# An Experimental Study on Neighbourhood Component Analysis

Ting-Shuo Yo
0501050

**Abstract**

In this experimentation project, we introduce the *Neighbourhood Component Analysis* (NCA), a classification method combining *k*-nearest neighbour (KNN) and learned distance metric, originally proposed by Goldberger et al. in 2004. With an implementation in the R environment, we perform experiments on several datasets. Results of our experiments suggest that with the learned distance metric, KNN classification can be improved, especially for skewed and noisy data. For certain type of data, NCA also outperforms traditional dimension reduction methods. However, the comparisons to other classifiers, i.e. linear and quadratic discriminant analysis, do not show significant strength in this method.

## 1 Introduction

Neighbourhood component analysis (NCA) [4] is a method for learning a Mahalanobis distance measure to be used in the k-nearest-neighbour (KNN) classification. The traditional KNN classifier suffers from two draw backs: one is the computational/storage demand for the training data, and the other is on the difficulty to define a suitable distance measure for the data. With a learned distance measure, the later problem can be solved. If the learning process can be done accompanied with a rank reduction, the computational/storage demand may also be reduced.

Besides NCA, there are also several distance-measure-learning methods proposed and applied in face recognition, namely Principle Component Analysis (PCA) combined with other classification methods, Discriminant Common Vectors(DCV) [1, 2], and Laplacianfaces(LAP) [5]. While both NCA and DCV focus on optimizing the distance metric on certain objective functions, LAP emphasizes more on dimension reduction. Among these methods, NCA seems also suitable to serve as a general purpose classifier, regardless of its original application to face recognition.

In this research, we study NCA as a general classifier. With an implementation in the R environment, we compare its performance with other classifiers. In section 2, we give a more detailed description of NCA, and the detailed derivation can be found in appendix A. The algorithms of our implementation are described in section 3, and the test results of 6 datasets (4 from UCI Machine Learning Repository and 2 self-generated) are shown in section 4. Finally, we present some discussions and a few concluding remarks in section 5.

## 2 Learning the distance measures for KNN

### 2.1 Form of distance measure and the objective

In the NCA framework, the distance measure is limited within Mahalanobis (quadratic) distance metrics, which can always be represented as symmetric positive-definite matrices. The distance between two vectors, $x$ and $y$, can be represented as:

$$d(x, y) = (x - y)^T Q(x - y) = (Ax - Ay)^T (Ax - Ay) = ||Ax - Ay||^2,$$

where $A = Q^{1/2}$ is the learned metric. As shown in the formula above, $A$ can be seen as a transformation matrix from the original feature space.

For a dataset having $n$ data points and $m$ attributes, $A$ is a $m \times m$ matrix. With certain dimension reduction techniques and a desired size of the transformed dimension $d(d < m)$, it is possible to restrict A to a $d \times m$ matrix. By using dimension reduction, the demand for computation/storage can also be moderated.

To learn a distance metric, we need to specify a proper objective to be optimized. Since the leave-one-out(LOO) error is generally regarded as a good estimator for the true error rate, it might be a good choice of the objective. That is to say, we want to learn a transformation matrix, $A$, that minimizes LOO for KNN in the training data.

### 2.2 Stochastic neighbour selection

In order to minimize the LOO error, a mechanism for neighbour selection is required. Instead of selecting the neighbour directly base on the learned distance, Goldberger et al. [4] suggested to use a probability distribution for neighbour selection. The probability that point $x_i$ being selected as a neighbour of point $x_j$ is defined as:

$$p_{ij} = \frac{\exp(-||Ax_i - Ax_j||^2)}{\sum_{k \neq i} \exp(-||Ax_i - Ax_k||^2)} \qquad , \qquad p_{ii} = 0 \tag{1}$$

which is a softmax of the Euclidean distances in the reformed space. Whenever the point $x_j$ is in the same class as $x_i$, the classification is correct. We hence define the collection of such $x_j$ as the *correctly selected neighbours*, $C_i = \{x_j | class(x_j) = class(x_i)\}$. So, the probability of correct classification, $p_i$, can be denoted as:

$$p_i = \sum_{j \in C_i} p_{ij} \tag{2}$$

As stated previously, the learned $A$ should minimize the LOO error, which is equivalent to maximizing the probability that the selected neighbour $x_j$ is in the same class as each test point $x_i$. By summing up equation (2) for all data points, we have the *expected number of points correctly classified*, which is used as the objective function to be maximized in the NCA.

$$f(A) = \sum_i \sum_{j \in C_i} p_{ij} = \sum_i p_i \tag{3}$$

### 2.3 Procedure of Neighbourhood Component Analysis

Given the objective function (3), and letting $x_{ij} = x_i - x_j$, the gradient of the objective function can be derived as:

$$\frac{\partial f(A)}{\partial A} = 2A \sum_i \left( p_i \sum_k p_{ik} x_{ik} x_{ik}^T - \sum_{j \in C_i} p_{ij} x_{ij} x_{ij}^T \right) \tag{4}$$

The detailed derivation of the objective function and its gradient can be found in appendix A. With a proper optimizer given (3) and (4), an optimized transformation matrix $A$ can be found. Also with the defined probability distribution, $p_{ij}$, a real-value estimate of the optimal number of neighbours($k$) can be found as the effective perplexity, $\hat{K}$, of this distribution:

$$\hat{K} = \exp(-\sum_{ij} p_{ij} \log p_{ij} / N) \tag{5}$$

, where $N$ denotes the size of the data. With the learned matrix $A$ and $\hat{K}$, the original data can be projected into the transformed space, and a KNN with Euclidean distance can be used for classification.

Thus, the NCA algorithm can be summarized as follows:

1. Use the objective function and its gradient to learn the transformation matrix $A$ and $\hat{K}$ from the training data, $D_{train}$(with or without dimension reduction).

2. Project the test data, $D_{test}$, into the transformed space.

3. Perform traditional KNN (with $\hat{K}$ and $AD_{train}$) on the transformed test data, $AD_{test}$.

# 3 Implementing NCA in R

## 3.1 Algorithms and corresponding functions

In this study, the NCA algorithm has been implemented in the S-language and tested in the R environment. The basic matrix manipulations and KNN methods are already supported in R, as well as a multiple-parameter optimizer, **optim** [6]. Hence the implementation work can be reduced to equation (1) - (5), combined with calling of a few supported functions. For convenience, the objective function and its gradient, (3) and (4), are named as **f(A)** and **g(A)**, respectively, while (5) is named as **Perplexity(A)**. Algorithms used for implementation are shown below, namely **NCA** and **NCAKNN**.

**Algorithm 1** NCA($D_{train}$, InitialGuess, ....)

f(A) $\leftarrow$ eqn(3)
g(A) $\leftarrow$ eqn(4)
Perplexity(A) $\leftarrow$ eqn(5)
$A_0 \leftarrow$ InitialGuess
$A_{opt} \leftarrow$ optim($A_0$,f(A),g(A),$D_{train}$)
$\hat{K} \leftarrow$ Perplexity($A_{opt}$)
**return** $(A_{opt}, \hat{K})$

**Algorithm 2** NCAKNN($D_{train}, D_{test}$)

TM $\leftarrow$ NCA($D_{train}$).$A_{opt}$ {learned transformation matrix}
K $\leftarrow$ NCA($D_{train}$).$\hat{K}$ {learned scaling parameter}
$D'_{train} \leftarrow D_{train} \cdot$ TM
$D'_{test} \leftarrow D_{test} \cdot$ TM
PRED $\leftarrow$ KNN($D'_{train}, D'_{test}$, K)
**return** PRED

Algorithm (1) is used for learning the the distance metric from training data, and the classification procedure described in previous section is implemented as algorithm (2).

## 3.2 Issues in implementation

There are three points worth mentioning in our implementation. First, the headers of two functions are:

- nca(train ,class-label, rd, wt, start, rseed)
- ncaknn(train, test, class-label, k,...)

As shown above, several parameters (rd, wt, and rseed) are used to control the learning process (dimension-reduction, weights-for-each-point, and random-seed). And the initial guess for optimization is also implemented as options to choose, among $\Sigma^{-1/2}, \Sigma_{pooled}^{-1/2}$, and random matrices, where $\Sigma$ denotes the covariance matrix of training data, and $\Sigma_{pooled}$ denotes the pooled covariance matrix by class labels.

| Dataset | Description | Attributes | Classes | Size |
|---|---|---|---|---|
| **bal** | UCI balance-scale data | 4 | 3 | 625 |
| **ion** | UCI ionosphere data | 33 | 2 | 351 |
| **iris** | UCI iris data | 4 | 3 | 150 |
| **wine** | UCI wine data | 13 | 3 | 178 |
| **n2d** | 2-d normal distribution | 2 | 2 | 100 |
| **ring** | 2-d ring plus 8-d noise | 10 | 2 | 200 |

Table 1: Summary of used datasets. Including their names, short descriptions, number of attributes, and number of classes.

Second, the **optim** function provides several methods for optimization, namely: **DEFAULT, BFGS, CG, L-BFGS-B,** and **SANN**. During the test phase, **CG**(conjugate gradients method) appears to have problems with convergence, while **DEFAULT** and **SANN** are much slower than **BFGS** (quasi-Newton method, also known as a variable metric algorithm). Therefore, **BFGS** is used by default in our implementation of NCA, and this option can be change by giving an extra parameter during the function call.

Finally, several types of initial guess are used for optimization: the identity matrix ($I$), $\Sigma^{-1/2}$, $\Sigma_{pooled}^{-1/2}$), and random numbers in $[0, 1]$. In our pilot experiments (which are not shown here), using $I$ only rescales the original dimension of the data, and hence produces identical results as in the untransformed space. This is because the KNN function in R will normalize the data by default. As for other initial guesses, none of them always outperforms others. By using $\Sigma^{-1/2}$ as initial guess for reduced dimension, KNN can be seen as an optimization procedure starting from first few Principal Components from a Principal Components Analysis (PCA). Therefore, $\Sigma^{-1/2}$ and $\Sigma_{pooled}^{-1/2}$ will be used for experiments in the next section.

In addition to these issues, we encounter another difficulty. Because the parameters to be optimized is in forms of matrices, the optimizers would encounter problems of "local optima". As a result, for a few datasets, our NCA algorithm fails to give a higher classification accuracy than traditional KNN. However, the **optim** function does not allow inspection and modification to its source code, and matrix optimization belongs to current research topics in numerical linear algebra. Considering the scope of this study, this problem is therefore left as future work.

# 4 Experiments with NCA

## Datasets

In order to compare our results to [4], the same four datasets from the UCI repository [3] has been chosen for testing, namely balance-scale, ionosphere, iris, and wine. In addition, two self-generated datasets are also tested, to better understand the theoretical behavior of NCA. A short summary of these datasets is shown in table 1.

Figure 1 demonstrates the structure of two self-generated datasets. More details about the UCI datasets can be found at the UCI repository [3].

Each of these six datasets will be tested with several classification methods, i.e., NCA, KNN, Linear Discriminant Analysis (LDA), and Quadratic Discriminant Analysis (QDA). In addition, they are also tested with PCA+KNN to be compared with the results of NCA with dimension reduction. All tests are performed with a 10-fold cross validation method, and the results of the experiments are summarized bellow.

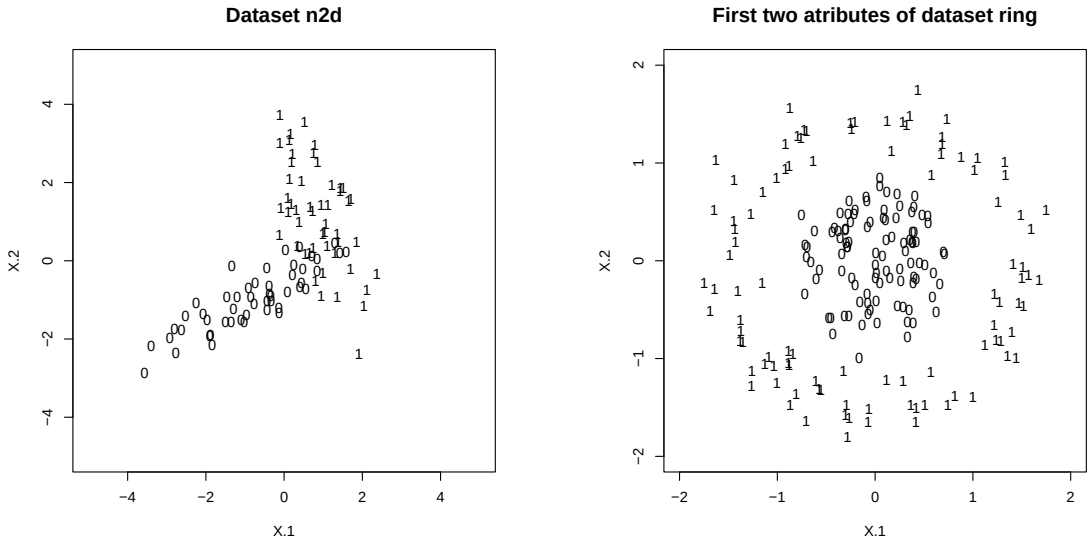


Figure 1: Illustration of two self-generated datasets. **n2d** is a mixture of two bivariate normal distributions with different means and covariance matrices. **ring** consists of 2-d concentric rings and 8 dimensions of uniform random noise.

| | **bal** | **ion** | **iris** | **wine** | **n2d** | **ring** |
|---|---|---|---|---|---|---|
| **k** | 11 | 2 | 9 | 1 | 4 | 2 |
| **KNN** | 0.1023 | 0.1537 | 0.0533 | 0.0556 | 0.1600 | 0.5300 |
| **NCA** | 0.0847 | 0.1567 | 0.0267 | 0.0222 | 0.1600 | 0.3050 |

Table 2: Error rates of KNN and NCA with the same K. It is shown that generally NCA does improve the performance of KNN.

### KNN on original and transformed space

First of all, we want to know "whether the transformation improves the performance of KNN". As shown in table 2 and figure 2, the learned transformation does improve the accuracy of KNN, although the improvement seems not significant in some datasets.

### Comparing NCA with other classifiers

While NCA outperforms traditional KNN in previous described experiments, it may still be outperformed by other classifiers. To compare the performance of NCA to other classifiers, the accuracy among LDA, QDA, 1NN and NCA is shown in figure 3. It is obvious that NCA never outperforms QDA, and hence NCA may not be the best classifier.

### Dimension reduction

One advantage of NCA is that it includes the dimension reduction (or feature selection) and distance metric learning into one single procedure. As discussed in section 3, with one single

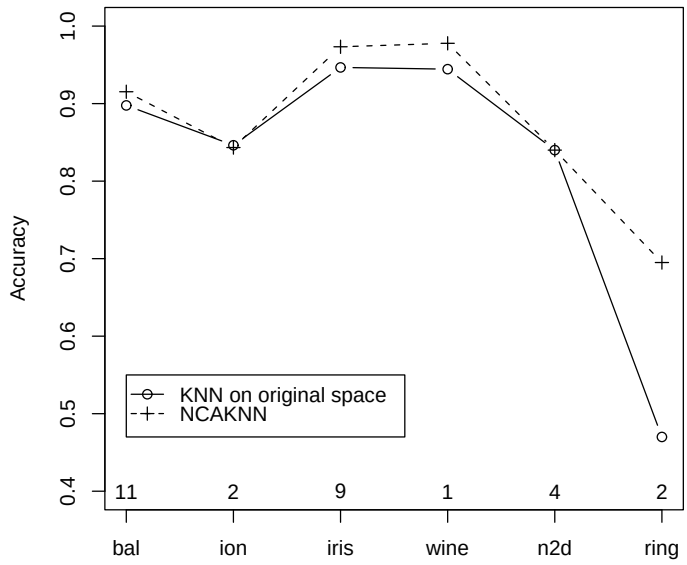


Figure 2: Averaged accuracy in 10-fold cross validation of KNN and NCA with the same K, which is shown above the horizontal axis.

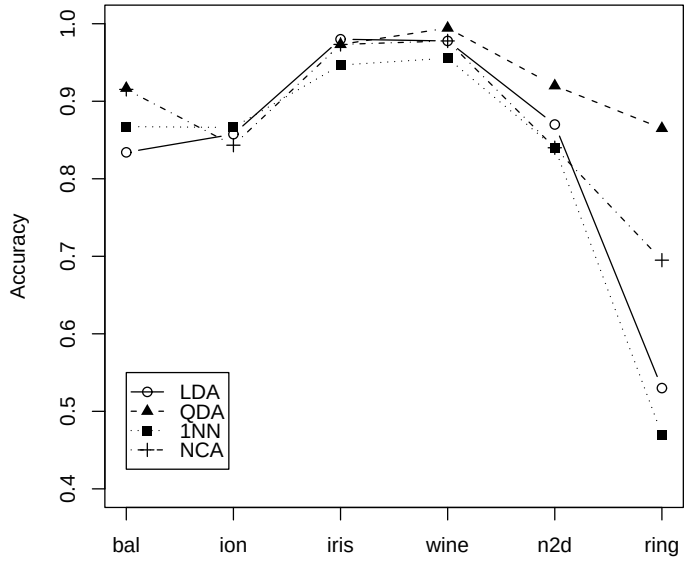


Figure 3: Averaged accuracy in 10-fold cross validation of LDA, QDA, 1NN and NCA.

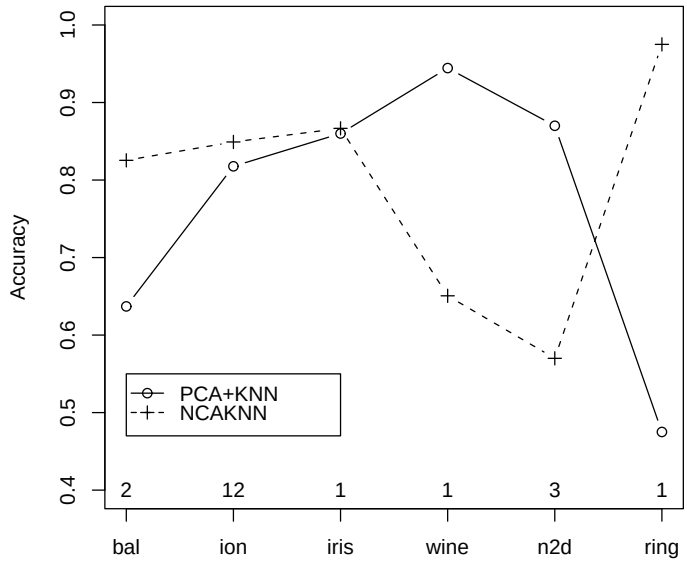


Figure 4: Averaged accuracy of rank 2 transformation by PCA+KNN and NCA with the same K, which is shown above the horizontal axis.

parameter $d$, the transformation matrix will be reduced from $m \times m$ ($m$ represents the original dimension) to $d \times m$, and hence produce a space with a lower number of dimension, $d$. This procedure is called **rank $d$ transformation** in [4]. So, is NCA better than other dimension reduction methods?

Principal Component Analysis (PCA) is the most widely used technique in dimension reduction. It is also mentioned in previous section that NCA using $\Sigma^{-1/2}$ as the initial guess can be seen as optimizing on the first $d$ principal components. As a results, it is expected that NCA outperforms PCA+KNN with the same K, if the there is no problem in the optimization procedure.

Figure 4 shows the results of two methods with the reduced dimension $d = 2$ ($d = 1$ for n2d dataset). For datasets **bal**, **ion**, **iris** and **ring**, NCA does give better performance. However, for datasets **wine** and **n2d**, NCA performs much worse than PCA+KNN. This result conflicts with [4] on UCI **wine** dataset, in which NCA is reported to outperform PCA+KNN (with an accuracy ratio of 0.87 to 0.67). This disagreement will be further discussed in next section.

# 5 Discussion and conclusion

## 5.1 Dataset UCI wine and n2d

As mentioned in the previous section, NCA gives a worse performance for dataset **wine** and **n2d**. This discouraging result requires further exploration and explanation.

Figure 5 shows the rank 2 transformation of **wine** dataset by PCA and NCA. The illustration of the first two PCs shows a well separated pattern of three different classes, but the NCA transformed space is kind of mixed up. This result disagrees with [4]'s result for the same comparison, for they reported the opposite result.

The datasets are standardized before testing in our experiments (which is not mentioned in [4]), and possibly the optimizers used are different, but all other procedures in two study

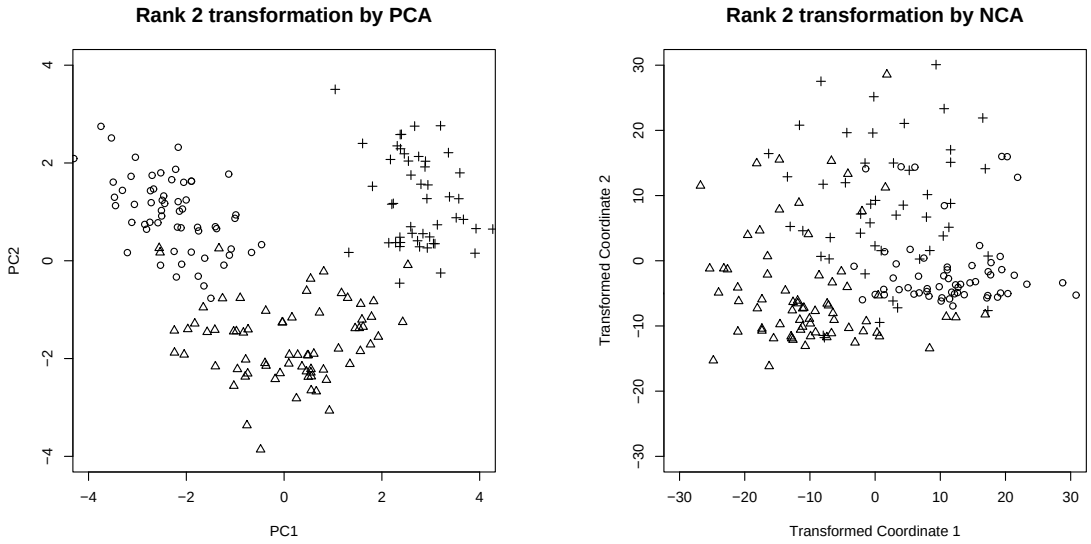


Figure 5: Rank 2 transformation of **wine** dataset by PCA and NCA.

are identical. Therefore this suggests some exploration on the optimization algorithms may be necessary in further study.

A similar comparison for **n2d** dataset is shown in figure 6. Because the transformed space by PCA shows a smaller mixed-up region than NCA, it is reasonable that PCA gives a better classification accuracy. Again, it is hard to explain why an optimizer gives worse result than the initial value without looking into the optimization algorithms. However, matrix optimization is beyond the scope of this study, and hence will not be discussed here.

## 5.2 Performance issue

As shown in section 4, NCA improves the accuracy of KNN method, with a price of more computation. By observation on our experiments, the optimization algorithm usually converges after $600 \sim 1000$ iterations, and the computation time is therefore also around 1000 times more.

Given the fact that some less costly classifiers (e.g., LDA and QDA) do outperform NCA, this distance metric learning algorithm should only be used when more knowledge about the datasets is known.

Another way to improve NCA's computing time may lie on better optimization algorithms. But this issue is again beyond the reach of this study.

## 5.3 Concluding remarks

In this study, NCA algorithm is studied. With an implementation in R, several experiments are conducted to explore the performance of NCA. A summary of important findings and a few concluding remarks are listed below.

1. Results of experiments suggest that with the learned distance metric by NCA algorithm, KNN classification can be improved.

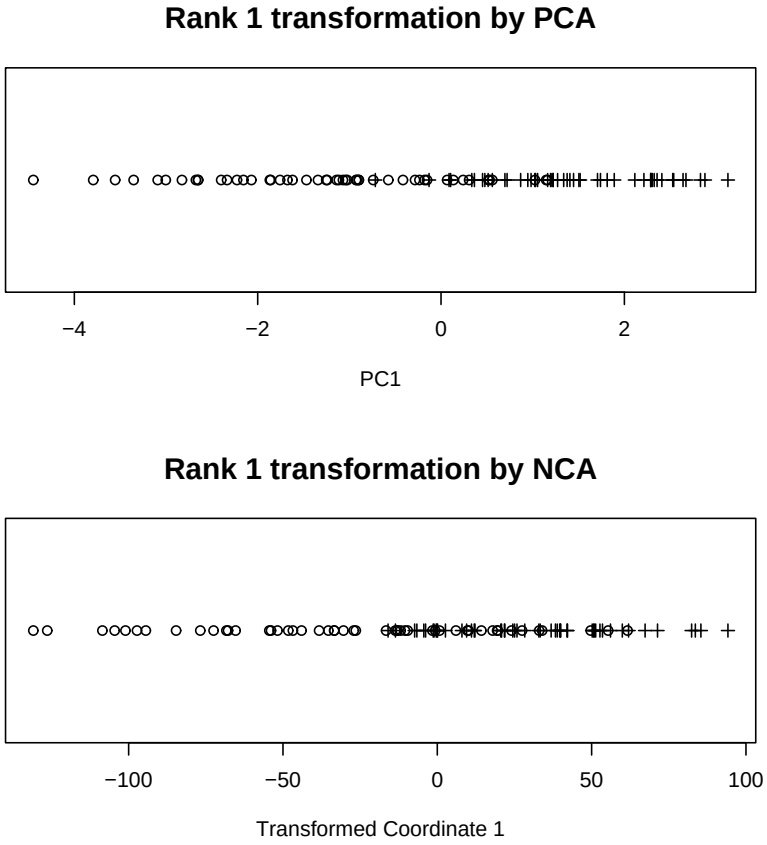


Figure 6: Rank 1 transformation of **n2d** dataset by PCA and NCA.

2. NCA also outperforms traditional dimension reduction methods for several datasets.

3. Comparing to other classification methods (i.e. LDA and QDA), NCA usually does not give the best accuracy.

4. Some odd performance on dimension reduction suggests that a further investigation on the optimization algorithm is necessary.

## Appendix A: Deriving the objective function for NCA

In this appendix we provide a detail derivation of the objective function and its gradient. Given the definition described in section 2, we have:

- the probability that point $j$ is selected as a neighbour of point $j$, $p_{ij}$:

$$p_{ij} = \frac{\exp(-||Ax_i - Ax_j||^2)}{\sum_{k \neq i} \exp(-||Ax_i - Ax_k||^2)} \qquad , \qquad p_{ii} = 0 \tag{a.1}$$

- the probability of correct classification, $p_i$:

$$p_i = \sum_{j \in C_i} p_{ij} \tag{a.2}$$

- the objective function to be maximized, $f(A)$:

$$f(A) = \sum_i \sum_{j \in C_i} p_{ij} = \sum_i p_i \tag{a.3}$$

For easier denote, we let $x_{ij} = x_i - x_j$, the gradient of objective function is:

$$\frac{\partial f(A)}{\partial A} = \frac{\partial}{\partial A} \sum_i \sum_{j \in C_i} \frac{\exp(-||Ax_i - Ax_j||^2)}{\sum_{k \neq i} \exp(-||Ax_i - Ax_k||^2)} \tag{a.4}$$

Given the fact that:

$$\frac{\partial}{\partial A} \exp(-||Ax_i - Ax_j||^2) = \frac{\partial}{\partial A} \exp(-A^2 x_{ij}^2) = -2Ax_{ij}^2 \exp(-A^2 x_{ij}^2) \tag{a.5}$$

, and

$$\frac{\partial}{\partial A} \sum_{k \neq i} \exp(-||Ax_i - Ax_j||^2) = -2A \sum_{k \neq i} (x_{ik}^2 \exp(-A^2 x_{ik}^2)) \tag{a.6}$$

then (a.4) becomes (a.7):

$$\begin{aligned} & \sum_i \sum_{j \in C_i} \left( \frac{1}{\sum_{k \neq i} \exp(-A^2 x_{ik}^2)} \right) (-2A) x_{ij}^2 \exp(-A^2 x_{ij}^2) \\ + & \sum_i \sum_{j \in C_i} \left( \frac{1}{\sum_{k \neq i} \exp(-A^2 x_{ik}^2)} \right) (-1)(-2A) \frac{\sum_{k \neq i} x_{ik}^2 \exp(-A^2 x_{ik}^2)}{\left[ \sum_{k \neq i} \exp(-A^2 x_{ik}^2) \right]^2} \end{aligned}$$

To simplify the equation above, we use the relation in (a.1) that:

$$\exp(-A^2 x_{ik}^2)) = p_{ik} \sum_{k \neq i} \exp(-A^2 x_{ik}^2)$$

since $p_{ii} = 0$ by definition, the summation on all pairs $(i, j)$ will be the same as on $(i, k | k \neq i)$. And hence the second term in (a.7) can be simplified as:

$$\frac{\sum_{k \neq i} x_{ik}^2 \exp(-A^2 x_{ik}^2)}{\left[ \sum_{k \neq i} \exp(-A^2 x_{ik}^2) \right]^2} = \frac{\sum_{k \neq i} x_{ik}^2 p_{ik} \sum_{l \neq i} \exp(-A^2 x_{il}^2)}{\left[ \sum_{k \neq i} \exp(-A^2 x_{ik}^2) \right]^2} = \frac{\sum_{k \neq i} x_{ik}^2 p_{ik}}{\sum_{k \neq i} \exp(-A^2 x_{ik}^2)}$$

Thus, the gradient for the objective function becomes (a.8):

$$\begin{aligned}
\frac{\partial f(A)}{\partial A} &= (a.4) = (a.7) \\
&= -2A \sum_i \sum_{j \in C_i} \left( x_{ij}^2 p_{ij} - (\sum_{k \neq i} x_{ik}^2 p_{ik}) p_{ij} \right) \\
&= -2A \sum_i \sum_{j \in C_i} p_{ij} (x_{ij}^2 - \sum_{k \neq i} p_{ik} x_{ik}^2) \\
&= 2A \sum_i (p_i \sum_{k \neq i} p_{ik} x_{ik}^2 - \sum_{j \in C_i} p_{ij} x_{ij}^2)
\end{aligned}$$

In our implementation, equation (a.3) and (a.8) are used for optimizing the tranformation matrix, $A$.

## References


[1] Hakan Cevikalp, Marian Neamtu, Mitch Wilkes, and Atalay Barkana. Discriminative common vectors for face recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(1):4–13, 2005.

[2] Hakan Cevikalp and Mitch Wilkes. Face recognition by using discriminative common vectors. In *ICPR (1)*, pages 326–329, 2004.

[3] C.L. Blake D.J. Newman, S. Hettich and C.J. Merz. UCI repository of machine learning databases, 1998.

[4] Jacob Goldberger, Sam T. Roweis, Geoffrey E. Hinton, and Ruslan Salakhutdinov. Neighbourhood components analysis. In *Advances in Neural Information Processing Systems 17*, 2004.

[5] Xiaofei He, Shuicheng Yan, Yuxiao Hu, Partha Niyogi, and Hong-Jiang Zhang. Face recognition using Laplacianfaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(03):328–340, 2005.

[6] W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S.* Springer, September 2003.