COMP61021: Modelling and Visualization of High Dimensional Data

Lab Work: Applications of PCA

Yan Liu

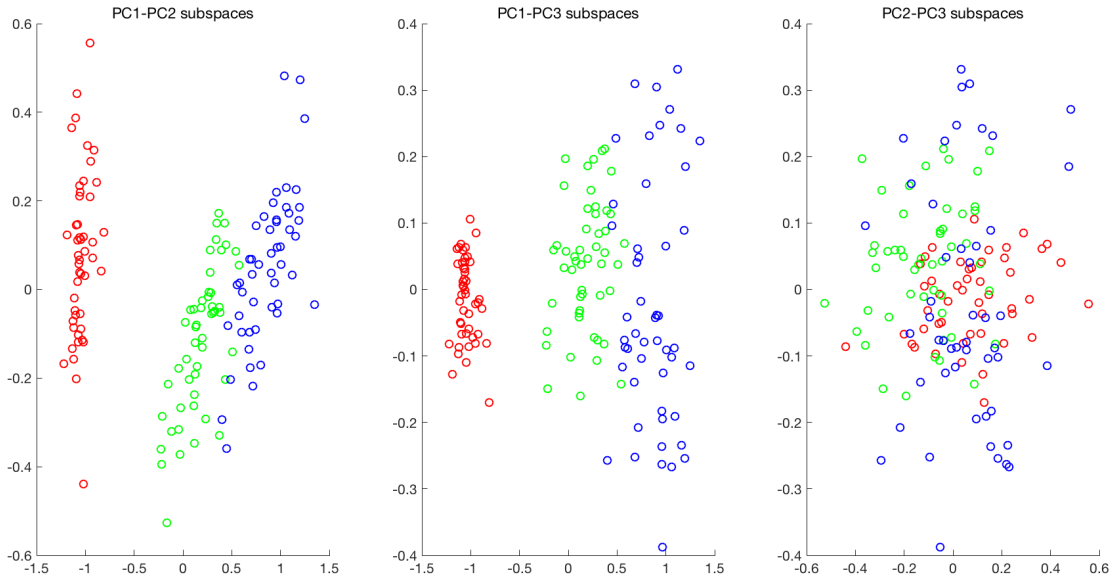
ID: 10443939

2019.11.18

Computer Scinece Department

PART 1 – Visualisation

In this part, I choose PCA derived from the co-variance matrix. PC_1, PC_2 and PC_3 denote top three principal components of the data set. The visualized results are shown below. The first plot is $PC_1 - PC_2$ subspaces, second plot is $PC_1 - PC_3$ subspaces and third plot is $PC_2 - PC_3$ subspaces. The red dot is example that the label is 1, green dot is example that the label is 2, blue dot is example that the label is 3.



From the result, we can see that when we choose the top two principal components of the data set (PC_1, PC_2) and project all 150 data points onto $PC_1 - PC_2$ subspace, the result is very linear separable, the distribution of the examples of the same label is very concentrated (almost on the same line). When choosing PC_1 and PC_3, although examples of different labels are still separable, examples of the same label are very discrete. While choosing PC_2 and PC_3, this is the worst way to reduce the dimension because all the points are mixed and not separable. Therefore, the choice of PC should be from large to small. Choosing the top PC of dataset is more conducive to data classification and maximum retention of data characteristics.

PART 2 – Image compression

1. What is the appropriate implementation of PCA used in this application?

In my opinion, the Matlab function of PCA derived from SVD is more appropriate for this digit dataset. Because there are 300 examles in raining subset while the dimension of every example is $28 * 28 = 784$ which is much bigger than the number of samples and will cost a lot of time while SVD has less computation.

2. How many principal components are needed to establish a satisfactory compression system?

The number of principal components is 47. I use proportion of variance(PoV) to determine it in this part. As we know, the eigenvalues in PCA tell us how much variance can be explained by its associated eigenvector. According to this, if we take all eigenvectors together, we can explain all the variance in the data sample. Based on this concept, PoV is calculated as below.

$$PoV = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{d} \lambda_i} = \frac{\lambda_1 + \cdots + \lambda_k}{\lambda_1 + \cdots + \lambda_k + \cdots + \lambda_d}$$

Actually, PoV (Proportion of variation) represents how much information of data will remain relatively to using all of them. If POV is high then less information will be lose. Therefore, I set $PoV >= 90\%$, then the corresponding k will be the best number of principal components. For this train subset, I use the loop, let k start at 0, gradually increase, until the PoV is greater than or equal to 0.9, and finally get $K = 47$. So, I need the top 47 principal components.

3. For 10 images in the test set, what are their low-dimensional representations?

All the low-dimensional representations of those 10 images in a table that are put in the appendix. Because the table is too big, so I used a picture to show it on one page

4. How can you reconstruct those images from their low-dimensional representations?

First of all, because the dimension reduction(compression) method I used is:

$$z = U_M^T (x - \bar{x})$$

where z is a M-dimensional vector, x is original data, $\bar{x}$ is the mean of x. So, in order to reconstruct those images, we need to invert the formula like this:

$$x_{reconstruted} = \bar{x} + U_M z$$

where $\bar{x}$ is the mean of train dataset. The test samples must minus the mean value of the training samples before we do dimensionality reduction on the test samples. And the transformation matrix obtained from the training samples must be used. Only in this way, can we ensure that the test samples are converted to the same sample space as training samples. Then, we cam get the low-dimensional representations of every 10 test images which is $784 * 1$ matrix. So, in order to plot the reconstructed images, I convert every image to $28 * 28$ matrix. The 10 original and their corresponding reconstructed images for comparison are shown below.

10 original images

10 corresponding reconstructed images (based on SVD)

5. How can you estimate their reconstruction errors for 10 test images?

The average reconstruction error is 5.8611. I use Quadratic Reconstruction Error method (shown below) to estimate their reconstruction errors.

$$E = \frac{1}{N}\sum_{n=1}^{N} \|x_n - \tilde{x_n}\|^2$$

where E is the reconstruction error, N is the number of samples, x_n is the original data point and $\tilde{x_n}$ is the reconstruction data point. First of all, I calculate the difference between original data pointand and reconstruction data point and do norm and square. Then use a matrix called rc_err to store the reconstruction error of every image.

the reconstruction error of each 10 test images are :

Image Num	image1	image2	image3	image4	image5
reconstruction error	3.99326754629869	4.20522867136251	4.25681047269034	4.63309932170772	7.56324175760717
Image Num	image6	image7	image8	image9	image10
reconstruction error	4.22895330711064	11.5287068422716	3.61706582801801	8.44255573472444	6.14207019238235

Then I calculate the average reconstruction error of these ten images, which is 5.8611 (when $k = 47$ and $pov <= 0.9$).

PART 3 – Bonus marks

In this part, I use Classical MDS (multidimensional scaling) to apply dimensionality reduction to problems described in Part 2 for a better performance. MDS is a form of non-linear dimensionality reduction and the distance standard of classical multi-dimensional scale transformation is Euclidean distance. What MDS wants to do is that the points in the high-dimensional coordinates are projected into the low-dimensional space and the similarity between the points is kept as unchanged as possible. So it is reasonable to choose MDS.

The algorithm of Classical MDS is :

(1) Set up the matrix of squared proximities $D^{(2)} = [d_{ij}^2]$

(2) Apply the double centering:$B = -\frac{1}{2}JD^2J$ using the matrix $J = I - n^{-1}11'$, where n is the number of objects.

(3) Extract the m largest positive eigenvalues $\lambda_1 \cdots \lambda_m$ of B and and the corresponding m eigenvectors $e_1 \cdots e_m$.

(4) A m-dimensional spatial configuration of the n objects is derived from the coordinate matrix $X = E_m \wedge \frac{1}{m}^2$, where E_m is the matrix of m eigenvectors and $_m$ is the diagonal matrix of m eigenvalues of B, respectively.

Then, In this part, I compress the image into 2 dimensions. Then reconstruct the images. And calculate the reconstruction error. The original and reconstruct 10 images are shown below:

10 original test images

10 corresponding reconstructed images (based on MDS)

From the pictures above, we can clearly see that, MDS not only compress the images, but also makes test images look more like digit "6". Then I calculate the reconstruction error, which is large because we compress the 784 dimensions image into 2 dimensions. But comparing the two pictures above, we found that most of the test images lost only the background (black) color, and the overall classification was improved. So, it has a better performance.

Next page is appendix page

Appendix :low-dimensional representations of 10 images

low-dimensional representations of 10 test images (47*10)									
Image1	image2	Image3	Image4	Image5	Image6	Image7	Image8	Image9	Image10
2.65107551	2.891163091	2.3139524	0.718266	-1.95338	-2.42588	-0.84461	-0.49474	3.274476	-3.35863
-1.57948756	-0.25482009	-3.545794	1.909137	-0.54896	-0.74325	-0.04825	2.640281	-3.73024	-0.89695
1.17659996	1.304441127	0.0491599	0.000498	-1.37304	-1.91917	-0.02704	0.09605	1.783502	2.027746
1.435489	0.820618249	0.5273486	1.923971	-0.0859	-1.50704	-0.4383	2.729446	-0.61715	1.804066
0.47122961	-0.89056002	-0.176172	1.146686	0.899535	-2.86243	3.301614	0.210383	1.263477	0.724879
-1.70520227	-0.82851642	-0.070157	-0.06382	2.430791	0.803214	2.72118	1.147327	0.267592	1.232527
1.67026962	0.350843564	-2.117249	0.630327	0.468291	1.658319	-1.67724	-0.88301	0.44128	0.091156
0.02441315	0.097952733	-0.975521	-1.30086	-1.94762	-1.30146	0.975312	-0.51201	-1.16594	-0.85658
-0.4739459	0.464680594	0.5973014	-0.73478	-0.38516	-0.18187	1.584731	-1.11163	1.022631	0.95875
0.47461732	1.326946812	0.8538434	0.422571	-0.47067	0.357166	0.2701	-0.0394	0.340277	-0.48321
0.17024926	-0.42433779	0.4333955	0.039439	0.733774	0.464226	0.318655	-0.10215	0.343291	-0.03985
-0.98950022	-1.2126494	-0.086338	-1.14281	-0.59561	0.923132	-1.27327	0.573677	0.568587	1.792414
-0.33885384	0.082312063	0.6582818	0.382149	-0.74467	-1.02939	-0.81477	-0.94683	0.404533	0.351295
-0.06981628	-0.76817604	-0.416099	0.008094	-0.95839	-0.30737	0.288508	-1.09041	-0.53925	-1.12912
0.04210334	1.041268472	1.1279925	0.309183	0.535581	0.252369	0.309158	-0.35819	-0.03817	1.02962
0.58281405	0.604581308	-0.393421	-0.32347	0.452034	0.498061	0.036652	-0.29922	1.361063	0.22317
-1.20387251	1.595211708	-1.040274	-1.25534	1.253387	-0.41338	-0.60246	0.024848	-0.19318	0.9416
0.01824239	-0.29177026	1.2087679	0.875558	0.725205	-0.54344	-0.19492	0.169827	0.625747	0.056999
0.26388896	-0.64588262	-0.063708	-0.2671	0.546895	0.310714	-0.37889	-0.3528	-0.09628	0.844397
0.09511732	0.653348332	-0.35146	-0.88863	0.312328	-0.33853	-1.31663	0.290414	-0.33232	0.492529
-0.2511068	0.130166287	0.1102852	-1.05925	-0.36834	-0.23489	-0.02244	0.020801	0.182273	0.205354
0.31251232	-0.17813906	-0.9993	-0.18324	0.425572	-0.1056	0.298782	-0.46207	-0.32676	-0.23557
0.38174837	0.526702289	0.2544024	0.540242	0.394885	1.027096	-0.26333	0.28858	-0.65028	-0.14432
-1.21740889	-0.33968466	-0.116783	0.407241	0.342767	-0.00434	-0.16741	-0.39763	-0.57016	-0.50822
-0.03803354	-0.34261011	-0.669788	-0.1449	-0.66296	-0.47947	-0.05731	-0.7306	0.005485	-0.18889
-0.68970611	0.35046923	-0.507231	0.513597	-0.14397	-0.14861	0.963571	1.006346	0.599917	-0.16182
0.13645575	0.155530626	0.6022415	0.394404	0.14693	0.28007	0.058784	0.521493	-0.35811	0.198151
0.09595673	-0.38928439	-0.244131	-0.17854	0.776238	0.295418	-0.4551	-0.30479	-0.16211	-0.78752
0.46916174	-0.10258491	0.619346	0.166594	0.615195	-0.09378	-0.22172	-0.2665	0.121009	-0.46518
0.61480439	-0.34757558	0.5099471	0.042229	0.350256	-0.04007	0.417064	-0.12662	0.260396	0.539208
-0.7119315	-0.11959055	0.0198108	-0.61813	0.541068	0.571006	0.027522	0.198613	-0.78745	0.29723
0.12139015	-0.04578465	-0.219	-0.17043	-0.15592	-0.16261	0.132649	-0.22792	-0.3465	-0.04803
-0.43895219	0.697794558	0.0383882	-0.26767	-0.64748	0.100522	0.264314	-0.34376	0.522739	0.778225
-0.09822461	-0.0739542	0.1501182	-0.36741	-0.41718	-0.09154	-0.76865	0.750277	-0.47514	0.799915
0.26249669	0.134606526	-0.533882	0.027719	0.730728	-0.30775	-0.31515	0.366104	0.124591	0.440724
0.45847351	-0.68357195	-0.019075	-0.18217	-0.72613	0.087712	-0.17038	0.359437	0.965924	0.302326
-0.60416419	0.091124357	0.2137942	0.486978	0.370055	0.43478	-0.20837	-0.1619	-0.45395	-1.03624