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

1 Introduction **1**

2 Background **4**

2.1 Theoretical background 4

2.2 Related methods 6

3 Neighbourhood component analysis **8**

3.1 General presentation 8

3.2 Class-conditional kernel density estimation interpretation 10

3.3 Practical notes 12

3.3.1 Optimization methods 12

3.3.2 Initialization 13

3.3.3 Numerical issues 14

3.3.4 Regularization 15

3.3.5 Doing classification 15

3.3.6 Dimensionality annealing 16

4 Reducing the computational cost **17**

4.1 Sub-sampling 17

4.2 Mini-batches 18

4.3 Stochastic learning 21

4.4 Approximate computations 22

4.4.1 k-d trees 23

4.4.2 Approximate kernel density estimation 26

4.4.3 Approximate KDE for NCA 27

4.5 Exact computations 28

4.6 NCA with compact support kernels and background distribution . 30

Bibliography **33**

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 It has its origins in an unpublished technical report by **?** and since then it became standard textbook material (Russell et al., 1996; Mitchell, 1997; Bishop, 2006).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) is another method that has the goal of learning a Mahalobnis-like metric. It was developed with kNN in mind; from a practical point of view NCA can be viewed as a recommended additional step when doing classification with kNN.

3.1 General presentation

Because the goal is to enhance the kNN performance, the first idea the authors had was to maximize the leave one out cross validation performance: we take each point in the data set and try to classify it using kNN. Unfortunately, there does not exist an exact correlation between the linear transformation $\mathbf{A}$ and the nearest neighbours for a given point: a small perturbation of $\mathbf{A}$ might cause strong changes or, conversely, it might leave the neighbours unchanged. This means that any function of the neighbours is piecewise constant and discontinuous and, hence, hard to optimize.

So they proposed a function that behave better by introducing the concept of stochastic nearest neighbours. These are achieved by adding randomization to the classification process. In the classical case, the 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. They used a function a probability function that is reminiscent to the softmax activation used for neural network or to the generalized logistic function.

The probability that the point j is selected the nearest neighbour of the point

i is given by:

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

Also we have to define that $p_{ii} = 0$: point i cannot pick itself as the nearest neighbour. Note that these stochastic assignment are function of the distances and implicitly, they are depend on the point positions. Because the point position change with the linear transformation $\mathbf{A}$, it means that p_{ij} are function of $\mathbf{A}$ and, furthermore, p_{ij} are differentiable with respect to $\mathbf{A}$.

Because we want to improve the classification accuracy, we want to maximize the probability of our point i of being selected by points that belong to its true class (equivalently, the probability of the point of being correctly classified). This quantity is denoted by p_i:

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

By considering each point and the probability of belonging to its true class, we get the objective function:

$$f(\mathbf{A}) = \sum_{i=1}^{N} p_i \tag{3.3}$$

$$= \sum_{i=1}^{N} \sum_{j \in c_i} \frac{\exp(-d_{ij}^2)}{\sum_{k \neq i} \exp(-d_{ik}^2)}. \tag{3.4}$$

This can also be interpreted as the expected number of the correctly classified points.

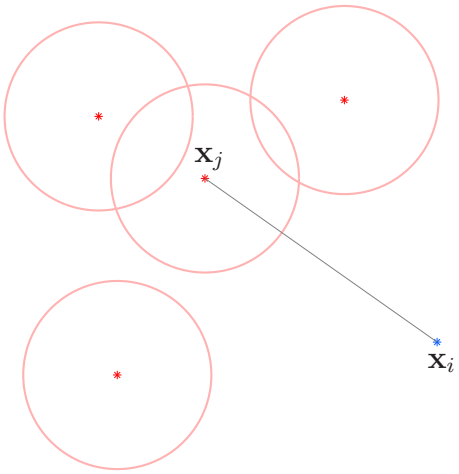
By differentiating with respect to A, we obtain the gradient:

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

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

The derivation of the gradient can be found in the appendix. The gradient offers us the possibility of optimizing the function using a gradient based solver.

NCA does not assume anything about the classes structure, permits very diverse metrics. It comes at some cost: the objective function is not convex. There appear some problems like initialization and optimization which are treated more in detail in Section 3.3. Another problem is the computational cost. For



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

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

each point we need to compute all the distances to other points. This scales in N^2 which is intractable for large data sets. Actually, the a function evaluation is in $\mathcal{O}(dDN^2)$. Chapter 4 presents various solutions to mitigate this problem.

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:

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

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

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 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:

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

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

However, we are interested 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:

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

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):

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

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.

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

.

The gradient of the objective function is the following:

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

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

- We have the first-order gradient information; so, we can use any gradient based method.
- Gradient descent is an iterative algorithm that uses first-order information to find a minimum of a function. At each step, it proposes to go in the steepest direction, *i.e.*, the direction with largest gradient. It is a very simple to implement method, but suffers from known convergence drawbacks. For example, there might appear the zig-zagging effect it depends on some critical parameters which make it difficult to use in practice. These are the learning rate η and the convergence conditions.
- Common choices for η are either using a constant step or decrease it gradually. Using a constant step size can make the algorithm diverge.
- There are different other heuristics that make this more efficient. One of these is the bold-driver trick.
- A learning rate decreasing procedure is to set $\eta = \frac{\eta_0}{t+t_0}$, where t represents the iteration number, η_0 and t_0 are constants. So, we end up with two hyper-parameters instead of one. There are various trick of tuning them. Leon Bottou mentions that a common value for η_0 is to be equal to the regularization variable and t_0 should then be selected such that the updates . In our implementation, because we did not use a regularization term, we set η_0 to a fix value and then we did an exponential search for t_0.

- One might also try to use momentum. For further useful advice on optimization one should consult BackProp.
- An improved version of the gradient descent is the conjugate gradients algorithm. This has better convergence.

3.3.2 Initialization

- Important, because the function is not convex; we obtain different final solutions by using different starting points. In general, it is good idea to try multiple initial seeds and then select that $\mathbf{A}$ that achieved the highest score.
- The simplest solution is to randomly choose values for the projection matrix $\mathbf{A}$. We also tried to initialize with other linear transformations whose computational cost is cheap compared to NCA: 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 the equations and further notes for these methods here:
  - 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 $\mathbf{S}_W^{-1}\mathbf{S}_B$.

    We note that, 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 3.17. The whitening transformation is then $\mathbf{A} = \mathbf{S}_W^{-1/2}$.

The methods above can be used for low-rank initialization as well: select only the top d most discriminative eigenvectors, *i.e.*, those that have the highest eigenvalues associated.

From our experiments, we can conclude that a good initialization reflects in a good solution. For small and simple data sets this is not that obvious, but for large data sets the differences are more pregnant.

3.3.3 Numerical issues

- There are situations when we end up in the undetermined case $\frac{0}{0}$ while computing the soft assignments p_i. This happens when the point $\mathbf{x}_i$ is far away from the rest of the points such that p_{ij} is 0 in numerical precision. To make an idea of how big the distance between two points need to be such the contribution is zero, we give the example for MATLAB: if `d>30` then `exp(-d^2)=0`

- Unfortunately, this happens often in practice. The most obvious case is when we are dealing with outliers. But there are also other scenarios. The scale of the axes can be large and the linear projection $\mathbf{A}$ cannot compensate for this. In this situation, we end up with almost all the points having this problem. This can also happen during training: a point might get "thrown away" during an update of $\mathbf{A}$.

- A simple way to alleviate this is by normalizing the data, *i.e.*, centering and making it unit variance:

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

In this case we have to store these characteristics and then apply them on the test data. The total projection will the combination of two successive

linear transformations:

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

- Because this does not guarantee the lack of numerical problem, we further corrected this issue by replacing each `NaN` with a very small value. This was inspired by Laurens van der Maaten implementation and we made use of it due to its simplicity. A more rigorous way of dealing with this would be by using the log-sum-exp trick. This idea is presented in detail in the appendix.

3.3.4 Regularization

- NCA favours 1NN and large scale linear projections $\mathbf{A}$. This is not usually the optimal thing.
- We can correct this using regularization, as pointed out in (Singh-Miller, 2010).

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

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

- On the other hand, it is not clear how to set the regularization parameter λ and how to choose the number of nearest neighbours for classification.

3.3.5 Doing classification

- We optimize the objective function 3.13. 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 mentioned in section , evaluating the objective function needs computing all the pairwise distances between the points. Also, the evaluating the gradient is expensive. This is done in $\mathcal{O}(N^2D^2)$ flops. So it is not trivial to successfully use NCA on large data sets.
- This chapter presents a wide palette of ideas and methods that can be applied to speed up the computations. Most of the methods rely on the fact that the learnt metric is low ranked.
- Every method presented can be regarded as an alteration of the original objective function. We basically change our objective function such that the new objective will have a reduced cost.

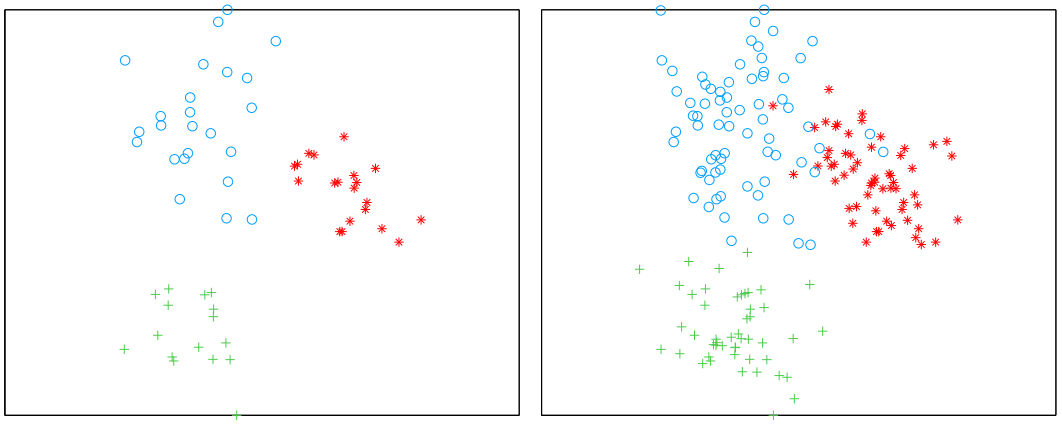
4.1 Sub-sampling

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 $\mathcal{O}(n^2D^2)$. After the projection matrix $\mathbf{A}$ is learnt, it can be applied to the whole data set and all the data points are used for classification.

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 distribution than the real data.



(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 misclassifications. The effects are even more acute if we use smaller sub-sets.

4.2 Mini-batches

The next obvious idea is to use sub-sets in an iterative manner, similar 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 $\mathcal{O}(n^2D^2)$ 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 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

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.

clustering (FPC; Gonzalez, 1985) and recursive projection clustering (RPC; Chalupka, 2011). Algorithm

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.

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

This 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 too learn.

So, as an alternative, RPC was especially developed to mitigate this problem. It constructs the clusters similarly to how the k-d trees are build (see

section). However instead of splitting the data set across axis aligned direction it chooses the splitting directions randomly (see 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. The complexity of this algorithm is $\mathcal{O}()$.

Algorithm 4.3 Recursive projection clustering

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

if $N < n$ **then**

New cluster: $i \leftarrow i + 1$.

return current points as a cluster: $\mathcal{M}_i \leftarrow \mathcal{D}$.

else

Randomly select two points $\mathbf{x}_j$ and $\mathbf{x}_k$ from $\mathcal{D}$.

Project all data points onto the line defined by $\mathbf{x}_j$ and $\mathbf{x}_k$. (Give equation?)

Select the median value $\tilde{\mathbf{x}}$ from the projected points.

Recurse on the data points above and below $\tilde{\mathbf{x}}$: RPC($\mathcal{D}_{>\tilde{\mathbf{x}}}$) and RPC($\mathcal{D}_{\leq\tilde{\mathbf{x}}}$).

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 periodically or we could cluster only once in the original space. 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.

Further notes:

- The learning rate can be updated using various heuristics as presented in the section related to optimization.
- The convergence can be tested in various ways: stop when there is not enough momentum in the parameter space, when the function value doesn't vary too much or by using early stopping or a maximum number of iterations. Discuss all of these in practical issues section.

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 $\{p_i\}_{i=1}^n$ using all the N points. It should be further stressed the difference to the mini-batches approach; there, the contributions 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:

$$f_{\text{sNCA}}(\mathbf{A}) = \sum_{i=1}^{n} p_i \tag{4.1}$$

$$\frac{\partial f_{\text{sNCA}}}{\partial \mathbf{A}} = \sum_{i=1}^{n} \frac{\partial p_i}{\partial \mathbf{A}} \tag{4.2}$$

This means that the theoretical cost of the stochastic learning method will scale 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.

repeat

Split data $\mathcal{D}$ into groups $\mathcal{M}_i$ of size n.

for all $\mathcal{M}_i$ **do**

Update parameter using gradient given by Equation 4.2:

$\mathbf{A} \leftarrow \mathbf{A} - \eta \frac{\partial f_{\text{sNCA}}(\mathbf{A}, \mathcal{M}_i)}{\partial \mathbf{A}}$.

Update learning rate η.

end for

until convergence.

Note: this idea might be used easily and robustly for on-line learning. Given a new point $\mathbf{x}_{N+1}$ we update $\mathbf{A}$ using the derivative $\frac{\partial p_{N+1}}{\partial \mathbf{A}}$.

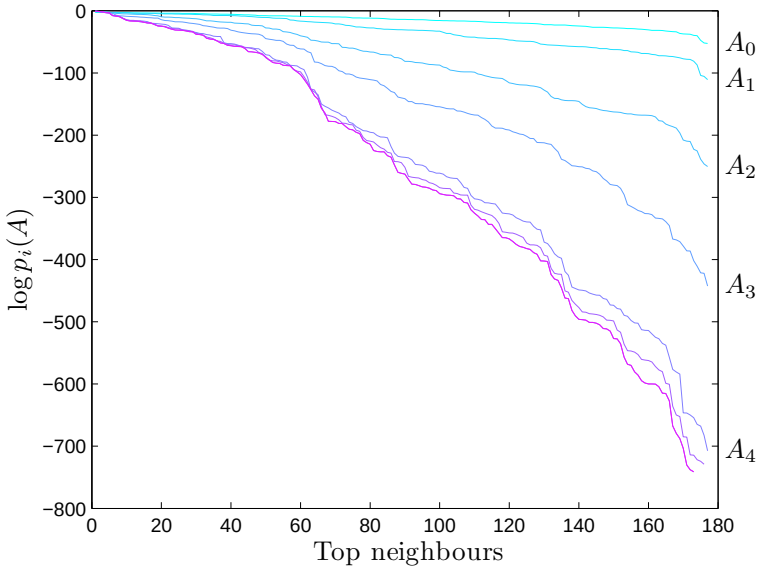


Figure 4.2: Evolution of the stochastic assignments p_{ij} during training for a given point $\mathbf{x}_i$.

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 $\mathbf{x}_i$.

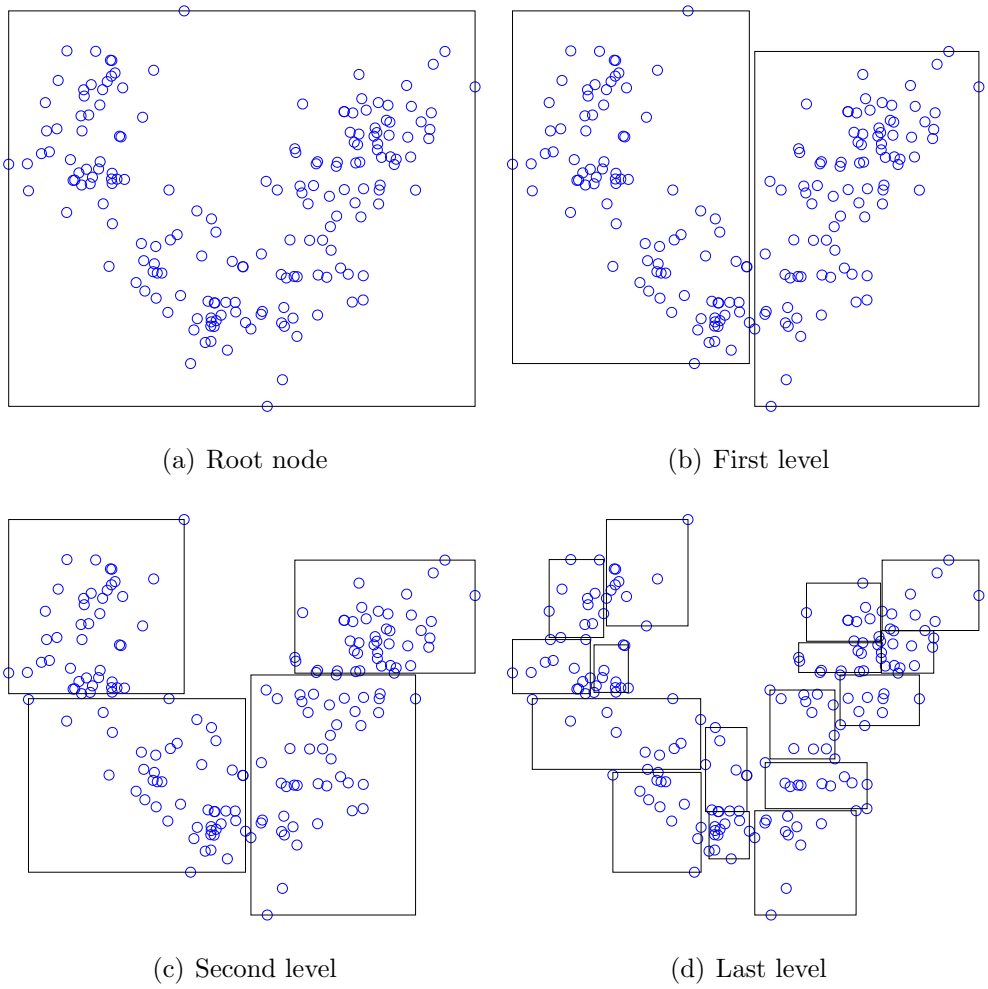
4.4.1 k-d trees

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:

$$\vec{d} \leftarrow \text{argmax}_d(\max_i x_{id} - \min_i x_{id}). \tag{4.3}$$

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:

$$\mathcal{D}_{\leq \tilde{x}_d} = \{\mathbf{x} \in \mathcal{D}_{\mathbf{x}_i} | x_d \leq \tilde{x}_d\}, \tag{4.4}$$

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:

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

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.

if $N < m$ **then**

Mark node i as leaf: `splitting_direction(i)=-1`.

Add points to leaf: `points(i)` $\leftarrow \mathcal{D}$.

return

end if

Choose direction $\vec{d}$ using Equation 4.3:

`splitting_direction(i)=`d.

Find the median value $\tilde{x}_d$ on the direction $\vec{d}$:

`splitting_value(i)=`$\tilde{x}_d$.

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 4.4 and 4.5.

Build left child `build_kdtree(`$\mathcal{D}_{\leq \tilde{\mathbf{x}}}$`,2*i)`.

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 (Gray and Moore, 2003; Shen et al., 2006).

- The goal is to compute $p(\mathbf{x})$. In the kernel density estimation, we estimate the unknown probability density function as follows: $p(\mathbf{x}_i) = \frac{1}{N} \sum_{j=1} N k(\mathbf{x}_i | \mathbf{x}_j)$. If the number of samples N is large then $p(\mathbf{x}_i)$ can approximated to high degree of precision by discarding lots of the data. A question still remains: how can we do this in a sensible manner?

- Given a query point $\mathbf{x}$ and a group of points G we can replace each individual contribution $k(\mathbf{x}|\mathbf{x}_j), \mathbf{x}_j \in G$, with $k(\mathbf{x}|\mathbf{x}_g)$. This last quantity is specific for the group and will be common for all points in G. There are two ways

to obtain a reasonable value for $k(\mathbf{x}|\mathbf{x}_g)$: either approximate $\mathbf{x}_g$ first and then compute $k(\mathbf{x}|\mathbf{x}_g)$; here $\mathbf{x}_g$ can be the mean of the points in the group. The second possibility is to approximate $k(\mathbf{x}|\mathbf{x}_g)$ directly; one possibility is $k(\mathbf{x}|\mathbf{x}_g) = \frac{\min_j k(\mathbf{x}|\mathbf{x}_j)+\max_j k(\mathbf{x}|\mathbf{x}_j)}{2}$. We prefer the second option, because it does not need to store mean of the points from G. Also, we will see that the minimum and maximum contributions have to be computed to decide whether to prune or not; so no further computational expense is introduced by using this approximation.

- We can see that the maximum error for each point in the group that is introduced by such an approximation is $\frac{\max_j k(\mathbf{x}|\mathbf{x}_j)-\min_j k(\mathbf{x}|\mathbf{x}_j)}{2}$. This can be controlled to be small; for example, we can decide to approximate only when the group is far away from the query point or when the kernel contribution is almost constant for the points within the group. However, it is better to consider the error relative to the total quantity we want to estimate. But we do not know the total sum before hand so we will use an upper bound $p(\mathbf{x}) < p_{\text{SoFar}}(\mathbf{x}) + \max k(\mathbf{x}|\mathbf{x}_j)$. This means that the order in which we accumulate it is important.

- By using k-d trees our groups will be hyper-rectangles. So in order to prune we test that the largest and the smallest contribution within the hyper-rectangle varies a little. We start at with a large group, the root of the tree, that contains all the points and then recurs on its children until there is no need to and we can approximate.

4.4.3 Approximate KDE for NCA

- NCA was formulated as a class-conditional kernel density estimation problem. So we evaluate $p(\mathbf{x}|c), \forall c$ and for each class we build a k-d tree. We will obtain an approximated version of the objective function. To obtain the gradient of this new objective function we can use Equation 3.14. The derivative of $p(\mathbf{Ax}|c)$ will be different only for those groups where we do approximations. So, for such a group we obtain the following gradient:

$$\begin{aligned}\frac{\partial}{\partial \mathbf{A}} \sum_{j \in G} k(\mathbf{Ax}|\mathbf{Ax}_j) &\approx \frac{\partial}{\partial \mathbf{A}} \frac{1}{2} \left\{ \min_{j \in G} k(\mathbf{Ax}|\mathbf{Ax}_j) + \max_{j \in G} k(\mathbf{Ax}|\mathbf{Ax}_j) \right\} \\ &= \frac{1}{2} \left\{ \frac{\partial}{\partial \mathbf{A}} k(\mathbf{Ax}|\mathbf{Ax}_c) + \frac{\partial}{\partial \mathbf{A}} k(\mathbf{Ax}|\mathbf{Ax}_f) \right\}, \end{aligned} \tag{4.6}$$

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. So the closest point gives the maximum contribution, while the farthest point contributes the least.

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.

for all classes c **do**

Build k-d tree for the points in class c.

end for

for all data points $\mathbf{x}_i$ **do**

for all classes c **do**

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:

end for

Compute soft probability $\hat{p}_i \equiv \hat{p}(c|\mathbf{A}\mathbf{x}_i) = \frac{\hat{p}(\mathbf{A}\mathbf{x}_i|c_i)}{\sum_c \hat{p}(\mathbf{A}\mathbf{x}_i|c)}$.

Compute gradient $\frac{\partial}{\partial \mathbf{A}}\hat{p}$ using Equation 3.14.

Update function value and gradient value.

end for

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).

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:

$$k_{\mathrm{CS}}(u) = \begin{cases} c\,(a^2 - u^2)^2 & \text{if } u \in [-a; +a] \\ 0 & \text{otherwise,} \end{cases} \tag{4.7}$$

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 4.9 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:

$$k_{\mathrm{CS}}(d_{ij}) = (1 - d_{ij}^2)^2\ \mathrm{I}(|d_{ij}| \leq 1), \tag{4.8}$$

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:

$$q_{ij} = \frac{k_{\mathrm{CS}}(d_{ij})}{\sum_{k \neq i} k_{\mathrm{CS}}(d_{ik})}. \tag{4.9}$$

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:

$$f_{\mathrm{CS}}(\mathbf{A}) = \sum_i \sum_{j \in c_i} q_{ij}. \tag{4.10}$$

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

$$\begin{aligned} \frac{\partial}{\partial \mathbf{A}} k_{\mathrm{CS}}(d_{ij}) &= \frac{\partial}{\partial \mathbf{A}} \left[(1 - d_{ij}^2)^2 \cdot \mathrm{I}(|d_{ij}| \leq 1)\right] \\ &= -4\mathbf{A}(1 - d_{ij}^2)\mathbf{x}_{ij}\mathbf{x}_{ij}^{\mathrm{T}} \cdot \mathrm{I}(|d_{ij}| \leq 1), \end{aligned} \tag{4.11}$$

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

The gradient of the new objective function is:

$$\frac{\partial f_{\mathrm{CS}}}{\partial \mathbf{A}} = 4\mathbf{A} \sum_{i=1}^{N} \left(q_i \sum_{k=1}^{N} \frac{q_{ik}}{1 - d_{ik}^2} \mathbf{x}_{ik}\mathbf{x}_{ik}^{\mathrm{T}} - \sum_{j \in c_i} \frac{q_{ij}}{1 - d_{ij}^2} \mathbf{x}_{ij}\mathbf{x}_{ij}^{\mathrm{T}} \right). \tag{4.12}$$

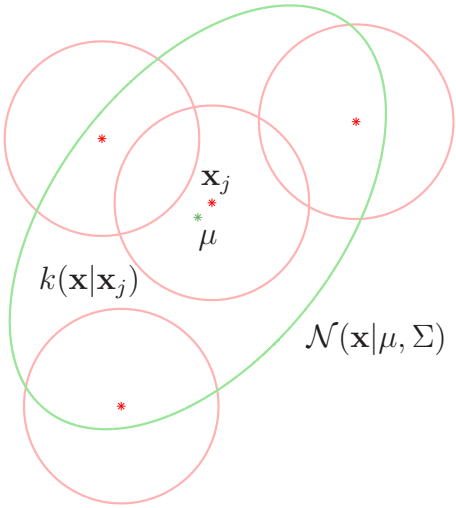


Figure 4.4: Neighbourhood component analysis with compact support kernels and background distribution. The main assumption is that each class is a mixture of compact support distributions $k(\mathbf{x}|\mathbf{x}_j)$ plus a normal background distribution $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$.

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 4.12.

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 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:

$$p(\mathbf{x}_i|c) = \frac{1}{N}\sum_{j\in c} k_{\text{CS}}(\mathbf{x}_i|\mathbf{x}_j), \tag{4.13}$$

where $k_{\text{CS}}(\mathbf{x}_i|\mathbf{x}_j) = k_{\text{CS}}(d_{ij})$ and is defined by Equation 4.7. 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:

$$p(\mathbf{x}_i|c) = \beta\mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c) + (1-\beta)\frac{1}{N_c}\sum_{j\in c} k_{\text{CS}}(\mathbf{x}_i|\mathbf{x}_j), \tag{4.14}$$

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 4.14 into the set of Equations 3.11, 3.13 and 3.14. 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 $k_{\text{CS}}(\mathbf{x}_i|\mathbf{x}_j)$ with respect to $\mathbf{A}$ is very similar to what is given in Equation 4.11. 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:

$$\frac{\partial}{\partial \mathbf{A}}\mathcal{N}(\mathbf{A}\mathbf{x}_i|\mathbf{A}\boldsymbol{\mu}_c, \mathbf{A}\boldsymbol{\Sigma}_c\mathbf{A}^{\mathrm{T}}) = \mathcal{N}(\mathbf{A}\mathbf{x}_i|\mathbf{A}\boldsymbol{\mu}_c, \mathbf{A}\boldsymbol{\Sigma}_c\mathbf{A}^{\mathrm{T}}) \times \{-(\mathbf{A}\boldsymbol{\Sigma}_c\mathbf{A}^{\mathrm{T}})^{-1}\mathbf{A}\boldsymbol{\Sigma}_c + \mathbf{v}\mathbf{v}^{\mathrm{T}}\mathbf{A}\boldsymbol{\Sigma}_c - \mathbf{v}(\mathbf{x} - \boldsymbol{\mu}_c)^{\mathrm{T}}\}, \quad (4.15)$$

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:

$$\frac{\partial}{\partial \beta}p(\mathbf{x}_i|c) = \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c) - \frac{1}{N_c}\sum_{j \in c} k_{\mathrm{CS}}(\mathbf{x}_i|\mathbf{x}_j). \quad (4.16)$$

Bibliography

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

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. (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.