TP4: LDA and *k*means

by Martino Ferrari

1 Linear discriminant analysis

I implemented a simple 2-classes LDA algorithm.

I tested the algorithm with the class “3” and “7” of the *mnist_all* dataset with 4000 training data (2000 for each class) on 500 test images, all reduced to the first 50 principal components.

The results are exposed in table 1.

Class	Total	Predicted “3”	Predicted “7”	Recall	Missrate
“3”	245	240	4	0.979592	0.016327
“7”	255	5	251	0.984314	0.019608
Precision	0.982				
Time	0.74s				

Table 1. Confusion matrix of LDA

Later I classified the same data using the n-KK classifier I implemented for the last TP.

The results are exposed in table 2.

Class	Total	Predicted “3”	Predicted “7”	Recall	Missrate
“3”	245	242	3	0.987755	0.012245
“7”	255	3	252	0.988235	0.011765
Precision	0.988				
Time	12.64s				

Table 2. Confusion matrix of PCA

The n-KK classifier is slightly more precise, 99% against 98% of the LDA, but is much slower, 12.64s against 0.74s. The ratio precision time of the LDA is really impressive.

2 *k*Means classification

*k*Means is another clustering (classification) method.

First I analyzed the evolution of $J(y)$, the total distance of the samples from the cluster center, over the number of clusters.

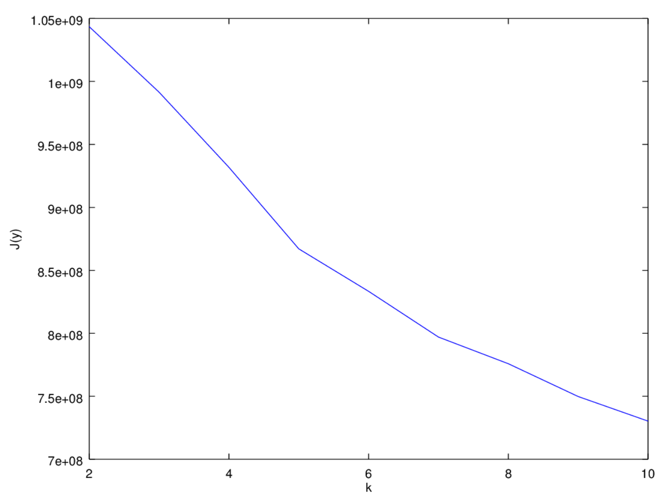

Figure 1. evolution of J with k

As predictable the distance is decreasing with the number of clusters, as the space will be divided in more and more clusters with smaller areas.

Using this method over our data set give us very interesting results (table 3).

Class	Total	Predicted “3”	Predicted “7”	Recall	Missrate
“3”	251	241	14	0.960159	0.055777
“7”	249	10	235	0.943775	0.040161
Precision	0.952				
Time	0.04s				

Table 3. Confusion matrix of *k*Means

This method is much more faster ($\times 18$) than the LDA but is slightly less precise. This performance are reached by a higher optimization of my Octave code and by the fact that itself the algorithm is simplier. However the perforance are really impressive, even more if compared to the *k*NN method (more then 300 times faster).

Using the same method over a different set of classes results in a very different distribution (table 4).

Class	Total	Predicted "3"	Predicted "5"	Recall	Missrate
"3"	319	222	41	0.960159	0.055777
"5"	181	97	140	0.943775	0.040161
Precision	0.724				
Time	0.13s				

Table 4. Confusion matrix of kMeans, class "3" and "5"

The performance and the quality of the results for this two classes are much worst than for the previous ones, this is probably due to the similarity of the digit 3 and 5, the bottom part of the digit is the same and only the upper part change.

The similitude of the two data set can be observed as well in the distance of the center of the two cluster:

$\|g(3) - g(7)\| = 1421.3$

$\|g(3) - g(5)\| = 1124.9$

as well from the mean distance of the mean distance of the samples form the respective cluster centers ($\bar{d}(l_1, l_2) = \frac{1}{n}\sum_{i=1}^{n} \|x_i - g_j\|$):

$\bar{d}(3, 7) = 64.76$

$\bar{d}(3, 5) = 68.40$

The two cluster in the second case are closer togheter while the samples are further away, this implies a higher probability of making mistakes.

3 Code

As already told, the code was developed for Octave, and never tested with Matlab (and I suspect some operation are not allowed in Matlab as the subtraction of a matrix and vector). However Octave is free and available for all platform.

Code/	folder contains all octave code
↪ pca.m	return the pca, eigen values and normalized eigen values
↪ lda.m	return a, b and S_w values of the lda
↪ kkmeans.m	return k clusters and sum of data distances from the cluster mean
↪ nkk.m	return the index of the
↪ rm.m	reconstruct an image using m pca components
↪ con_matrix.m	print confusion matrix
↪ tp4_e1.m	part 1 of the TP
↪ tp4_e2.n	part 2 of the TP