Chapter 2

Background

This chapter sets the theoretical basis for the following discussion on low-rank metrics. We start with a brief mathematical introduction into distance metrics. Then we outline the characteristics of the Mahalanobis-like metric. At the end of the chapter we review some of the most prominent work in the area. The method we study, neighbourhood component analysis (NCA), is presented in chapter 3.

2.1 Theoretical background

A distance metric is a mapping from a pair of inputs to a scalar. Let $d(x, y)$ denote the distance function between the arguments x and y. Given a set $\mathcal{S}$, we say that d is a metric on $\mathcal{S}$ if it satisfies the following properties for any $x, y, z \in \mathcal{S}$:

- Non-negativity: $d(x, y) \geq 0$.
- Distinguishability: $d(x, y) = 0 \Leftrightarrow x \equiv y$.
- Symmetry: $d(x, y) = d(y, x)$.
- Triangle Inequality: $d(x, y) \leq d(x, z) + d(z, y)$.

Essentially, a metric can be defined on any set $\mathcal{S}$ of objects. The distance metric is valid as long as the above properties hold for any elements in the set. However, for the scope of our applications we limit ourselves to points in D dimensional real space. The inputs will be represented by column vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^D$. In this case the distance d is a function from $\mathbb{R}^D \times \mathbb{R}^D$ to $\mathbb{R}$. A well known example of such a function is the Euclidean metric. This is defined by:

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})^{\mathrm{T}}(\mathbf{x} - \mathbf{y})}, \quad \text{with } \mathbf{x}, \mathbf{y} \in \mathbb{R}^D. \tag{2.1}$$

Albeit useful in many scenarios, the Euclidean metric has two properties that are problematic in machine learning applications:[1]

- **Sensitivity to variable scaling**. In geometrical situations, the values across all D dimensions are measured in the same unit of length. In practical situations however, each dimension encodes a different feature of the data set. The Euclidean metric is sensitive to scaling and it returns different results if we change the measurement units for one of the D variables. We compensate this differences by dividing the values on dimension i by the standard deviation s_i on that direction:

$$\begin{aligned} d(\mathbf{x},\mathbf{y}) &= \sqrt{\left(\frac{x_1 - y_1}{s_1}\right)^2 + \cdots + \left(\frac{x_D - y_D}{s_D}\right)^2} = \\ &= \sqrt{(\mathbf{x}-\mathbf{y})^{\mathrm{T}}\mathbf{S}^{-1}(\mathbf{x}-\mathbf{y})}, \end{aligned} \tag{2.2}$$

  where $\mathbf{S} = \mathrm{diag}(s_1^2, \cdots, s_D^2)$ and $s_i^2 = \mathrm{var}[x_i], \forall i = 1, \cdots, D$.

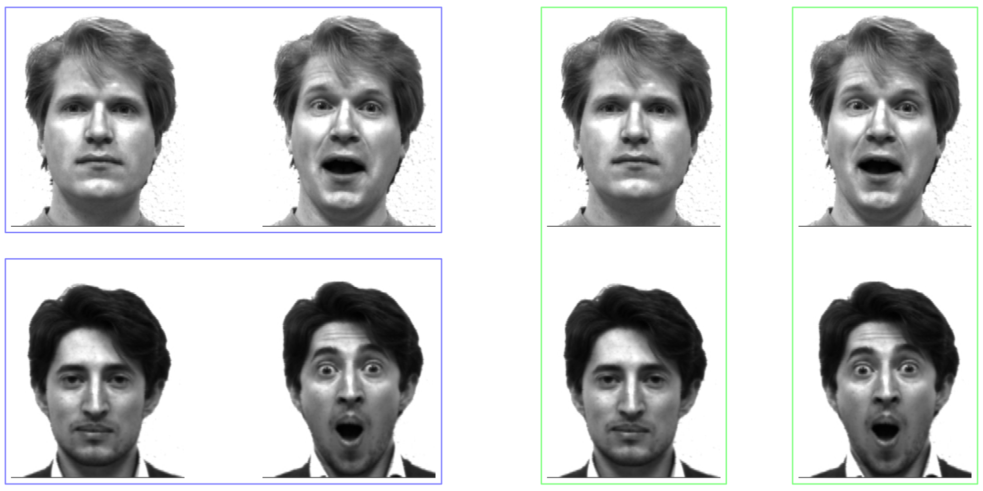
- **Invariance to correlated variables**. Euclidean distance treats equally the discrepancy across any of the D dimensions. Often many features of a data set are redundant and, because of their number, they significantly influence the distance. Take the example of images and face recognition. The pixels from the image background are highly correlated and they reflect the same information: the colour of the background. If the two pictures of the same subject have different backgrounds, we get a high distance between the images. This effect is alleviated if we replace $\mathbf{S}$ in equation (2.2) with the covariance matrix of the data:

$$d(\mathbf{x},\mathbf{y}) = \sqrt{(\mathbf{x}-\mathbf{y})^{\mathrm{T}}\mathbf{S}^{-1}(\mathbf{x}-\mathbf{y})}, \tag{2.3}$$

  where $\mathbf{S} = \mathrm{cov}[\mathcal{D}]$, $\mathcal{D}$ is the dataset. The metric defined in equation (2.3) is known as the Mahalanobis distance.

The Mahalanobis metric extends the Euclidean distance by incorporating data specific information. But a good metric should be even more flexible than that. Apart for the data set, we are also given a task to perform. We want our metric to extract and discriminate only between information that is relevant to our given task. Again, in the face recognition example we are not interested at all in the

[1]`http://matlabdatamining.blogspot.com/2006/11/mahalanobis-distance.html`



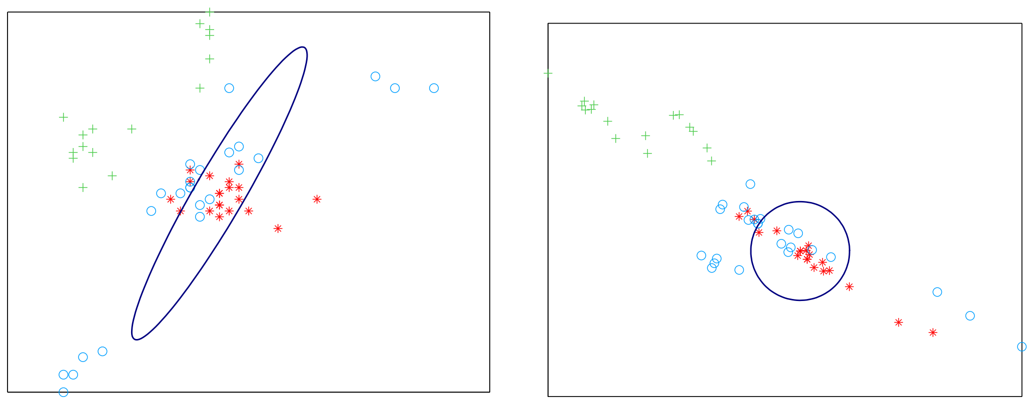
(a) Face recognition (b) Expression recogntion

Figure 2.1: On a given data set we might need to perform multiple tasks; in this example: face recognition and expression recognition. Euclidean metric is not optimal since we want different pairs of equivalence for the two tasks. A good distance metric should be task dependent. These images are an extract from Yale face database.

background colour so the importance of that feature should be down-weighted. In figure 2.1, we illustrate that for a given data set it is not optimal to use a fixed metric.

We can further generalize the result from equation (2.3). Instead of setting $\mathbf{S}$ as the covariance of the data, we let $\mathbf{S}$ be a different matrix. The problem of determining a suitable matrix $\mathbf{S}^{-1}$ is called Mahalanobis (or quadratic) distance metric learning. We note that $\mathbf{S}^{-1}$ ought to be positive semi-definite $\mathbf{x}^{\mathrm{T}}\mathbf{S}^{-1}\mathbf{x} \geq 0, \forall\, \mathbf{x}$. This ensures that the norm is a real value: $d(\mathbf{x},\mathbf{0}) \equiv \|\mathbf{x}\| = \sqrt{\mathbf{x}^{\mathrm{T}}\mathbf{S}^{-1}\mathbf{x}} \in \mathbb{R}$.

There is an interesting equivalence between a Mahalanobis-like metric and a linear transformation. Using Cholesky decomposition we can write $\mathbf{S}^{-1} = \mathbf{A}^{\mathrm{T}}\mathbf{A}$, which gives $\|\mathbf{x}\| = \sqrt{(\mathbf{A}\mathbf{x})^{\mathrm{T}}(\mathbf{A}\mathbf{x})}$, where $\mathbf{A}$ represents a linear transformation of the data. Hence, the problem of finding a suitable distance metric $\mathbf{S}$ is equivalent to the problem of finding a good linear transformation $\mathbf{A}$ and then applying Euclidean distance in the projected space (figure 2.2). We will discuss these two variants interchangeably and we will often parametrize our models in terms of $\mathbf{A}$.



(a) Mahalanobis metric in the original space (b) Euclidean metric in the projected space

Figure 2.2: Illustration of the equivalence between a Mahalanobis metric in the original data space and an Euclidean metric in the transformed data space. The metrics are denoted by an ellipse and, respectively, a circle whose contours indicate equal distance from the centre point to the rest of the points.

2.2 Related methods

The first attempts in metric learning were simple alterations of the Euclidean metric to improve kNN performance. For example, Mitchell (1997) suggests to weight each attribute of the data points such that the cross validation error is minimized. This technique is equivalent to stretching the axis of the relevant attributes and compressing the axis of the attributes that are less informative. The approach is similar to learning a diagonal metric, equation (2.2). Another method (Moore and Lee, 1994) is to eliminate the least relevant attributes: they are given a weight of zero. Albeit useful, these techniques are restrictive as the feature scaling does not reflect how the data covary.

Hastie and Tibshirani (1996) proposed discriminant adaptive nearest neighbours (DANN), a method that constructs a local metric for each particular query. The metric is chosen such that it provides the best discrimination amongst classes in the neighbourhood of the query point. This increases the already costly testing time of kNN. However, DANN method provides a non-linear metric if we take into account all the possible resulted local linear distance metrics. Even though non-linear transformations represent a very interesting domain, we will concentrate our attention on linear metrics as they are less prone to over-fitting and easier to learn and represent.

Xing et al. (2003) proposed learning a Mahalanobis-like distance metric for clustering in a semi-supervised context. There are specified pairs of similar data points and the algorithm finds the linear projection that minimizes the distance between these pairs without collapsing the whole data set to a single point. This approach is formulated as a convex optimization problem with constraints. The solution is attained using an iterative procedure, such as Newton-Raphson. This method assumes that the data set is unimodal and leverages the assumption-free kNN. Another drawback is the training time, because the iterative process is costly for large data sets.

Relevant component analysis (RCA; Bar-Hillel et al., 2003; Shental et al., 2002) is another method that makes use of the labels of the classes in a semi-supervised way to provide a distance metric. More precisely, RCA uses the so-called chunklet information. A chunklet contains points from the same class, but the class label is not known; data points that belong to the same chunklet have the same label, while data points from different chunklets do not necessarily belong to different classes. The main idea behind RCA is to find a linear transformation which "whitens" the data with respect to the averaged within-chunklet covariance matrix (Weinberger et al., 2006). Compared to the method of Xing et al. (2003), RCA has the advantage of presenting a closed form solution, but it has even stricter assumptions about the data: it considers that the data points in each class are normally distributed so they can be described using only second order statistics (Goldberger et al., 2004).

Large margin nearest neighbour (LMNN; Weinberger et al., 2006) is a metric learning method that was designed especially for kNN classification. LMNN can be regarded as the support vector machine (SVM) counter part of metric learning techniques: it brings together points from the same class trying to separate the classes by a certain margin. Also LMNN inherits ths convexity property and falls into the category of semi-definite programming. Weinberger and Saul (2008) worked on fast metric learning by introducing ball trees to speed up the computations. In (Weinberger and Saul, 2008), they extended LMNN to learn local metrics further improving its performance.

Metric learning is an ongoing research area in machine learning with an abundance of different techniques. For a good report that covers many related methods, the reader is referred to (Yang and Jin, 2006).

Chapter 3

Neighbourhood component analysis

This chapter can be regarded as a self-contained tutorial on neighbourhood component analysis (NCA; Goldberger et al., 2004). Much of the material is especially useful for the reader interested in applying the algorithm in practice. We start with a review of the NCA method (section 3.1). Next we recast the NCA model into a class conditional kernel density estimation problem (section 3.2). This new formulation will prove useful later when we want to alter the model in a principled way (section 4.6). Lastly, in section 3.3 we present the learnt lessons from our experience with NCA. Both section 3.2 and 3.3 offer new insights into the NCA method and they also bring together related ideas from previous work.

3.1 General presentation

Neighbourhood component analysis learns a Mahalanobis metric that improves the performance of k nearest neighbours (kNN). From a practical point of view, NCA is regarded as a desirable additional step before doing classification with kNN. It improves the classification accuracy and it is also provides good low-dimensional representation of the data.

Because the goal is to enhance the kNN performance, the first idea Goldberger et al. (2004) had was to maximize the leave one out cross validation performance with respect to a linear projection $\mathbf{A}$. The procedure can be described as follows: apply the linear transformation $\mathbf{A}$ to the whole data set, then take each point $\mathbf{Ax}_i$ and classify it using kNN on the transformed data set $\{\mathbf{Ax}_j\}_{j=1}^{N}$. The matrix $\mathbf{A}$ that achieves the highest number of correct classifications will be used for testing. However, finding the optimal $\mathbf{A}$ is not easy. Any objective function based on the

k nearest neighbours is piecewise constant and discontinuous and, hence, hard to optimize. The reason is that there does not exist an exact correlation between $\mathbf{A}$ and the neighbours of a given point: a small perturbation in $\mathbf{A}$ might cause strong changes or, conversely, it might leave the neighbours unchanged.

The authors' solution lies in the concept of *stochastic* nearest neighbours. Remember that in the classical 1NN scenario, a query point gets the label of the closest point. In the stochastic nearest neighbour case, the query point inherits the label of a neighbour with a probability that is inverse proportional with the distance. The stochastic function is reminiscent of the generalized logistic function or of the softmax activation used for neural networks. Let p_{ij} denote the probability that the point j is selected as the nearest neighbour of the point i. The value of p_{ij} is given by:

$$p_{ij} = \frac{\exp(-d_{ij}^2)}{\sum_{\substack{k=1 \\ k \neq i}}^{N} \exp(-d_{ik}^2)}, \tag{3.1}$$

where d_{ij} represents the distance between point i and point j in the projected space, i.e. $d_{ij} = d(\mathbf{A}\mathbf{x}_i; \mathbf{A}\mathbf{x}_j) = (\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\mathbf{A}(\mathbf{x}_i - \mathbf{x}_j)$. Also we have to set $p_{ii} = 0$: point i cannot pick itself as the nearest neighbour since we assume its label is not known.

Now we can construct a continuous objective function using the stochastic assignments p_{ij} which are differentiable with respect to $\mathbf{A}$. A suitable quantity to optimize is the probability of each point of getting correctly classified. Point i is correctly classified when it is selected by a point j that has the same label as i:

$$p_i = \sum_{j \in c_i} p_{ij}. \tag{3.2}$$

The objective function considers each point in the data set and incorporates their probability of belonging to the true class:

$$\begin{aligned} f(\mathbf{A}) &= \sum_{i=1}^{N} p_i \\ &= \sum_{i=1}^{N} \sum_{j \in c_i} \frac{\exp(-d_{ij}^2)}{\sum_{k \neq i} \exp(-d_{ik}^2)}. \end{aligned} \tag{3.3}$$

The score given by the objective function can be interpreted as the expected number of the correctly classified points.

To obtain the best linear projection $\mathbf{A}$, we maximise $f(\mathbf{A})$ using an iterative gradient based solver such as gradient ascent, conjugate gradients or delta-bar-delta (subsection 3.3.1). For these methods, we can use the analytical expression of the gradient. If we differentiate with respect to $\mathbf{A}$, we obtain:

$$\frac{\partial f}{\partial \mathbf{A}} = 2\mathbf{A} \sum_{i=1}^{N} \left(p_i \sum_{k=1}^{N} p_{ik} \mathbf{x}_{ik} \mathbf{x}_{ik}^{\mathrm{T}} - \sum_{j \in c_i} p_{ij} \mathbf{x}_{ij} \mathbf{x}_{ij}^{\mathrm{T}} \right), \tag{3.4}$$

where $\mathbf{x}_{ij} = \mathbf{x}_i - \mathbf{x}_j$. The interested reader can find the derivation in the appendix.

At test time, we use the learnt matrix $\mathbf{A}$ to transform the points before doing classification with kNN. Given a query point $\mathbf{x}^*$, we assign it the label c_j of the closest point j in the projected space, where $j = \operatorname{argmin}_i d(\mathbf{A}\mathbf{x}^*, \mathbf{A}\mathbf{x}_i)$.

Further remarks

There are some interesting observations that can be made about NCA. It is useful to note that NCA model is still valid even if we set the matrix $\mathbf{A}$ to be rectangular, i.e. $\mathbf{A} \in \mathbb{R}^{d \times D}$, with $d < D$. In this case the linear transformation $\mathbf{A}$ is equivalent to a low dimensional projection. Restricting $\mathbf{A}$ to be non-square gives rise to a couple of advantages. Firstly, the optimization process is less prone to overfitting since a rectangular matrix has fewer parameters than a square one. Secondly, the low dimensional representation of the data set reduces the memory requirements and the computational cost at test time. In this thesis we concentrate mostly on low-rank metrics partly motivated by the above reasons, but also because many techniques that are subsequently developed (chapter 4) work best in low dimensions.

We need to underline that NCA's objective function is not convex. The first consequence is that the parameter space is peppered with local optima and care must be taken to avoid poor solutions. Factors such as the choice of initialization or the optimization procedure affect the quality of the final result. Section 3.3 discusses these practical issues in detail. The other side of non-convexity is that it allows NCA to cope well with data sets that have multi-modal classes. This is an important advantage over classical methods such as principal component analysis, linear discriminant analysis or relevant component analysis. All of these assume that the data is normally distributed which harms the classification performance on complicated data sets.

Another important characteristic of NCA is its computational cost. For each point, we need to calculate the distances to all the other points. So the number of operations is quadratic in the number of data points. The evaluation of the objective function is done in two steps. First we find the point projections $\{\mathbf{A}\mathbf{x}_i\}_{i=1}^N$; this operation has $\mathcal{O}(dDN)$ cost. Next we compute all the distances d_{ij} in the low dimensional space; this is done in $\mathcal{O}(dN^2)$ flops. The total cost is then $\mathcal{O}(dDN + dN^2)$. Another way of evaluating the function f is to parametrize it in terms of the Mahalanobis metric $\mathbf{S} = \mathbf{A}^{\mathrm{T}}\mathbf{A}$. Then we can compute the distances in the D-dimensional space in $\mathcal{O}(DN^2)$ flops. This option is slower than the previous case for $d < D$ and $D \ll N$.

Since the optimization is based on the gradient, we are also interested in its cost. The evaluation of gradient given in the original paper and in equation (3.4) scales in D^2N^2. We improve this by "pushing" the matrix $\mathbf{A}$ into the sum as in (Singh-Miller, 2010):

$$\frac{\partial f}{\partial \mathbf{A}} = 2\sum_{i=1}^{N}\left(p_i \sum_{k=1}^{N} p_{ik}(\mathbf{A}\mathbf{x}_{ik})\mathbf{x}_{ik}^{\mathrm{T}} - \sum_{j\in c_i} p_{ij}(\mathbf{A}\mathbf{x}_{ij})\mathbf{x}_{ij}^{\mathrm{T}}\right), \tag{3.5}$$

If we consider that the projections $\mathbf{A}\mathbf{x}_i$ were already calculated for the function evaluation, the gradient has a cost of $\mathcal{O}(dDN^2)$. However, in theory, evaluating the gradient should have the same complexity as evaluating the objective function. Automatic differentiation (AD; Rall, 1981) is a technique that achieve this theoretical result. AD uses the chain rule to derivative the objective function and split the gradient into elementary functions that can be readily computed. So instead of evaluating the complicated analytical gradient, we work on a series of cheap operations. This method is very useful in practice and there exist libraries for this in most programming languages. In our implementation we used the analytical differentiation, because we were not aware of AD until recently.

The computational drawback represents an important setback in applying NCA in practice. For example, Weinberger and Tesauro (2007) reported that the method ran out of memory on large data sets and Weizman and Goldberger (2007) used only 10% of an hyper-spectral data set to train the NCA classifier. We present a number of solutions to this problem in chapter 4.

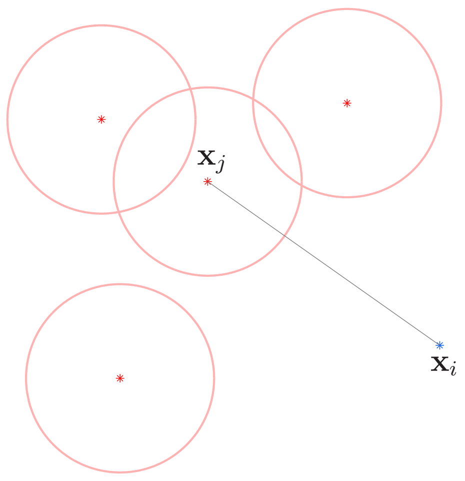


Figure 3.1: NCA as a class-conditional kernel density estimation model. The figure represents a schematic illustration of a mixture of Gaussians. The red circles denote isotropic Gaussians. These are centred onto the points that belong to a certain class. A point i is generated by either of the components of the class with a probability inverse proportional with the distance.

3.2 Class-conditional kernel density estimation interpretation

In this section we recast NCA into a class-conditional kernel density estimation framework. This interpretation will allow us to understand the assumptions behind NCA method. After this we are in position to alter the model in a suitable way that is efficient for computations (sections 4.4 and 4.5). Similar ideas were previously presented by Weinberger and Tesauro (2007) and Singh-Miller (2010), but the following were derived independently and they offer different insights. Our following interpretation was inspired by the probabilistic kNN presented by Barber (2011).

We start with the basic assumption that each class can be modelled by a mixture of Gaussians. For each of the N_c data points in class c we consider a Gaussian "bump" centred around it (figure 3.1). From a generative perspective, we can imagine that each point j can generate a point i with a probability given by an isotropic normal distribution with variance σ^2:

$$p(\mathbf{x}_i|\mathbf{x}_j) = \mathcal{N}(\mathbf{x}_i|\mathbf{x}_j, \sigma^2 \mathrm{I}_D) \tag{3.6}$$

$$= \frac{1}{(2\pi)^{D/2}\sigma^D} \exp\left\{-\frac{1}{2\sigma^2}(\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}}(\mathbf{x}_i - \mathbf{x}_j)\right\}. \tag{3.7}$$

By changing the position of the points through a linear transformation $\mathbf{A}$, the probability changes as follows:

$$p(\mathbf{A}\mathbf{x}_i|\mathbf{A}\mathbf{x}_j) \propto \exp\left\{-\frac{1}{2\sigma^2}(\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\mathbf{A}(\mathbf{x}_i - \mathbf{x}_j)\right\}. \tag{3.8}$$

We note that this is similar to the p_{ij} from NCA. Both $p(\mathbf{A}\mathbf{x}_i|\mathbf{A}\mathbf{x}_j)$ and p_{ij} are directly proportional with the same quantity.

Using the mixture of Gaussians assumption, we have that the probability of a point of being generated by class c is equal to the sum of all Gaussians in class c:

$$p(\mathbf{x}_i|c) = \frac{1}{N_c}\sum_{\mathbf{x}_j \in c} p(\mathbf{x}_i|\mathbf{x}_j) \tag{3.9}$$

$$= \frac{1}{N_c}\sum_{\mathbf{x}_j \in c} \mathcal{N}(\mathbf{x}_i|\mathbf{x}_j, \mathrm{I}_D). \tag{3.10}$$

However, we are interested in the inverse probability, given a point $\mathbf{x}_i$ what is the probability of $\mathbf{x}_i$ belonging to class c. We obtain the expression for $p(c|\mathbf{x}_i)$ using Bayes' theorem:

$$p(c|\mathbf{x}_i) = \frac{p(\mathbf{x}_i|c)p(c)}{p(c|\mathbf{x}_i)} = \frac{p(\mathbf{x}_i|c)p(c)}{\sum_c p(\mathbf{x}_i|c)p(c)}. \tag{3.11}$$

Now if further consider the classes to be equal probable (which is a reasonable assumption if we have no a priori information) we arrive at result that resembles the expression of p_i (equation (3.2)):

$$p(c|\mathbf{A}\mathbf{x}_i) = \frac{\frac{1}{N_c}\sum_{\mathbf{x}_j \in c} \exp\left\{-\frac{1}{2\sigma^2}(\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\mathbf{A}(\mathbf{x}_i - \mathbf{x}_j)\right\}}{\frac{1}{N_{c'}}\sum_{c'}\sum_{\mathbf{x}_k \in c'} \exp\left\{-\frac{1}{2\sigma^2}(\mathbf{x}_i - \mathbf{x}_k)^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\mathbf{A}(\mathbf{x}_i - \mathbf{x}_k)\right\}} \tag{3.12}$$

We are interested in predicting the correct class of the point i. So we try to find that linear transformation $\mathbf{A}$ that maximises the class conditional probability of this point $p(c_i|\mathbf{A}\mathbf{x}_i)$ to its true class c_i.

$$f(\mathbf{A}) = \sum_i p(c_i|\mathbf{A}\mathbf{x}_i). \tag{3.13}$$

.

As in section 3.1, we can optimize the function using the gradient information. The gradient is the following:

$$\frac{\partial f}{\partial \mathbf{A}} = \sum_i \left\{ \frac{\frac{\partial p(\mathbf{A}\mathbf{x}_i|c)}{\partial \mathbf{A}} p(c)}{\sum_c p(\mathbf{x}_i|c)p(c)} - \underbrace{\frac{p(\mathbf{A}\mathbf{x}_i|c)p(c)}{\sum_c p(\mathbf{A}\mathbf{x}_i|c)p(c)}}_{p(c|\mathbf{A}\mathbf{x}_i)} \frac{\sum_c \frac{\partial p(\mathbf{A}\mathbf{x}_i|c)}{\partial \mathbf{A}} p(c)}{\sum_c p(\mathbf{A}\mathbf{x}_i|c)p(c)} \right\}. \tag{3.14}$$

The main advantage of the above formulation is that we can construct different models simply by modifying the class-conditional probability (equation (3.9)). Once $p(\mathbf{x}_i|c)$ is altered in a suitable way, we use Bayes rule in conjuction equations (3.13) and (3.14) to get the new function and its corresponding gradient. We optimize for parameter $\mathbf{A}$ exactly in the same way as for classical NCA.

Examples of this technique are in sections 4.5 and 4.6. There the alterations allowed considerable speed-ups as we shall see in chapter 5.

Other benefits of CC-KDE model are the possibility to incorporate prior information about the class distributions $p(c)$ and the possibility to classify a point $\mathbf{x}^*$ using $p(c|\mathbf{A}\mathbf{x}^*)$ (subsection 3.3.5).

3.3 Practical notes

While NCA is not that hard to implement (appendix A.1), there is needed certain care in order to achieve good solutions. In this section we try to answer some of the questions that can be raised when implementing NCA.

3.3.1 Optimization methods

The function $f(\mathbf{A})$, equation (3.3), can be maximised using any gradient based optimizer. We considered two popular approaches for our implementation: gradient ascent and conjugate gradients. These are only briefly presented here. The interested reader is pointed to (Bishop, 1995).

Gradient ascent

Gradient ascent is one of the simplest optimisation methods. It is an iterative algorithm that aims to find the maximum of a function by following the gradient direction at each step. The entire procedure is summarized by algorithm 3.1.

Besides this version, an alternative is to compute the gradient on a fewer points and not on the entire data set. This variant is known as stochastic gradient ascent and it has the advantage to be much faster than the batch version. Different alterations of the stochastic gradient ascent were tried in chapter 4 to reduce the computational cost.

Algorithm 3.1 Gradient ascent (batch version)

```
Require: Data set $\mathcal{D} = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$.
1: Get initial $\mathbf{A}_0$
2: repeat
3:     Update parameter: $\mathbf{A}_{t+1} \leftarrow \mathbf{A}_t + \eta \frac{\partial f(\mathbf{A}_t, \mathcal{D})}{\partial \mathbf{A}}$
4:     Update learning rate $\eta$
5:     $t \leftarrow t + 1$
6: until convergence
7: return $\mathbf{A}_t$.
```

For the algorithm 3.1, there are three aspects we need to carefully consider:

Initialization. The algorithm starts the search in the parameter space from the an initial parameter $\mathbf{A}_0$. Different values for $\mathbf{A}_0$ will give different final solutions. In the NCA context, initialization is related to finding a good initial linear transformation; this is discussed separately in subsection 3.3.2. For the moment, we can assume the values of $\mathbf{A}_0$ are randomly chosen.

Learning rate. The parameter η is called *step size* or, in neural networks related literature, it also known as *learning rate*. As name suggests, η controls how much we move the parameters in the gradient direction.

The learning rate can be either fixed or adaptive. For the first case, choosing the correct value for η is critical. If η is set too large, the algorithm diverges. On the other hand, a small η results in slow convergence.

Usually, a variable learning rate η is preferred since it is more flexible. The "bold driver" (Vogl et al., 1988) is a reasonable heuristic for automatically modifying η during training. The idea is to gradually increase the learning rate after each iteration as long as the objective function keeps improving. Then we go back to the previous position and start decreasing the learning rate until a new increase in the function is found. The common way to increase the step size η is by multiplying it with a constant $\rho > 1$, usually $\rho = 1.1$. To decrease the step size, we can multiple it with a constant $\sigma < 1$, for example $\sigma = 0.5$.

Another way is to change the learning rate η using a decreasing rule proportional to the number of current iteration: $\eta = \frac{\eta_0}{t+t_0}$. Now we have two parameters instead of one. Intuitively, η_0 can be regarded as the initial learning rate and t_0 as the number of iterations after which η_0 starts to drop off. Using a learning rate of such form is especially motivated in the stochastic gradient case. If $\eta \propto 1/t$, the algorithm converges to the true minimum as t goes to infinity. However, in practice the convergence is very slow; so setting the constants η_0 and t_0 is important for reaching a good solution in a short time. Leon Bottou[1] recommends to set η_0 to be the regularization value and then select t_0 such that the resulted update will modify $\mathbf{A}$ with a sensible amount. Since we did not use regularization, we followed some of the tricks presented in (LeCun et al., 1998). We fixed the value for η_0 and we did an exponential search for t_0 using a cross validation set.

Convergence. The gradient ascent algorithm can be stopped either when we hit a maximum number of iterations or when the learning rate η falls below a certain threshold. Also we can look at the variation in the objective function. If the value of the objective function stops improving we can halt the algorithm. Another popular choice to check convergence is "early stopping". We monitor the performance of the new $\mathbf{A}$ on a cross validation set. If the objective function did not increase for more than a preset number of iterations we stop and return to the previous best solution. This solution prevents overfitting since we stop when we achieve a minimum on a test error and not on the training error. We illustrate this in figure 3.2.

Gradient ascent has known convergence problems and determining good learning rates is often difficult. But it has the advantage of being quite flexible, especially in the stochastic optimization scenario, where we can select any noisy estimate of the gradient in order to replace the total gradient.

[1]Oral communication: `http://videolectures.net/mmdss07_bottou_lume/`

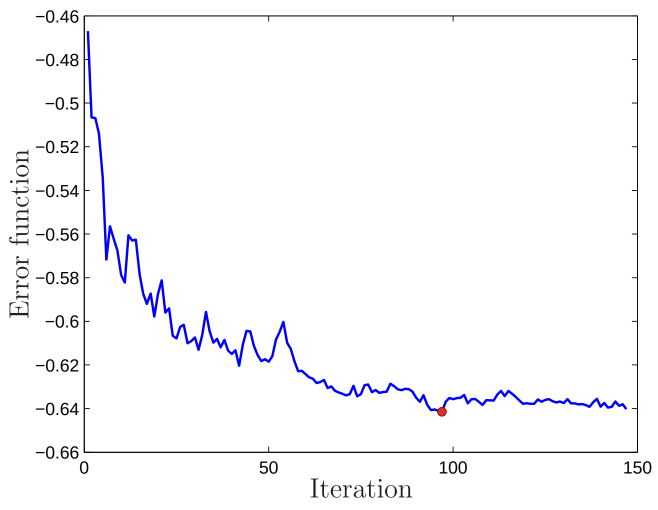


Figure 3.2: This figure illustrates the early stopping procedure. During training we keep a small part of the data set apart, for cross-validation. On this cross-validation data set we compute the error function at each iteration. When the error function stops decreasing we stop the learning procedure and return to the parameter that achieved the best value. This figure resulted while applying NCA on the `mnist` data set. The best value was obtained at iteration 98, as indicated by the red dot.

Conjugate gradients

Conjugate gradient (CG) method solves some of the convergence issues. Instead of selecting the search directions to be the gradient directions, CG method selects the next direction depending on both the previous directions and on the curvature of surface. This permits this method to reach the maximum of a quadratic energy function in only two iterations. Also CG eliminates the manual set-up for the learning rates. As Bishop (1995) notes, the CG can be regarded as a more complex version of gradient ascent that automatically chooses the parameters. The details of the conjugate gradient method are out of the scope of this thesis; a gentle introduction into the subject is given by Shewchuk (1994).

In practice it is considerably easier to use an existing implementation of CG (for example, Carl E. Rasmussen's `minimize.m`[2]) than trying to find the optimal parameters for the learning rate for gradient ascent. For small and medium-sized data sets, "bold driver" heuristic gives comparable scores to the more fancier CG method (tables B.1 and B.2).

[2]Source code can be found at the following URL: `http://www.gaussianprocess.org/gpml/code/matlab/util/minimize.m`

3.3.2 Initialization

Initialization is important because the function $f(\mathbf{A})$ is not convex. The quality of the final solution relies on the starting point of the optimisation algorithm. A general rule of thumb is to try multiple initial seeds and select that final $\mathbf{A}$ that gives the highest score.

We have already mentioned random initialization in subsection 3.3.1. Beside this, we can try linear transformations that are cheap to compute. Such examples include principal component analysis (PCA; Pearson, 1901), linear discriminant analysis (LDA; Fisher, 1936) and relevant component analysis (RCA; Bar-Hillel et al., 2003). For completeness, we give here the equations and also include further notes:

- PCA finds an orthogonal linear transformation of the data. This is obtained by computing the eigendecomposition of the outer covariance matrix:

$$\mathbf{S} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}-\boldsymbol{\mu})(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}. \tag{3.15}$$

- LDA finds a linear transformation $\mathbf{A}$ by maximizing the variance between classes $\mathbf{S}_B$ relative to the amount of within-class variance $\mathbf{S}_W$:

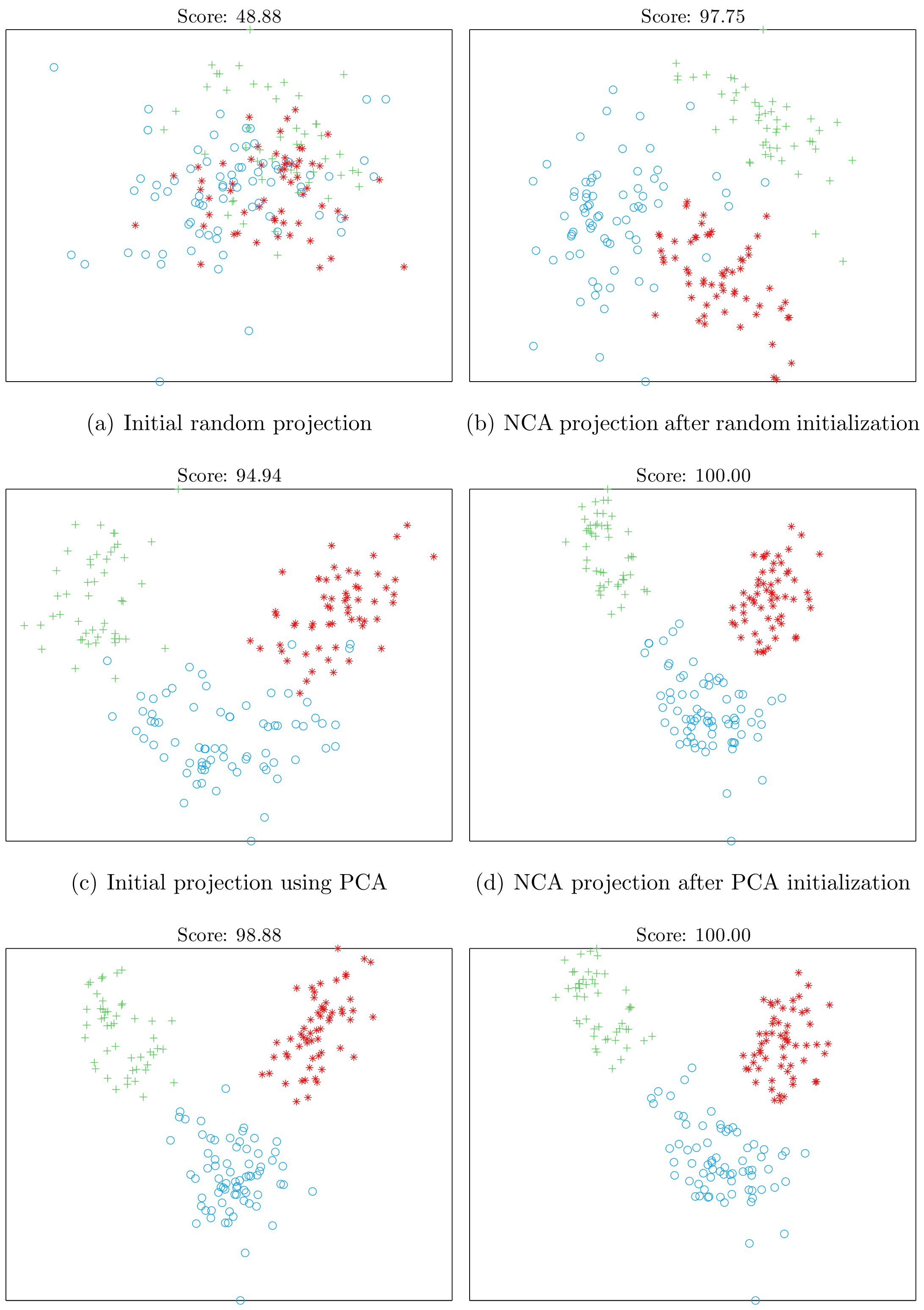
$$\mathbf{S}_B = \frac{1}{C}\sum_{c=1}^{C}\boldsymbol{\mu}_c\boldsymbol{\mu}_c^{\mathrm{T}} \tag{3.16}$$

$$\mathbf{S}_W = \frac{1}{N}\sum_{c=1}^{C}\sum_{i\in c}(\mathbf{x}_i-\boldsymbol{\mu}_c)(\mathbf{x}_i-\boldsymbol{\mu}_c)^{\mathrm{T}}. \tag{3.17}$$

  The projection matrix $\mathbf{A}$ that achieves this maximization consists of the eigenvectors of $\mathbf{S}_W^{-1}\mathbf{S}_B$.

  Unlike PCA, LDA makes use of the class labels and this usually guarantees a better initial projection.

- RCA finds a linear transformation $\mathbf{A}$ that decorelates the points within a chunklet, i.e. it makes the within-chunklet covariance to the identity matrix. Because for NCA we restrict ourselves to fully labelled data, the within-chunklet covariance is the within-class covariance $\mathbf{S}_W$, equation (3.17). The "whitening" transformation $\mathbf{A}$ is then $\mathbf{A} = \mathbf{S}_W^{-1/2}$.



(a) Initial random projection

(b) NCA projection after random initialization

(c) Initial projection using PCA

(d) NCA projection after PCA initialization

(e) Initial projection using LDA

(f) NCA projection after LDA initialization

Figure 3.3: Results for different initializations (random, PCA and LDA) on `wine` data set. The images on the left side represent the projections of the data set using the initial $\mathbf{A}$. The images on the right side are data set projection with the final $\mathbf{A}$. Above each figure there is presented the LOOCV score which is normalized to 100.

If the projection is full-rank, $\mathbf{A} \in \mathbb{R}^{D\times D}$, other obvious initializations are the identity matrix $\mathbf{A} = \mathrm{I}_D$ and the Mahalanobis linear transformation $\mathbf{A} = \mathbf{S}^{-1/2}$.

If we want to learn a low-rank projection, $\mathbf{A} \in \mathbb{R}^{d\times D}, d < D$, then we can still use the eigendecomposition based methods. We construct $\mathbf{A}$ using only the top d most discriminative eigenvectors, *i.e.*, those eigenvectors that have the highest eigenvalues associated.

From our experiments, we conclude that a good initialization reflects in a good solution and a better convergence; this benefits are more evident on large data sets. As advertised in (Butman and Goldberger, 2008), we found RCA to work the best. Figure 3.3 depicts initialization effects on a small data set. More illustrations are in the appendix B.1. Occasionally random initialization gives better final scores than the other techniques (figure B.2).

3.3.3 Numerical issues

Numerical problems can easily appear when computing the soft assignments p_i. If a point $\mathbf{x}_i$ is far away from the rest of the points, the stochastic probabilities $p_{ij}, \forall j$, are all 0 in numerical precision. Consequently, the result p_i is undetermined: $\frac{0}{0}$. To make an idea of how far $\mathbf{x}_i$ has to be for this to happen, let us consider an example in MATLAB. The answer to `exp(-d^2)` is `0` whenever `d` exceeds `30` units. This is problematic in practice, since distances larger than 30 often appear. Some common cases include data sets that contain outliers or data sets that present a large scale.

The large scale effect can be partially alleviated if we initialize $\mathbf{A}$ with small values. This is idea was used by Laurens van der Maaten in his implementation: `A = 0.01*randn(d,D)`. However, this does not guarantee to compensate the scale variation for any data set.

A more robust solution is to normalize the data, *i.e.*, centre and making it unit variance:

$$x_{ij} \leftarrow \frac{x_{ij} - \mu_j}{\sigma_j}, \quad i = \{1, \cdots, N\},\ j = \{1, \cdots, D\}. \tag{3.18}$$

In this case, we have to store the initial mean and variance of the data and, at test time, transform the test data accordingly: subtract the mean and scale it using the variance coefficients. The variance scaling can be regarded as a linear transformation. We can combine this transformation with the learnt transformation

A and get:

$$\mathbf{A}_{\text{total}} \leftarrow \mathbf{A} \cdot \begin{pmatrix} \sigma_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_D \end{pmatrix}. \tag{3.19}$$

In general, data normalization avoids numerical issues for the first iterations. But during training, the scale of **A** increases and data points can be "thrown" away. We adopt a rather simplistic approach to avoid any numerical problems: replace p_i with a very small value whenever we are in the $\frac{0}{0}$ case, as in Laurens van der Maaten implementation. In MATLAB, this is done using the following command `max(p_i,eps)`.

A more rigorous way of dealing with this by multiplying both the numerator and denominator of p_i with a certain quantity $\exp(L)$:

$$p_i = \frac{\sum_{j \in c_i} \exp(L - d_{ij}^2)}{\sum_{k \neq i} \exp(L - d_{ik}^2)}, \tag{3.20}$$

where $L = \min_{k \neq i} d_{ik}^2$. This value of L ensures that at least one term in the denominator does not undeflow.

In our implementation, we preferred the first trick because we can vectorize the code. In `Matlab`, it is important to have vectorized code to achieve better speed-ups. For a C implementation, we recommend the second option.

3.3.4 Regularization

Although the original paper (Goldberger et al., 2004) claimed that there were no problems with overfitting, we observed that NCA objective function has the tendency to increase the scale of the linear projection **A**. Butman and Goldberger (2008) pointed out that this is especially problematic for data sets whose size N is significantly smaller than the dimensionality D. In such a case, the linear transformation **A** can be chosen to project each point $\mathbf{x}_i$ to a pre-specified a location $\mathbf{y}_i$. The values of **A** are obtained by solving the linear system:

$$\mathbf{A}\mathbf{x}_i = \mathbf{y}_i, \quad i = \{1, \cdots, N\}. \tag{3.21}$$

Because the system has more equations dN than unknowns dD, it has exact solutions. If we set the same positions $\mathbf{y}_i$ for points in the same class, we can virtually increase the scale of the projection to infinity and get an error-free classification. This degeneracy can be corrected with the help of regularization. We alter the objective function by subtracting a regularization term proportional with the magnitude of $\mathbf{A}$. This gives:

$$g(\mathbf{A}) = f(\mathbf{A}) - \lambda \sum_{i=1}^{d} \sum_{j=1}^{D} A_{ij}^2. \tag{3.22}$$

$$\frac{\partial g}{\partial \mathbf{A}} = \frac{\partial f}{\partial \mathbf{A}} - 2\lambda \mathbf{A}, \tag{3.23}$$

where λ is a positive constant that is tuned via cross-validation.

Another effect of a large scale linear projection $\mathbf{A}$ is the fact that only the nearest neighbour is considered for each point. This is not usually the optimal thing to do, so regularization is also used to prevent this (Singh-Miller, 2010).

In our implementation, we use "early stopping" technique which has a similar effect to regularization.

3.3.5 Doing classification

For testing, Goldberger et al. (2004) considered only kNN decision rule. Since we are optimizing a particular function $f(\mathbf{A})$, another sensible tool for classification is an NCA based function. If we are using the probabilistic interpretation (section 3.2) a query point $\mathbf{x}_i$ is labelled with most probable class c:

$$c = \operatorname{argmax}_c p(c|\mathbf{x}_i). \tag{3.24}$$

This can be re-written in NCA specific notation as:

$$c = \operatorname{argmax}_c \frac{\sum_{j \in c} p_{ij}}{\sum_k p_{ik}}. \tag{3.25}$$

In our experiments, 1NN and the NCA classification rule described by equation (3.25) give very similar results if we train $\mathbf{A}$ om the un-regularized function $f(\mathbf{A})$. When we used early stopping, the NCA classification rule yielded better performances (chapter 5). In this case, kNN with $k > 1$ should also perform better than simple 1NN.

3.3.6 Dimensionality annealing

Dimensionality annealing is an approach to learning a low-rank metric in an way that avoids local optima (Murray and Hinton, oral communication). We start with a full rank projection **A** and gradually reduce the rank by introducing regularization terms on the lines of **A**. Compared with the classical regularization procedure, subsection 3.3.4, here we use a regularization term on each of the dimension d. The new objective function and its gradient are given by the following equations:

$$g(\mathbf{A}) = f(\mathbf{A}) - \sum_{i=1}^{d} \lambda_i \sum_{j=1}^{D} A_{ij}^2 \tag{3.26}$$

$$\frac{\partial g}{\partial \mathbf{A}} = \frac{\partial f}{\partial \mathbf{A}} - 2 \begin{pmatrix} \lambda_1 A_{11} & \cdots & \lambda_1 A_{1D} \\ \vdots & \ddots & \vdots \\ \lambda_d A_{d1} & \cdots & \lambda_d A_{dD} \end{pmatrix}. \tag{3.27}$$

A large value of λ_d will impose a small magnitude of **A** on dimension d. We increase λ_d slowly; this permits for the rest of the values of **A** to readjust. We clarify these steps in algorithm 3.2.

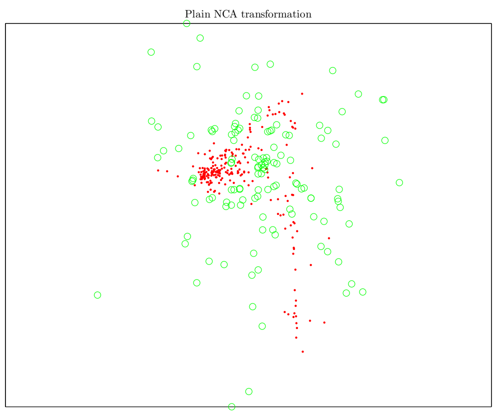
Algorithm 3.2 Dimensionality annealing

```
Require: Data set $\mathcal{D} = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$, initial linear transformation $\mathbf{A} \in \mathbb{R}^{D \times D}$
    and low dimension $d$.
1: for $i = 1, \cdots, D - d$ do
2:     Select dimension $d$ to anneal
3:     for $j = 1, \ldots, P$ do
4:         Increase regularization coefficient $\lambda_d \leftarrow \lambda_d + \Delta\lambda$
5:         Optimize function $g$: $\mathbf{A} = \mathbf{A} + \eta \frac{\partial g(\mathbf{A}, \mathcal{D}, \boldsymbol{\lambda})}{\partial \mathbf{A}}$
6:     end for
7: end for
```

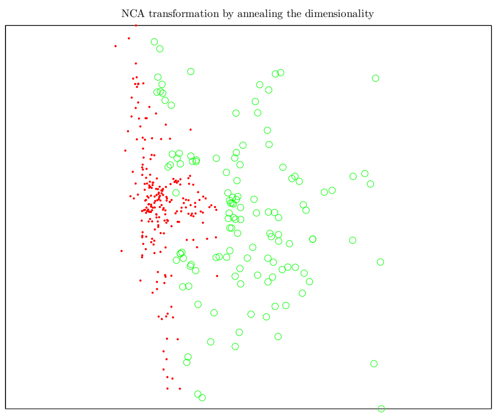
There are several notes to be made. There are alternatives for selecting the dimension d we want to anneal, step 2 of algorithm 3.2. We can choose d to be the direction that has:

- minimum variance in our data set $\{\mathbf{A}\mathbf{x}_i\}_{i=1}^{N}$
- minimum variance in the projected data set $\{\mathbf{A}\mathbf{x}_i\}_{i=1}^{N}$
- smallest magnitude in **A**.

The function optimization step can be done either by gradient ascent or conjugate gradients. It is possible to do conjugate gradients for a small number of iterations, typically 2 or 3.



(a) Simple NCA transformation



(b) NCA with dimensionality annealing projection

Figure 3.4: Example of applying the dimensionality annealing procedure to `ionosphere` data set. The method using dimensionality annealing is more visually appealing than the classical approach.

After a dimension is annealed, we reach the end of the inner for loop, we can remove that particular dimension; for example set the elements on the d row equal to 0.

A further idea would be to run conjugate gradients until convergence initialized with low dimensional $\mathbf{A}$ returned by algorithm 3.2.