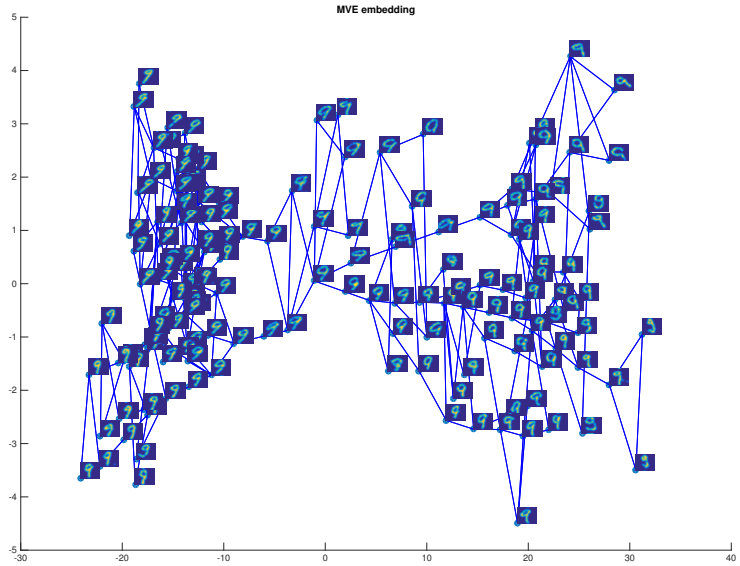

Figure 10: linear-kernel-MVE, k=2

2.3 Polynomial kernel

We compute the affinity matrix A as follows:

$$A = (1 + X^T \cdot X)^d, d \in N^\star \tag{4}$$

where X is the $p \times N$ data-matrix ($p = 27^2$ number of parameters and $N = 155$ numbers of points).

For each k=2..4 nearest-neighbors, we find the best d-parameter for the kernel, $d = 1..7$. For this purpose we compute the MVE fidelity vs d-parameter.

2.3.1 k=2 nearest-neighbors

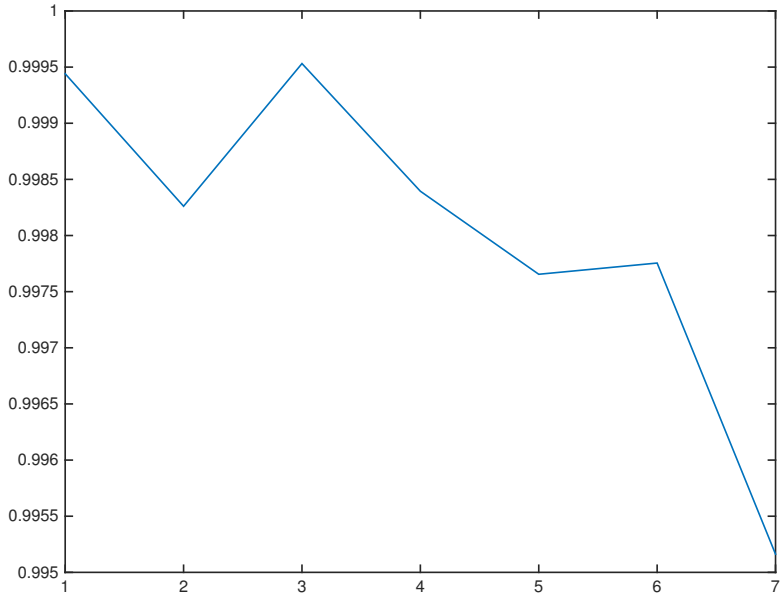

Figure 11: Fidelity vs d-parameter

The best parameter for 2d-visualization is $d = 3$. This gives a fidelity after MVE $f = 0.9995$ versus $f = 0.2760$ for KPCA with the same kernel.

2.3.2 k=3 nearest-neighbors

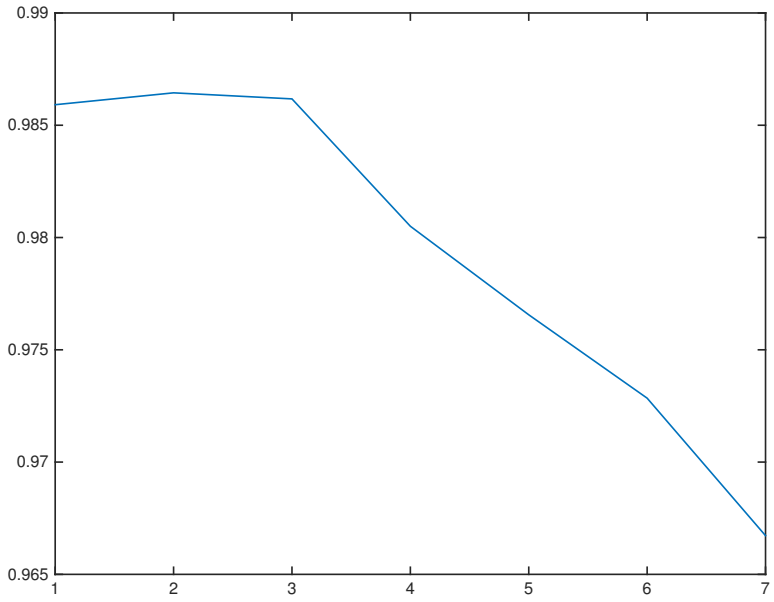

Figure 12: Fidelity vs d-parameter

The best parameter is $d = 2$. This gives a fidelity after MVE $f = 0.9864$ versus $f = 0.3094$ for KPCA with the same kernel.

2.3.3 k=4 nearest-neighbors

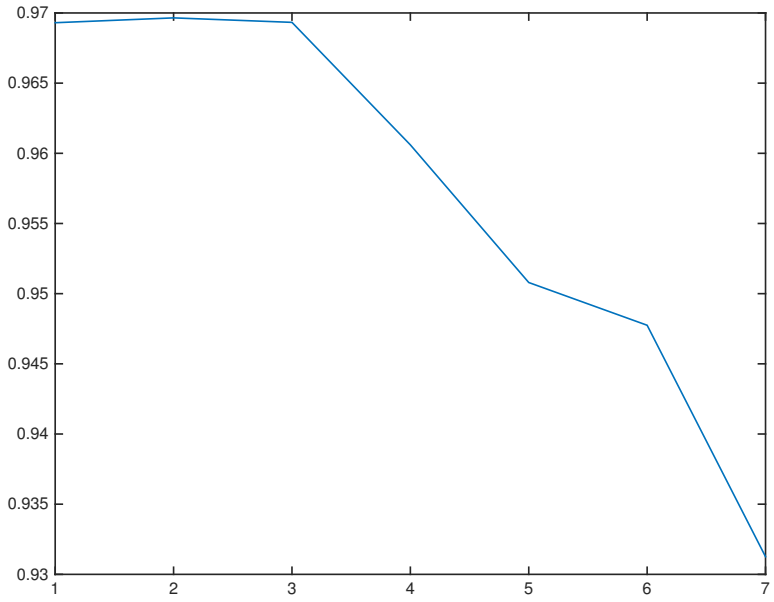

Figure 13: Fidelity vs d-parameter

The best parameter is $d = 2$. This gives a fidelity after MVE $f = 0.9696$ versus $f = 0.3094$ for KPCA with the same kernel.

2.3.4 Results

In the table below are the fidelity results. We give for each k-nearest-neighbors the fidelity after MVE, the fidelity of the KPCA corresponding to the same kernel, and finally the parameter that gives the best result.

k-neighbors	MVE-f	KPCA-f	d
2	0.9995	0.2760	3
3	0.9864	0.3094	2
4	0.9696	0.3094	2

2.3.5 Visualization

The best result is given by k=2 and a degree-3-polynomial kernel:

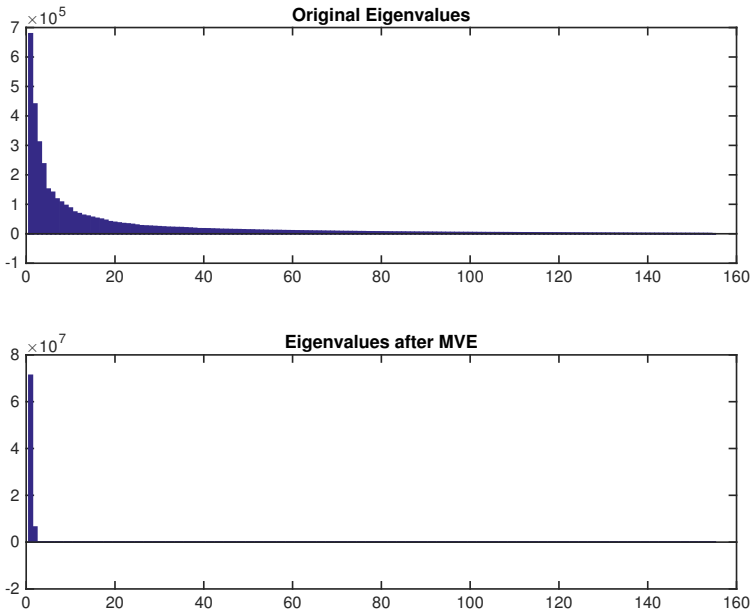

Figure 14: Eigenvalues, degree-3 polynomial-kernel, k=2

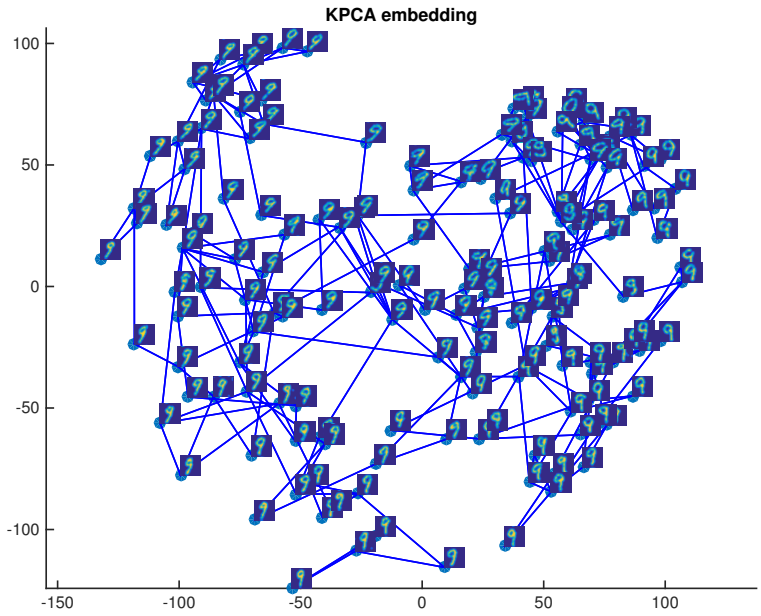

Figure 15: KPCA, degree-3 polynomial-kernel, k=2

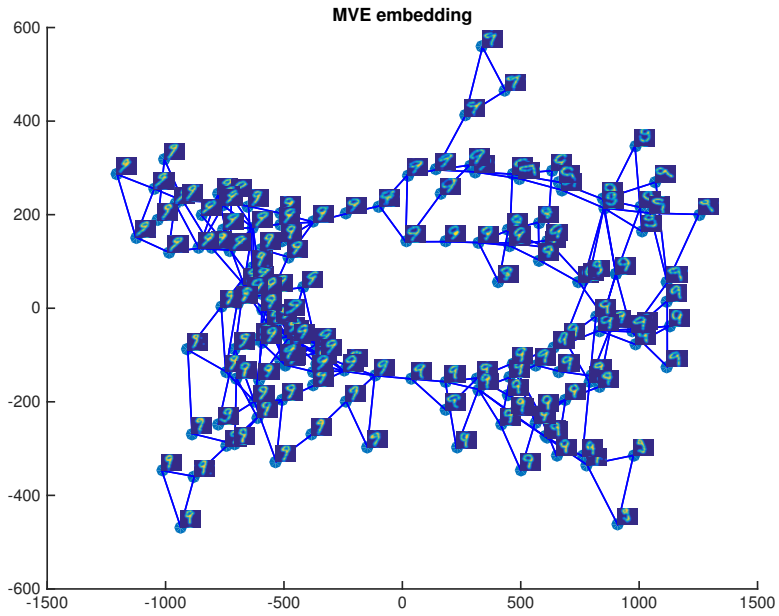

Figure 16: degree-3 polynomial-kernel, k=2

2.4 Gaussian kernel

We compute the affinity matrix A as follows:

$$A_{i,j} = K(X_i, X_j), (i,j) \in [1,155]^2 \tag{5}$$

$$K : (x,y) \mapsto e^{-\frac{x^T \cdot x + y^T \cdot y - 2x^T \cdot y}{2\sigma}} = e^{-\frac{\|x-y\|^2}{2\sigma}} \tag{6}$$

For each k=2..4 nearest-neighbors, we find the best σ-parameter for the kernel, $\sigma = \left[1, 10^{0.5}, 10^1, .., 10^3\right]$. For this purpose we compute the MVE-fidelity vs σ-parameters.

2.4.1 k=2 nearest-neighbors

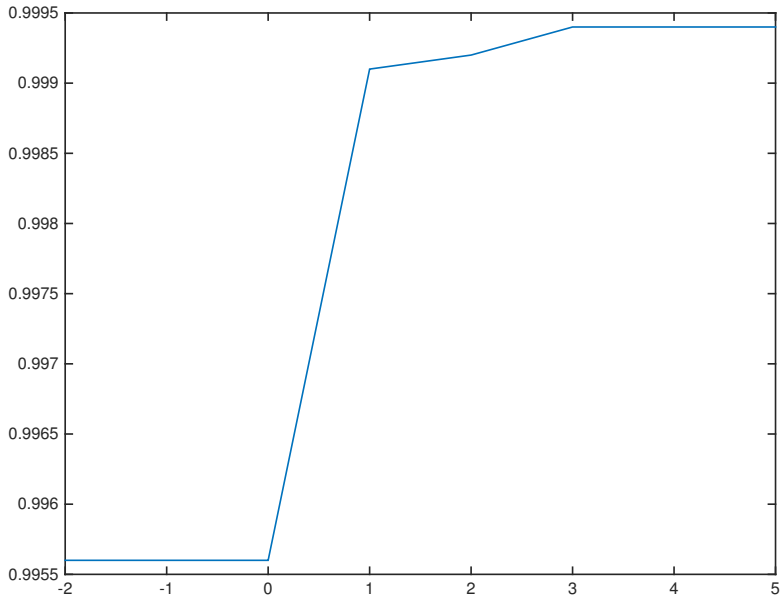

Figure 17: Fidelity vs $log(\sigma)$

2.4.2 k=3 nearest-neighbors

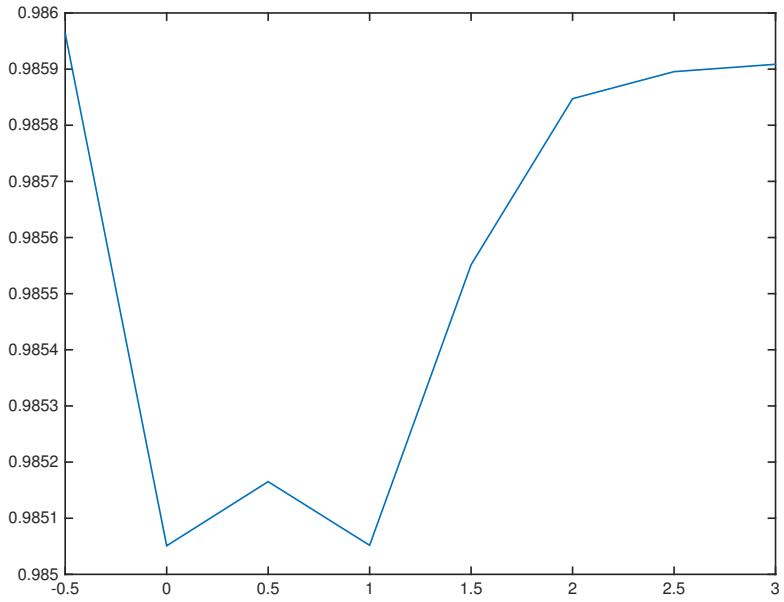

Figure 18: Fidelity vs $log(\sigma)$

2.4.3 k=4 nearest-neighbors

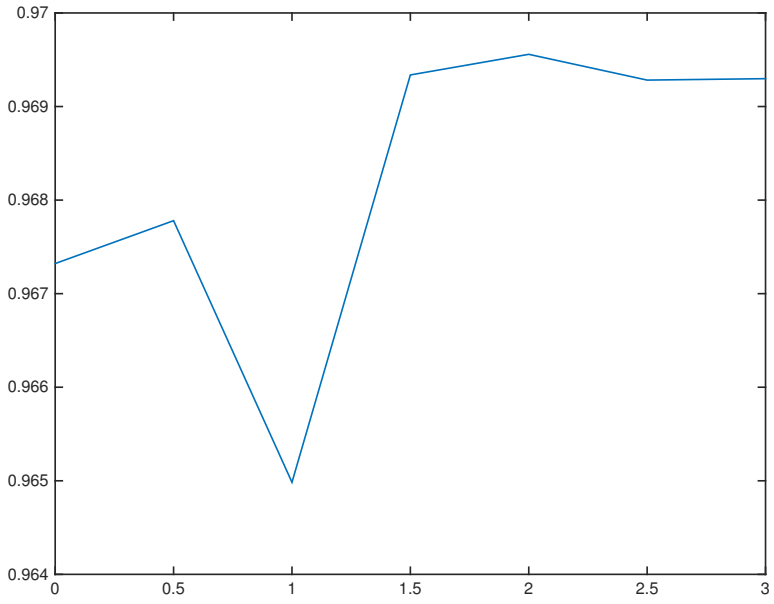

Figure 19: Fidelity vs $log(\sigma)$

2.4.4 Results

Below are the kernel that gives the best performances for $k = 2..4$.

k-neighbors	MVE-f	KPCA-f	σ
2	0.9994	0.3507	10^3
3	0.9859	0.3507	10^3
4	0.9696	0.3360	10^2

2.4.5 Visualization

The best result is found with k=2 nearest-neighbors and a gaussian-kernel with $\sigma = 10^3$:

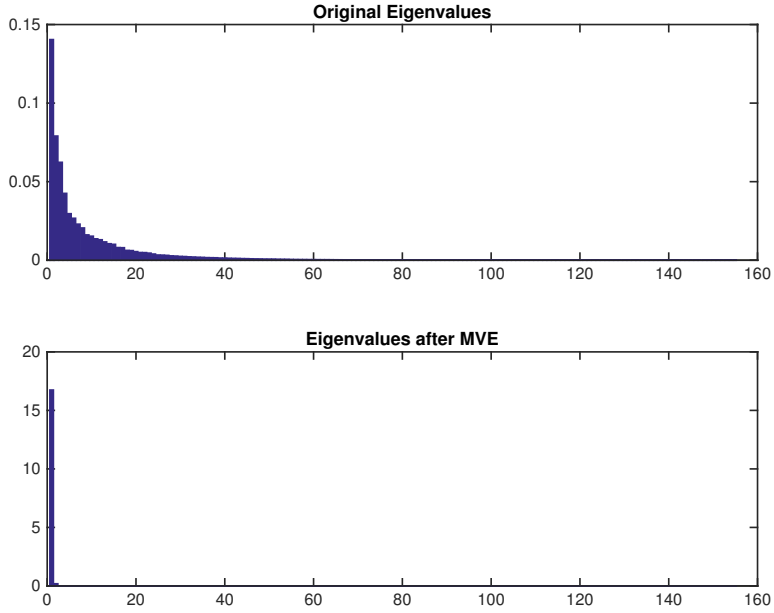

Figure 20: Eigenvalues, gaussian-kernel, $\sigma = 10^3$, k=2

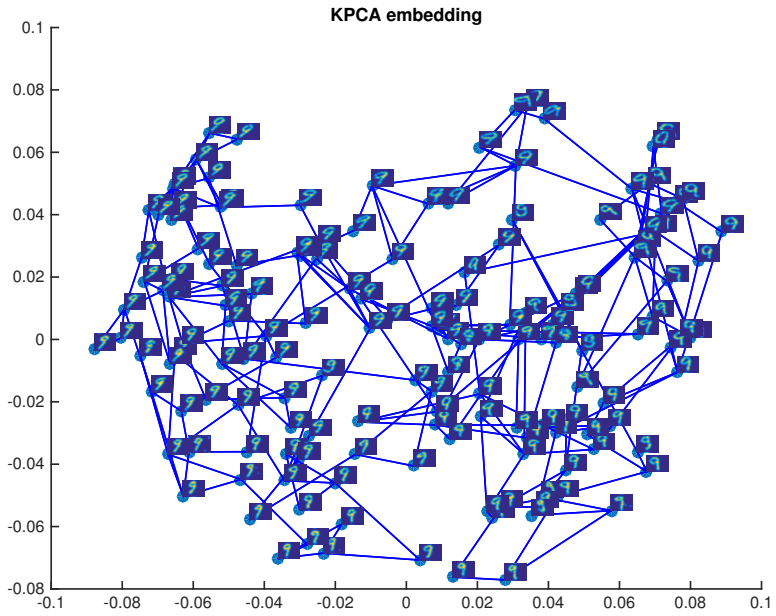

Figure 21: KPCA, gaussian-kernel, $\sigma = 10^3$, k=2

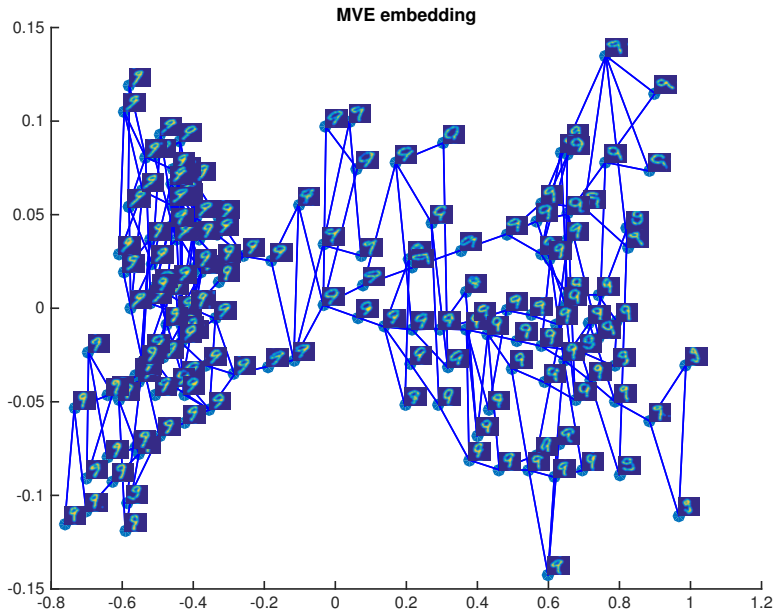

Figure 22: MVE, gaussian-kernel, $\sigma = 10^3$, k=2

2.5 Results

We can see that the main eigenvector represent the orientation of the 9-digit. A low value means that the 9 is bent to the left, and a high value means a 9 bent to the right.
In this case, with MVE, we only need two dimensions to capture the variance, we have

$$\lambda_1 + \lambda_2 \gg \sum_{i=3}^{N} \lambda_i \tag{7}$$

With PCA, we need around 20 dimensions to capture most of the variance.

3 Part 3

The MVE objective function is:

$$max\{F(K)\} \tag{8}$$

$$F(K) = \frac{\lambda_1 + \lambda_2}{\sum_{i=1}^{N} \lambda_i} \tag{9}$$

We can instead solve the following spectral-SDP:

$$max_{\beta}\{max_{K}\big(\lambda_1 + \lambda_2 - \beta \cdot \sum_{i=1}^{N} \lambda_i\big)\} \tag{10}$$

We thus use the following Matlab code to solve the MVE with respect to β:

```
function [Y, K, eigVals, mveScore] = mveB(A, neighbors, tol, targetd, beta)

    ...

    [v, d] = eig(K);

    [dnew,idx] = sort(real(diag(d)), 'ascend'); v = real(v(:,idx));

    B = zeros(N, N);
    for i=1:N-targetd
        B = B + beta*v(:, i) * v(:, i)';
    end
    for i=(N-targetd + 1):N
        B = B + ( beta - 1 )*v(:, i) * v(:, i)';
    end

    ...

end
```

Below are the computation of fidelity vs β for several k-nearest-neighbors with a linear kernel.

3.1 k=2 nearest-neighbors

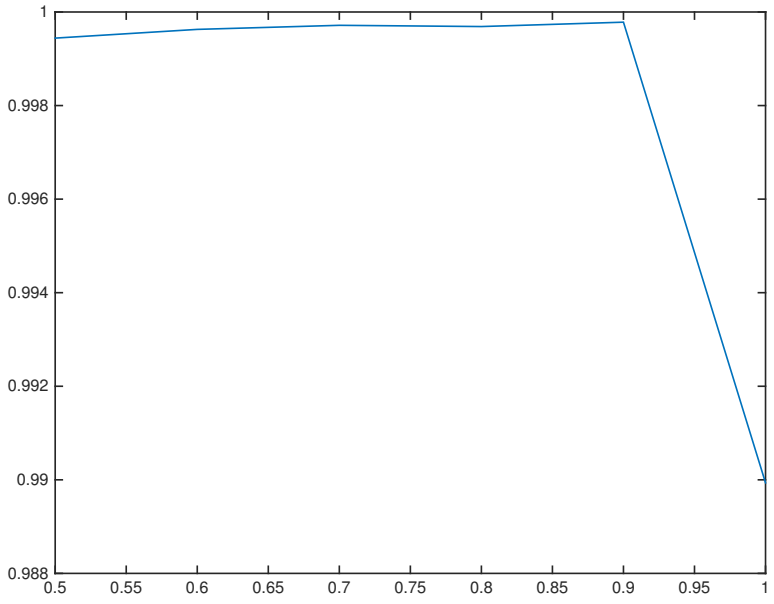

Figure 23: fidelity vs β for linear-kernel MVE, k=2

3.2 k=3 nearest-neighbors

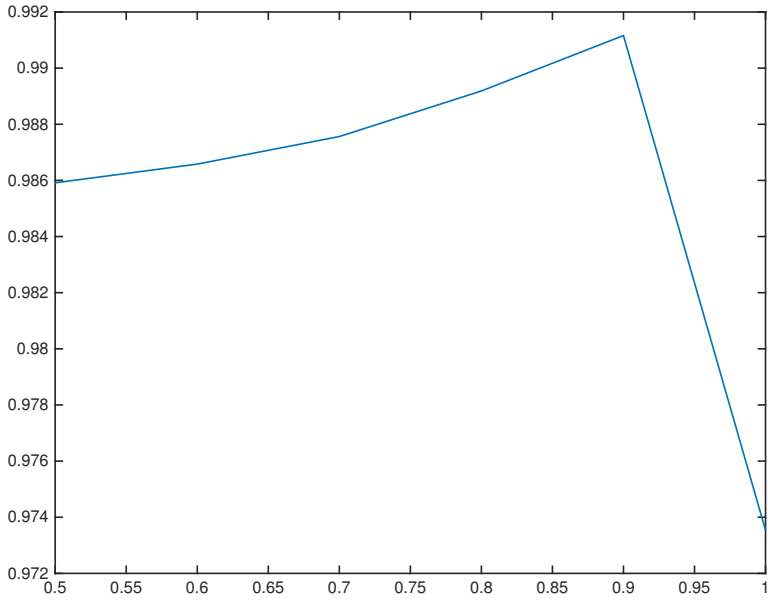

Figure 24: fidelity vs β for linear-kernel MVE, k=3

3.3 k=4 nearest-neighbors

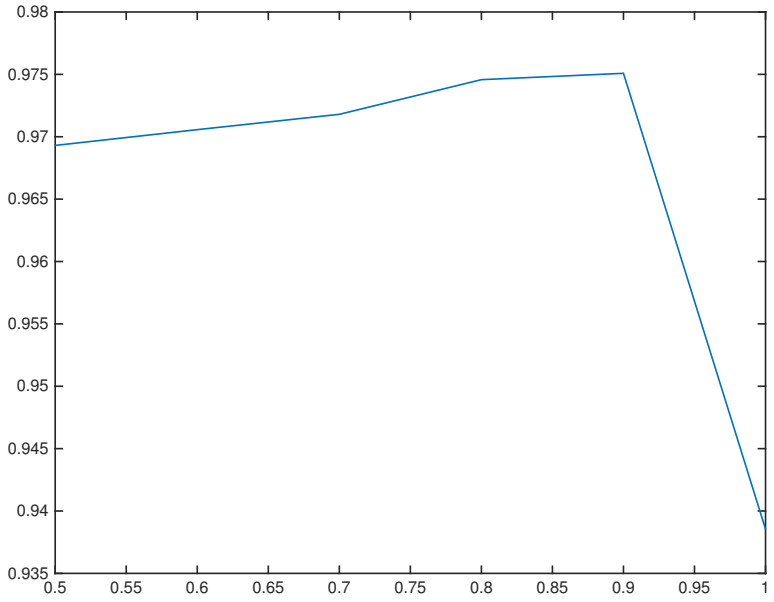

Figure 25: fidelity vs β for linear-kernel MVE, k=4

3.4 Visualization

The best result with linear kernel is given by $k = 2$ and $\beta = 0.9$.

We now apply the previous method to improve the best of the computed models: we use a degree-2-polynomial kernel and k=2 nearest-neighbors. The KPCA with this kernel gives $f = 0.3094$.

With $\beta = 0.95$ we find $f = 0.9999$ for MVE-fidelity.

Visualization of the results:

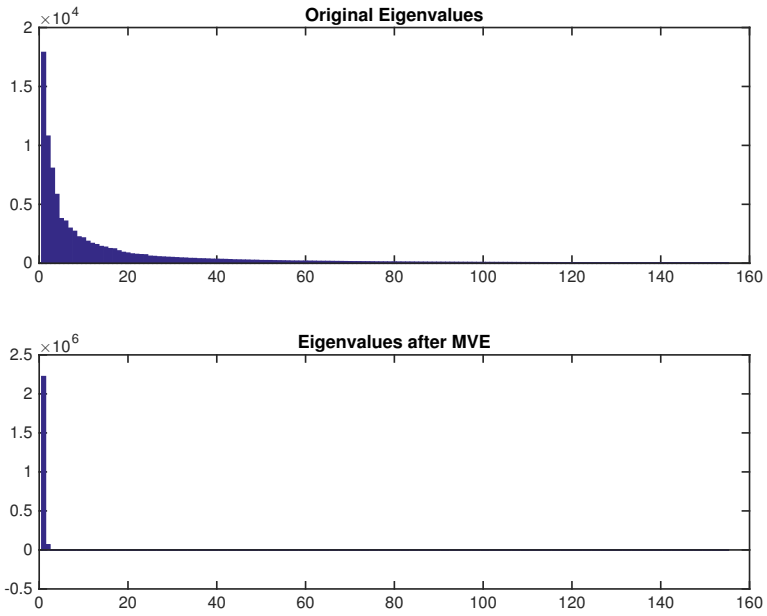

Figure 26: Eigenvalues degree-2-polynomial-kernel MVE, with k=2 and $\beta = 0.95$

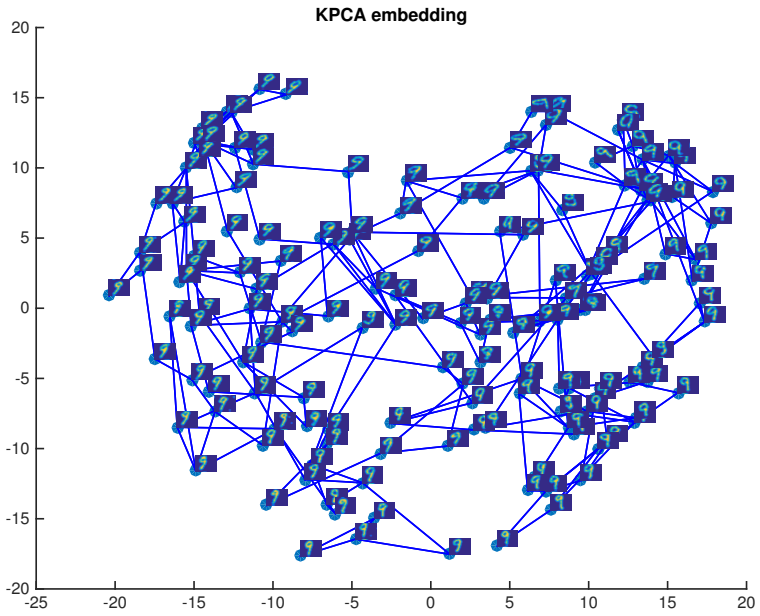

Figure 27: KPCA with degree-2-polynomial-kernel

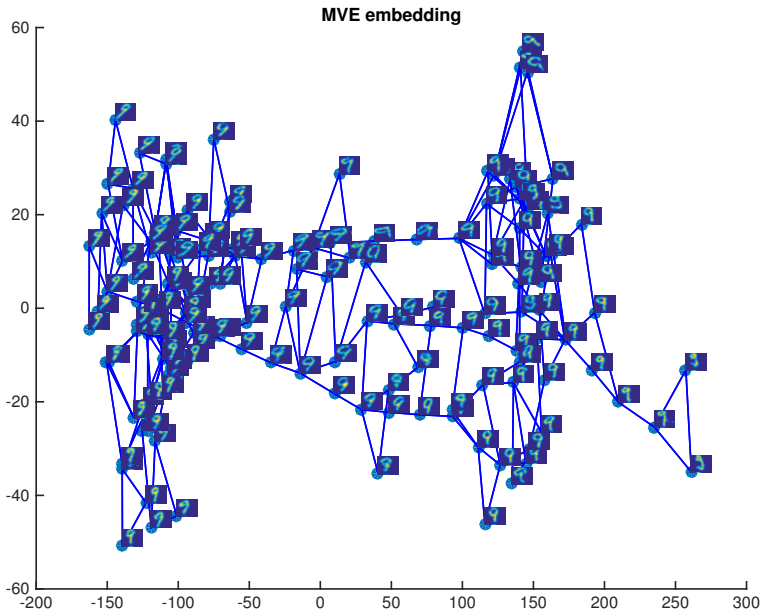

Figure 28: Degree-2-polynomial-kernel MVE, with k=2 and $\beta = 0.95$

4 Part 4

The data used for this part is a set of 300 faces as 150×150-JPG images. We first convert these color-150×150-images to black and white 150^2-vectors. This provides us with the 22500×300-matrix X.

4.1 Model selection

We first use a linear kernel to select the best k-nearest-neighbors model with k=2..4.

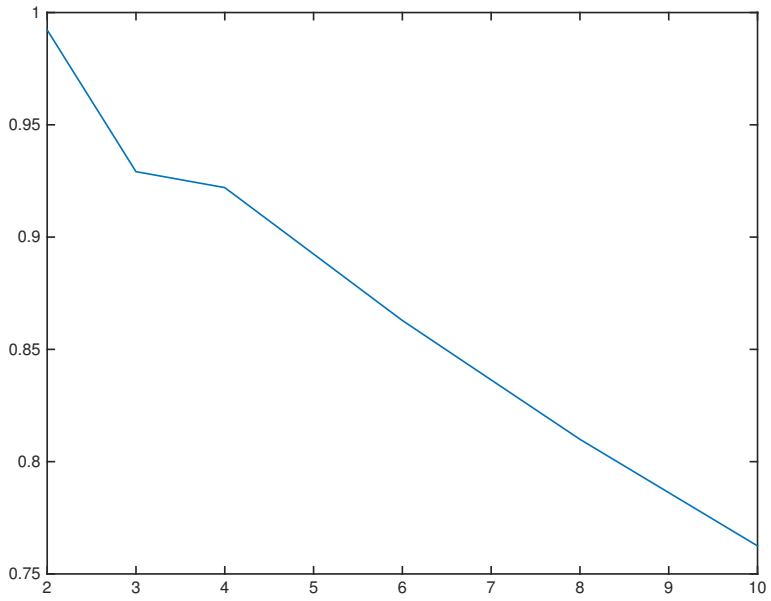

Figure 29: Fidelity vs k-nearest-neighbors, linear-kernel MVE

The best result is for k=2. We will chose k=2 nearest-neighbors in what follows.

4.2 Kernel selection

We then try MVE with different polynomial and gaussian kernels and k=2 in order to find the best performing one.

4.2.1 Polynomial kernels

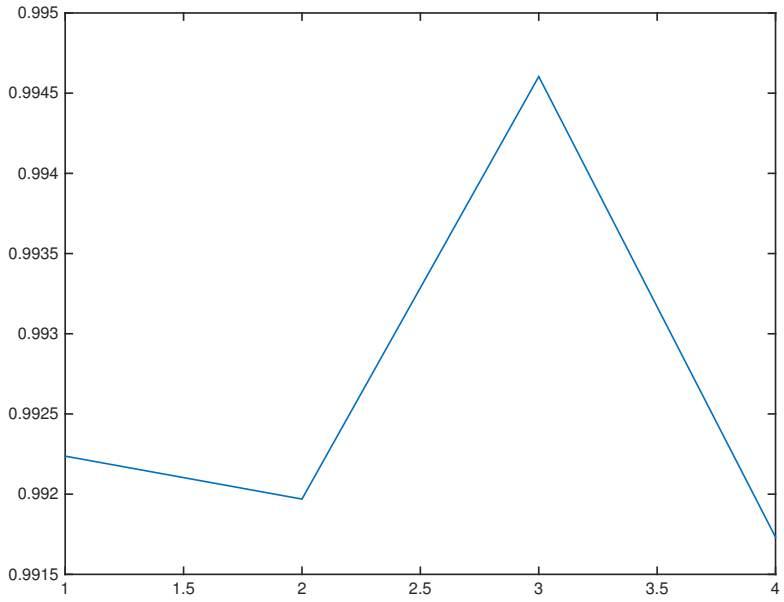

Figure 30: Fidelity vs d-parameter, degree-d-polynomial-kernel MVE, k=2

4.2.2 Gaussian kernels

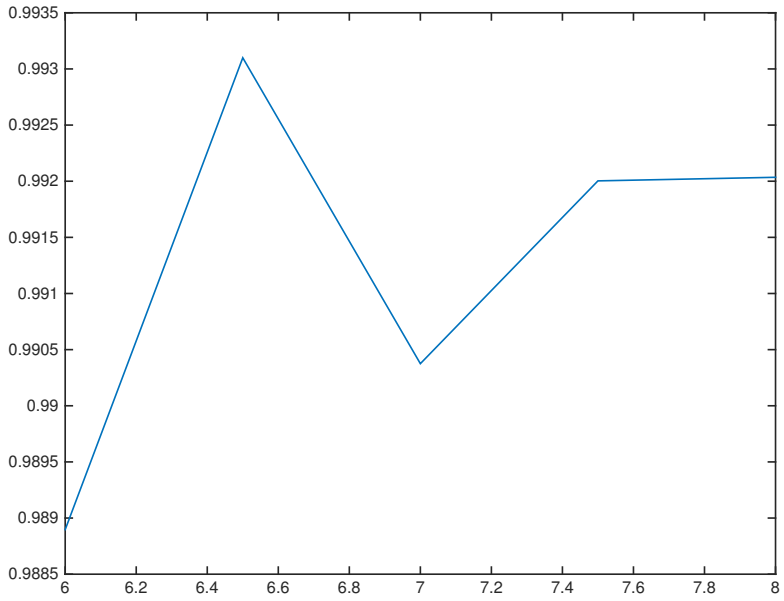

Figure 31: Fidelity vs $log(\sigma)$, gaussian-kernel MVE, k=2

The kernel that gives the best result is the degree-3-polynomial kernel. We choose this kernel for the following part.

4.3 Fidelity improvement

Using the previous degree-3-polynomial-kernel, with k=2, we compute MVE-fidelity for different β.

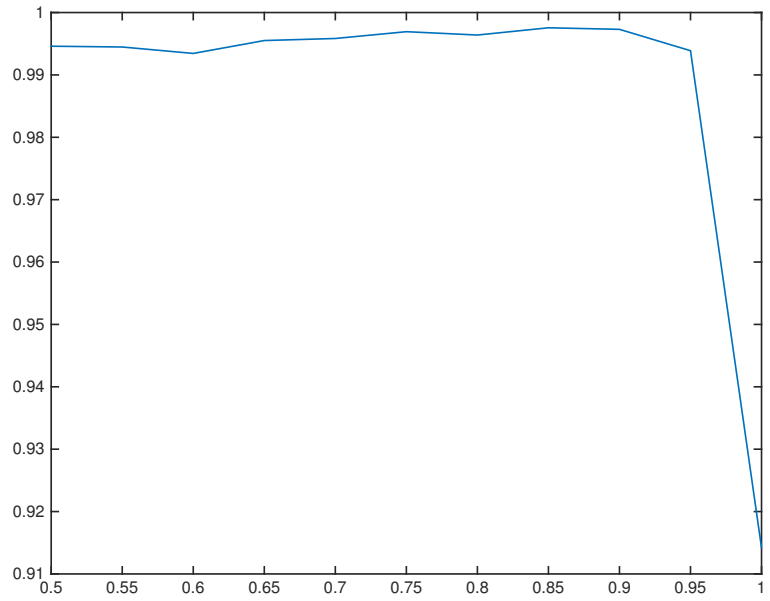

Figure 32: Fidelity vs β-parameter, degree-3-polynomial-kernel MVE, k=2

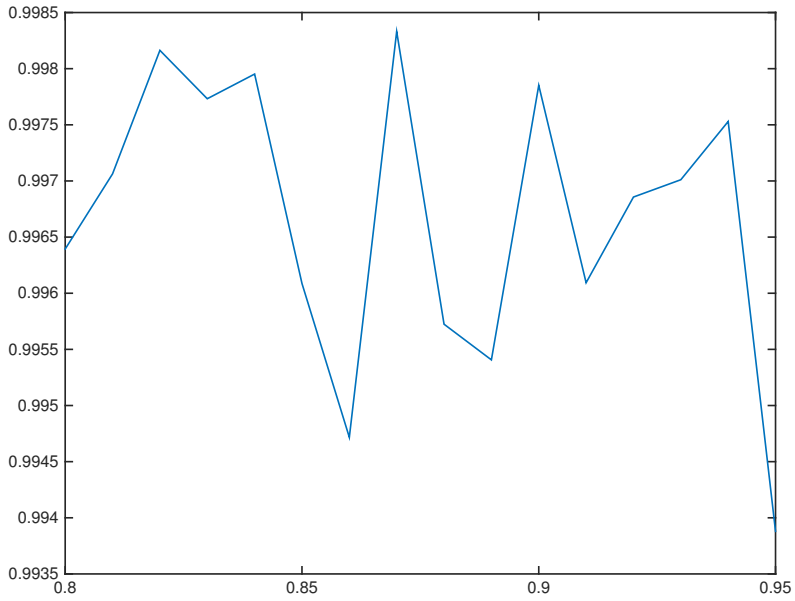

Figure 33: Fidelity vs β-parameter, degree-3-polynomial-kernel MVE, k=2

The best result is obtained for $\beta = 0.87$:

4.4 Visualization

We use the degree-3-polynomial kernel, with k=2 and $\beta = 0.9$.
Below are the results obtained:

	Fidelity
PCA	0.3413
KPCA	0.4620
SDE	0.9315
MVE	0.9973

4.4.1 PCA

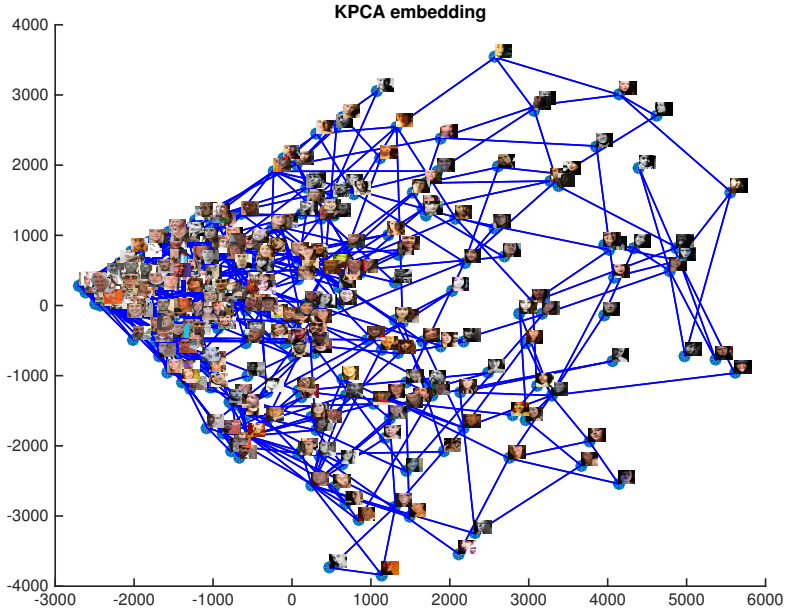

Figure 34: PCA result

4.4.2 KPCA

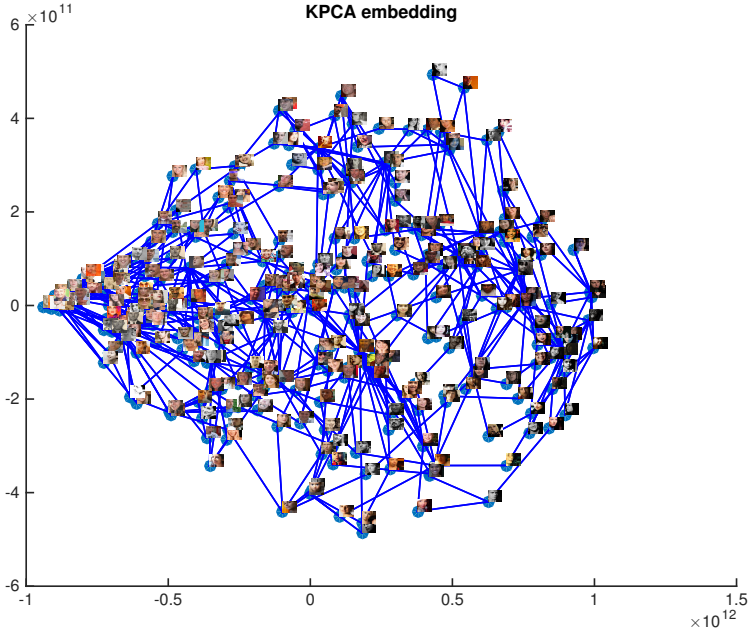

Figure 35: KPCA with degree-3-polynomial-kernel

4.4.3 SDE

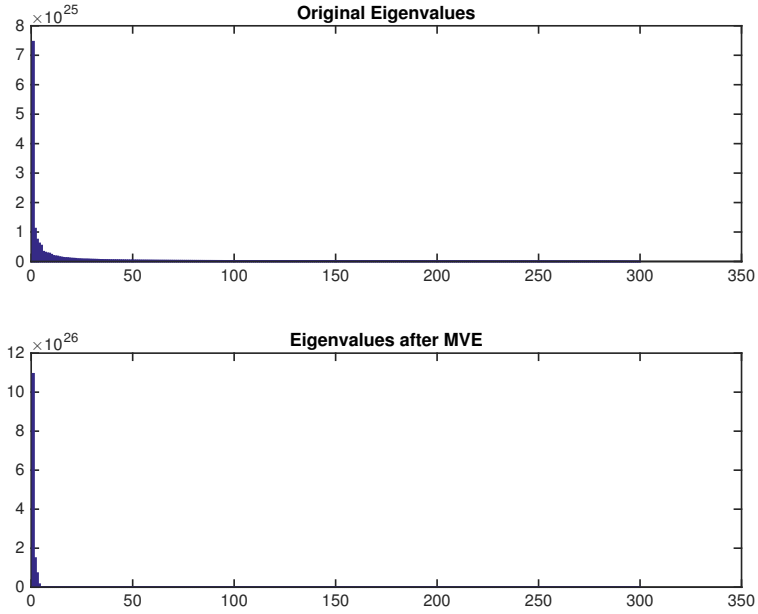

Figure 36: Eigenvalues for degree-3-polynomial-kernel SDE, k=2 and $\beta = 0.9$

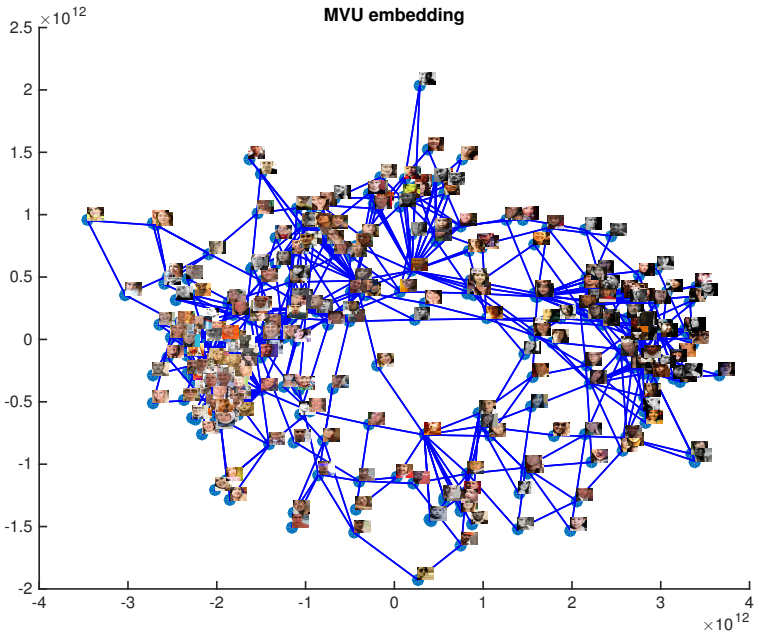

Figure 37: Degree-3-polynomial-kernel SDE, k=2 and $\beta = 0.9$

4.4.4 MVE

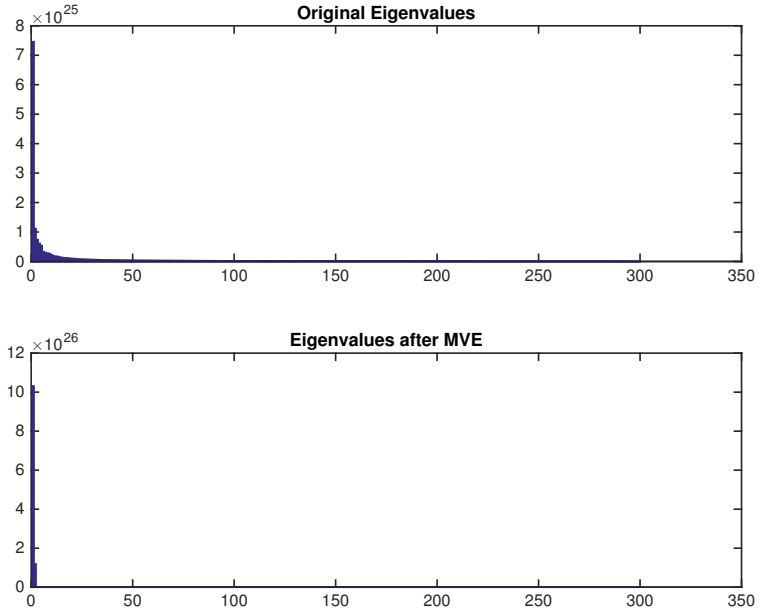

Figure 38: Eigenvalues for degree-3-polynomial-kernel MVE, k=2 and $\beta = 0.9$

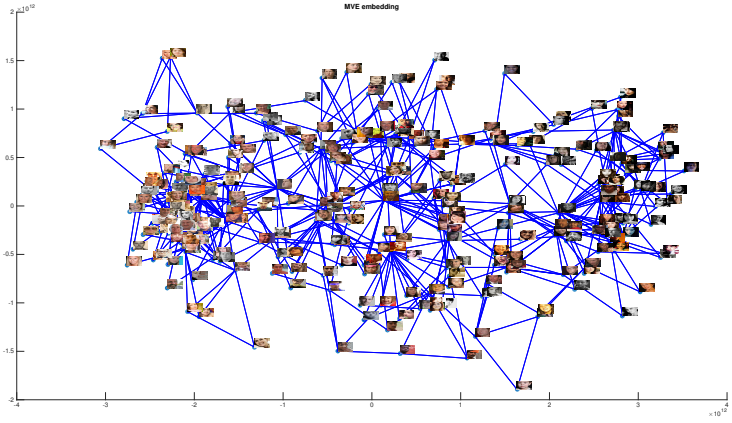

Figure 39: Degree-3-polynomial-kernel MVE, k=2 and $\beta = 0.9$

4.5 Results

The main eigenvector seems to correspond to the darkness of the pictures. On the left-side there is a cluster of light pictures while on the right there is a cluster of dark ones.

We can see that after MVE, $\lambda_1 \gg \lambda_2 \gg \lambda_i, i > 2$. Therefore with *MVE the data only need 2 dimensions*. However, with SDE, $\lambda_2 \approx \lambda_3 \gg \lambda_i, i > 3$, therefore *SDE needs 3 dimensions* to fully capture variance and efficiently visualize the data. In this case MVE is more efficient to focus variance in only 2 dimensions.

Also, we can see from Figure 29 that the fidelity steadily decreases with the number of nearest-neighbors. Indeed if we take into account too many neighbors, the MVE is too constrained and its structure can not be squeezed in 2 dimensions.

If we take the extreme case of N-nearest-neighbors where N is the number of points, the result should be very close to KPCA, because the N-dimensional structure of data-points will be kept.