Fast low-rank metric learning

Dan-Theodor Oneaţă

Master of Science
Artificial Intelligence
School of Informatics
University of Edinburgh
2011

Abstract

This doctoral thesis will present the results of my work into the reanimation of lifeless human tissues.

Acknowledgements

Many thanks to my mummy for the numerous packed lunches; and of course to Igor, my faithful lab assistant. DP IM.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Dan-Theodor Oneaţă*)

Table of Contents

Chapter 1

Introduction

k Nearest Neighbours (kNN) is one of the oldest and simplest classification methods. It has its origins in an unpublished technical report by **?** and since then it became standard textbook material (Russell et al., 1996; Mitchell, 1997; Bishop, 2006). In the last 50 years, kNN was present in most of the machine learning related fields (pattern recognition, statistical classification, data mining, information retrieval, data compression) and it plays a role in many attractive applications (e.g., face recognition, plagiarism detection, vector quantization).

The idea behind kNN is intuitive and straightforward: classify a given point according to a majority vote of its neighbours; the selected class is the one that is the most represented amongst the k nearest neighbours. This is easy to implement and it usually represents a good way to approach new problems or data sets. Despite its simplicity kNN is still a powerful tool, performing surprisingly well in practice, as most of the simple classifiers (Holte, 1993).

Yet there are also other characteristics that make kNN an interesting method. First of all, kNN makes no assumptions about the underlying structure of the data, but lets the data "speak for themselves". This means that no a priori knowledge is needed beforehand and, more importantly, the accuracy increases with the number of points in the data set (in fact, it approaches Bayes optimality as the cardinality of the training set approaches infinity and k is sufficiently large; Cover and Hart, 1967). Secondly, kNN is able to represent complex functions with non-linear decision boundaries by using only simple local approximations (this behaviour is hardly captured by any parametric method). Lastly, kNN operates in a "lazy" fashion: the training data is just stored and their use is delayed until the testing. The quasi-inexistent training allows to easily add new

training examples.

On the other hand, kNN has some drawbacks that influence both the computational performance and also the quality of its predictions. Firstly, because it has to memorise all the exemplars, the storage requirements are directly proportional with the number of instances in the training set. Furthermore, a serious practical disadvantage is represented by the fact that all the computations are done at testing. The cost for this is also linear in the cardinality of the training set and it is often prohibitive for large data sets. However, more important are the issues related to the accuracy. On one hand, there is not clear how we should choose a dissimilarity metric and how notions as "close"/"far" are defined for our data set. This is non-trivial and usually the standard Euclidean metric is not satisfactory (see Subsection **??**, for a more detailed discussion). On the other hand, there have been raised concerns with the usefulness of Nearest Neighbours (NN) methods for high dimensional data (Beyer et al., 1999; Hinneburg et al., 2000). The curse of dimensionality arises for kNN, because the distances become indiscernible for many dimensions; alternatively formulated, for a given distribution the maximum and the minimum distance between points become equal in the limit of dimensions.

All these problems are even more acute nowadays when we have to operate on huge sets of data with many attributes (e.g., images, videos, DNA sequences, etc.). There is an entire literature that tries to come up with possible solutions (some of the most prominent papers are discussed in Section **??**). One elegant answer is provided by Goldberger et al. (2004). They proposed a new method, Neighbourhood Component Analysis (NCA), that copes with the drawbacks in a unified manner by learning a low-rank metric. This reduces both the storage and the computational cost, because the algorithm uses the data set projected into a lower subspace, with fewer attributes needed. Also the accuracy is improved, because the label information is used for constructing a proper metric that selects the relevant attributes. However, NCA introduces a consistent training time, because we now have an objective function whose gradient is quadratic in the number of data points that is optimized using iterative methods.

The main aim of this project is to see whether NCA's training and testing time can be reduced without significant loses in accuracy. We will try to achieve this by making use of k-d trees (a space partitioning structure; Bentley 1975). These have been successfully applied for speeding up kNN at query time (Friedman et al.,

1977) and we will use them in a similar manner for NCA's testing operations. For training, we will apply k-d trees in a slightly different way (as in Deng and Moore, 1995); as k-d trees organize the space, they can be used to group together near points and quickly compute approximations of the gradient and objective function at each iteration. This idea was also followed by Weinberger and Saul (2009) in accelerating a different distance metric technique, Large Margin Nearest Neighbour (LMNN). As an alternative approach, we could interpret NCA as a class-conditional kernel-density estimate and then use different types of kernels instead of the Gaussian ones; we will focus especially on kernels with compact support, because they do not introduce approximation errors and computations can be done more efficiently than in the previous case. If time permits, it would be interesting to experiment with different tree structures, such as ball trees (Omohundro, 1989; Moore, 2000) (which can perform well in higher dimensional spaces) or dual-trees (Gray and Moore, 2003) (a faster representation of the typcal k-d tree). Also we could try this approach on other distance metric learning methods, with different objective functions (e.g., Xing et al., 2003).

Chapter 2

Background

2.1 Theoretical background

A distance metric represents a function or a mapping from a pair of inputs to a scalar proportional to the dissimilarity of the inputs. Additionally, in order to be a proper distance, the given function ought to be non-negative, symmetric and it should respect the triangle inequality. The most common and used metric is the standard Euclidean distance. It often appears and is very useful in many geometrical situations, when distances between points need to be calculated. However, it has two major drawbacks that are problematic especially in machine learning applications. First of all, Euclidean distance is sensitive to scaling. Whilst in mathematical problems this does not constitute an issue, in real situations, we may have features that are measured in different units (e.g., seconds, kilograms, etc.) and we will obtain different distances between our data points if we change the scalings on some axis. The other problem is the fact that it does not take into consideration the correlations in the data structure. It can often happen that more attributes reflect the same information present in the data and, consequently, the distance is strongly influenced by those attributes. Take the example of face recognition: there the pixels from the image background are highly correlated and they usually reflect the same information, i.e., the colour of the background.

The standard notation is $d(\mathbf{x}, \mathbf{y})$ and it represents a function that calculates the distance between two inputs $\mathbf{x}$ and $\mathbf{y}$ (column vectors in a D dimensional space $\mathbf{x}, \mathbf{y} \in \mathbb{R}^D$). Esentially, the metric maps a pair from $\mathbb{R}^D \times \mathbb{R}^D$ to a real number scalar.

Formally, $d : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$ is a metric if for any $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{R}^D$ the following properties hold **?**:

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

Now we can relate the previous definition with the most common and used example: Euclidean distance. This is defined as:

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

Unfortunately, the Euclidean distance has two major drawbacks that are problematic especially in machine learning applications[1]:

- **Sensitivity to variable scaling**. In geometrical situations, all variables are measured in the same units of length; for some of the real data variability is stronger on certain dimensions than on others. Naturally, we try to compensate the acute difference; so we want components with high variability to receive less weight than those with low variability. We can obtain this by simply rescaling the components with corresponding values $(s_i)_{i=\overline{1,D}}$:

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

  where $S^{-1} = \text{diag}(s_1^2, \cdots, s_D^2)$.

- **Invariance to correlated variables**. Euclidean distance does not take into account the correlations in the data structure. For face images, for example, the pixels in the background, usually have the same colour, especially if they are close to each other; if these pixels differ strongly for other picture of the same person, we will be heavily penalized, because their number is large and the distance will be influenced by them. Therefore, we should not take into account all these variables since they reflect

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

the same information (i.e., the color of the background). Intuitively, we want our distance metric to reflect the correlation in the data. This can be easily achieved by replacing S in Equation (2.2) with the covariance matrix of the data.

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x}-\mathbf{y})^T S^{-1} (\mathbf{x}-\mathbf{y})}, \text{ with } S = \text{cov}(\mathcal{X}) \tag{2.3}$$

where $\mathcal{X}$ is the dataset $\mathcal{X} = (\mathbf{x}_i)_{i=\overline{1,N}}$.

Equation (2.3) is the standard definition of the Mahalanobis distance. Since we do not care for any variance in the data, but just for that is representative for our particular task (as discussed in Section **??**), we will consider different matrices S that are suitable for the given query.

However there are some conditions that S should respect. First of all, we note that $d(\mathbf{x}, 0) = \|\mathbf{x}\| = \sqrt{\mathbf{x}^T S^{-1} \mathbf{x}}$ which imposes $\mathbf{x}^T S^{-1} \mathbf{x} \geq 0, \forall \mathbf{x}$—this means that S^{-1} must be a positive semi-definite matrix.

We note that we can use Cholesky decomposition and write $S^{-1} = L^T L$, which gives $\|\mathbf{x}\| = \sqrt{(L\mathbf{x})^T (L\mathbf{x})}$. L can be viewed as a linear transformation of the data. Also, using L we can define parametrize our problem by defining a family of metrics over the input space. So the problem of finding a suitable distance metric is virtually equivalent to the problem of finding a good linear transformation and we will discuss these two variants interchangeably.

2.2 Related methods

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 tries to find that 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 with constraints which can be solved using an iterative procedure, such as Newton-Raphson. The solution is local optima free, but this also means that the method assumes that the data set is uni-modal. This is a strong assumption and it does not coagulate well with the more liberal kNN. Apart from this, the training time is also a problem as the iterative process is costly for large data sets.

Relevant Component Analysis (RCA; Bar-Hillel et al., 2003; **?**) 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 and Saul, 2009). 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).

Chapter 3

Neighbourhood component analysis

Neighbourhood Component Analysis (NCA; Goldberger et al., 2004) 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.

3.1 General presentation

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 $\mathbf{A}\mathbf{x}_i$ and classify it using kNN on the transformed data set $\{\mathbf{A}\mathbf{x}_j\}_{j=1}^{N}$. The matrix $\mathbf{A}$ that achieves the highest number of correct classifications will be used for testing. Unfortunately, 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 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 softmax activation used

for neural networks or the generalized logistic function. So p_{ij} is the probability that the point $\mathbf{x}_j$ is selected as the nearest neighbour of the point $\mathbf{x}_i$ and it 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 $d_{ij} = d(\mathbf{A}\mathbf{x}_i; \mathbf{A}\mathbf{x}_j) = (\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}} \mathbf{A}^{\mathrm{T}} \mathbf{A} (\mathbf{x}_i - \mathbf{x}_j)$; also we set $p_{ii} = 0$: point $\mathbf{x}_i$ cannot pick itself as the nearest neighbour. 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 maximize is the probability of each point of getting correctly classified. A point $\mathbf{x}_i$ is correctly classified when it is selected by a point $\mathbf{x}_j$ that has the same label as $\mathbf{x}_i$:

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

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

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

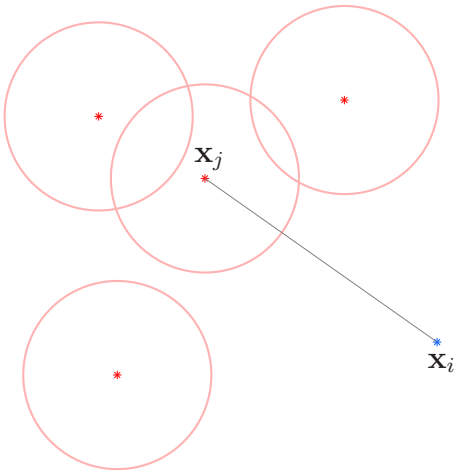
The score given by the objective function can be interpreted as the expected number of the correctly classified points.

We maximise $f(\mathbf{A})$ using an iterative gradient based solver such as gradient ascent, conjugate gradients or delta-bar-delta, see Subsection 3.3.1. For the optimisation, we need the numerical 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 of the gradient in the appendix.

It is useful to note that NCA's objective function is not convex. So care must be taken to avoid poor local optima. The choice of initialization and of the



(a) Illustration of the class as a mixture of Gaussians.

Figure 3.1: Formulating NCA as a class-conditional kernel density estimation framework.

optimization method affect the quality of the final solution. Section 3.3 discusses these and other practical issues. On the other hand, the non-convexity property has its advantage. It allows NCA to get good results on more complicated data sets that are not non convex.

Another important aspect is the computational cost of the method. For each point, we need to compute all the pairwise distances d_{ij} which has the cost of $\mathcal{O}(dN^2)$. But prior to that, we have to compute the point projections $\{\mathbf{A}\mathbf{x}_i\}_{i=1}^N$; this operation is done in $\mathcal{O}(dDN)$ flops. So the total cost of evaluating the objective function $f(\mathbf{A})$ is $\mathcal{O}(dDN + dN^2)$. Chapter 4 treats NCA's computational drawback; there are presented different ideas of speeding up the computations.

The general scenario in which NCA is applied can be described by the training and testing steps: describe algorithm.

3.2 Class-conditional kernel density estimation interpretation

In this section we will present NCA into a class-conditional kernel density estimation framework. This interpretation will allow us to understand what are the assumptions behind NCA. Moreover, this also offers the possibility of altering the model in a suitable way that is efficient for computations. We will see this in the

sections 4.4 and 4.5. Similar ideas were previously presented by and , but the following were derived independently and they offer different insights. The 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. From a generative perspective, we can view that each point $\mathbf{x}_j$ can generate a point $\mathbf{x}_i$ with a probability given by an isotropic normal distribution with variance σ^2:

$$p(\mathbf{x}_i|\mathbf{x}_j) = \mathcal{N}(\mathbf{x}_i|\mathbf{x}_j, \sigma^2 \mathrm{I}_D) \tag{3.5}$$

$$= \frac{1}{(2\pi)^{D/2}} \exp\left\{-\frac{1}{2\sigma^2}(\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}}(\mathbf{x}_i - \mathbf{x}_j)\right\}. \tag{3.6}$$

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

$$p(\mathbf{A}\mathbf{x}_i|\mathbf{A}\mathbf{x}_j) \propto \exp\left\{-\frac{1}{2\sigma^2}(\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\mathbf{A}(\mathbf{x}_i - \mathbf{x}_j)\right\}. \tag{3.7}$$

We can 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:

$$p(\mathbf{x}_i|c) = \frac{1}{N_c}\sum_{\mathbf{x}_j \in c} p(\mathbf{x}_i|\mathbf{x}_j) \tag{3.8}$$

$$= \frac{1}{N_c}\sum_{\mathbf{x}_j \in c} \mathcal{N}(\mathbf{x}_i|\mathbf{x}_j, \mathrm{I}_D). \tag{3.9}$$

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

$$p(c|\mathbf{x}_i) = \frac{p(\mathbf{x}_i|c)p(c)}{p(c|\mathbf{x}_i)} = \frac{p(\mathbf{x}_i|c)p(c)}{\sum_c p(\mathbf{x}_i|c)p(c)}. \tag{3.10}$$

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

$$p(c|\mathbf{A}\mathbf{x}_i) = \frac{\frac{1}{N_c}\sum_{\mathbf{x}_j \in c} \exp\left\{-\frac{1}{2\sigma^2}(\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\mathbf{A}(\mathbf{x}_i - \mathbf{x}_j)\right\}}{\frac{1}{N_{c'}}\sum_{c'}\sum_{\mathbf{x}_k \in c'} \exp\left\{-\frac{1}{2\sigma^2}(\mathbf{x}_i - \mathbf{x}_k)^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\mathbf{A}(\mathbf{x}_i - \mathbf{x}_k)\right\}} \tag{3.11}$$

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

$$f(\mathbf{A}) = \sum_i p(c_i|\mathbf{A}\mathbf{x}_i). \tag{3.12}$$

.

The gradient of the objective function is the following:

$$\frac{\partial f}{\partial \mathbf{A}} = \sum_i \left\{ \frac{\frac{\partial p(\mathbf{A}\mathbf{x}_i|c)}{\partial \mathbf{A}} p(c)}{\sum_c p(\mathbf{x}_i|c)p(c)} - \underbrace{\frac{p(\mathbf{A}\mathbf{x}_i|c)p(c)}{\sum_c p(\mathbf{A}\mathbf{x}_i|c)p(c)}}_{p(c|\mathbf{A}\mathbf{x}_i)} \frac{\sum_c \frac{\partial p(\mathbf{A}\mathbf{x}_i|c)}{\partial \mathbf{A}} p(c)}{\sum_c p(\mathbf{A}\mathbf{x}_i|c)p(c)} \right\}. \tag{3.13}$$

3.3 Practical notes

This section provides some practical advice for the questions that can be raised while implementing NCA. While NCA is not that hard to implement, there is needed certain care in order to achieve good solutions.

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.

There are three aspects we need to carefully consider:

1. 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.

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 η
5: $t \leftarrow t + 1$
6: **until** convergence
7: **return** $\mathbf{A}_t$.

2. 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 parameter 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 its more flexible. The "bold driver" is a heuristic for modifying η during training.

3. Stopping criterion.

Conjugate gradients

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 linear transformations that are cheap to compute. Such examples include principal component analysis (PCA; Pearson, 1901), linear discriminant analysis (LDA; Fisher and Others, 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:

$$\mathbf{S} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}-\boldsymbol{\mu})(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}. \tag{3.14}$$

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

$$\mathbf{S}_B = \frac{1}{C}\sum_{c=1}^{C}\boldsymbol{\mu}_c\boldsymbol{\mu}_c^{\mathrm{T}} \tag{3.15}$$

$$\mathbf{S}_W = \frac{1}{N}\sum_{c=1}^{C}\sum_{i\in c}(\mathbf{x}_i-\boldsymbol{\mu}_c)(\mathbf{x}_i-\boldsymbol{\mu}_c)^{\mathrm{T}}. \tag{3.16}$$

  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 "whitens" the data with respect to the within-chunklet covariance matrix. Because for NCA we restrict ourselves to fully labelled data, the within-chunklet covariance is the within-class covariance $\mathbf{S}_W$, Equation 3.16. The whitening transformation is then $\mathbf{A} = \mathbf{S}_W^{-1/2}$.

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 large data sets. As advertised in , we found RCA to work the best. Figure depicts initialization effects on a small data set.

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:

$$x_{ij} \leftarrow \frac{x_{ij} - \mu_j}{\sigma_j}, i = \{1, \cdots, N\}, j = \{1, \cdots, D\}. \tag{3.17}$$

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 $\mathbf{A}$ and get:

$$\mathbf{A}_{\text{total}} \leftarrow \mathbf{A} \cdot \begin{pmatrix} \sigma_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_D \end{pmatrix}. \tag{3.18}$$

In general, data normalization avoids numerical issues for the first iterations. But during training, the scale of $\mathbf{A}$ 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 robust more rigorous way of dealing with this by multiplying both the numerator and denominator of p_i with a certain quantity $\exp(L)$:

$$p_i = \frac{\sum_{j \in c_i} \exp(L - d_{ij}^2)}{\sum_{k \neq i} \exp(L - d_{ik}^2)}, \tag{3.19}$$

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 previous trick because of its simplicity and speed.

3.3.4 Regularization

NCA original objective function has the tendency to increase the scale of the linear projection $\mathbf{A}$. A point is happy if it has a neighbour from the same class as him and if they are very far away from the rest. So NCA favours 1 nearest neighbour. In practice, this is not however the optimal thing to do. Butman and Goldberger (2008) and Singh-Miller (2010) suggested the introduction of a regularization term to prevent degeneracy

$$g(\mathbf{A}) = f(\mathbf{A}) - \lambda \sum_{i=1}^{d} \sum_{j=1}^{D} A_{ij}^2. \tag{3.20}$$

$$\frac{\partial g}{\partial \mathbf{A}} = \frac{\partial f}{\partial \mathbf{A}} - 2\lambda \mathbf{A}. \tag{3.21}$$

Because we use

3.3.5 Doing classification

- We optimize the objective function 3.12. We found that for large data sets it is better to do classification using the probabilistic based approach: given a query point $\mathbf{x}^*$ we assign to the most probable class $c = \text{argmax}_c \, p(c|\mathbf{x}^*)$.

3.3.6 Dimensionality annealing

-

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

$$\frac{\partial g}{\partial \mathbf{A}} = \frac{\partial f}{\partial \mathbf{A}} - 2 \begin{pmatrix} \lambda_1 A_{11} & \cdots & \lambda_1 A_{1D} \\ \vdots & \ddots & \vdots \\ \lambda_d A_{d1} & \cdots & \lambda_d A_{dD} \end{pmatrix}. \tag{3.23}$$

Chapter 4

Reducing the computational cost

As emphasized in Section 3.1, Neighbourhood Component Analysis (NCA) is a computationally expensive method. The evaluation of its objective function is quadratic in the size of the data set. Given that the optimization is done iteratively, NCA becomes prohibitively slow when applied on large data sets. There is only little previous work that uses NCA for large scaled applications. One example is Singh-Miller (2010), who parallelizes the computations across multiple computers and adopts various heuristics to prune terms of the objective function and the gradient.

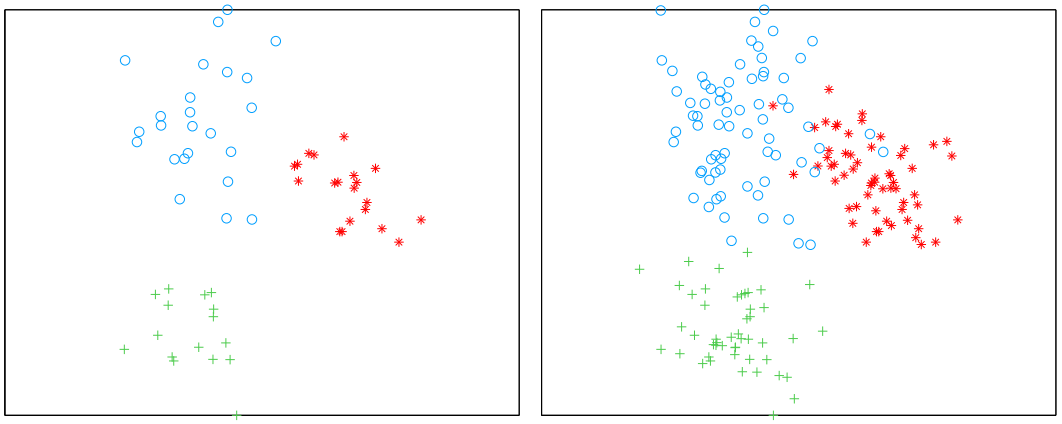
This chapter aims to complete the existing work. We present a series of methods that can improve NCA's speed. Most of these ideas are new in the context of NCA. They are also quite versatile. Each method proposes a new objective function and can be regarded as an independent model on its own.

4.1 Sub-sampling

Sub-sampling is the simplest idea that can help speeding up the computations. For the training procedure we use a randomly selected sub-set $\mathcal{D}_n$ of the original data set $\mathcal{D}$:

$$\mathcal{D}_n = \{\mathbf{x}_{i_1}, \cdots, \mathbf{x}_{i_n}\} \subseteq \mathcal{D}.$$

If n is the size of the sub-set then the cost of the gradient is reduced to $\mathcal{O}(dDn^2)$. After the projection matrix $\mathbf{A}$ is learnt, we apply it to the whole data set $\{\mathbf{x}_i\}_{i=1}^N$. Then all the new data points $\{\mathbf{A}\mathbf{x}_i\}_{i=1}^N$ are used for classification. The cost of the classification is $\mathcal{O}(dN)$; it is only linear in the total number of points N.



(a) Learnt projection $\mathbf{A}$ on the sub-sampled data set $\mathcal{D}_n$.

(b) The projection $\mathbf{A}$ applied to the whole data set $\mathcal{D}$.

Figure 4.1: Result of sub-sampling method on `wine`. There were used one third of the original data set for training, *i.e.*, $n = N/3$. We note that the points that belong to the sub-set $\mathcal{D}_n$ are perfectly separated. But after applying the metric to the whole data there appear different misclassification errors. The effects are even more acute if we use smaller sub-sets.

While easy to implement, this method discards a lot of information available. Also it is affected by the fact the sub-sampled data has a thinner density than the real data. The distances between the randomly selected points are larger than they are in the full data set. This causes the scale of the projection matrix $\mathbf{A}$ not to be large enough, Figure 4.1.

4.2 Mini-batches

The next obvious idea is to use sub-sets in an iterative manner, similarly to the stochastic gradient descent method: split the data into mini-batches and train on them successively. Again the cost for one evaluation of the gradient will be $\mathcal{O}(dDn^2)$ if the mini-batch consists of n points.

There are different possibilities for splitting the data-set:

1. Random selection. In this case the points are assigned randomly to each mini-batch and after one pass through the whole data set another random allocation is done. As in section 4.1, this suffers from the thin distribution problem. In order to alleviate this and achieve convergence, large-sized

Algorithm 4.1 Training algorithm using mini-batches formed by clustering

Require: Data set $\mathcal{D} = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$ and initial linear transformation $\mathbf{A}$.

1: **repeat**
2: Project each data point using $\mathbf{A}$: $\mathcal{D}_\mathbf{A} = \{\mathbf{A}\mathbf{x}_1, \cdots, \mathbf{A}\mathbf{x}_N\}$.
3: Use either algorithm 4.2 or 4.3 on $\mathcal{D}_\mathbf{A}$ to split $\mathcal{D}$ into K mini-batches $\mathcal{M}_1, \cdots, \mathcal{M}_K$.
4: **for all** $\mathcal{M}_i$ **do**
5: Update parameter: $\mathbf{A} \leftarrow \mathbf{A} + \eta \frac{\partial f(\mathbf{A}, \mathcal{M}_i)}{\partial \mathbf{A}}$.
6: Update learning rate η.
7: **end for**
8: **until** convergence.

mini-batches should be used (similar to Laurens van der Maaten's implementation). The algorithm is similar to Algorithm 4.1, but lines 2 and 3 will be replaced with a simple random selection.

2. Clustering. Constructing mini-batches by clustering ensures that the point density in each mini-batch is conserved. In order to maintain a low computational cost, we consider cheap clustering methods, *e.g.*, farthest point clustering (FPC; Gonzalez, 1985) and recursive projection clustering (RPC; Chalupka, 2011).

   FPC gradually selects cluster centres until it reaches the desired number of clusters K. The point which is the farthest away from all the current centres is selected as new centre. The precise algorithm is presented below.

Algorithm 4.2 Farthest point clustering

Require: Data set $\mathcal{D} = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$ and K number of clusters.

1: Randomly pick a point that will be the first centre $\mathbf{c}_1$.
2: Allocate all the points in the first cluster $\mathcal{M}_1 \leftarrow \mathcal{D}$.
3: **for** $i = 1$ to K **do**
4: Select the i-th cluster centre $\mathbf{c}_i$ as the point that is farthest away from any cluster centre $\mathbf{c}_1, \cdots, \mathbf{c}_{i-1}$.
5: Move to the cluster $\mathcal{M}_i$ those points that are closer to its centre than to any other cluster centre: $\mathcal{M}_i = \{\mathbf{x} \in \mathcal{D} \mid d(\mathbf{x}; \mathbf{c}_i) < d(\mathbf{x}; \mathbf{c}_j), \forall j \neq i\}$
6: **end for**

The computational cost of this method is $\mathcal{O}(NK)$. However, we do not have

any control on the number of points in each cluster, so we might end up with very unbalanced clusters. A very uneven split has a couple of obvious drawbacks: too large mini-batches will maintain high cost, while on too small clusters there is not too much to learn.

An alternative is RPC which was especially designed to mitigate this problem. It constructs the clusters similarly to how the k-d trees are build, Subsection 4.4.1. However instead of splitting the data set across axis aligned direction it chooses the splitting directions randomly, Algorithm 4.3. So, because it uses the median value it will result in similar sized clusters and we can easily control the dimension of each cluster.

Algorithm 4.3 Recursive projection clustering

Require: Data set $\mathcal{D} = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$ and n size of clusters.

1: **if** $N < n$ **then**
2: New cluster: $i \leftarrow i + 1$.
3: **return** current points as a cluster: $\mathcal{M}_i \leftarrow \mathcal{D}$.
4: **else**
5: Randomly select two points $\mathbf{x}_j$ and $\mathbf{x}_k$ from $\mathcal{D}$.
6: Project all data points onto the line defined by $\mathbf{x}_j$ and $\mathbf{x}_k$. (Give equation?)
7: Select the median value $\tilde{\mathbf{x}}$ from the projected points.
8: Recurs on the data points above and below $\tilde{\mathbf{x}}$: RPC($\mathcal{D}_{>\tilde{\mathbf{x}}}$) and RPC($\mathcal{D}_{\leq\tilde{\mathbf{x}}}$).
9: **end if**

Note that we are re-clustering in the transformed space after one sweep through the whole data set. There are also other alternatives. For example, we could cluster in the original space. This can be done either only once or periodically. However the proposed variant has the advantage of a good behaviour for a low-rank projection matrix $\mathbf{A}$. Not only that is cheaper, but the clusters resulted in low dimensions by using RPC are closer to the real clusters then applying the same method in a high dimensional space.

4.3 Stochastic learning

The following technique is theoretically justified by stochastic approximation arguments. The main idea is to get an unbiased estimator of the gradient by looking only at a few points and how they relate to the entire data set.

More precisely, in the classical learning setting, we update our parameter $\mathbf{A}$ after we have considered each point $\mathbf{x}_i$ in the data set. In the stochastic learning procedure, we update $\mathbf{A}$ more frequently by considering only n randomly selected points. As in the previous case, we still need to compute the soft assignments $\{p_i\}_{i=1}^n$ using *all* the N points. To stress this further, this solution differs from the mini-batch approach. For the previous method, the contributions $\{p_i\}_{i=1}^n$ are calculated only between the n points that belong to the mini-batch.

The objective function that we need to optimize at each iteration and its gradient are given by the next equations:

$$f_{\text{sNCA}}(\mathbf{A}) = \sum_{l=1}^{n} p_{i_l} \tag{4.1}$$

$$\frac{\hat{\partial f}}{\partial \mathbf{A}} = \frac{\partial f_{\text{sNCA}}}{\partial \mathbf{A}} = \sum_{l=1}^{n} \frac{\partial p_{l_i}}{\partial \mathbf{A}}, \tag{4.2}$$

$$\text{with } p_i = \sum_{\substack{j=1 \\ j \in c_i}}^{N} p_{ij}. \tag{4.3}$$

This means that the theoretical cost of the stochastic learning method scales with nN.

Algorithm 4.4 Stochastic learning for NCA (sNCA)

Require: Data set $\mathcal{D} = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$, n number of points to consider for the gradient estimation, $\mathbf{A}$ initial linear transformation.

1: **repeat**
2: Split data $\mathcal{D}$ into groups $\mathcal{M}_i$ of size n.
3: **for all** $\mathcal{M}_i$ **do**
4: Update parameter using gradient given by Equation 4.2:
5: $\mathbf{A} \leftarrow \mathbf{A} + \eta \frac{\partial f_{\text{sNCA}}(\mathbf{A}, \mathcal{M}_i)}{\partial \mathbf{A}}$.
6: Update learning rate η.
7: **end for**
8: **until** convergence.

This method comes with an additional facility. It can be used for on-line learning. Given a new point $\mathbf{x}_{N+1}$ we update $\mathbf{A}$ using the derivative $\frac{\partial p_{N+1}}{\partial \mathbf{A}}$.

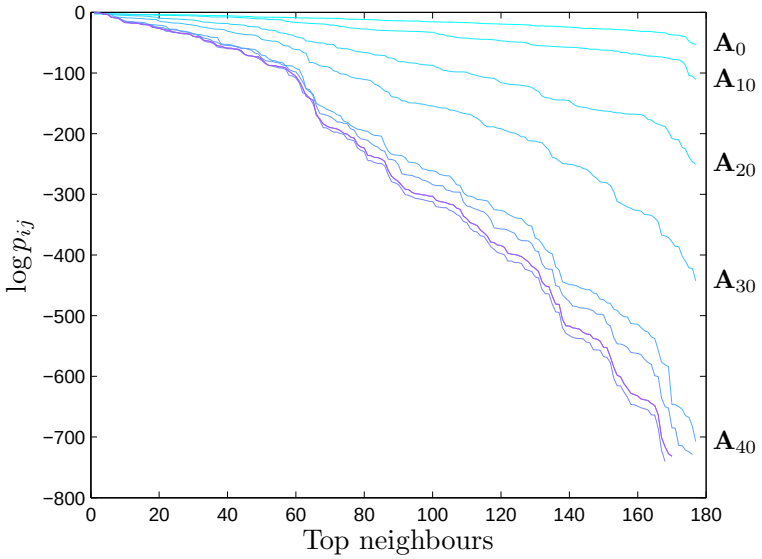


Figure 4.2: Evolution of the stochastic assignments p_{ij} during training for a given point $\mathbf{x}_i$.

4.4 Approximate computations

A straightforward way of speeding up the computations was previously mentioned in the original paper (Goldberger et al., 2004) and in some NCA related work (Weinberger and Tesauro, 2007; Singh-Miller, 2010). The observations involve pruning small terms in the original objective function. We then use an approximated objective function and its corresponding gradient for the optimization process.

The motivation lies in the fact that the contributions p_{ij} decay very quickly with distance:

$$p_{ij} \propto \exp\{-d(\mathbf{A}\mathbf{x}_i; \mathbf{A}\mathbf{x}_j)^2\}.$$

The evolution of the contributions during the training period is depicted in Figure 4.2. We notice that most of the values p_{ij} are insignificant compared to the largest contribution. This suggests that we might be able to preserve the accuracy of our estimations even if we discard a large part of the neighbours.

So, Weinberger and Tesauro (2007) choose to use only the top $m = 1000$ neighbours for each point $\mathbf{x}_i$. Also they disregard those points that are farther away than $d_{\max} = 34$ units from the query point: $p_{ij} = 0, \forall \mathbf{x}_j$ such that $d(\mathbf{A}\mathbf{x}_i; \mathbf{A}\mathbf{x}_j) > d_{\max}$. While useful in practical situations, these suggestions lack of a principled description: how can we optimally choose m and $d_{\max}$ in a general

setting? We would also like to be able to estimate the error introduced by the approximations.

We correct those drawbacks by making use of the KDE formulation of NCA (see Section 3.2) and adapting existing ideas for fast KDE (Deng and Moore, 1995; Gray and Moore, 2003) to our particular application. We will use a class of accelerated methods that are based on data partitioning structures (*e.g.*, k-d trees, ball trees). As we shall shortly see, these provide us with means to quickly find only the neighbours $\mathbf{x}_j$ that give significant values p_{ij} for any query point $\mathbf{x}_i$.

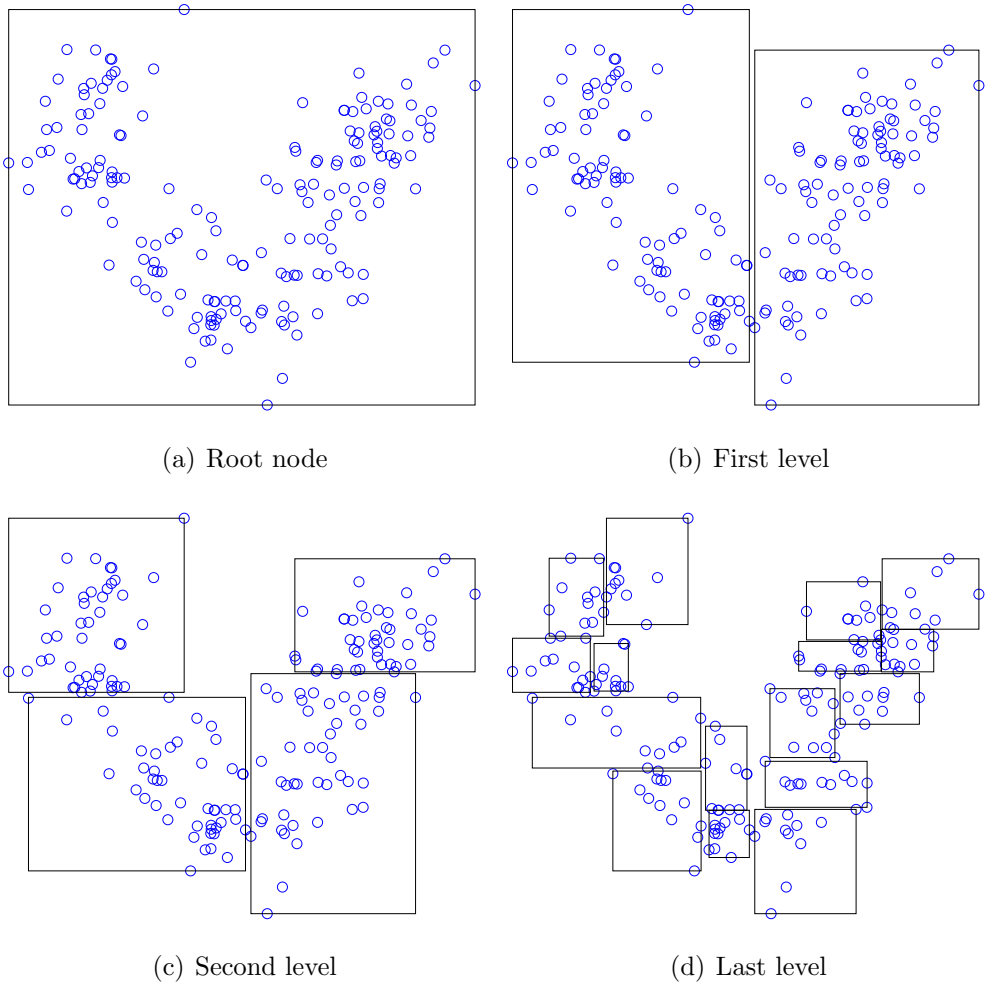
4.4.1 k-d trees

The k dimensional tree structure (k-d tree; Bentley, 1975) organises the data in a binary tree using axis-aligned splitting planes. The k-d tree has the property to place close in the tree those points that live nearby in the original geometrical space. This makes such structures efficient mechanisms for nearest neighbour searches (Friedman et al., 1977) or range searches (Moore, 1991).

There are different flavours of k-d trees. We choose for our application a variant of k-d tree that uses bounding boxes to describe the position of the points. Intuitively, we can imagine each node of the tree as a bounding hyper-rectangle in the D dimensional space of our data. The root node will represent the whole data set and it can be viewed as an hyper-rectangle that contains all the data points, see Figure 4.3(a). In the two-dimensional presented exampled, the points are enclosed by rectangles. From Figure 4.3(a) to 4.3(d), there are presented the existing bounding boxes at different levels of the binary tree. To understand how these are obtained, we discuss the k-d tree construction.

The building of the tree starts from the root node and is done recursively. At each node we select which of the points from the current node will be allocated to each of the two children. Because these are also described by hyper-rectangles, we just have to select a splitting plane. Then the two successors will consist of the points from the two sides of the hyper-plane.

A splitting hyper-plane can be fully defined by two parameters: a direction $\vec{d}$ on which the plane is perpendicular and a point P that is in the plane. Given that the splits are axis aligned, there are D possible directions $\vec{d}$. We can either choose this randomly or we can use each of the directions from 1 to D in a successive manner. A more common approach is to choose $\vec{d}$ to be the dimension



(a) Root node (b) First level

(c) Second level (d) Last level

Figure 4.3: Illustration of the k-d tree with bounding boxes at different levels of depths. This figure also outlines the building phases of the tree.

that presents the largest variance:

$$\vec{d} \leftarrow \text{argmax}_d(\max_i x_{id} - \min_i x_{id}). \tag{4.4}$$

This results in a better clustering of the points and the shape of the bounding boxes will be closer to the shape of a square. Otherwise it might happen that points situated in the same node can still be further away.

Regarding the splitting value P, a usual choice is the median value $\tilde{x}_d$ on the previously selected direction $\vec{d}$. This choice of P guarantees a balanced tree which offers several advantages. We can allocate static memory for the entire data structure. This is faster to access than dynamical allocation. Also a balanced tree has a better worst case complexity than an unbalanced one. Other useful implementation tricks that can be applied to balanced k-d trees are suggested by Lang (2009).

After the splitting plane is chosen, the left child will contain the points that are on the left of the hyper-plane:

$$\mathcal{D}_{\leq \tilde{x}_d} = \{\mathbf{x} \in \mathcal{D}_{\mathbf{x}_i} | x_d \leq \tilde{x}_d\}, \tag{4.5}$$

where $\mathcal{D}_{\mathbf{x}_i}$ denotes the data points bounded by the current node $\mathbf{x}_i$. Similarly, the right child will contain the points that are placed on the right of the hyper-plane:

$$\mathcal{D}_{> \tilde{x}_d} = \{\mathbf{x} \in \mathcal{D}_{\mathbf{x}_i} | x_d > \tilde{x}_d\}. \tag{4.6}$$

This process is repeated until the number of points bounded by the current node goes below a threshold m. These nodes are the leaves of the tree and they store the effective points. The other non-leaf nodes store information regarding the bounding box and the splitting plane. Note that a hyper-rectangle is completely defined by only two D-dimensional points, one for the "top-right" corner and the other for the "bottom-left" corner.

The most used operation on a k-d tree is the nearest neighbour (NN) search. While we will not apply pure NN for the next method, we will use similar concepts. However, we can do NN retrieval with k-d trees after we applied NCA, as suggested by Goldberger et al. (2004). The search in the k-d tree is done in a depth-first search manner: start from the root and traverse the whole tree by selecting the closest node to the query point. In the leaf, we find the nearest neighbour from the m points and store it and the corresponding distance $d_{\min}$. Then we recurs up the tree and look at the farther node. If this is situated at

Algorithm 4.5 k-d tree building algorithm

Require: Data set $\mathcal{D} = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$, i position in tree and m number of points in leaves.

1: **if** $N < m$ **then**
2: Mark node i as leaf: `splitting_direction(i)=-1`.
3: Add points to leaf: `points(i)` $\leftarrow \mathcal{D}$.
4: **return**
5: **end if**
6: Choose direction $\vec{d}$ using Equation 4.4: `splitting_direction(i)=`d.
7: Find the median value $\tilde{x}_d$ on the direction $\vec{d}$: `splitting_value(i)=`$\tilde{x}_d$.
8: Determine the subsets of points that are separated by the resulting splitting plane: $\mathcal{D}_{\leq \tilde{\mathbf{x}}}$ and $\mathcal{D}_{> \tilde{\mathbf{x}}}$, see Equations 4.5 and 4.6.
9: Build left child `build_kdtree(`$\mathcal{D}_{\leq \tilde{\mathbf{x}}}$`,2*i)`.
10: Build right child `build_kdtree(`$\mathcal{D}_{> \tilde{\mathbf{x}}}$`,2*i+1)`.

a minimum distance that is smaller than $d_{\min}$, we have to investigate also that node. In the other case, we can ignore the node and all the points it contains. Usually, a large fraction of the points can be omitted, especially when the data has structure. It is important to stress that the performance of k-d trees quickly degrades with the number of dimensions the data lives.

4.4.2 Approximate kernel density estimation

The following ideas are mostly inspired by previous work on fast kernel density estimators with k-d trees (Gray and Moore, 2001, 2003; Alexander Gray, 2003) and fast Gaussian Processes regression with k-d trees (Shen et al., 2006).

In kernel density estimation (KDE), we are given a data set $\mathcal{D}$ and the goal is to compute the probability denisty at a given point using a sum of the contributions from all the points:

$$p(\mathbf{x}_i) = \frac{1}{N} \sum_{j=1}^{N} k(\mathbf{x}_i | \mathbf{x}_j). \tag{4.7}$$

A common class of KDE problems are the N-body problems (Gray and Moore, 2001). This imposes to estimate $p(\mathbf{x}_i)$ for each data point $\{\mathbf{x}_i\}_{i=1}^{N}$ in the data set $\mathcal{D}$. Note that this operation is quadratic in the number of points, as it is the case of NCA.

If the number of samples N in $\mathcal{D}$ is sufficiently large, we expect to accurately approximate $p(\mathbf{x}_i)$ by using only nearby neighbours of $\mathbf{x}_i$.

To illustrate how this works, let us assume we are given a query point $\mathbf{x}_i$ and a group of points G. We try to reduce computations by replacing each individual contribution $k(\mathbf{x}_i|\mathbf{x}_j), \mathbf{x}_j \in G$, with a fixed quantity $k(\mathbf{x}_i|\mathbf{x}_g)$. The value of $k(\mathbf{x}_i|\mathbf{x}_g)$ is group specific and since it is used for all points in G, it is chosen such that it does not introduce a large error. A reasonable value for $k(\mathbf{x}_i|\mathbf{x}_g)$ is obtained by approximating each point $\mathbf{x}_i$ with a fixed $\mathbf{x}_g$, for example, the mean of the points in G. Then we compute the kernel value $k(\mathbf{x}_i|\mathbf{x}_g)$ using the estimated $\mathbf{x}_g$. A second possibility is to directly approximate $k(\mathbf{x}_i|\mathbf{x}_g)$. For example:

$$k(\mathbf{x}_i|\mathbf{x}_g) = \frac{1}{2}\left(\min_j k(\mathbf{x}_i|\mathbf{x}_j) + \max_j k(\mathbf{x}_i|\mathbf{x}_j)\right). \tag{4.8}$$

We prefer the this option, because it does not introduce any further computational expense. Both the minimum and the maximum contributions are previously calculated to decide whether to prune or not. Also it does not need storing any additional statistic, such as the mean.

We see that the error introduced by each approximation is bounded by the following quantity:

$$\epsilon_{\max} \leq \frac{1}{2}\left(\max_j k(\mathbf{x}_i|\mathbf{x}_j) - \min_j k(\mathbf{x}_i|\mathbf{x}_j)\right). \tag{4.9}$$

This can be controlled to be small if we approximate only for those groups that are far away from the query point or when the variation of the kernel value is small within the group. It is better still to consider the error relative to the total quantity we want to estimate. Of course we do not know the total sum we want to estimate in advance, but we can use a lower bound $p_{\text{SoFar}}(\mathbf{x}_i) + \max k(\mathbf{x}_i|\mathbf{x}_j)$. Hence, a possible cut-off rule is:

Note that in this case it is important the order in which we accumulate. A large $p_{\text{SoFar}}(\mathbf{x}_i)$ in the early stage will allow more compuational savings.

We use k-d trees to form groups of points G that will be described as hyper-rectangles. To compute the probability density function $p(\mathbf{x}_i)$, we start at the root, the largest group, and traverse the tree going through the nearest node each time until we reach the leaf. This guarantees that we add large contributions at the begining. Then we recurs up the tree and visit other nodes only if necessary, when the cut-off condition is not satisfied.

4.4.3 Approximate KDE for NCA

We recall that NCA was formulated as a class-conditional kernel density estimation problem, Section 3.2. By combining ideas from the previous two subsections, we can develop an NCA specific approximation algorithm.

There are some differences from the classical KDE approximation. In this case, we deal with class-conditional probabilities $p(\mathbf{A}\mathbf{x}_i|c), \forall c$. So each class c needs to be treated separately: we build a k-d tree with the projected data points $\{\mathbf{A}\mathbf{x}_j\}_{j \in c}$ and calculate the estimated probability $\hat{p}(\mathbf{A}\mathbf{x}_i|c)$ for each class. Another distictintion is that for NCA our final interest are the objective function and its gradient. We can easily obtain an approximated version of the objective function by replacing $p(\mathbf{A}\mathbf{x}_i|c)$ with the approximated $\hat{p}(\mathbf{A}\mathbf{x}_i|c)$ in Equation 3.12. To obtain the gradient of this new objective function we can use Equation 3.13. The derivative of $\frac{\partial}{\partial \mathbf{A}}\hat{p}(\mathbf{A}\mathbf{x}|c)$ will be different only for those groups where we do approximations. So, for such a group we obtain:

$$\begin{aligned}\frac{\partial}{\partial \mathbf{A}} \sum_{j \in G} k(\mathbf{A}\mathbf{x}|\mathbf{A}\mathbf{x}_j) &\approx \frac{\partial}{\partial \mathbf{A}} \frac{1}{2} \left\{ \min_{j \in G} k(\mathbf{A}\mathbf{x}|\mathbf{A}\mathbf{x}_j) + \max_{j \in G} k(\mathbf{A}\mathbf{x}|\mathbf{A}\mathbf{x}_j) \right\} \\ &= \frac{1}{2} \left\{ \frac{\partial}{\partial \mathbf{A}} k(\mathbf{A}\mathbf{x}|\mathbf{A}\mathbf{x}_c) + \frac{\partial}{\partial \mathbf{A}} k(\mathbf{A}\mathbf{x}|\mathbf{A}\mathbf{x}_f) \right\}, \end{aligned} \tag{4.10}$$

where $\mathbf{A}\mathbf{x}_c$ denotes the closest point in G to the query point $\mathbf{A}\mathbf{x}$ and $\mathbf{A}\mathbf{x}_f$ is the farthest point in G to $\mathbf{A}\mathbf{x}$. Here we made use of the fact the kernel function is a monotonic function of the distance. This means that the closest point gives the maximum contribution, while the farthest point the minimum.

4.5 Exact computations

Exact methods are the counterpart of approximate methods. We can have both efficient and exact computations just by modifying the NCA model. Again, the idea is motivated by the rapid decay of the exponential function. Instead of operating on very small values, we will make them exactly zero. This is achieved by replacing the squared exponential kernel with a compact support function. So, the points that lie outside the support of the kernel are ignored and just a fraction of the total number of points is used for computing the contributions p_{ij}. Further gains in speed are obtained if the search for those points is done with k-d trees (the range search algorithm is suitable for this task; Moore, 1991).

Algorithm 4.6 Approximate NCA objective function and gradient computation

Require: Projection matrix $\mathbf{A}$, data set $\mathcal{D} = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$ and error ϵ.

1: **for all** classes c **do**
2: Build k-d tree for the points in class c.
3: **end for**
4: **for all** data points $\mathbf{x}_i$ **do**
5: **for all** classes c **do**
6: Compute estimated probability $\hat{p}(\mathbf{A}\mathbf{x}_i|c)$ and the corresponding derivatives $\frac{\partial}{\partial \mathbf{A}}\hat{p}(\mathbf{A}\mathbf{x}_i|c)$ using approximated KDE Algorithm:
7: **end for**
8: Compute soft probability $\hat{p}_i \equiv \hat{p}(c|\mathbf{A}\mathbf{x}_i) = \frac{\hat{p}(\mathbf{A}\mathbf{x}_i|c_i)}{\sum_c \hat{p}(\mathbf{A}\mathbf{x}_i|c)}$.
9: Compute gradient $\frac{\partial}{\partial \mathbf{A}}\hat{p}_i$ using Equation 3.13.
10: Update function value and gradient value.
11: **end for**

The choice of the compact support kernel is restricted by a single requirement: differentiability. We will use the simplest polynomial function that has this property. This is given by the following expression:

$$k_{\mathrm{CS}}(u) = \begin{cases} c\,(a^2 - u^2)^2 & \text{if } u \in [-a; +a] \\ 0 & \text{otherwise,} \end{cases} \tag{4.11}$$

where c is a constant that controls the height of the kernel and a is a constant that controls the width of the kernel. In the given context, the kernel will be a function of the distance between two points: $k_{\mathrm{CS}}(u) = k_{\mathrm{CS}}(d_{ij})$, where $d_{ij} = d(\mathbf{A}\mathbf{x}_i; \mathbf{A}\mathbf{x}_j)$. Note that the constant a can be absorbed by the linear projection $\mathbf{A}$. This means that the scale of the learnt metric will compensate for the kernel's width. Also the value for c is not important: from Equation 4.13 we see that this reduces. For convenience, we set both $a = 1$ and $c = 1$. So, we obtain the following simplified version of the kernel:

$$k_{\mathrm{CS}}(d_{ij}) = (1 - d_{ij}^2)^2\ \mathrm{I}(|d_{ij}| \leq 1), \tag{4.12}$$

where $\mathrm{I}(\cdot)$ denotes the indicator function: $\mathrm{I}(\cdot)$ return 1 when its argument is true and 0 when its argument is false.

Now we reiterate the steps of the NCA algorithm (presented in Section 3.1), and replace $\exp(\cdot)$ with $k_{\mathrm{CS}}(\cdot)$. We obtain the following new stochastic neighbour

assignments:

$$q_{ij} = \frac{k_{\text{CS}}(d_{ij})}{\sum_{k \neq i} k_{\text{CS}}(d_{ik})}. \tag{4.13}$$

These can be compared to the classical soft assignments given by Equation 3.1. Next we do not need to change the general form of the objective function:

$$f_{\text{CS}}(\mathbf{A}) = \sum_i \sum_{j \in c_i} q_{ij}. \tag{4.14}$$

In order to derive the gradient of the function f_{CS}, we start by computing the gradient of the kernel:

$$\begin{aligned} \frac{\partial}{\partial \mathbf{A}} k_{\text{CS}}(d_{ij}) &= \frac{\partial}{\partial \mathbf{A}} \left[(1 - d_{ij}^2)^2 \cdot \mathrm{I}(|d_{ij}| \leq 1) \right] \\ &= -4\mathbf{A}(1 - d_{ij}^2)\mathbf{x}_{ij}\mathbf{x}_{ij}^{\mathrm{T}} \cdot \mathrm{I}(|d_{ij}| \leq 1), \end{aligned} \tag{4.15}$$

where $\mathbf{x}_{ij} = \mathbf{x}_i - \mathbf{x}_j$.

The gradient of the new objective function is:

$$\frac{\partial f_{\text{CS}}}{\partial \mathbf{A}} = 4\mathbf{A} \sum_{i=1}^{N} \left(q_i \sum_{k=1}^{N} \frac{q_{ik}}{1 - d_{ik}^2} \mathbf{x}_{ik}\mathbf{x}_{ik}^{\mathrm{T}} - \sum_{j \in c_i} \frac{q_{ij}}{1 - d_{ij}^2} \mathbf{x}_{ij}\mathbf{x}_{ij}^{\mathrm{T}} \right). \tag{4.16}$$

This method can be applied in same way as classic NCA: learn a metric $\mathbf{A}$ that maximizes the objective function $f_{\text{CS}}(\mathbf{A})$. Since the function is differentiable, any gradient based method is suitable for optimization and can be used on Equation 4.16.

There is one concern with the compact support version of NCA. There are situations when a point $\mathbf{x}_i$ is placed outside the support of any other point in the data set. Intuitively, this means that the point $\mathbf{x}_i$ is not selected by any point, hence it is not assigned any class label. Also this causes mathematical problems: as in Subsection 3.3.3, the contributions p_{ij} will have an indeterminate value $\frac{0}{0}$. Except of the log-sum-exp trick, the advice from Subsection 3.3.3 can be applied here as well. A more robust way of dealing with this is discussed in the next Section.

4.6 NCA with compact support kernels and background distribution

We extend the previous model to handle cases where points fall outside the support of any other neighbours. The idea is to use for each class a background

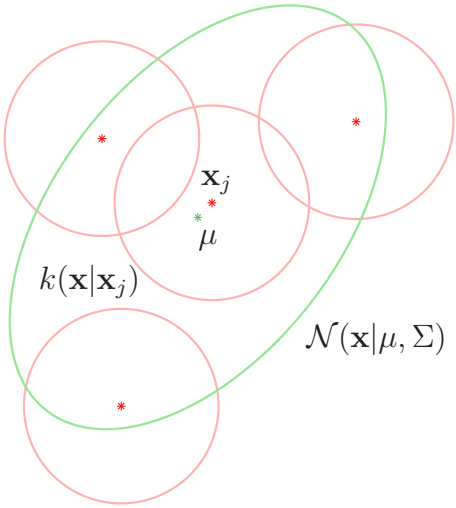


Figure 4.4: Neighbourhood component analysis with compact support kernels and background distribution. The main assumption is that each class is a mixture of compact support distributions $k(\mathbf{x}|\mathbf{x}_j)$ plus a normal background distribution $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$.

distribution that explains the unallocated points. The background distribution should have an infinite support and an obvious example is the normal distribution.

To introduce a background distribution in a principled manner, we return to the class conditional kernel density estimation (CC-KDE) formulation of NCA, Section 3.2. First, we recast the compact support NCA in the probabilistic framework and consider each class as mixture of compact support distributions:

$$p(\mathbf{x}_i|c) = \frac{1}{N} \sum_{j \in c} k_{\text{CS}}(\mathbf{x}_i|\mathbf{x}_j), \tag{4.17}$$

where $k_{\text{CS}}(\mathbf{x}_i|\mathbf{x}_j) = k_{\text{CS}}(d_{ij})$ and is defined by Equation 4.11. Because $k_{\text{CS}}(\mathbf{x}_i|\mathbf{x}_j)$ denotes a distribution, it ought to integrate to 1. For $c = \frac{15}{16}$ and $a = 1$ the requirement is satisfied.

We further change the model and incorporate an additional distribution in the class-conditional probability $p(\mathbf{x}_i|c)$. From a generative perspective this can be interpreted as follows: a point $\mathbf{x}_i$ is generated by either the compact support distribution from each point $k_{\text{CS}}(\mathbf{x}_i|\mathbf{x}_j)$ or by a class-specific normal distribution $\mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$. So, the distribution $p(\mathbf{x}_i|c)$ can be written as the sum of these components:

$$p(\mathbf{x}_i|c) = \beta \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c) + (1 - \beta) \frac{1}{N_c} \sum_{j \in c} k_{\text{CS}}(\mathbf{x}_i|\mathbf{x}_j), \tag{4.18}$$

where β is the mixing coefficient between the background distribution and the compact support model, $\boldsymbol{\mu}_c$ is the sample mean of the class c and $\boldsymbol{\Sigma}_c$ is the sample covariance of the class c. The constant β can be set to $\frac{1}{N_c+1}$. This will give equal weights to the background distribution and to each compact support distribution. It might be better to treat β as a parameter and fit it during training. We expect β to adapt to the data set: for example, β should increase for data sets with convex classes.

To finalize this method, we just need to plug Equation 4.18 into the set of Equations 3.10, 3.12 and 3.13. The only difficulty is the gradient computation. We give here only derivatives for each individual component (the full derivations and equations can be found in the Appendix):

- The gradient of the compact support distribution $k_{\mathrm{CS}}(\mathbf{x}_i|\mathbf{x}_j)$ with respect to $\mathbf{A}$ is very similar to what is given in Equation 4.15. The only difference is that in this case we have everything multiplied by the constant $c = \frac{15}{16}$.

- For the gradient of the background distribution it is useful to note that projecting the points $\{\mathbf{x}_i\}_{i=1}^N$ into a new space $\{\mathbf{A}\mathbf{x}_i\}_{i=1}^N$ will change the sample mean $\boldsymbol{\mu}_c$ to $\mathbf{A}\boldsymbol{\mu}_c$ and the sample covariance $\boldsymbol{\Sigma}_c$ to $\mathbf{A}\boldsymbol{\Sigma}_c\mathbf{A}^{\mathrm{T}}$. Hence, we have:

$$\frac{\partial}{\partial \mathbf{A}}\mathcal{N}(\mathbf{A}\mathbf{x}_i|\mathbf{A}\boldsymbol{\mu}_c, \mathbf{A}\boldsymbol{\Sigma}_c\mathbf{A}^{\mathrm{T}}) = \mathcal{N}(\mathbf{A}\mathbf{x}_i|\mathbf{A}\boldsymbol{\mu}_c, \mathbf{A}\boldsymbol{\Sigma}_c\mathbf{A}^{\mathrm{T}}) \times \{-(\mathbf{A}\boldsymbol{\Sigma}_c\mathbf{A}^{\mathrm{T}})^{-1}\mathbf{A}\boldsymbol{\Sigma}_c + \mathbf{v}\mathbf{v}^{\mathrm{T}}\mathbf{A}\boldsymbol{\Sigma}_c - \mathbf{v}(\mathbf{x}-\boldsymbol{\mu}_c)^{\mathrm{T}}\}, \quad (4.19)$$

  where $\mathbf{v} = (\mathbf{A}\boldsymbol{\Sigma}_c\mathbf{A}^{\mathrm{T}})^{-1}\mathbf{A}(\mathbf{x}-\boldsymbol{\mu}_c)$.

- If we also consider β a parameter, we also need the derivative of the objective function with respect to β. This can be easily obtained, if we use the derivative of the class conditional distribution with respect to β:

$$\frac{\partial}{\partial \beta}p(\mathbf{x}_i|c) = \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c) - \frac{1}{N_c}\sum_{j\in c} k_{\mathrm{CS}}(\mathbf{x}_i|\mathbf{x}_j). \quad (4.20)$$

Bibliography

Alexander Gray, A. M. (2003). Rapid evaluation of multiple density models. In *Artificial Intelligence and Statistics*.

Bar-Hillel, A., Hertz, T., Shental, N., and Weinshall, D. (2003). Learning distance functions using equivalence relations. In *Proceedings of the Twentieth International Conference on Machine Learning*, volume 20, page 11.

Barber, D. (2011). *Bayesian Reasoning and Machine Learning*. Cambridge University Press. In press.

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18:509–517.

Beyer, K., Goldstein, J., Ramakrishnan, R., and Shaft, U. (1999). When is "nearest neighbor" meaningful? *Database TheoryICDT99*, pages 217–235.

Bishop, C. (1995). *Neural networks for pattern recognition*. Oxford university press.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Butman, M. and Goldberger, J. (2008). Face recognition using classification-based linear projections. *EURASIP Journal on Advances in Signal Processing*, 2008:1–7.

Chalupka, K. (2011). Empirical evaluation of Gaussian Process approximation algorithms.

Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. 13:21– 27.

Deng, K. and Moore, A. (1995). Multiresolution instance-based learning. In *Proceedings of the Twelfth International Joint Conference on Artificial Intellingence*, pages 1233–1239, San Francisco. Morgan Kaufmann.

Fisher, R. and Others (1936). The use of multiple measurements in taxonomic problems. In *Annals of Eugenics*, volume 7, pages 179–188.

Friedman, J. H., Bentley, J. L., and Finkel, R. A. (1977). An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3:209–226.

Goldberger, J., Roweis, S., Hinton, G., and Salakhutdinov, R. (2004). Neighbourhood components analysis. In *Advances in Neural Information Processing Systems*. MIT Press.

Gonzalez, T. (1985). Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306.

Gray, A. and Moore, A. (2001). 'N-Body' problems in statistical learning. *Advances in neural information processing systems*, pages 521–527.

Gray, A. and Moore, A. (2003). Nonparametric density estimation: Toward computational tractability. In *SIAM International Conference on Data Mining*, volume 2003.

Hinneburg, E., Aggarwal, C., Keim, D., and Hinneburg, A. (2000). What is the nearest neighbor in high dimensional spaces?

Holte, R. (1993). Very simple classification rules perform well on most commonly used datasets. *Machine learning*, 11(1):63–90.

Lang, D. (2009). *Astrometry.net: Automatic recognition and calibration of astronomical images*. PhD thesis, University of Toronto.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.

Moore, A. (1991). A tutorial on kd-trees. Extract from PhD Thesis. Available from `http://www.cs.cmu.edu/~awm/papers.html`.

Moore, A. (2000). *The anchors hierarchy: Using the triangle inequality to survive high dimensional data*. Citeseer.

Omohundro, S. (1989). *Five balltree construction algorithms*.

Pearson, K. (1901). On lines and planes of closest fit to systems of points in space. *Philosophical Magazine Series 6*, 2(11):559–572.

Russell, S. J., Norvig, P., Candy, J. F., Malik, J. M., and Edwards, D. D. (1996). *Artificial intelligence: a modern approach*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Shen, Y., Ng, A., and Seeger, M. (2006). Fast gaussian process regression using kd-trees. *Advances in neural information processing systems*, 18:1225.

Singh-Miller, N. (2010). *Neighborhood Analysis Methods in Acoustic Modeling for Automatic Speech Recognition*. PhD thesis, Massachusetts Institute of Technology.

Weinberger, K. and Saul, L. (2009). Distance metric learning for large margin nearest neighbor classification. *The Journal of Machine Learning Research*, 10:207–244.

Weinberger, K. and Tesauro, G. (2007). Metric learning for kernel regression. In *Eleventh international conference on artificial intelligence and statistics*, pages 608–615.

Xing, E., Ng, A., Jordan, M., and Russell, S. (2003). Distance metric learning with application to clustering with side-information. In *Advances in Neural Information Processing Systems*, pages 521–528.