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

Neighbourhood component analysis

1.1 General presentation

$$\begin{aligned} f(A) &= \frac{1}{N}\sum_{i=1}^{N} p_i = \frac{1}{N}\sum_{i=1}^{N}\sum_{j\in c_i} p_{ij} \\ &= \frac{1}{N}\sum_{i=1}^{N}\sum_{j\in c_i} \frac{e^{-d_{ij}^2}}{\sum_{k\neq i} e^{-d_{ij}^2}}, \end{aligned} \tag{1.1}$$

where $d_{ij}^2 = \|A\mathbf{x}_i - A\mathbf{x}_j\|^2$.

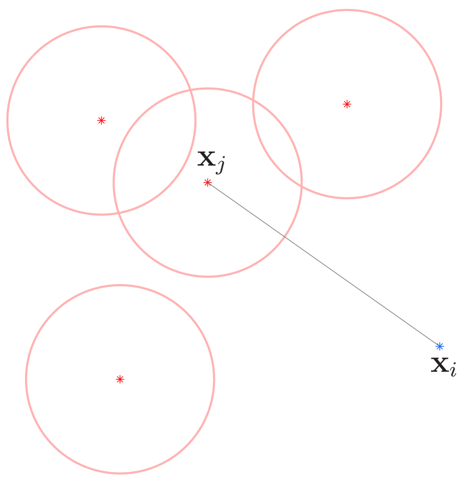
By differentiating with respect to A, we obtain the gradient:

$$\frac{\partial f}{\partial A} = 2A\sum_{i=1}^{N}\left(p_i \sum_{k=1}^{N} p_{ik}\mathbf{x}_{ik}\mathbf{x}_{ik}^{\mathrm{T}} - \sum_{j\in c_i} p_{ij}\mathbf{x}_{ij}\mathbf{x}_{ij}^{\mathrm{T}}\right), \tag{1.2}$$

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

1.2 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 2.4 and 2.5. 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).



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

Figure 1.1: Formulating NCA as a class-conditional kernel density estimation framework.

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:

$$p(\mathbf{x}_i|\mathbf{x}_j) = \mathcal{N}(\mathbf{x}_i|\mathbf{x}_j, \sigma^2 \mathrm{I}_D) \tag{1.3}$$
$$= \frac{1}{(2\pi)^{D/2}} \exp\left\{ -\frac{1}{2\sigma^2}(\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}}(\mathbf{x}_i - \mathbf{x}_j) \right\}. \tag{1.4}$$

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

$$p(\mathbf{A}\mathbf{x}_i|\mathbf{A}\mathbf{x}_j) \propto \exp\left\{ -\frac{1}{2\sigma^2}(\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\mathbf{A}(\mathbf{x}_i - \mathbf{x}_j) \right\}. \tag{1.5}$$

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:

$$p(\mathbf{x}_i|c) = \frac{1}{N_c}\sum_{\mathbf{x}_j\in c} p(\mathbf{x}_i|\mathbf{x}_j) \tag{1.6}$$

$$= \frac{1}{N_c}\sum_{\mathbf{x}_j\in c} \mathcal{N}(\mathbf{x}_i|\mathbf{x}_j, \mathrm{I}_D). \tag{1.7}$$

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:

$$p(c|\mathbf{x}_i) = \frac{p(\mathbf{x}_i|c)p(c)}{p(c|\mathbf{x}_i)} = \frac{p(\mathbf{x}_i|c)p(c)}{\sum_c p(\mathbf{x}_i|c)p(c)}. \tag{1.8}$$

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):

$$p(c|\mathbf{A}\mathbf{x}_i) = \frac{\frac{1}{N_c}\sum_{\mathbf{x}_j\in c}\exp\left\{-\frac{1}{2\sigma^2}(\mathbf{x}_i-\mathbf{x}_j)^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\mathbf{A}(\mathbf{x}_i-\mathbf{x}_j)\right\}}{\frac{1}{N_{c'}}\sum_{c'}\sum_{\mathbf{x}_k\in c'}\exp\left\{-\frac{1}{2\sigma^2}(\mathbf{x}_i-\mathbf{x}_k)^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\mathbf{A}(\mathbf{x}_i-\mathbf{x}_k)\right\}} \tag{1.9}$$

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.

$$f(\mathbf{A}) = \sum_i p(c_i|\mathbf{A}\mathbf{x}_i). \tag{1.10}$$

.

The gradient of the objective function is the following:

$$\frac{\partial f}{\partial \mathbf{A}} = \sum_i \left\{ \frac{\frac{\partial p(\mathbf{A}\mathbf{x}_i|c)}{\partial \mathbf{A}}p(c)}{\sum_c p(\mathbf{x}_i|c)p(c)} - \underbrace{\frac{p(\mathbf{A}\mathbf{x}_i|c)p(c)}{\sum_c p(\mathbf{A}\mathbf{x}_i|c)p(c)}}_{p(c|\mathbf{A}\mathbf{x}_i)} \frac{\sum_c \frac{\partial p(\mathbf{A}\mathbf{x}_i|c)}{\partial \mathbf{A}}p(c)}{\sum_c p(\mathbf{A}\mathbf{x}_i|c)p(c)} \right\}. \tag{1.11}$$

Chapter 2

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.

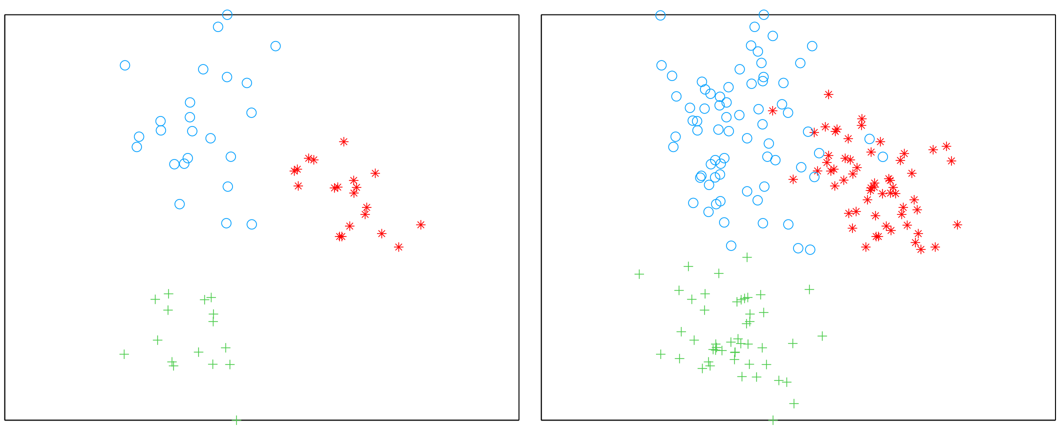
2.1 Sub-sampling

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 2.1: 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.

2.2 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 2.1, 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 2.1, 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 2.1 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 2.2 or 2.3 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 $\eta$.
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 2.2 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 2.3). 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 2.3 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.

2.3 Stochastic learning

2.4 Approximate computations

2.5 Exact computations

The counterpart of the approximate methods are the exact computations. These are also obtained at some cost. We have to change our model such that it allows efficient exact methods. Again motivated by the rapid decaying squared exponential function, some of the contributions are almost zero. The idea is just to use a compact support kernel instead of the squared exponential kernel. After this is done, there will be a large number of points that will lie outside the support of the kernel and their contribution will be exactly zero instead of a very small value in the previous case.

The kernel needs to be differentiable, so we will use the simplest polynomial compact support kernel that satisfies this requirement. Of course, any other kernel that satisfies this two requirements can be used. This is given by the following expression:

$$k(u) = \begin{cases} \frac{15}{16}(a^2 - u^2)^2 & \text{if } u \in [-a; +a] \\ 0 & \text{otherwise.} \end{cases} \tag{2.1}$$

The expression from equation 2.1 also integrates to one, so it can be used in the kernel density estimation context: $p(\mathbf{x}_i|\mathbf{x}_j) = k(d(\mathbf{x}_i; \mathbf{x}_j))$.

For the simple replacement of the exponential kernel, we have preferred a simplified version of equation 2.1:

$$k(u) = \begin{cases} (1 - u^2)^2 & \text{if } u \in [-1; 1] \\ 0 & \text{otherwise.} \end{cases} \tag{2.2}$$

The width of the kernel's support a will be integrated in the projection matrix $\mathbf{A}$, similarly as the widths for the Gaussian kernel are absorbed by $\mathbf{A}$ for the classic NCA.

$$p_{ij} = \frac{k(d_{ij})}{\sum_{k \neq i} k(d_{ik})}. \tag{2.3}$$

Objective function will be:

$$f_{\mathrm{CS}}(\mathbf{A}) = \sum_i \sum_{j \in c_i} p_{ij} \tag{2.4}$$

The gradient of the kernel:

$$\frac{\partial}{\partial \mathbf{A}} k(d_{ij}) = \frac{\partial}{\partial \mathbf{A}} \left[(1 - d_{ij}^2)^2 \cdot \mathrm{I}(|d_{ij}| \leq 1) \right] \tag{2.5}$$

$$= -4\mathbf{A}(1 - d_{ij}^2)^2 \mathbf{x}_{ij} \mathbf{x}_{ij}^{\mathrm{T}} \cdot \mathrm{I}(|d_{ij}| \leq 1), \tag{2.6}$$

where $d_{ij}^2 = d(\mathbf{A}\mathbf{x}_i; \mathbf{A}\mathbf{x}_j)$, $\mathbf{x}_{ij} = \mathbf{x}_i - \mathbf{x}_j$ and $\mathrm{I}(\cdot)$ is the indicator function that return 1 when its argument is true and 0 when its argument is false.

The gradient of the new objective function:

$$\frac{\partial f_{\mathrm{CS}}}{\partial \mathbf{A}} = 2\mathbf{A} \sum_{i=1}^{N} \left(p_i \sum_{k=1}^{N} \frac{p_{ik}}{1 - d_{ik}^2} \mathbf{x}_{ik} \mathbf{x}_{ik}^{\mathrm{T}} - \sum_{j \in c_i} \frac{p_{ij}}{1 - d_{ij}^2} \mathbf{x}_{ij} \mathbf{x}_{ij}^{\mathrm{T}} \right), \tag{2.7}$$

The main concern with this method is what happens when points lie outside the compact support of any other point in the data set. So care must be taken at initialisation. One way would be to initialise with a very small scale **A**. However, this means that there is no gain in speed for at least the first iterations. It might be better to use the principal components for initialisation.

Bibliography

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.

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

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.

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

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

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

Omohundro, S. (1991). *Bumptrees for efficient function, constraint, and classification learning.*