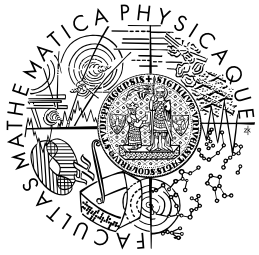

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Bc. Ondřej Švec

Evolutionary Algorithms for Data Transformation

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the master thesis: Mgr. Martin Pilát, Ph.D.

Study programme: Computer Science

Study branch: Artificial Intelligence

Prague 2017

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ..... signature of the author

Title: Evolutionary Algorithms for Data Transformation

Author: Bc. Ondřej Švec

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: Mgr. Martin Pilát, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract: In this work, we propose a novel method for a supervised dimensionality reduction, which learns weights of a neural network using an evolutionary algorithm, CMA-ES, optimising the success rate of the k-NN classifier. If no activation functions are used in the neural network, the algorithm essentially performs a linear transformation, which can also be used inside of the Mahalanobis distance. Therefore our method can be considered to be a metric learning algorithm. By adding activations to the neural network, the algorithm can learn non-linear transformations as well. We consider reductions to low-dimensional spaces, which are useful for data visualisation, and demonstrate that the resulting projections provide better performance than other dimensionality reduction techniques and also that the visualisations provide better distinctions between the classes in the data thanks to the locality of the k-NN classifier.

Keywords: distance metric learning, Mahalanobis distance, dimensionality reduction, evolutionary algorithms, visualisation, data transformation, neural networks

I would like to thank my team at Avast Software, especially Ing. Martin Vejmelka, Ph.D., for inspiring me to investigate this area of machine learning and for his guidance from the very beginning of my research. Additionally, I would like to acknowledge the academic and technical support of Avast Software, namely for the provided hardware used for evaluating various deep learning methods.

I would like to thank my supervisor, Mgr. Martin Pilát, Ph.D., for his guidance, the considerable time spent on consultations, proof reading, and for inspiring me to investigate the combination of metric learning and evolutionary algorithms.

I also want to thank my family for their continued support and encouragement during my Master studies and especially during the time spent working on this thesis.

Contents

1 Introduction **3**

1.1 Metric learning 5

1.2 Applications 7

1.3 Mahalanobis distance 8

1.4 Outline 12

2 Related work **14**

2.1 MMC Xing method 14

2.2 k-Nearest Neighbour classifier 15

2.3 Large Margin Nearest Neighbour 16

2.4 Neighbourhood Components Analysis 17

2.5 Local Fisher Discriminant Analysis 18

2.6 Evolutionary distance metric learning 19

2.7 Dimensionality Reduction 22

2.8 Summary of DML methods 23

3 Generalising Evolutionary Distance Learning **24**

4 Experiments and Examples **28**

4.1 Experimental settings 28

4.1.1 Datasets 28

4.1.2 Data preprocessing 29

4.2 Experiment: Classification 30

4.2.1 Hyperperameter search 31

4.2.2 Results 33

4.3 Experiment: Generalisation of EAs 42

4.3.1 Results 42

4.4 Experiment: jDE with log population 50

4.4.1 Results 50

4.5 Experiment: Learning time 52

4.5.1 Results 52

4.6 Experiment: Dimensionality reduction 57

4.6.1 Results 57

5 Conclusion **71**

6 Future work **72**

Bibliography **73**

List of Figures **79**

List of Algorithms **81**

List of Tables **82**

List of Abbreviations **83**

List of Notations **84**

Attachments **85**

1. Introduction

A metric is ubiquitous in machine learning. For instance, in classification, k-Nearest Neighbour (k-NN) classifier [Cover and Hart, 1967] uses a metric to find the nearest neighbours. In clustering, many clustering algorithms, including popular k-Means [Hartigan and Wong, 1979] algorithm, calculate distance between data points and group similar instances together. In information retrieval, when searching for the most relevant document, the documents themselves are ranked according to their similarity scores. Also security systems use a metric for face verification. Obviously, the performance of all these algorithms depends on the quality of the metric defined on the input space. The better the metric judges how close or how similar any two instances in the space are, the better the final machine learning algorithm will perform.

Several general purpose metrics exist, such as Euclidean distance, defined in Equation 1.1, or its generalised Minkowski distance, defined in Equation 1.2 and cosine similarity defined in Equation 1.3. These are all used for measuring distances between vectors of real numbers. Levenshtein distance measures edit distance between strings and χ^2 distance (in Equation 1.4) measures distances between histograms.

$$d_{\mathbf{2}}(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2 = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{1.1}$$

$$d_{\mathbf{p}}(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_p = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}} \tag{1.2}$$

$$\mathrm{sim}(\mathbf{x}, \mathbf{y}) = \cos\phi = \frac{\mathbf{x}\mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|} \tag{1.3}$$

$$\chi^2(\mathbf{x}, \mathbf{y}) = \sum_i \frac{(x_i - y_i)^2}{(x_i + y_i)} \tag{1.4}$$

These metrics, however, do not take any advantage of a prior knowledge of the task being solved or the structure of the data. Improved results are expected when the metric is designed for a particular task. Nevertheless, manually hand-crafting a metric is a strenuous and tedious process, which requires expert knowledge of the task and the data. This has led to the introduction of distance metric learning (DML) where the distance metrics or similarities are learnt from the data themselves.

The goal of metric learning is to adapt a pairwise metric function to the task at hand by using the training examples. This process requires almost no prior knowledge about the data and is very fast, compared to hand-crafting a metric

function, which may take months to develop. That is why learning a distance metric or a similarity metric directly from the data has obtained a lot of traction in the recent years.

Another important area of machine learning is dimensionality reduction, whose goal is to embed high-dimensional space in a low-dimensional space so that most of the information contained in the data is preserved (such as the variance or local distance measurements). Some metric learning algorithms do not scale well when the data are high-dimensional and so the dimensionality reduction can be used to transform data into a low-dimensional space just to save computational cost. Another dimensionality reduction application is visualisation, where the data are reduced into a 2-dimensional or 3-dimensional space, in which they can be plotted and inspected more easily. The reason why we include dimensionality reduction in this work is because some of the metric learning algorithms search for a projection into a low-dimensional space and thus can also be used for dimensionality reduction.

One of the most common unsupervised dimensionality reduction methods is Principal Component Analysis (PCA) [Jolliffe, 1986], which decomposes the dataset in a set of successive orthogonal components that explain a maximum amount of the variance inside of the data. Other well-known methods are for example Independent Component Analysis (ICA) [Comon, 1994] and Linear Discriminant Analysis (LDA) [Fisher, 1936].

A specific area of non-linear dimensionality reduction is called manifold learning, which is based on the idea that the dimensionality of many datasets is only artificially high but the intrinsic dimensionality is low and the data lie on an embedded low-dimensional manifold within the higher-dimensional space.

A popular manifold algorithm receiving a lot of traction recently is t-distributed Stochastic Neighbor Embedding (t-SNE) [Maaten and Hinton, 2008], which tries to preserve a local structure of the data and therefore is especially useful in visualisation. Other known manifold methods are Locally linear embedding (LLE) [Roweis and Saul, 2000], Multi-dimensional Scaling (MDS) [Kruskal and Wish, 1978] and many others. More dimensionality reduction methods were reviewed by [Fodor, 2002] and [Van Der Maaten et al., 2009].

In this thesis, we review existing state-of-the-art methods for learning global Mahalanobis distance metric and propose a new method using evolutionary algorithms (EA). In the experiments the methods are compared on several popular datasets to see how they improve classification. Learning the full Mahalanobis distance means learning d^2 parameters, however by restricting the methods to learn only a diagonal of the Mahalanobis distance metric, only d parameters are needed. Moreover, we explore the usage of metric learning algorithms for visuali-

sation of the data (dimensionality reduction to 2 dimensions) and compare them with traditional approaches.

1.1 Metric learning

The origins of metric learning can be traced back to [Short and Fukunaga, 1981], however, the first work that received a lot of attention was done by [Xing et al., 2002], where the metric learning is formulated as a convex optimisation problem. Since then, metric learning has obtained a lot of attention and appeared at ICML 2010[1], ECCV 2010[2] coferences and ICCV 2011[3], NIPS 2011[4] and ICML 2013[5] workshops.

The goal of the metric learning is to find a pairwise real valued metric function $d(x, y)$ for the given problem such that it leverages the most information it can using the training samples. Every metric learning algorithm has some defining properties. These properties are well described and categorised in a survey done by [Bellet et al., 2013]. All these properties and their categories are summarised in Figure 1.1.

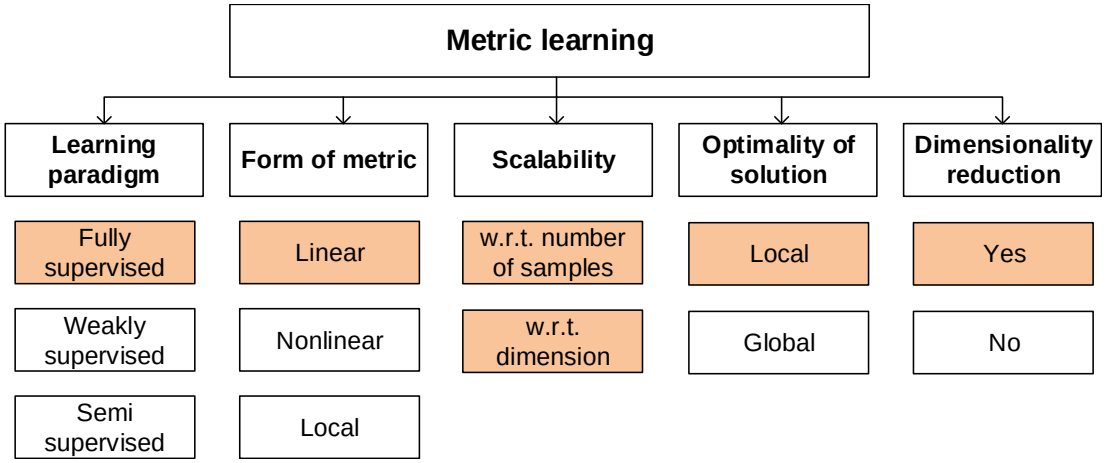

Figure 1.1: Properties of metric learning algorithms. This work focuses on the highlighted categories.

[1] http://www.icml2010.org/program.html
[2] http://www.ics.forth.gr/eccv2010/tutorials.php
[3] http://www.iccv2011.org/authors/workshops
[4] https://nips.cc/Conferences/2011/Schedule
[5] http://icml.cc/2013/?page_id=41

Learning paradigm can be categorized into three categories, depending on what kind of information about the data is avaiable:

- in *fully supervised* paradigm, every sample of the dataset $z_i = (x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$ is composed of the instance $x_i \in \mathcal{X}$ and its label (class) $x_y \in \mathcal{Y}$, which is a discrete and finite set of $|\mathcal{Y}|$ labels.
- in *weakly supervised* paradigm, the instances are unlabeled, but they come with a side information, typically in a form of constraints $\mathcal{S}, \mathcal{D}$ defined as

$$\mathcal{S} = \{(x_i, x_j) : x_i \text{ and } x_j \text{ should be similar}\}, \tag{1.5}$$

$$\mathcal{D} = \{(x_i, x_j) : x_i \text{ and } x_j \text{ should be dissimilar}\}. \tag{1.6}$$

  The metric learning algorithm tries to exploit these constraints to find the best parameters of a metric that best agree with these constraints. While obtaining labelled instances can be hard or costly, obtaining similar and dissimilar pairs is easier.
- in *semi-supervised* paradigm, there is some information (fully supervised or weakly supervised) about a portion of the data, but some (usually large) portion of the data comes with no information.

Form of metric is one of the defining features of a metric learning algorithm. It categorizes the metrics into three main categories:

- *Linear metrics* where the distance measures a length of a straight line between two points in Euclidean space. Typical example is Euclidean distance or Mahalanobis distance.
- *Nonlinear metrics* can capture nonlinear dependencies inside data, typical example is χ^2 histogram distance, which was defined in Equation 1.4.
- *Local metrics* where multiple different metrics are used for different parts of the input space. Local metrics can be linear or nonlinear and all metrics are usually trained in parallel.

Scalability is an important factor to be considered for any machine learning algorithm. Having a great method that is intractable to calculate for large amounts of data is useless. Therefore the algorithms need to scale well with respect to *number of samples*. Also, metric learning algorithms should scale well with respect to *number of dimensions* of the data. However, since many metric learning algorithms are trying to learn a $d \times d$ matrix, designing an algorithm that scales well with an increasing dimensionality is a challenge.

Optimality of the Solution refers to the ability of the algorithm to find a solution. Ideally the algorithm is capable of finding a *global optimum*, which means that the solution is always the best one of all feasible solutions available. This is usually the case for algorithms defined as convex optimisations. On the other hand, the algorithm might only be able to find *local optimum*, which means that no better solution exists in an immediate neighbourhood of the current solution. This is typical for nonconvex formulations of the algorithms.

Dimensionality reduction is an interesting "by-product" of metric learning algorithms that search for a projection of the data into a low-dimensional space, typically achieved by applying a strong regularisation on the algorithm so that the final solution is low-rank. An advantage of supervised dimensionality reduction via metric learning is that it tries to maximize the separation of labelled data. Dimensionality reduction has numerous applications, ranging from visualisation to more compact representations of the data.

This work is mainly focused on the metric learning algorithms that belong to the categories highlighted with orange background in Figure 1.1: fully supervised with a linear (Mahalanobis) metric that scale well and are capable of dimensionality reduction.

1.2 Applications

DML can potentially be beneficial whenever a metric between instances plays an important role. Recently, it has been successfully applied to many different problems, such as link prediction in networks [Shaw et al., 2011], state representation in reinforcement learning [Taylor et al., 2011], music recommendation [McFee et al., 2012], identity verification [Ben et al., 2012], webpage archiving [Law et al., 2012], cartoon synthesis [Yu et al., 2012], and assessing the efficacy of acupuncture [Liang et al., 2012].

There are three large fields that benefit from DML. In *Computer vision* DML is not only used to find distances between images and videos, but it has also been successfully applied in problems such as image classification [Mensink et al., 2012], object recognition [Frome et al., 2007]; [Verma et al., 2012], face recognition [Guillaumin et al., 2009b]; [Lu et al., 2014], visual tracking [Li et al., 2012]; [Jiang et al., 2012] or image annotation [Guillaumin et al., 2009a]. In *Information retrieval*, the objective is to try to provide the most relevant documents to a specific query, or rank documents and other media. This is achieved by training a metric between

the documents or a document and a query. Recent applications of DML in this field were done by [Lebanon, 2006], [Lee et al., 2008], [McFee and Lanckriet, 2010] and [Lim et al., 2013]. *Bioinformatics* is another large field where DML is used extensively to compare protein or DNA sequences. Typically edit distance is used to measure distance between these sequences, but adapting a metric to a specific problem can greatly improve the results as demonstrated in the works of [Xiong and Chen, 2006], [Saigo et al., 2006], [Kato and Nagano, 2010] and [Wang et al., 2012].

Kernel learning is another research area related to metric learning, but it is outside the scope of this work. Compared to metric learning (for example a Mahalanobis distance) where parameters of a given form of metric are to be learnt, kernel learning is typically nonparametric. The main problem with this approach is that it is transductive, which means that the distances between the pairs of the instances are learnt as values, not functions. Therefore the kernel learning can hardly be applied to new data. A recent survey on kernel learning was done by [Abbasnejad et al., 2012].

Multiple Kernel Learning (MKL) [Gönen and Alpaydın, 2011] is an extension of kernel learning, which is parametric and the goal is to learn a predefined set of kernels. Compared to kernel learning, MKL has efficient formulations and can be applied in the inductive setting (reasoning from observed data to make general rules). MKL is very popular because it can select an optimal kernel from a larger set of kernels and it can combine data that have different notion of similarity and require different kernels. MKL has been used in many applications such as recognition in video [Chen et al., 2013], object recognition in images [Bucak et al., 2014] or biomedical data fusion [Yu et al., 2010]. A recent survey on MKL was done by [Gönen and Alpaydın, 2011].

1.3 Mahalanobis distance

The problem with the Euclidean distance is that it does not account for variances in different directions and correlations between pairs of dimensions in the data. For normally distributed data, the distance from the mean can be specified by computing the z-score, defined as $z = \frac{x-\mu}{\sigma}$, where μ is the population mean and σ is the population standard deviation. The z-score of x can seen as a dimensionless quantity and can be interpreted as the number of standard deviations of x away from the mean.

Figure 1.2 demonstrates this problem in two dimensions: the figure visualises a dataset with both dimensions highly correlated and with the first dimension

highly variable. In the plot the coloured ellipses show the probability density function of the Gaussian mixture model, which is high for ellipses near the origin and low for ellipses further away. In the figure there are three red points: origin O at coordinates $[0, 0]$, A at coordinates $[2, 0]$, and B at coordinates $[0, 1]$.

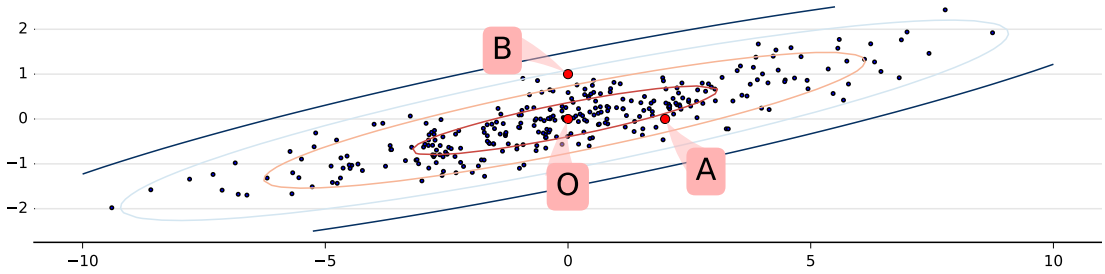

Figure 1.2: Correlated data centred around origin

It is clear that in Euclidean distance $d_2(A, O) > d_2(B, O)$. However, because the first axis has larger variance, it is clear that seeing point A is much more likely in our dataset than it is seeing point B because A lies on the edge of the first ellipsis, meanwhile point B lies beyond the edge of the second ellipsis. Using the terminology of the z-score, point B is more standard deviations away from the origin than point A.

The data can be uncorrelated using the covariance matrix $\boldsymbol{\Sigma}$, which is guaranteed to be a positive semi-definite (PSD) matrix. Matrix $\boldsymbol{A}$ is PSD if it fulfils

$$\boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x} \geq 0 \quad \forall \boldsymbol{x} \in \mathbb{R}^d, \boldsymbol{x} \neq 0.$$

Sometimes, however the covariance matrix is even positive definite (PD) and thus the Cholesky decomposition $\boldsymbol{\Sigma} = \boldsymbol{L}^\top \boldsymbol{L}$ can be used to obtain a lower triangular matrix $\boldsymbol{L}$. The inverse of this triangular matrix $\boldsymbol{L}$ can be used to remove the correlation from the data defined by $\boldsymbol{\Sigma}$. Figure 1.3 demonstrates the effect of multiplying the original data by $\boldsymbol{L}^{-1}$. In this figure with uncorrelated data, it is now clear that $d_2(A, O) < d_2(B, O)$ even in Euclidean distance.

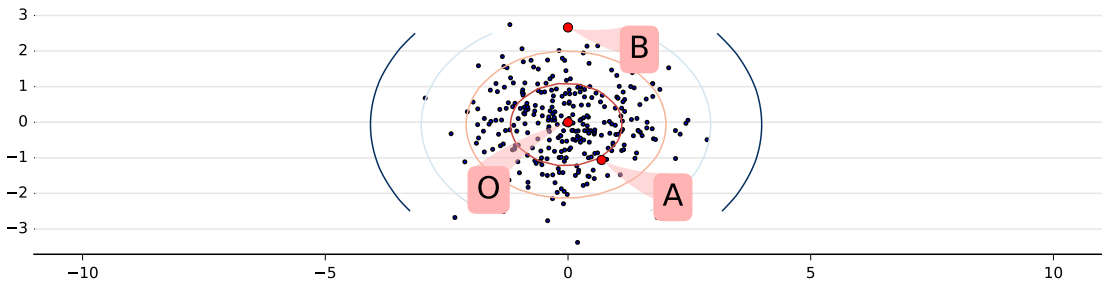

Figure 1.3: Uncorrelated data using Cholesky decomposition of covariance matrix

Many researchers tackle metric learning by adapting a Mahalanobis distance metric first described by [Mahalanobis, 1936] and defined in Equation 1.7, where

$\boldsymbol{\Sigma}$ is a covariance matrix of the data defined in Equation 1.8. However this name is now overloaded and Mahalanobis distance now stands for any metric parameterised by a matrix $\boldsymbol{M}$ as defined in Equation 1.9.

$$d_{Mahalanobis}(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \mathbf{y})} \tag{1.7}$$

$$\boldsymbol{\Sigma} = \mathrm{E}\left[(\mathbf{X} - \mathrm{E}[\mathbf{X}]) \, (\mathbf{X} - \mathrm{E}[\mathbf{X}])^{\mathrm{T}} \right] \tag{1.8}$$

$$d_{\boldsymbol{M}}(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})^\top \boldsymbol{M} (\mathbf{x} - \mathbf{y})} \tag{1.9}$$

The problem of correlation between dimensions does not arise in the Mahalanobis distance (Equation 1.7) because it decorrelates the data intrinsically before calculating the distance itself. If the Cholesky decomposition is applied directly inside of the Equation 1.7, then the Mahalanobis distance of the original data is equal to the Euclidean distance of the decorrelated data as shown in Equation 1.10.

$$\begin{aligned} d_{Mahalanobis}(\mathbf{x}, \mathbf{y}) &= \sqrt{(\mathbf{x} - \mathbf{y})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \mathbf{y})} \\ &= \sqrt{(\mathbf{x} - \mathbf{y})^T (\boldsymbol{L}\boldsymbol{L}^T)^{-1} (\mathbf{x} - \mathbf{y})} \\ &= \sqrt{(\mathbf{x} - \mathbf{y})^T \boldsymbol{L}^{-T} \boldsymbol{L}^{-1} (\mathbf{x} - \mathbf{y})} \\ &= \sqrt{(\boldsymbol{L}^{-1}\mathbf{x} - \boldsymbol{L}^{-1}\mathbf{y})^T (\boldsymbol{L}^{-1}\mathbf{x} - \boldsymbol{L}^{-1}\mathbf{y})} \\ &= d_{\mathbf{2}}(\boldsymbol{L}^{-1}\mathbf{x}, \boldsymbol{L}^{-1}\mathbf{y}) \end{aligned} \tag{1.10}$$

Therefore from Equation 1.10 it is possible to see that the Mahalanobis distance has the following properties:

1. it automatically accounts for the scaling of the coordinate axes
2. it corrects for correlation between different features
3. it reduces to the Euclidean distance for uncorrelated data with unit variance

The main weakness with the original Mahalanobis distance using the covariance matrix (Equation 1.7) is that it does not take the labels of the data instances into account (it can be thought of as "unsupervised" method). For this reason new metric learning algorithms exploiting additional information about data were developed and the term Mahalanobis distance now refers to not only decorrelation of the data using the Covariance matrix but to any linear transformation using different matrices as defined in Equation 1.9.

From a mathematical perspective, a metric is a function $d(x, y)$ that defines a distance or dissimilarity between two instances from the input space: $d : \mathcal{X} \times \mathcal{X} \mapsto [0, \inf)$. For a function d to be a metric, it has to fulfil these conditions:

$$d(\mathbf{x}, \mathbf{y}) \geq 0 \qquad \textit{nonnegativity} \tag{1.11}$$

$$d(\mathbf{x}, \mathbf{y}) = 0 \iff \mathbf{x} = \mathbf{y} \qquad \textit{identity of indiscernibles} \tag{1.12}$$

$$d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x}) \qquad \textit{symmetry} \tag{1.13}$$

$$d(\mathbf{x}, \mathbf{z}) \leq d(\mathbf{x}, \mathbf{y}) + d(\mathbf{y}, \mathbf{z}) \qquad \textit{triangle inequality} \tag{1.14}$$

In order for the Mahalanobis distance (Equation 1.9) to be a metric, the matrix $\mathbf{M}$ that parametrizes the distance needs to fulfil the conditions in Equations 1.11—1.14. A strictly PD matrix $\boldsymbol{M} \succ 0$ fulfils all conditions and therefore the Mahalanobis distance with a PD matrix is a distance metric.

Sometimes, however, it is not easy to guarantee the positive definiteness of a matrix. For this reason the *identity of indiscernibles* condition (Equation 1.12) is usually relaxed into a condition of *identity* (Equation 1.16) which allows $d(x, y) = 0$ for two distinct values $x \neq y$. Together with the rest of the conditions, this rule forms a new set of conditions (Equations 1.15—1.18) which together define a *pseudo-metric*. Therefore a Mahalanobis distance with a PSD matrix $\mathbf{M} \succeq \mathbf{0}$ is a pseudo-metric, which is enough in most applications.

$$d(\mathbf{x}, \mathbf{y}) \geq 0 \qquad \textit{nonnegativity} \tag{1.15}$$

$$d(\mathbf{x}, \mathbf{x}) = 0 \qquad \textit{identity} \tag{1.16}$$

$$d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x}) \qquad \textit{symmetry} \tag{1.17}$$

$$d(\mathbf{x}, \mathbf{z}) \leq d(\mathbf{x}, \mathbf{y}) + d(\mathbf{y}, \mathbf{z}) \qquad \textit{triangle inequality} \tag{1.18}$$

If the matrix $\boldsymbol{M}$ in the generalised Mahalanobis distance is PD, it can be decomposed using a Cholesky decomposition into $\boldsymbol{\Sigma} = \boldsymbol{L}^\top \boldsymbol{L}$ and then the same trick from Equation 1.10 can be used as shown in Equation 1.19. Therefore every Mahalanobis distance with a PD matrix can be replaced by a linear transformation of the original space using a matrix $\boldsymbol{L}$ and then calculating the Euclidean distance in the transformed space. This interesting feature of Mahalanobis distance allows us to look at metric learning algorithms also as a data transformation or data preprocessing.

$$
\begin{aligned}
d_{\boldsymbol{M}}(\mathbf{x}, \mathbf{y}) &= \sqrt{(\mathbf{x}-\mathbf{y})^T \boldsymbol{M} (\mathbf{x}-\mathbf{y})} \\
&= \sqrt{(\mathbf{x}-\mathbf{y})^T \boldsymbol{L}^T \boldsymbol{L} (\mathbf{x}-\mathbf{y})} \\
&= \sqrt{(\boldsymbol{L}\mathbf{x}-\boldsymbol{L}\mathbf{y})^T (\boldsymbol{L}\mathbf{x}-\boldsymbol{L}\mathbf{y})} \\
&= d_{\boldsymbol{2}}(\boldsymbol{L}\mathbf{x}, \boldsymbol{L}\mathbf{y})
\end{aligned} \tag{1.19}
$$

Note that if the $\boldsymbol{M}$ matrix is low rank, i.e., $rank(\boldsymbol{M}) = k < d$ then the linear projection using the matrix $\boldsymbol{L}$ will transform the data onto a lower-dimensional (k-dimensional) space. This allows for a more compact representation of the data and cheaper distance computations inside of the low-dimensional space.

Learning Mahalanobis distance comes with two challenges:

1. ensuring that the matrix $\boldsymbol{M}$ stays PSD during the learning process. A simple way to ensure this constraint is to use a projection onto a PSD cone by setting the negative eigenvalues to zero, which is described in [Qian et al., 2015]. Still, the eigenvalue decomposition takes $\mathcal{O}(d^3)$ and therefore is intractable for high-dimensional datasets.

2. learning a low-rank matrix $\boldsymbol{M}$ instead of the full-rank one. This would imply having the transformation to the low-dimensional space with all the advantages described earlier. Unfortunately, finding a minimum-rank solution of a semidefinite program is NP-hard as described in [Lemon et al., 2016].

1.4 Outline

This work extends upon the ideas from DML and applies them to the problem of dimensionality reduction and visualisation. The connection between dimensionality reduction and DML and the motivation for this work is quite simple. Given a generalised Mahalanobis distance $d_{\boldsymbol{M}}(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x}-\mathbf{y})^T \boldsymbol{M} (\mathbf{x}-\mathbf{y})}$ with a PD matrix $\boldsymbol{M}$ and its Cholesky decomposition $\boldsymbol{M} = \boldsymbol{L}^\top \boldsymbol{L}$, it holds that $d_{\boldsymbol{M}}(\mathbf{x}, \mathbf{y}) = d_2(\boldsymbol{L}\mathbf{x}, \boldsymbol{L}\mathbf{y})$, where d_2 is the Euclidean distance. Therefore, the matrix $\boldsymbol{L}$ is a mapping from an original space O to another space T. This notion can be generalised further to look for general mappings $f : O \to T$. An example of such a mapping is a linear mapping. In such a case the matrix $\boldsymbol{L}$ may have a lower rank and thus it maps the points of the original space to a lower-dimensional space. Another example of mapping may be a neural network, which makes a non-linear transformation of O to T. The distances of points in O are then computed as Euclidean distances of their projections to T. Mostly mappings to low-dimensional

spaces suitable for visualisation are discussed in this work, however, most of the ideas can be extended to higher dimensions, and thus to dimensionality reduction in general. The techniques described here are supervised, and the mappings seek to optimize the accuracy of the k-NN classifier applied in the low-dimensional space. The motivation is that in case the k-NN classifier has a high accuracy, instances from the same class are close together.

Current approaches to DML are described in Chapter 2, the notion of DML is generalised to any general mapping and our own method for classification and dimensionality reduction is proposed in Chapter 3. In Chapter 4 our method is compared to other state-of-the-art methods in several different tasks such as classification, visualisation, and time-performance. We conclude our work in Chapter 5 and suggest some ideas for further investigation and research in Chapter 6.

2. Related work

As already mentioned in Chapter 1, DML has been gaining traction in the recent years and it has found many applications in many fields as discussed in Section 1.2. The survey from [Bellet et al., 2013] is a comprehensive review of the supervised Mahalanobis distance learning methods. In the survey categorises DML methods into the following categories:

- *Linear methods* are the simplest methods, which are easy to optimise and are less prone to overfitting. Typical example is Mahalanobis distance.
- *Online methods* process the samples piece-by-piece, without having the entire input available from the start.
- *Multi-Task methods* learn the metric for multiple related tasks at once to improve the performance.
- *Nonlinear methods* no longer have linear metric; it is typically achieved by kernelisation of a linear method.
- *Local Metric methods* learn multiple metrics for different parts of the space.
- *Semi-Supervised methods* in are designed to work well in semi-supervised paradigm, where most of the instances do not have labels.

This work mostly focuses on linear DML methods because the rest of the categories extend these simpler methods and build on top of them.

The Mahalanobis distance originally proposed by [Mahalanobis, 1936] (Equation 1.7) uses a Covariance matrix of the data and can be seen as a first DML method. Even though it is unsupervised, it still learns from the data. Since then the term Mahalanobis distance has been generalised and is now used with any PSD matrix, not just Covariance, discussed in more detail in Section 1.3. This generalised Mahalanobis distance is now used in many linear DML methods.

The individual DML methods that were mentioned here will be described in the next sections with more detail.

2.1 MMC Xing method

The work of [Xing et al., 2002] is the first Mahalanobis distance learning algorithm leveraging information about the data. It is a weakly supervised method, which takes information in form of pairs of similar and dissimilar instances as defined in Equations 1.5 and 1.6. The authors formulate a convex optimisation problem

that tries to maximize the distance between dissimilar pairs of instances while keeping the sum of the distances between similar pairs as low as possible (≤ 1):

$$\begin{aligned} \max_{\boldsymbol{M} \in \mathbb{S}_+^d} \quad & \sum_{(x_i, x_j) \in \mathcal{D}} d_{\boldsymbol{M}}(x_i, x_j) \\ \text{s.t.} \quad & \sum_{(x_i, x_j) \in \mathcal{S}} d^2_{\boldsymbol{M}}(x_i, x_j) \leq 1 \\ & \boldsymbol{M} \succeq 0 \end{aligned}$$

This convex program is solved by using gradient ascend to maximize the distances between dissimilar instances, followed by projections to ensure the conditions of the program are held (matrix $\boldsymbol{M}$ must stay PSD and the distances between similar pairs must stay low).

2.2 k-Nearest Neighbour classifier

k-NN algorithm is a simple non-parametric method for classification and regression. In classification, an unknown sample is classified by a majority vote of k nearest training samples. The pseudocode for classification is shown in Algorithm 1. In regression, the output is an average of the values of k nearest neighbours.

The performance of the k-NN largely depends on the metric used. Some DML methods are designed to directly improve k-NN classifier effectiveness. Any metric can be used in the algorithm simply by changing the distance metric (Line 4 of Algorithm 1). Modifying the k-NN to use a custom metric is trivial even for faster implementations using tree data structures for nearest neighbour search.

Algorithm 1: Pseudocode for classification using k-NN

Data: set of labelled instances $\mathcal{Z} = \{\boldsymbol{z}_i = (\boldsymbol{x}_i, y_i) : \boldsymbol{x}_i \in \mathcal{X}, y_i \in \mathcal{Y}\}_{i=1}^{n}$, where $\mathcal{X}$ are training data and $\mathcal{Y}$ are their class labels

Input: unknown sample $\boldsymbol{x} \in \mathcal{X}$, number of neighbours k

Result: class label of the unknown sample $\boldsymbol{x}$

```
1 begin
2     for i ← 1 to m do
4         distances[i] ← compute distance d(z_i, x)
5     end
6     return majority class from the k smallest distances
7 end
```

2.3 Large Margin Nearest Neighbour

Large Margin Nearest Neighbor (LMNN) method designed by [Weinberger and Saul, 2009] is a very popular method that has been improved by work of many other researchers, such as Multiple-Metrics LMNN [Weinberger and Saul, 2008], Multi-task LMNN [Parameswaran and Weinberger, 2010] and Gradient-Boosted LMNN [Kedem et al., 2012]. The goal of this method is to improve the performance of the k-NN classifier, the idea is that k-nearest neighbours with the same class should stay together while separating the imposters (instances with a different class) by a large margin. The algorithm works by defining the constraints in a local way: optimising only over similar points that are also in a close neighbourhood of each other as defined in Equations 2.1—2.2. This neighbourhood is usually determined using Euclidean distance.

$$\mathcal{S} = \{(\boldsymbol{x}_i, \boldsymbol{x}_j) : y_i = y_j \text{ and } \boldsymbol{x}_j \text{ is in the } k\text{-neighbourhood of } \boldsymbol{x}_i\} \tag{2.1}$$

$$\mathcal{R} = \{(\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{x}_k) : (\boldsymbol{x}_i, \boldsymbol{x}_j) \in \mathcal{S}, y_i \neq y_k\} \tag{2.2}$$

The distance metric is then learnt using the convex program defined in Equations 2.3—2.4. The authors exploited the locality of the constraints and designed their own solver, able to solve the program even with billions of constraints. The μ parameter controls the pull/push trade-off which influences how much the algorithm will push the imposters away and pull the similar instances together.

$$\min_{\boldsymbol{M} \in \mathbb{S}_+^d} \quad (1-\mu) \sum_{(\boldsymbol{x}_i, \boldsymbol{x}_j) \in \mathcal{S}} d_{\boldsymbol{M}}^2(\boldsymbol{x}_i, \boldsymbol{x}_j) + \mu \sum_{i,j,k} \xi_{ijk} \tag{2.3}$$

$$\text{s.t.} \quad d_{\boldsymbol{M}}^2(\boldsymbol{x}_i, \boldsymbol{x}_k) - d_{\boldsymbol{M}}^2(\boldsymbol{x}_i, \boldsymbol{x}_j) \geq 1 - \xi_{ijk} \quad \forall (\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{x}_k) \in \mathcal{R} \tag{2.4}$$

Figure 2.1 illustrates how LMNN works. The left diagram shows neighbourhood of the points before training and the right diagram after training. The distance metric is optimised so that the points lie inside of a smaller neighbourhood, while the instances with different classes are pushed outside of this neighbourhood.

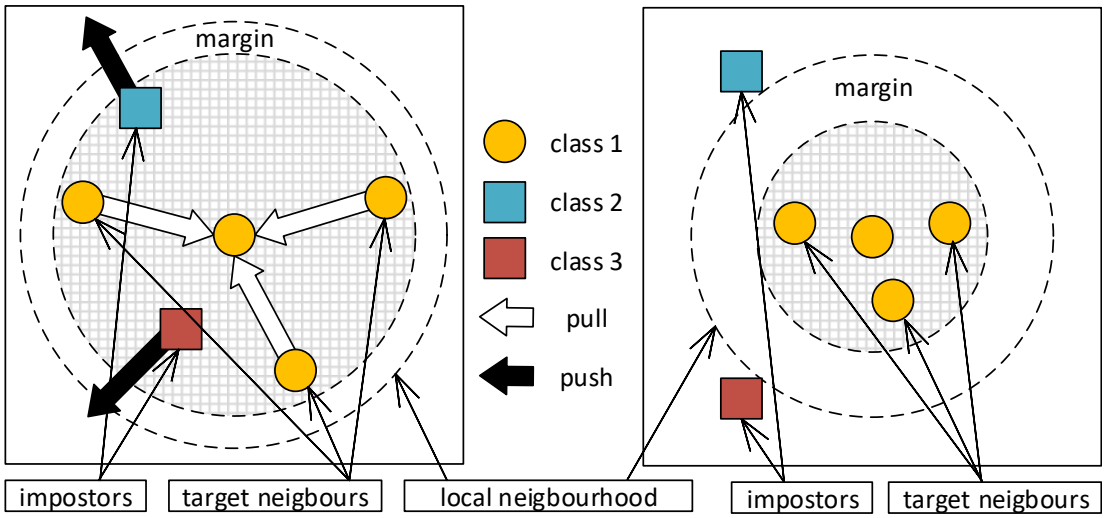

Figure 2.1: Illustration of the input space before and after training using LMNN

2.4 Neighbourhood Components Analysis

[Jacob Goldberger and Geoff Hinton, 2004] proposed another method designed to improve the performance of the k-NN classifier called Neighbourhood Components Analysis (NCA). The algorithm works by calculating the leave-one-out error for a stochastic version of the k-NN classifier which uses Mahalanobis distance. Instead of searching for k-nearest neighbours, it uses a softmax probability distribution, defined in Equation 2.5, which is used to weight all the neighbours while favouring the closer ones. Note that the hyperparameter k is no longer needed in this stochastic setting.

$$p_{ij} = \frac{\exp(-\|\boldsymbol{L}x_i - \boldsymbol{L}x_j\|_2^2)}{\sum_{l \neq i} \exp(-\|\boldsymbol{L}x_i - \boldsymbol{L}x_l\|_2^2)}, p_{ii} = 0 \tag{2.5}$$

The objective of NCA is to maximize the expected number of correctly classified points, defined in Equation 2.6. Thanks to the stochastic definition of the neighbour search, this function is continuous and differentiable. Therefore a gradient descend algorithm can be used to optimize it. The main disadvantage of this method is that the objective function is non-convex and therefore NCA optimisation can suffer from local minima and the performance depends on the

initialisation of matrix $\boldsymbol{L}$.

$$\max_{\boldsymbol{L}} \sum_{i} \sum_{j:y_j=y_i} p_{ij} \tag{2.6}$$

The algorithm is also valid if the matrix $\boldsymbol{L}$ is chosen to be rectangular and therefore NCA can also be used for dimensionality reduction by using the matrix $\boldsymbol{L}$ to project the original space into a lower-dimensional Euclidean space.

2.5 Local Fisher Discriminant Analysis

Local Fisher Discriminant Analysis (LFDA) [Sugiyama, 2007] is a combination of Linear Discriminant Analysis (LDA) method [Fisher, 1936] and Locality-Preserving Projection (LPP) method [He and Niyogi, 2003]. LDA works by maximising the amount of between-class variance relative to the amount of within-class variance. This might be problem when instances from one class form more clusters because these variances are calculated globally in LDA. LFDA improves this by introducing locality in the between-class and within-class variances using an affinity matrix from LPP. The affinity matrix is a square matrix where $\boldsymbol{A}_{i,j} \in [0, 1]$ and $\boldsymbol{A}_{i,j}$ is small when $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are close and it is large when they are far apart. If $\forall i, j : \boldsymbol{A}_{i,j} = 1$ then this algorithm degrades to the original LDA.

The within-class and between-class covariance matrices ($\widetilde{\boldsymbol{S}}^{(w)}$ and $\widetilde{\boldsymbol{S}}^{(b)}$, respectively) for LFDA are defined in Equations 2.7 and 2.8. Note that the affinity matrix $\boldsymbol{A}$ is not used to weight the instances from different classes in Equations 2.9 and 2.10 because these instances should be always separated.

$$\widetilde{\boldsymbol{S}}^{(w)} = \frac{1}{2} \sum_{i,j=1}^{n} \widetilde{\boldsymbol{W}}_{i,j}^{(w)} (x_i - x_j)(x_i - x_j)^\top \tag{2.7}$$

$$\widetilde{\boldsymbol{S}}^{(b)} = \frac{1}{2} \sum_{i,j=1}^{n} \widetilde{\boldsymbol{W}}_{i,j}^{(b)} (x_i - x_j)(x_i - x_j)^\top \tag{2.8}$$

$$\widetilde{\boldsymbol{W}}_{i,j}^{(w)} = \begin{cases} \boldsymbol{A}_{i,j}/n_l & \text{if } y_i = y_j = l \\ 0 & \text{if } y_i \neq y_j \end{cases} \tag{2.9}$$

$$\widetilde{\boldsymbol{W}}_{i,j}^{(b)} = \begin{cases} \boldsymbol{A}_{i,j}(1/n - 1/n_l) & \text{if } y_i = y_j = l \\ 1/n & \text{if } y_i \neq y_j \end{cases} \tag{2.10}$$

The final transformation matrix $\boldsymbol{L}$ for LFDA is defined exactly the same way as for LDA (Equation 2.11). This matrix transforms the space so that nearby data pairs with the same class are pulled closer together, meanwhile the pairs with different classes are pushed further apart. The data pairs with the same class but far away from each other in the original space are not forced closer

together. The transformation matrix can be obtained by solving the generalised eigenvalue decomposition of the scatter matrices and the authors offer an efficient implementation.

$$\boldsymbol{L}_{LFDA} = \underset{\boldsymbol{L}\in\mathbb{R}^{d\times r}}{\operatorname{argmax}} \left[\operatorname{tr}\left((\boldsymbol{L}^\top \widetilde{\boldsymbol{S}}^{(w)} \boldsymbol{L})^{-1} \boldsymbol{L}^\top \widetilde{\boldsymbol{S}}^{(b)} \boldsymbol{L} \right) \right] \tag{2.11}$$

Both LDA and LPP methods can be used for dimensionality reduction and so can be LFDA by restricting the matrix $\boldsymbol{L}$ to be rectangular.

2.6 Evolutionary distance metric learning

DML paired together with evolutionary algorithms has also been gaining more attention recently. For example the work of [Koloseni et al., 2012] uses evolutionary algorithms to select the best possible metric out of a predefined list to improve a classification error. The work of [Fukui et al., 2013] also uses evolutionary algorithms, but takes a different approach and instead of choosing a metric from a predefined list, they try to evolve a Mahalanobis distance matrix. Differential evolution is used in which an individual encodes the Mahalanobis matrix, which is symmetric and so it can be encoded as a real-valued vector of length $d(d+1)/2$.

Evolutionary algorithm is typically used when the objective function is non-differentiable and non-continuous. A typical schema of an evolutionary algorithm is shown in Figure 2.2: a population of individuals is randomly initiated and evaluated using the objective (fitness) function. Then the algorithm breeds new individuals through mutation and crossover, the individuals are evaluated and a new population of the individuals is selected. This whole process is repeated until a termination rule is met.

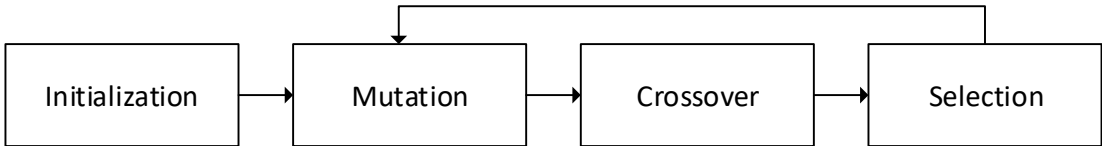

Figure 2.2: Schema of an evolutionary algorithm

Differential evolution is a special case of an evolutionary algorithm which is a stochastic, population based algorithm suitable for solving a real-valued objective function with many parameters. Differential evolution is an evolutionary algorithm with special mutation and crossover operations. The pseudocode for differential evolution is shown in Algorithm 2. It follows the schema from Figure 2.2 and the special crossover operation is defined on Line 8, where three different individuals are combined to create a new offspring.

Algorithm 2: Pseudocode for differential evolution

Input: fitness function f, crossover probability $CR \in [0, 1]$, differential weight $F \in [0, 2]$

Result: individual $\boldsymbol{x}$ that maximizes f

```
Initialise all agents with random positions in the search-space.
repeat
    for each agent x in the population do
        Randomly select three distinct agents a, b, c from the population
        Randomly select index R ∈ {1, ..., n}
        for i ← 1 to n do
            if rand() ≡ U(0,1) ≤ CR or i = R then
                y_i = a_i + F × (b_i − c_i)
            else
                y_i = x_i
            end
        end
        if f(y) < f(x) then replace x with y in the population
    end
until a termination criterion is met
```

Differential evolution algorithm has two hyperparameters apart from the size of the population: crossover probability CR and differential weight F. Differential evolution can be very sensitive to the choice of these two hyperparameters. [Fukui et al., 2013] uses an improved version of differential evolution called Self-adaptive Differential Evolution (jDE) [Brest et al., 2006], which alleviates this problem by encoding the hyperparameters CR and F inside of every single individual. When an offspring is created using the mutation and crossover, the CR and F parameters from the parent individuals are used. Therefore these two hyperparameters do not need to be specified for jDE and the algorithm itself is more robust.

To ensure that the evolved matrix is PSD during the evolution, the authors "repair" the matrix so that it is diagonally dominant using a rule defined in Equation 2.12. Diagonally dominant matrix with positive diagonal elements is PSD.

$$d_{i,j}^{repair} = \frac{m_{i,j}}{\sum_j |m_{i,j}|}, \ (i \neq j) \tag{2.12}$$

[Fukui et al., 2013] focuses on improving clustering algorithms and thus uses external clustering indexes as the fitness function. External clustering indexes,

compared to internal clustering indexes, use the class labels of the data for the evaluation. In their work they use purity, F-measure (harmonic mean of precision and recall of the classes assigned by clustering) and entropy. The authors also use a so called "neighbour relation" to improve the performance of the indexes by incorporating a weighted function inside of the indexes.

F-measure performed the best in their experimental results and so we chose to experiment with this index. F-measure is defined in Equations 2.13—2.18 where C is a cluster set, Y are class labels, $N_{s,i}$ is number of data points with class $s \in Y$ and in i^{th} cluster, $y(k)$ and $c(k)$ denote the class/cluster assignment for x_k.

$$Fme(s, C_i) = \frac{2 \cdot Prec(s, C_i) \cdot Rec(s, C_i)}{Prec(s, C_i) + Rec(s, C_i)} \tag{2.13}$$

$$Prec(s, C_i) = N_{s,i}/N_i \tag{2.14}$$

$$Rec(s, C_i) = N_{s,i}/N_s \tag{2.15}$$

$$N_{s,i} = \#\{x_k | y(k) = s, c(k) = C_i\} \tag{2.16}$$

$$N_s = \#\{x_k | y(k) = s \in Y\} \tag{2.17}$$

$$N_i = \#\{x_k | c(k) = C_i\} \tag{2.18}$$

The weighted versions of the previous definitions are shown in Equations 2.19—2.24. $h_{i,j}$ can be any monotonically decreasing function, the authors used Gaussian function $h_{i,j} = \exp(-r_{i,j}/\sigma)$, where $r_{i,j}$ is inter-cluster distance between C_i and C_j, and σ is a smoothing radius.

$$wFme(C) = \sum_{s \in Y} \frac{N_s}{N} \max_{C_i \in C} Fme(s, C_i) \tag{2.19}$$

$$Prec(s, C_i) = \widetilde{N}_{s,i}/\widetilde{N}_i \tag{2.20}$$

$$Rec(s, C_i) = \widetilde{N}_{s,i}/N_s \tag{2.21}$$

$$\widetilde{N}_{s,i} = \sum_{C_j \in C} h_{i,j} N_{s,j} \tag{2.22}$$

$$\widetilde{N}_i = \sum_{s \in Y} \widetilde{N}_{s,i} = \sum_{s \in Y} \sum_{C_j \in C} h_{i,j} N_{s,j} \tag{2.23}$$

$$\widetilde{N} = \sum_{C_i \in C} \widetilde{N}_i = \sum_{C_i \in C} \sum_{s \in Y} \sum_{C_j \in C} h_{i,j} N_{s,j} \tag{2.24}$$

After certain number of generations of the evolutionary algorithm, the individual with the best fitness value is selected and the matrix encoded in this individual is used for the final Mahalanobis distance.

2.7 Dimensionality Reduction

Metric learning can also be used for dimensionality reduction as already discussed in Chapter 1. This work mainly focuses on dimensionality reduction into two dimensions, where it is easy to visualise the data. NCA and LFDA were described earlier in this chapter. These algorithms perform dimensionality reduction by restricting the shape of the transformation matrix to be a rectangular with the number of columns corresponding to the dimension of the target space.

One of the most common and the simplest unsupervised dimensionality reduction method is PCA [Jolliffe, 2002] which decomposes the dataset in a set of successive orthogonal components that explain maximum amount of the variance inside of the data. The first principal component can be obtained by solving Equation 2.25; it can be shown that the solution is the eigenvector of $\boldsymbol{X}^\top \boldsymbol{X}$ corresponding to its largest eigenvalue.

$$\boldsymbol{w}_{(1)} = \underset{||\boldsymbol{w}||=1}{\operatorname{argmax}} \boldsymbol{w}^\top \boldsymbol{X}^\top \boldsymbol{X} \boldsymbol{w} \tag{2.25}$$

If all principal components are found and PCA is used to transform the data while keeping all the components (not performing dimensionality reduction), then the Euclidean distances between the samples in the new PCA-transformed space will equal their Mahalanobis distances in the observed-variable space.

Another dimensionality reduction method is t-SNE [Maaten and Hinton, 2008], which maps the points from the high-dimensional space to a low-dimensional one. The mapping aims to minimize the Kullback-Leibler divergence of the distribution of distances between points in the high and low-dimensional spaces.

t-SNE is gaining a lot of attention recently, although it does not provide any explicit mapping between the two spaces, therefore it cannot be used to map new data into the lower-dimensional space without re-running the whole method. In the experiments, however, we also show the visualisations obtained by t-SNE and compare them to those obtained by the method presented in this work.

2.8 Summary of DML methods

The most prominent DML methods were described in the previous sections. We experimented with many other DML methods from the survey by [Bellet et al., 2013], however, in our preliminary experiments we found that methods that were summarised here, perform the best.

Table 2.1 shows methods described in this chapter. These methods are used further in our experiments.

Name	Year	Supervision	Optimum	Regularizer	Notes
Covariance	1936	None	Global	None	Unsupervised
MMC	2002	Weak	Global	None	—
LMNN	2005	Full	Global	None	Improving k-NN
NCA	2004	Full	Local	None	Improving k-NN
LFDA	2007	Full	Global	None	—
jDE.wFme	2013	Full	Local	None	Evolutionary

Table 2.1: Summary of related metric learning methods

3. Generalising Evolutionary Distance Learning

The aim of this work is to find a mapping from the original space O of the input data, to a (potentially low-dimensional) space T. The mapping $f : O \to T$ is then used in the distance metric found by the algorithm. The evolved metric is defined as $d_f(x, y) = d_2(f(x), f(y))$, where d_2 is the Euclidean metric in the T space. It was already mentioned that in case f is represented by a lower triangular matrix $\boldsymbol{L}$, d_f represents a Mahalanobis distance with matrix $\boldsymbol{L}^\top \boldsymbol{L}$. However, the mappings may be more general. If the mapping is represented by a rectangular matrix, the technique can be used for dimensionality reduction. Moreover, we can also look for a non-linear mapping f, which is able to transform the input data in such a way that it is easier for the classifier to make the correct prediction.

Only mappings which can be represented as vectors or matrices of real numbers are considered in this work. Such mappings include both linear transformations and non-linear transformation such as neural networks (which can be represented as vectors containing their weights). The goal of the algorithm is to find a mapping that maximizes the performance of the classifier that follows the preprocessing step. In this work, k-NN classifier is considered, however different classifiers can also be used. The performance of the k-NN classifier is used as the fitness in the evolutionary algorithm. One of the reasons is that it is relatively fast, it takes $N \log N$ to build the underlying k-d tree structure with N training samples and then $M \log N$ to classify $\boldsymbol{M}$ testing samples. Another advantage of using the k-NN classifier is that the resulting mappings should also provide useful visualisations of the data. The k-NN classifier is local, and therefore works best in case instances from the same class are close together.

The k-NN classifier function is not differentiable and so we focus on finding the mapping using evolutionary algorithms. When the evolutionary algorithm is non-elitist (i.e. it can lose the best individual in the search), a concept called hall of fame (HOF) is used, which acts as a storage of the best individuals. It is updated with the best individuals after every generation and only keeps the best ones across all generations.

The high level pseudocode for an evolutionary algorithm evolving the mapping is shown in Algorithm 3. First, all parameters are initialised randomly along with an empty HOF. The individuals are vectors of real numbers that encode the mapping. In case of evolving the matrix $\boldsymbol{L}$ for the Mahalanobis distance, this is a vector of d^2 numbers. In case of evolving the neural networks, the individual

Algorithm 3: Evolving a mapping using an evolutionary algorithm

```
begin
    Initialise parameters of the algorithm
    Initialise empty HOF
    repeat
        Sample new individuals using the current parameters
        Evaluate fitnesses of the new individuals (described in Algorithm 4)
        Update best individuals in HOF
        Update the parameters of the evolutionary parameter
    until a termination criterion is met
    return the best individual from the HOF
end
```

contains all the weights from the network. Then the evolutionary algorithm is run until a stopping criterion is met. In each generation new individuals are sampled and the fitnesses of the individuals are evaluated. The HOF and the internal parameters of the evolutionary algorithm are updated. After the evolution, the best individual from the HOF is returned. The individual contains the weights that represent the corresponding mapping.

For evolving the mapping, we propose to use Covariance Matrix Adaptation Evolution Strategy (CMA-ES), which is an algorithm for non-linear non-convex optimisation problems in continuous domain. CMA-ES is considered to be state-of-the-art evolutionary algorithm for continuous optimisation. It does not require parameter tuning, the only hyperparameter is a population size, which comes with a reasonable default.

In our experiments the most common version (μ_W, λ)-CMA-ES is used, as described in [Hansen and Ostermeier, 2001]. In each iteration a weighted combination of μ best new candidates (out of λ) are used to update the distribution parameters. The reason for choosing CMA-ES is that it only needs a small population to perform well compared to other algorithms. Evaluation of the fitness function is time consuming in our setting and therefore the size of the population directly influences the training time.

Recommended population size for CMA-ES is $4 + \lfloor 3 \log d \rfloor$ [Hansen, 2006] whereas for jDE it is $10d$ [Brest et al., 2006], where d is number of parameters to evolve. This makes jDE intractable for learning Mahalanobis distance in high-dimensional space, because it has to evaluate $10 \cdot D(D+1)/2 \approx 5D^2$ (D is dimensionality of the data) fitnesses in each generation, while CMA-ES only needs to evaluate $4 + \lfloor 3 \log D^2 \rfloor \approx 4 + \lfloor 6 \log D \rfloor$ fitnesses. An experiment was

Algorithm 4: k-NN as a fitness function in an evolutionary algorithm

Data: set of labeled instances
Input: the individual $\boldsymbol{i}$ encoding the mapping
Result: the fitness value of the $\boldsymbol{i}$ individual

```
1 begin
2     Split instances into train-test sets
3     Extract the mapping to space T from the individual i
4     Map the instances to the space T
5     Initialise the k-NN classifier in the space T
6     Train the k-NN classifier using the train set
7     Perform the classification using the k-NN on the test set
8     return the success rate of the classification of the test set
9 end
```

performed to verify that jDE does not perform well with the same population size as CMA-ES. The experiment is discussed the next Chapter.

The most important step here is the evaluation of the fitness function, which is computed as the success rate of the k-NN classifier after transforming the data into space T. The pseudocode is shown in Algorithm 4.

It is important to note that the splitting of the data into training and testing sets is done for each fitness evaluation and therefore the sets are split differently for each individual. The idea behind this is that the evolution should find different individuals that generalise well.

DML algorithms can also be used for dimensionality reduction as already discussed in Chapter 2. Our mapping $f : O \to T$ is very general and can take any form. By evolving transformation matrix $\boldsymbol{L} \in \mathcal{R}^{m \times d}$, where $m < d$, a singular transformation matrix is obtained that will transform the original d-dimensional space into m-dimensional.

To perform a dimensionality reduction using the neural network, the shape and the activation functions need to be chosen beforehand, the only criteria is that the first layer has size d and the last layer has the size of the target dimension m. Without activation functions a fully connected neural network essentially reduces to a linear transformation and can therefore be used in Mahalanobis distance. The individuals in the evolution contain weights corresponding to all parameters of the neural network. Typical activation functions used in neural networks are sigmoid function ($\sigma(x)$), hyperbolic tangent (tanh) and rectified linear unit (ReLU), which are defined in Equations 3.1—3.3. Experiments with the activation functions and the depth of the neural network used for visualisation

are described in the next chapter.

$$\sigma(x) = 1/(1 + e^{-x}) \tag{3.1}$$

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 2\sigma(2x) - 1 \tag{3.2}$$

$$\mathrm{ReLU}(x) = \max(0, x) \tag{3.3}$$

The algorithm described here is similar to the algorithm described by [Fukui et al., 2013], however, there are some important differences:

- [Fukui et al., 2013] focuses on improving clustering and used the F-measure as the fitness function. On the other hand, we deal with classification and therefore we minimize the error rate of the k-NN classifier;
- a mapping is evolved instead of the matrix for the Mahalanobis distance, therefore the matrix $\boldsymbol{L}$ does not have to be "repaired" during the evolution. Matrix $\boldsymbol{M} = \boldsymbol{L}^\top \boldsymbol{L}$ can be used in Mahalanobis distance, because it is PSD:

$$\boldsymbol{x}^\top \boldsymbol{L}^\top \boldsymbol{L}\boldsymbol{x} = (\boldsymbol{L}\boldsymbol{x})^\top (\boldsymbol{L}\boldsymbol{x}) = (\boldsymbol{L}\boldsymbol{x}) \cdot (\boldsymbol{L}\boldsymbol{x}) \geq 0 \quad \forall \boldsymbol{x} \in \mathbb{R}^d, \boldsymbol{x} \neq 0 \tag{3.4}$$

- CMA-ES is used instead of jDE, therefore smaller population size can be used, leading to faster optimisation (recommended population size for CMA-ES is only $4 + \lfloor 3 \log d \rfloor$ [Hansen, 2006] whereas for jDE it is $10d$ [Brest et al., 2006], where d is number of parameters to evolve).

4. Experiments and Examples

Five different experiments were performed to assess the performance of the metric learning algorithm presented in this work. In Section 4.2 a metric for the k-NN classifier is learnt and the classification errors are compared on various datasets, which are described in Section 4.1.1. In the next experiment in Section 4.3 different evolutionary algorithms together with various fitnesses are compared and examined to see how well they evolve and generalise, while in Section 4.4 the performance of different population sizes of jDE is compared. In Section 4.5 the training times of the DML algorithms are measured and in the last experiment in Section 4.6 metric learning algorithms are used for visualisation and the resulting embeddings are compared with more traditional methods, PCA and t-SNE.

4.1 Experimental settings

In our experiments metric learning algorithms listed in Chapter 2 (LMNN, NCA, LFDA) are compared. They are implementated in Python *metric-learn* library[1]. We implemented a jDE-based method similar to [Fukui et al., 2013], and our algorithm described in Chapter 3. The implementation is an extension of *metric-learn* library and the source codes are available in our GitHub repository[2], but we hope the source codes will be accepted into the library. The scripts used to perform the experiments and create the results in this chapter are also available on GitHub[3].

4.1.1 Datasets

Datasets that were most commonly found in literature were used for the experiments, particularly in [Xing et al., 2002], [Weinberger and Saul, 2009], [Jacob Goldberger and Geoff Hinton, 2004] and [Fukui et al., 2013]. The most common among all these papers were *balance-scale*, *breast-cancer*, *iris*, *mice-protein*, *pima-indians*, *sonar* and *wine* datasets. All these datasets were obtained from a well-known UCI Machine Learning Repository[4]. Among these datasets *mice-protein* and *sonar* have high dimensionality, nevertheless, they are small in terms of number of samples. We also decided to add *digits6* and *digits10* datasets, also obtained from the same archive, which are relatively larger datasets containing

[1] `https://github.com/all-umass/metric-learn`

[2] `https://github.com/svecon/metric-learn/tree/ecml2017`

[3] `https://github.com/svecon/metric-learn-playground/tree/ecml2017`

[4] `https://archive.ics.uci.edu/ml/datasets/`

8×8 pixel images of digits with 6 and 10 classes, respectively.

In order to test the metric learning algorithms on a highly variable data, we also created an artificial dataset *gaussians* by sampling from four multinomial Gaussian distributions. Each Gaussian has a different centre, defined as a matrix in Equation 4.1 where each row corresponds to one centre. All Gaussians were generated sharing one covariance matrix, defined in Equation 4.2. Dimensions are uncorrelated, although, the first dimension has an enormous variability of 10^8.

$$means = \begin{pmatrix} 10 & 0 & 0 & 2 \\ 0 & 10 & 0 & -2 \\ 0 & 0 & 10 & -2 \\ 0 & 0 & -10 & 2 \end{pmatrix} \tag{4.1}$$

$$covariance = \begin{pmatrix} 10^8 & 0 & 0 & 0 \\ 0 & 100 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.2}$$

Table 4.1 summarizes datasets that were experimented on. The table shows number of samples, dimensionality and number of distinct classes for every dataset. All the datasets are meant for classification task: all of their instances belong to a class. All the datasets have a labelled instances and so they can be used in supervised methods. Most of the attributes of the datasets are real numbers, but some of them also contain integral values. The largest dataset in terms of number of samples and number of classes is *digits10* dataset with 1797 samples. However, the largest dataset in terms of number of dimensions is *mice-protein* dataset. Most of the datasets are relatively low in both number of samples, which is under a thousand, and their dimension is usually under 10.

4.1.2 Data preprocessing

Some of the datasets have missing attributes. There are several strategies for dealing with missing attributes, such as removing affected samples, filling missing values with a random value or zeros, their mean, median or the most frequent value. We chose to fill missing values using the mean for any given attribute as we found it performs well with all the chosen datasets.

Another difficulty with the datasets is that many of them do not have normalised attributes and their attribute values are unbounded and have high variance. From Table 4.2 it is clear that the most variable dataset is our artificial *gaussians* dataset, however, *pima-indians* and *wine* datasets are also highly variable with standard deviation of 58.37 and 215.75, respectively. The datasets were

Dataset	# samples	# dimensions	# classes
balance-scale	625	4	3
breast-cancer	699	9	2
digits6	1083	64	6
digits10	**1797**	64	**10**
gaussians	400	5	4
iris	150	4	3
mice-protein	1080	**77**	8
pima-indians	768	8	2
sonar	208	60	2
wine	178	13	3

Table 4.1: Datasets overall summary

Dataset	minimum	maximum	mean	std. deviation
balance-scale	1.00	5.00	3.00	1.41
breast-cancer	1.00	10.00	3.13	2.88
digits6	0.00	16.00	4.87	6.04
digits10	0.00	16.00	4.88	6.02
gaussians	**-33870.07**	**31033.73**	-7.65	**5285.16**
iris	0.10	7.90	3.46	1.97
mice-protein	-0.06	8.48	0.68	0.79
pima-indians	0.00	846.00	44.99	58.37
sonar	0.00	1.00	0.28	0.28
wine	0.13	1680.00	**69.13**	215.75

Table 4.2: Datasets samples statistics

normalised using standardisation defined in Equation 4.3, where μ is a mean vector for of each of the attributes and σ is a vector of their standard deviations.

$$\hat{X} = \frac{X - \mu}{\sigma} \tag{4.3}$$

4.2 Experiment: Classification

This experiment was designed to test how much a distance metric helps in classification task compared to standard Euclidean distance. The error rates were measured on k-NN classifier. It is one of the simplest and yet powerful classifiers and it is very easy to modify to use a custom metric, discussed in Section 2.2.

The metric learning algorithms tested in the experiments are

- Covariance (original Mahalanobis distance with a covariance matrix defined in Equation 1.7)
- LMNN, described in Section 2.3
- NCA, described in Section 2.4
- jDE with the F-measure fitness (jDE.fMe), which is based on the method presented by [Fukui et al., 2013]
- CMA-ES with the kNN fitness (CMAES.kNN) proposed in Chapter 3
- jDE with the kNN fitness (jDE.kNN)
- CMA-ES with the F-measure fitness (CMAES.fMe)
- Euclidean distance, acting as a baseline method

CMAES.kNN should be able to obtain good results in this experiment as the mapping created by the algorithm aims to optimize the performance of the classifier directly.

CMAES.fMe and jDE.kNN were tested as well, in order to tell, what is the effect of the different optimisation algorithm and what is the effect of the different fitness function.

To assess performance of the metric learning algorithms, the datasets listed in Section 4.1.1 were used. The datasets were split into training and testing sets using 10-fold cross validation. For each fold data were first preprocessed and standardised as described in Section 4.1.2, then the metric was trained using one of the methods and finally the metric was evaluated using k-NN classifier.

We also wanted to test how standardisation influences the learnt metrics and whether the metric learning algorithms can handle unnormalised data and so all of the experiments were run again, but without standardising the data. The experiments on preprocessed data are prefixed by "s:" in the tables with the results of the metric learning experiment.

4.2.1 Hyperperameter search

Every metric learning algorithm has a set of hyperparameters which highly influence the learnt metric. For each hyperparameter, sensible values from a reasonable range were selected. All hyperparameters for each algorithm are shown in Table 4.3. From the hyperparameter table it is clear that all the DML algorithms

(except Covariance and LFDA) are iterative (*max_iter* and *n_gen* hyperparameters) and the iterations were limited to 25, 50, 100, 250 and 1000 for all these iterative algorithms (only the maximum number of iterations was lowered to 500 for the evolutionary approaches on the larger datasets, as otherwise the computations take too long to finish). The transformer parameter means whether the algorithm uses the full matrix or it is restricted to a diagonal transformation matrix. Unfortunately the library did not offer learning metrics restricted to a diagonal matrix and so the restricted versions were tested only with the evolutionary methods that we implemented ourselves.

Method	Parameter name	Parameter values
k-NN classifier	n_neighbors	1, 2, 4, 8, 16, 32, 64, 128
LMNN	k	1, 2, 4, 8, 16, 32
	regularization	0.1, 0.5, 0.9
	max_iter	50, 250, 500, 1000
	learn_rate	1e-7, 1e-8, 1e-9
NCA	max_iter	50, 250, 500, 1000
	learn_rate	0.1, 0.01
LFDA	metric	weighted, orthonormalized
CMAES.kNN	transformer	full, diagonal
	n_gen	50, 100, 250, 1000
	knn_neighbors	1, 4, 8, 16
	knn_weights	uniform, distance
jDE.kNN	transformer	full, diagonal
	n_gen	50, 100, 250, 1000
	knn_neighbors	1, 4, 8, 16
	knn_weights	uniform, distance
CMAES.fMe	transformer	full, diagonal
	n_gen	50, 100, 250, 1000
jDE.fMe	transformer	full, diagonal
	n_gen	50, 100, 250, 1000

Table 4.3: Values of hyperparameters used for each algorithm

Even for these sensibly chosen hyperparameter values there are thousands of possible combinations for every single algorithm and thus thousands of different models would have to be trained. Considering LMNN as an example: there are 10 folds, 8 possible values for the final k-NN classifier hyperparameter, LMNN itself has 4 distinct hyperparameters with 6, 3, 4 and 3 values, which means there would be $10 * 8 * (6 * 3 * 4 * 3) = 17280$ different models to train just for LMNN

if a simple grid search was used. This is a major problem because some of the algorithms take a long time to train even on small datasets and trying that many combinations would take too long to test.

It can be noticed, however, that the metric does not need to be retrained for every hyperparameter of the final k-NN classifier. Instead the metric can be trained first and only then k-NN classifier can be trained and tested for each of the hyperparameters. Therefore, we designed our version of grid search that is hierarchical in a sense that it will train a metric with all combinations of hyperparameters first and only then it will evaluate these metrics using k-NN with its own hyperparameters. There will still be the same total number of models, however, there will be 8 times less evaluations (number of the final classifier hyperparameter combinations) for every metric learning algorithm, which roughly translates to 8 times less computation time as training a k-NN classifier is negligible compared to training DML algorithm.

The population sizes for CMA-ES and jDE were set to the recommended values ($4 + \lfloor 3 \log d \rfloor$ and $10d$, respectively). This gives an advantage to jDE in terms of number of fitness evaluations, however, in experiment in Section 4.4 we show that jDE does not perform well with the smaller population size.

4.2.2 Results

The error rates and their standard deviations (across 10 runs from the cross validation) of the final k-NN classifier for every dataset and every method are presented in Table 4.4. If the method name is prefixed with "s:" it means the dataset was standardised as described in Section 4.1.2. The best average performance for each dataset is in bold. NCA did not finish on the 3 largest datasets due to the memory limitations and so the corresponding cells are marked with "Memory".

Some other methods did not finish because a numerical error occurred, these are marked as "Error". The Covariance method failed for some datasets because the matrix was not PD and so it was not possible to use Cholesky decomposition. LFDA also failed for some datasets because a factorisation of a matrix failed during the calculation. NCA failed for *gaussians* and *mice-protein* datasets because the computation was unstable and the numbers overflowed to infinity during the computation.

The results are also plotted in Figures 4.1 and 4.2 using the box plots. Each dataset has its own graph where all the methods that finished correctly are drawn with their specific colour using the box plot graph. The median success rate is marked with a thick black line, the area of the box is an interquartile range (IQR)

defined as the third quartile minus first quartile: $IQR = Q_3 - Q_1$. The whiskers extend $1.5 * IQR$ to both sides of the box plot and anything beyond whiskers is considered to be an outlier value.

The results in the Table 4.4 and Figures 4.1 and 4.2 indicate that the standardisation of the data is generally a good idea, only datasets *digits6*, *digits10* and *iris* have the lowest error rates when unstandardised, although the standardised versions follow up with such a small margin that it is not a statistically significant difference. On the other hand, standardising data can have a large impact as can be seen on *gaussians*, *sonar* and *wine* datasets, where the difference between standardised and unstandardised data can be tens of percents. It shows that metric learning algorithms cannot fully replace the data standardisation.

Overall, three methods perform very well across all datasets: LMNN with 6 best results, followed by CMAES.kNN with 4 best results and LFDA with one best result. NCA method was not best in any dataset, but it closely followed the best methods. The differences between these methods are small and statistically insignificant. The statistical significance tables for all methods are attached in Attachments (Tables 6.1—6.10). The worst method is clearly Covariance algorithm, possibly because it is an unsupervised method. It is also important to note that the Euclidean distance (a baseline) is outperformed by a metric learning algorithm on almost every dataset. This suggests metric learning algorithms can have a large impact in the machine learning tasks.

When comparing jDE and CMA-ES, it can be noticed that the type of the fitness (F-measure or k-NN) affects the performance more than the optimisation algorithm. However, jDE has an advantage in the population size, therefore we believe CMA-ES is overall better than jDE for this task, as it has similar performance with much lower number of fitness evaluations. It is interesting to note that both jDE.fMe and jDE.kNN won on the artificial *gaussians* dataset, however, they were closely followed by CMAES.kNN method. CMAES.fMe and jDE.fMe methods perform the worst on most of the datasets.

	balance-scale	breast-cancer	digits6	digits10	gaussians
Euclidean	10.56 ± 1.07	3.58 ± 2.15	$\mathbf{0.09 \pm 0.27}$	1.22 ± 0.86	71.00 ± 5.72
s:Euclidean	9.92 ± 1.18	3.43 ± 1.72	0.55 ± 0.74	2.06 ± 0.90	0.75 ± 1.15
Covariance	10.09 ± 1.55	5.29 ± 1.79	Error	Error	0.75 ± 1.15
s:Covariance	10.09 ± 1.38	5.29 ± 1.79	Error	Error	0.75 ± 1.15
LMNN	5.78 ± 2.25	2.72 ± 1.63	$\mathbf{0.09 \pm 0.27}$	$\mathbf{0.94 \pm 1.03}$	71.00 ± 5.72
s:LMNN	5.45 ± 2.42	3.00 ± 2.07	0.28 ± 0.42	1.00 ± 0.65	0.25 ± 0.75
NCA	4.02 ± 2.12	3.29 ± 2.03	Memory	Memory	Error
s:NCA	4.03 ± 2.26	3.15 ± 1.66	Memory	Memory	0.50 ± 1.00
LFDA	7.53 ± 2.73	3.01 ± 2.08	Error	Error	0.75 ± 1.15
s:LFDA	7.38 ± 2.25	$\mathbf{2.43 \pm 1.57}$	Error	Error	0.75 ± 1.15
CMAES.kNN	3.55 ± 2.20	2.58 ± 1.67	0.18 ± 0.36	1.00 ± 0.89	3.25 ± 3.54
s:CMAES.kNN	$\mathbf{3.38 \pm 2.37}$	2.58 ± 1.90	0.64 ± 0.72	1.61 ± 0.91	0.50 ± 1.00
CMAES.fMe	9.61 ± 2.31	3.00 ± 1.49	0.64 ± 0.71	1.56 ± 0.82	35.50 ± 28.76
s:CMAES.fMe	9.75 ± 3.11	3.15 ± 2.28	1.20 ± 1.17	3.12 ± 1.40	0.75 ± 1.15
jDE.fMe	8.80 ± 2.78	3.15 ± 1.55	0.46 ± 0.62	2.56 ± 0.94	32.00 ± 27.11
s:jDE.fMe	8.66 ± 3.03	3.01 ± 1.76	0.93 ± 1.10	3.17 ± 1.47	$\mathbf{0.00 \pm 0.00}$
jDE.kNN	3.55 ± 2.20	2.72 ± 1.86	0.55 ± 0.73	1.55 ± 1.02	0.75 ± 1.15
s:jDE.kNN	3.55 ± 2.20	2.86 ± 1.57	0.84 ± 1.40	2.50 ± 1.06	$\mathbf{0.00 \pm 0.00}$

	iris	mice-protein	pima-indians	sonar	wine
Euclidean	2.67 ± 3.27	0.47 ± 0.64	24.61 ± 3.84	17.88 ± 11.43	22.80 ± 9.19
s:Euclidean	3.33 ± 3.33	0.38 ± 0.63	24.74 ± 3.91	13.50 ± 10.10	1.64 ± 2.50
Covariance	8.00 ± 6.53	Error	26.43 ± 3.84	17.70 ± 6.90	6.21 ± 6.78
s:Covariance	8.00 ± 6.53	Error	26.43 ± 3.84	17.70 ± 6.90	6.21 ± 6.78
LMNN	$\mathbf{1.33 \pm 2.67}$	$\mathbf{0.00 \pm 0.00}$	24.09 ± 5.15	13.53 ± 6.18	3.83 ± 4.24
s:LMNN	2.00 ± 3.06	$\mathbf{0.00 \pm 0.00}$	23.70 ± 4.11	$\mathbf{9.19 \pm 6.67}$	$\mathbf{0.56 \pm 1.67}$
NCA	2.00 ± 3.06	Memory	Error	14.41 ± 8.86	Error
s:NCA	2.00 ± 3.06	Memory	22.53 ± 4.19	12.00 ± 5.42	2.29 ± 3.75
LFDA	2.00 ± 3.06	Error	23.70 ± 3.28	14.37 ± 7.02	1.64 ± 2.50
s:LFDA	2.00 ± 3.06	Error	23.06 ± 4.71	13.50 ± 10.10	1.11 ± 2.22
CMAES.kNN	$\mathbf{1.33 \pm 2.67}$	$\mathbf{0.00 \pm 0.00}$	23.05 ± 4.03	13.48 ± 8.57	3.37 ± 4.55
s:CMAES.kNN	2.00 ± 3.06	0.10 ± 0.29	$\mathbf{21.48 \pm 3.13}$	10.17 ± 6.39	2.25 ± 3.72
CMAES.fMe	$\mathbf{1.33 \pm 2.67}$	0.65 ± 1.10	23.83 ± 4.11	17.84 ± 6.20	12.90 ± 8.52
s:CMAES.fMe	2.00 ± 4.27	0.47 ± 0.88	21.61 ± 4.03	10.07 ± 7.25	2.82 ± 2.83
jDE.fMe	2.00 ± 3.06	0.37 ± 0.45	23.30 ± 3.58	16.50 ± 9.31	5.59 ± 5.05
s:jDE.fMe	3.33 ± 4.47	0.38 ± 0.63	22.40 ± 4.69	18.30 ± 7.41	2.19 ± 3.66
jDE.kNN	2.00 ± 3.06	0.10 ± 0.29	23.83 ± 3.76	14.12 ± 8.75	4.45 ± 3.34
s:jDE.kNN	2.67 ± 3.27	0.19 ± 0.37	23.18 ± 3.89	12.96 ± 8.01	2.81 ± 3.75

Table 4.4: Error rates (in %) for the classification experiment

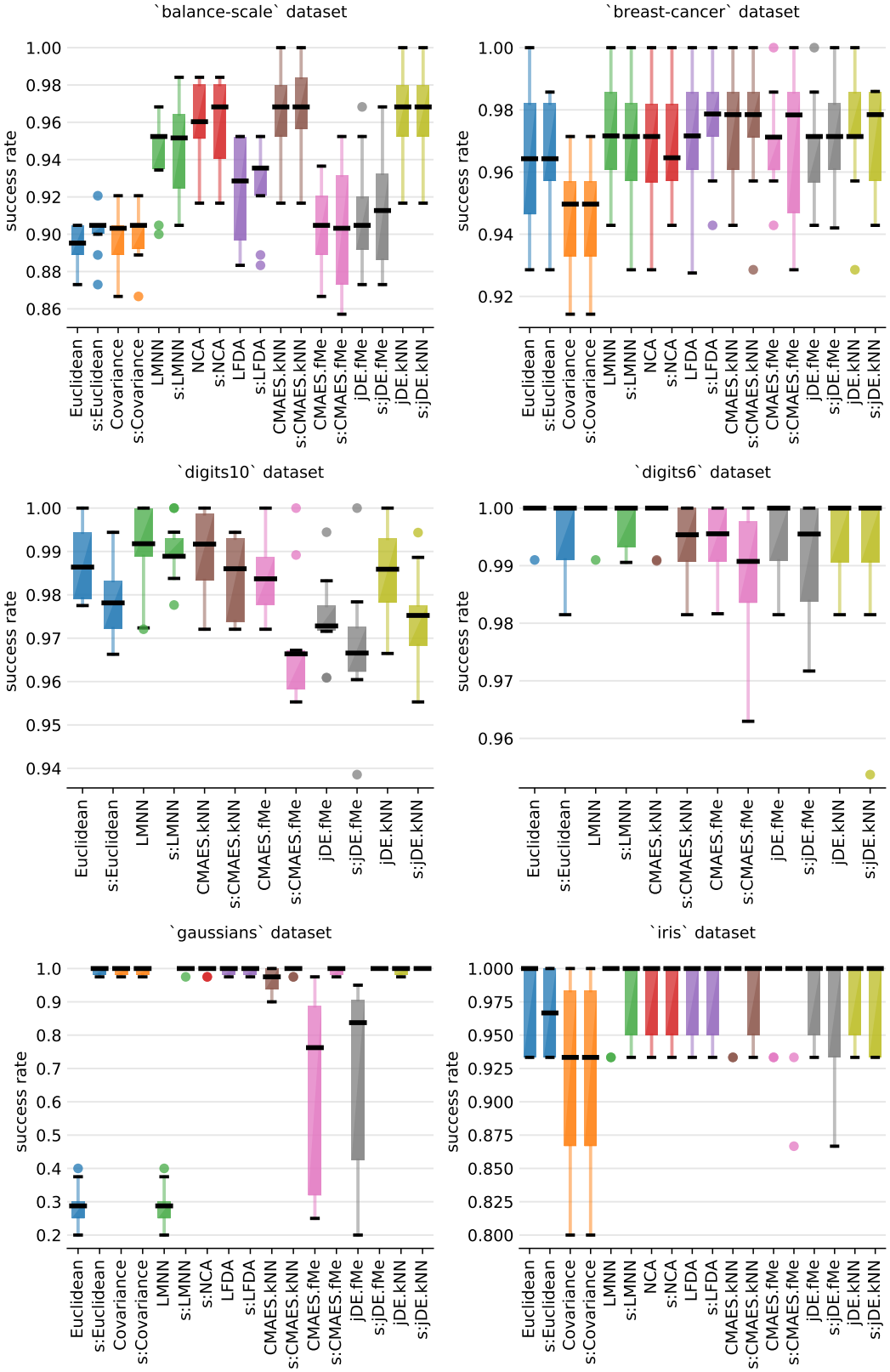

Figure 4.1: Success rates for classification experiment on *balance-scale*, *breast-cancer*, *digits6*, *digits10*, *gaussians*, and *iris* datasets

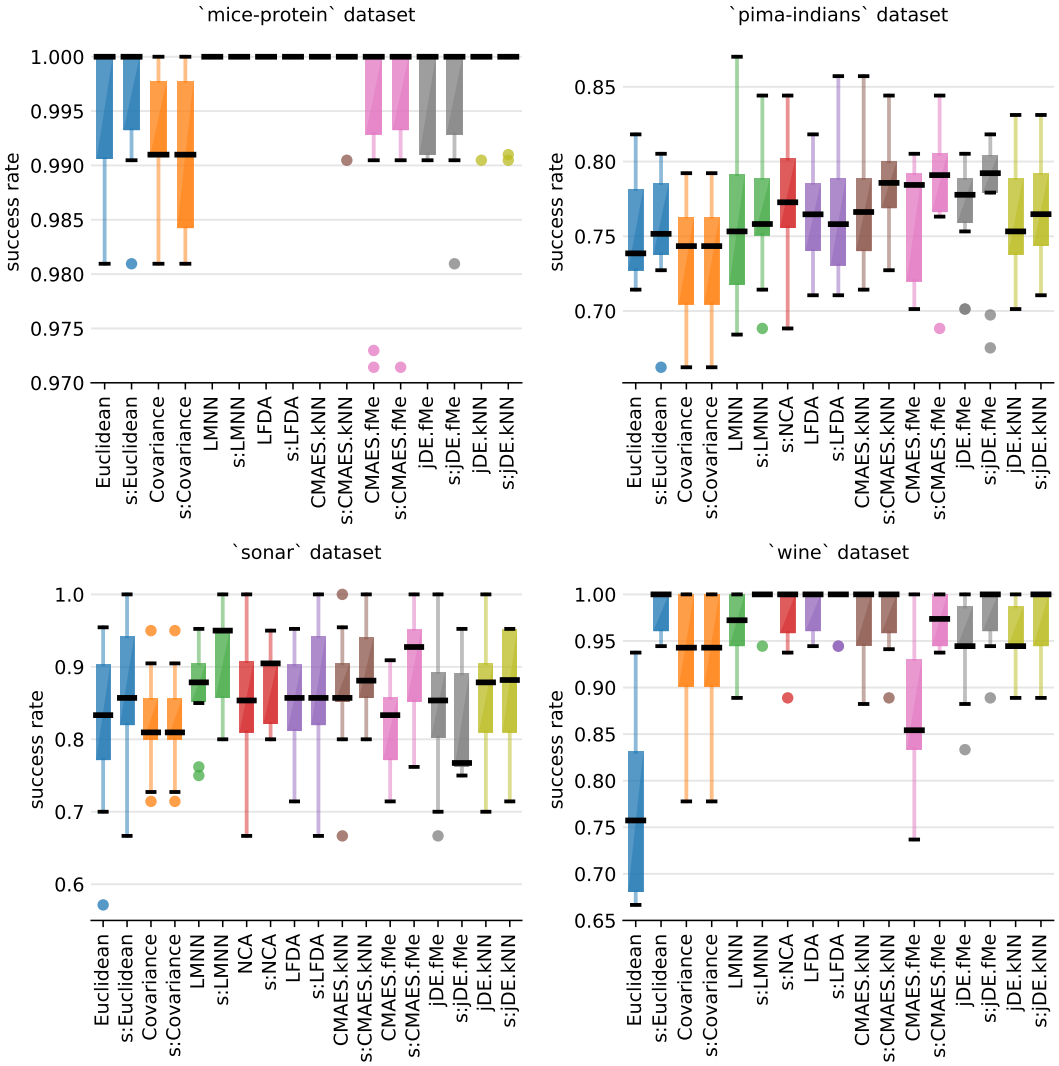

Figure 4.2: Success rates for classification experiment on *mice-protein*, *pima-indians*, *sonar*, and *wine* datasets

To see how the hyperparameters influence the success rate of k-NN classifier, one hyperparameter is inspected at a time. By fixing this hyperparameter and inspecting its values, the best success rate across the rest of the hyperparameters is searched.

First we look at the k hyperparameter in the final k-NN classifier. In Figure 4.3 there is one graph for every dataset (*digits6* dataset is omitted because it performs very similarly to *digits10* dataset) in which each line corresponds to one algorithm. The best value for this hyperparameter differs for each dataset, however, most of the curves have a tip near $k = 8$.

From individual graphs it is possible to see some trends among all the methods, in particular that the lines tend to copy a downwards parabola shape, meaning that k values of 1 and 2 are too small to generalise well and values 64 and 128 are too large for these small datasets.

The best shape for any particular method would be a flat line near the top of the graph, which would imply that the method separated the instances into a well separable and homogeneous clusters in which all instances have the same label.

Some of the methods seem to handle the change of this parameter more gracefully. This is the most apparent with LMNN and NCA methods in *mice-protein* dataset: these two methods have large success rates even when the k parameter increases, whereas the other methods' success rates drop significantly. This phenomenon is visible in other datasets and even in other methods, for example in *gaussians* dataset for CMAES.kNN as well as the two already mentioned.

The Covariance method has a bad success rate altogether, but it also drops down the fastest when the k hyperparameter increases.

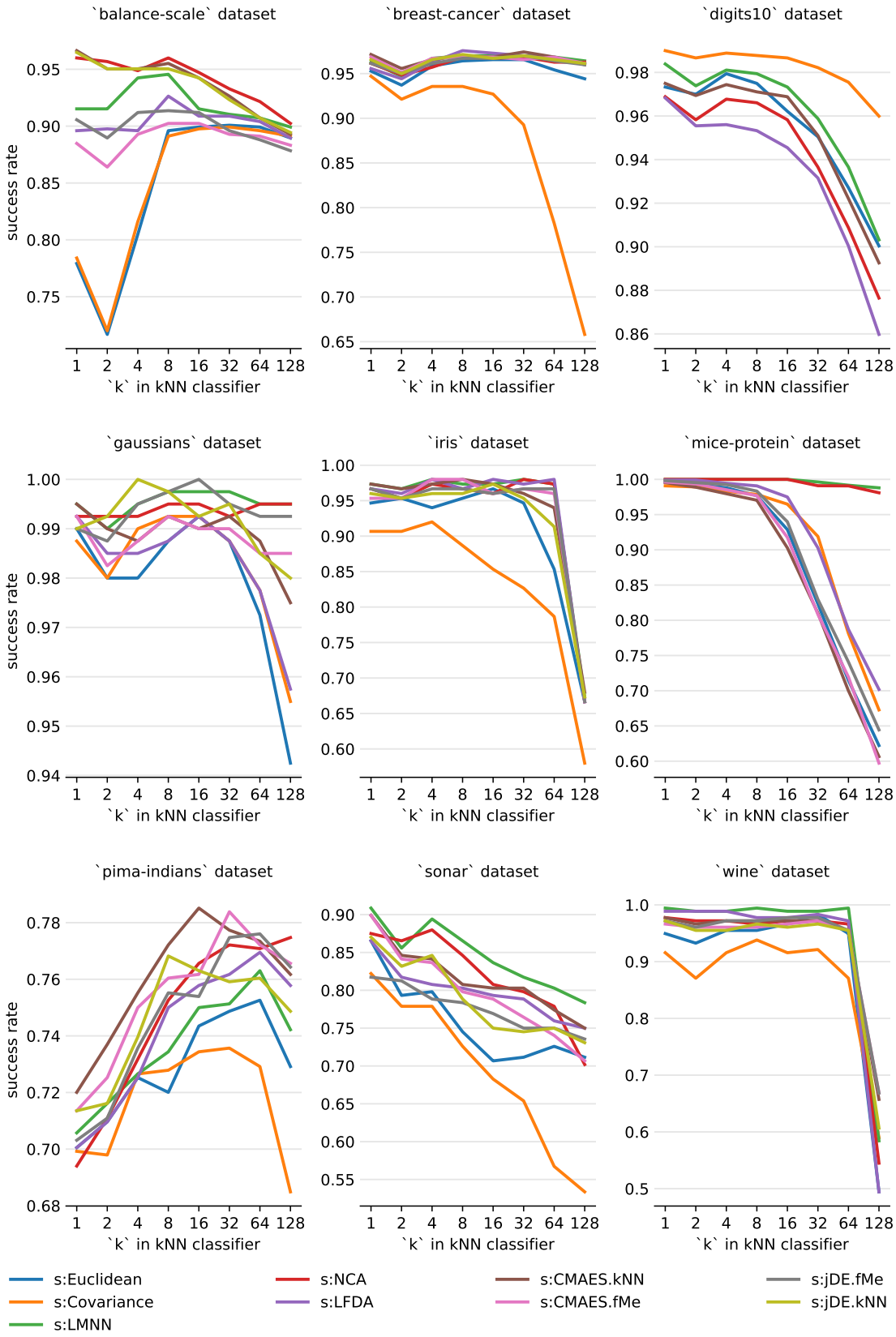

Figure 4.3: Success rates of DML algorithms while fixing k in the final k-NN classifier

The hyperparameters of the individual methods are shown in Figure 4.4. For each hyperparameter there is a different graph and each curve in a graph represents one dataset. These hyperparameters are shown in the nine graphs:

1. k hyperparameter for LMNN, which is number of neighbours that the method takes into account when optimising as discussed in Section 2.3. It is clear that this hyperparameter influences the success rate of this method by a large amount and it seems to peak at values between 4 and 16.

2. maximum number of iterations (max_iter) for LMNN from which it is clear that running the algorithm for 250 iterations works very well, but it starts to overfit slightly afterwards.

3. regularisation parameter for LMNN also improves the success rate when it is higher for all datasets except *sonar*.

4. learning rate does not affect the performance of NCA

5. metric initialization for LFDA shows that both weighted and orthonormalised intialisations perform equally well.

6. transformer shape for CMAES.kNN, shows a comparison between evolving a diagonal and full Mahalanobis matrix. For all except two datasets (*pima-indians* and *mice-protein*) evolving the full matrix worked better. Most likely CMA-ES could not explore the whole d^2-dimensional space well because *pima-indians* is a high-dimensional dataset.

7. number of neighbours in CMAES.kNN, shows it is best to select 8 neighbours.

8. weights in CMAES.kNN do not influence the performace too much.

9. transformer shape for jDE.fMe, evolving a full matrix works clearly better for all datasets.

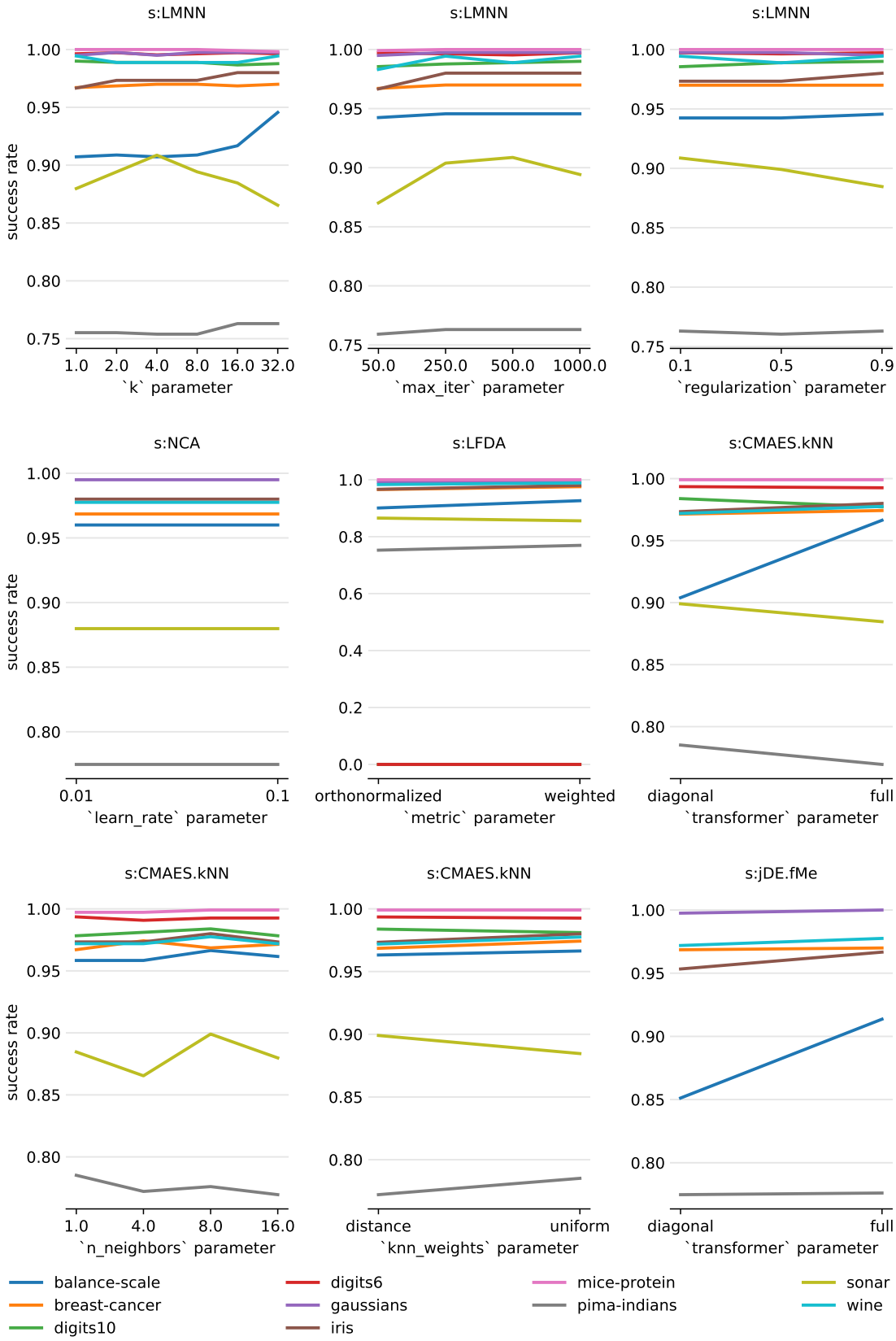

Figure 4.4: Success rates of DML algorithms when fixing given hyperparameters

4.3 Experiment: Generalisation of EAs

To test out how well solutions from evolutionary DML methods evolve and generalise, the methods are trained for 1000 generations and the fitness values, train, and test success rates are measured after every generation. From these values it is possible to see whether the method generalises well or whether it overfits. It is also possible to compare the search capabilities of the methods by looking at the individuals with maximal fitness.

CMA-ES and jDE were tested with both fitnesses (k-NN and F-measure): CMAES.kNN, CMAES.fMe, jDE.kNN, and jDE.fMe. All algorithms were tested with full and diagonal Mahalanobis matrices.

All methods have some hyperparameters and in order to avoid the time consuming hyperparameter search, the hyperparameters for the models in this experiment were selected according to results from the previous experiment in Section 4.2. Only one set of the hyperparameters was selected for all datasets. The number of nearest neighbours used in the k-NN fitness function was chosen to be 8 and the number of nearest neighbours in the final k-NN classifier was set to 4.

Datasets described in Section 4.1.1 were used as well for this experiment. Data were preprocessed and standardised the same way as in previous experiments, both methods are described in Section 4.1.2. Data were split into training and testing sets using a 67:33 ratio in a stratified fashion, keeping the sizes of classes balanced between the two splits.

To measure the performance of the fitness function, the fitness values of all individuals were logged in every generation and then the median, minimum, maximum, 25th and 75th percentiles (Q1 and Q3) were calculated in each generation. To see how well the individuals generalise, individuals with the best and the worst fitness values were selected in every generation and the metric encoded in these individuals was evaluated using the test set using the k-NN classifier after every single generation.

4.3.1 Results

Figures 4.5—4.14 show the results for each dataset. In each Figure, the top four methods learn a full matrix, meanwhile the bottom four methods learn a diagonal matrix. Both k-NN and F-measure fitness functions output values in a closed range $[0, 1]$. The final k-NN classifier used to measure performance outputs values in the same exact range and all values are plotted in the same graph. Therefore each individual graph shows minimal and maximal (both in orange) fitnesses, median fitness (in green) with the area between 25th and 75th

percentiles coloured in light green, the blue line represents the test success rate of the individual with the highest fitness and the pink line represents the test success rate of the individual with the lowest fitness in every generation. A black line representing a test success rate of k-NN using standard Euclidean distance is also added to the graphs. Unfortunately jDE did not finish on the four largest datasets and those graphs are marked with "Timeout".

Important aspect to also keep in mind when looking at the graphs is that the left four graphs all use k-NN for both fitness evaluation as well as test success rate evaluation, meanwhile the right four graphs use F-measure fitness for learning, but the test success rate is still evaluated using the k-NN classifier. This means that the left four graphs have fitnesses and their test success rates directly comparable, meanwhile in the right four graphs only general trends can be followed and the fitness values cannot be directly compared with the test success rates.

jDE updates the population only when a new individual with a better fitness value is generated, whereas CMA-ES regenerates the entire population from scratch every generation according to its internal covariance matrix. This is clearly visible on all the graphs: for jDE, the orange and blue lines representing fitnesses never drop, they only grow. On the other hand, CMA-ES' fitness lines are more unpredictable. Therefore, in order to smooth the lines and make the graphs more readable, one-dimensional Gaussian filter with a standard deviation 2.0 is used.

CMAES.kNN (full) methods performs very well on most datasets. From the graphs it is evident that it generalises also very well, which can be deduced from the fact that the blue and pink lines representing test success rate closely follow the green fitness value lines. CMAES.kNN (full) can overfit and plummet after some number of generations which can be seen in Figure 4.10 on *iris* dataset, where the test success rate first goes up to 94% and then starts dropping and finishes at 86% success rate. This suggests that there should be an early stopping mechanism, which would prevent overfitting and reduce the computation time because the algorithm would not have to finish a fixed number of generations. The full version should outperform the diagonal version. However the diagonal version performs better on some datasets, namely on *breast-cancer* (Figure 4.6), *digits10* (Figure 4.8) and *iris* (Figure 4.10). This might be because the algorithm was better at exploring the d-dimensional space compared to d^2-dimensional space in the case of the full matrix. Nevertheless, the full matrix outperforms the diagonal matrix on *balance-scale* (Figure 4.5), *mice-protein* (Figure 4.11), *sonar* (Figure 4.13) and *wine* (Figure 4.14) datasets. Moreover, in this experiment we are more interested in general trends of the evolution, not absolute values of the fitness functions.

If jDE.kNN is compared with CMAES.kNN, it generalises worse even though its fitness values are higher than fitness values of CMAES.kNN. In every single dataset the blue and pink lines representing the test success rates are far below the green fitness line and on most datasets even far below the lower orange fitness line representing the minimal fitness in the given generation. This problem concerns evolving both full and diagonal Mahalanobis matrices. Even though it does not generalise well and the fitness values tend to overestimate the success rate, the testing success rates are relatively comparable, although slightly worse, to those using CMAES.kNN.

The fitness values of CMAES.fMe have a very large range on all datasets, for examples on *breast-cancer* dataset (Figure 4.6), the fitness values range from nearly 0% to around 97%. This highly oscillating range suggests it is hard for the method to generate good individuals, however the test success rates perform relatively well. The issue with CMAES.fMe is that the fitness does not grow over time and therefore the test success rates do not grow either. This might be caused by the large fitness variability and because F-measure does not say anything about the structure of the clusters and together with the fact that CMA-ES regenerates the individuals in each generation, the generation does not improve the individuals but tends to be random. jDE.fMe overcomes this problem, because it does not regenerate the population but rather only replaces the individuals with better ones. For this reason the fitness values of jDE.fMe and jDE.kNN are only rising. The problem with jDE.fMe method is that the test success rates do not follow the growth of the fitness values, but they tend to stagnate as can be seen on *pima-indians* (Figure 4.12) and *wine* datasets (Figure 4.14). This might be a problem of F-measure not being suitable for representing the fitness of the clusters as CMAES.fMe suffered from the same problem. This concerns evolving both the full and the diagonal matrices.

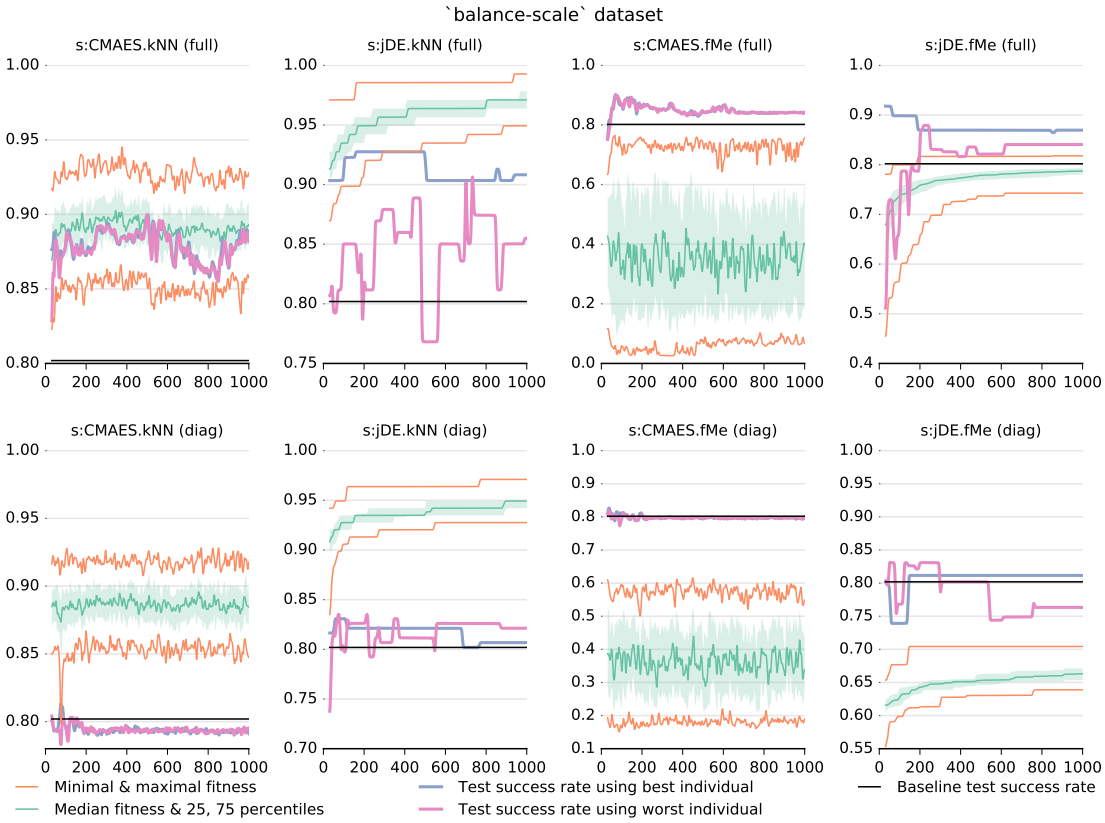

Figure 4.5: Fitness evolution and generalisation on *balance-scale* dataset

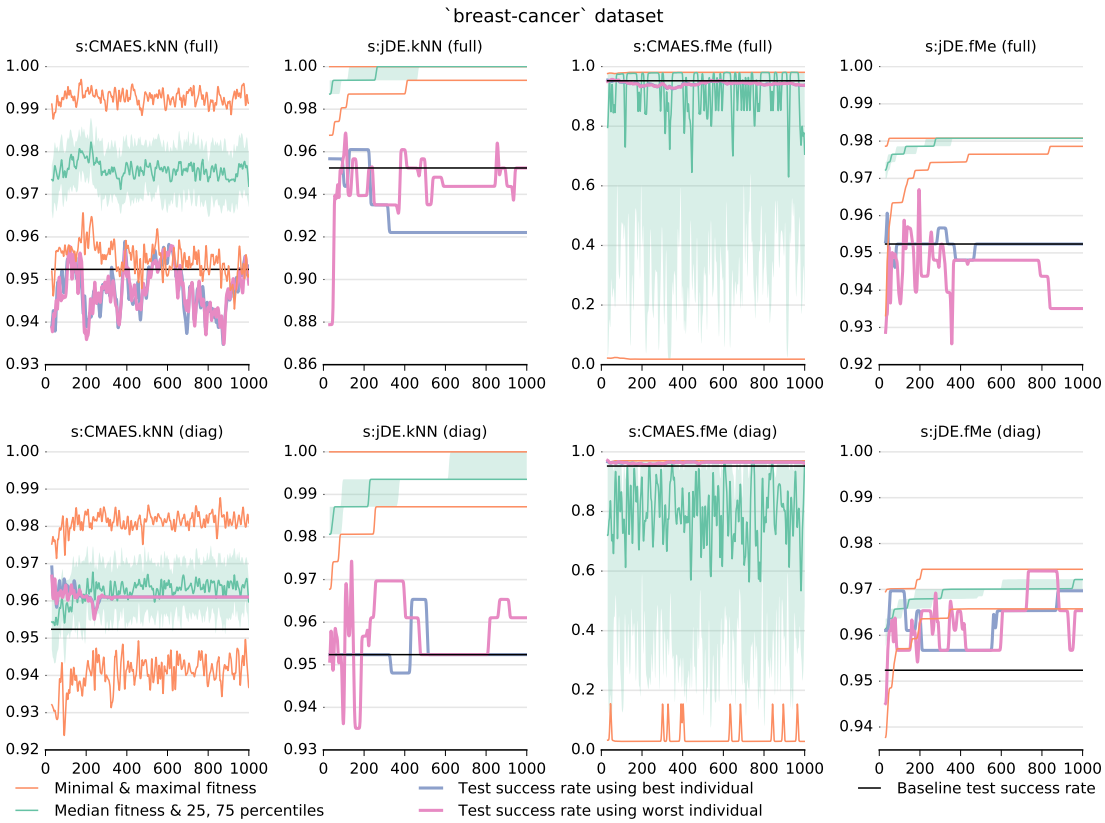

Figure 4.6: Fitness evolution and generalisation on *breast-cancer* dataset

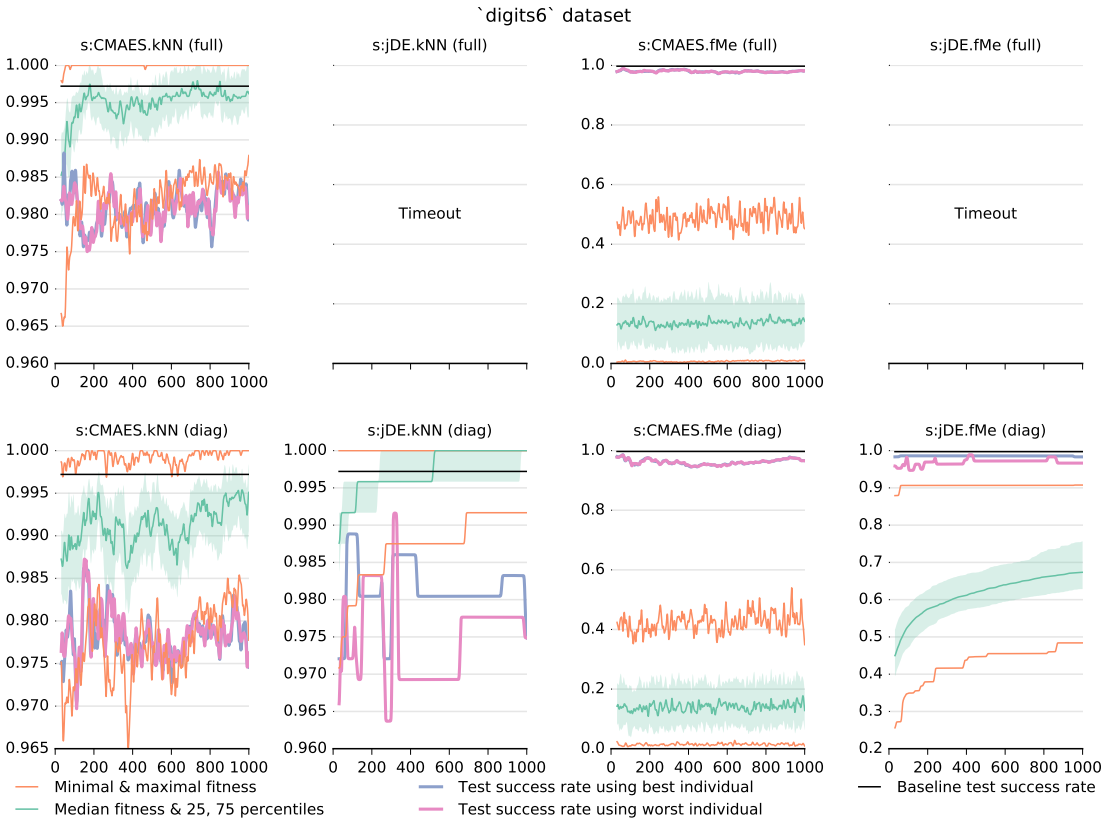

Figure 4.7: Fitness evolution and generalisation on *digits6* dataset

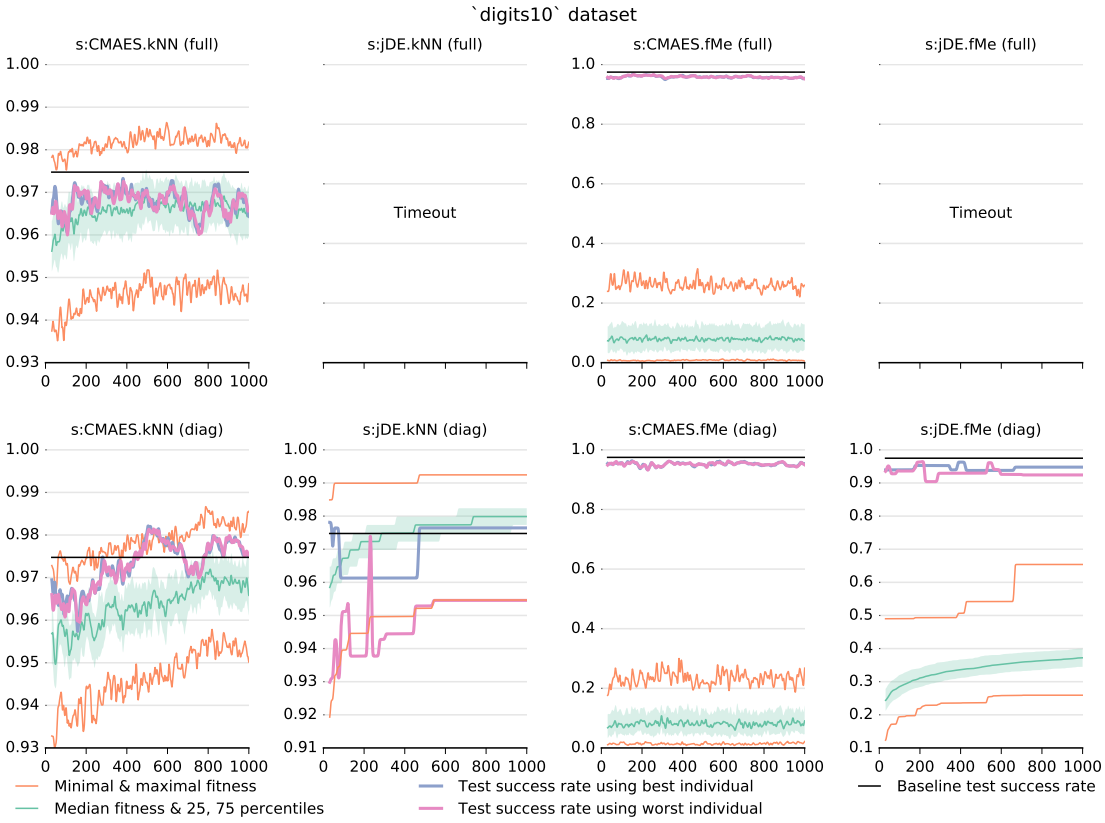

Figure 4.8: Fitness evolution and generalisation on *digits10* dataset

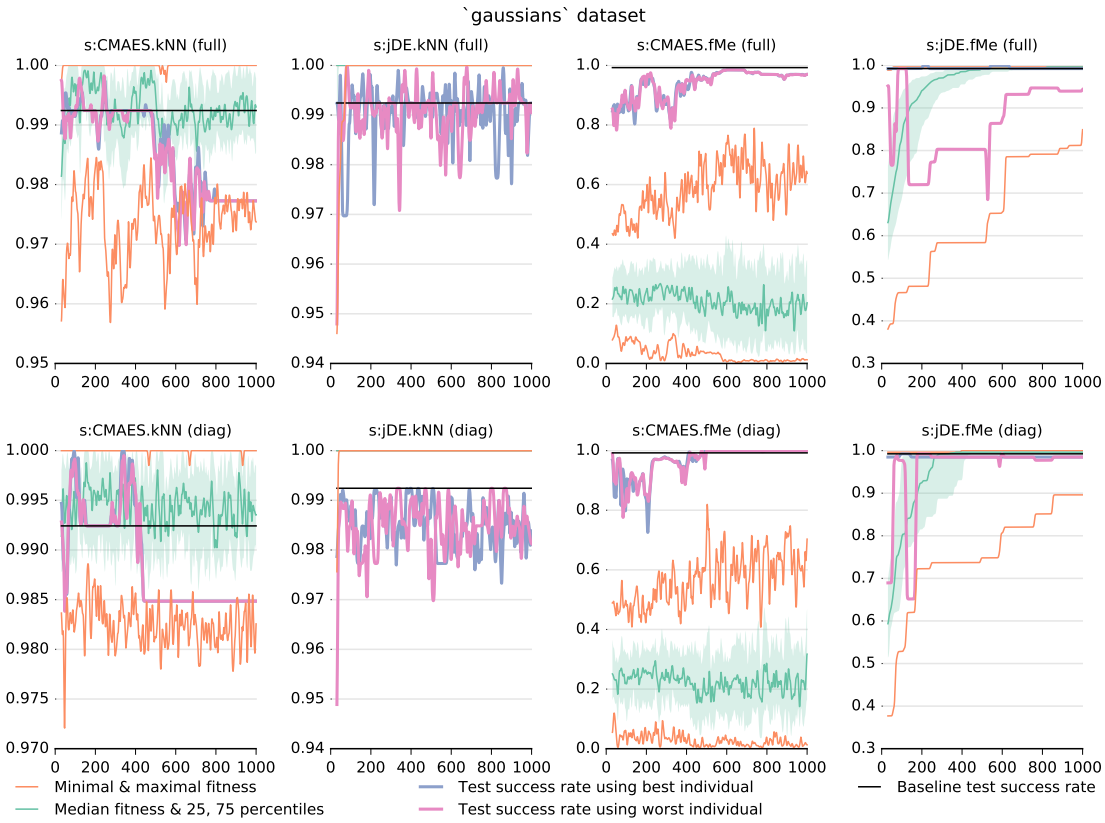

Figure 4.9: Fitness evolution and generalisation on *gaussians* dataset

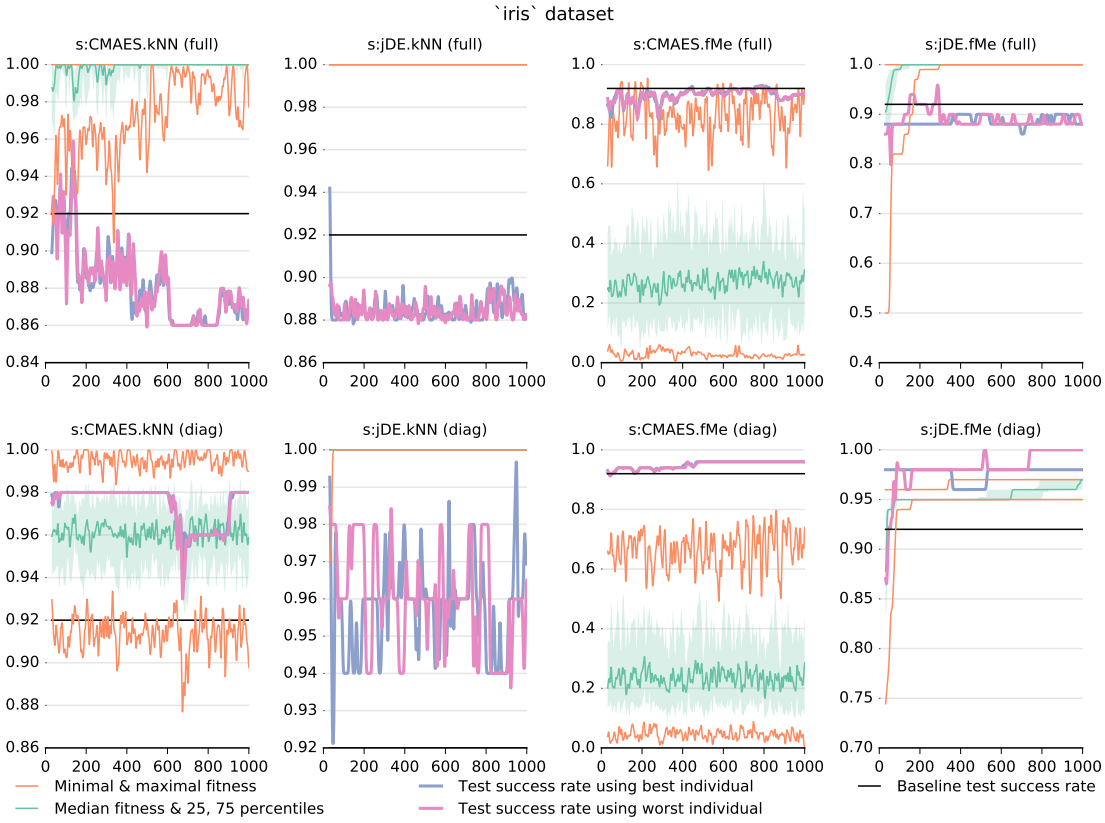

Figure 4.10: Fitness evolution and generalisation on *iris* dataset

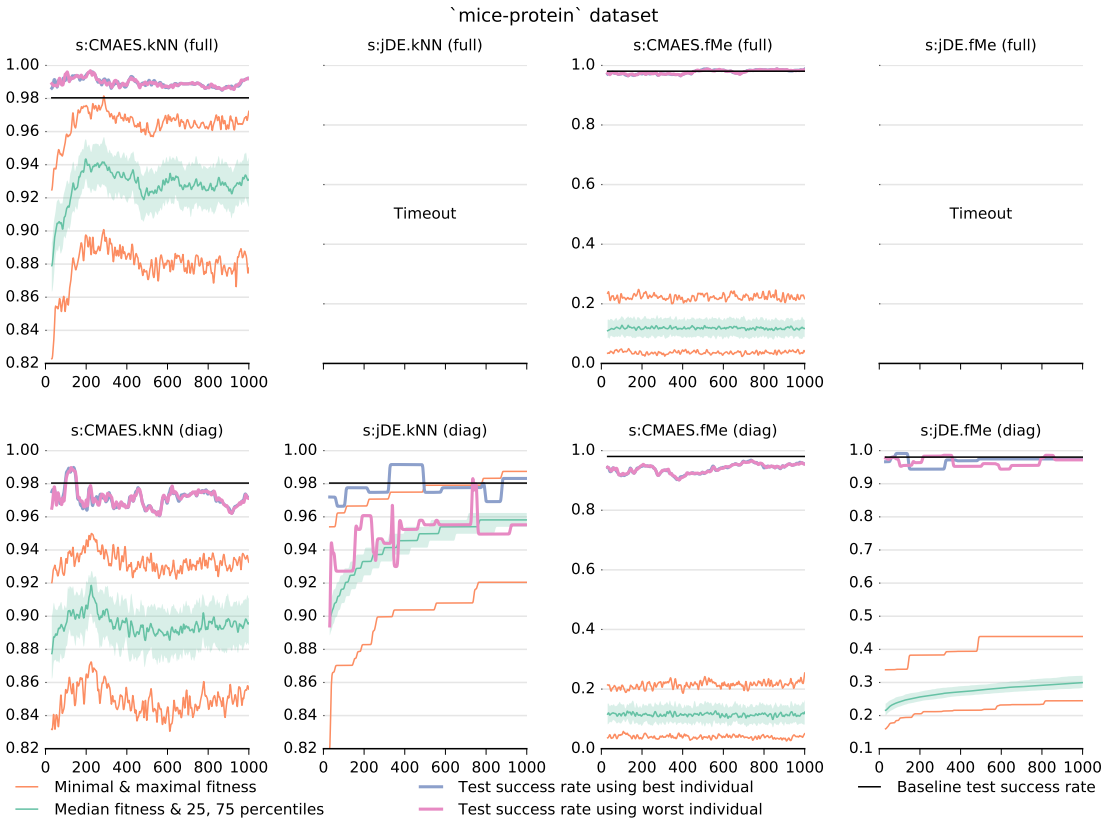

Figure 4.11: Fitness evolution and generalisation on *mice-protein* dataset

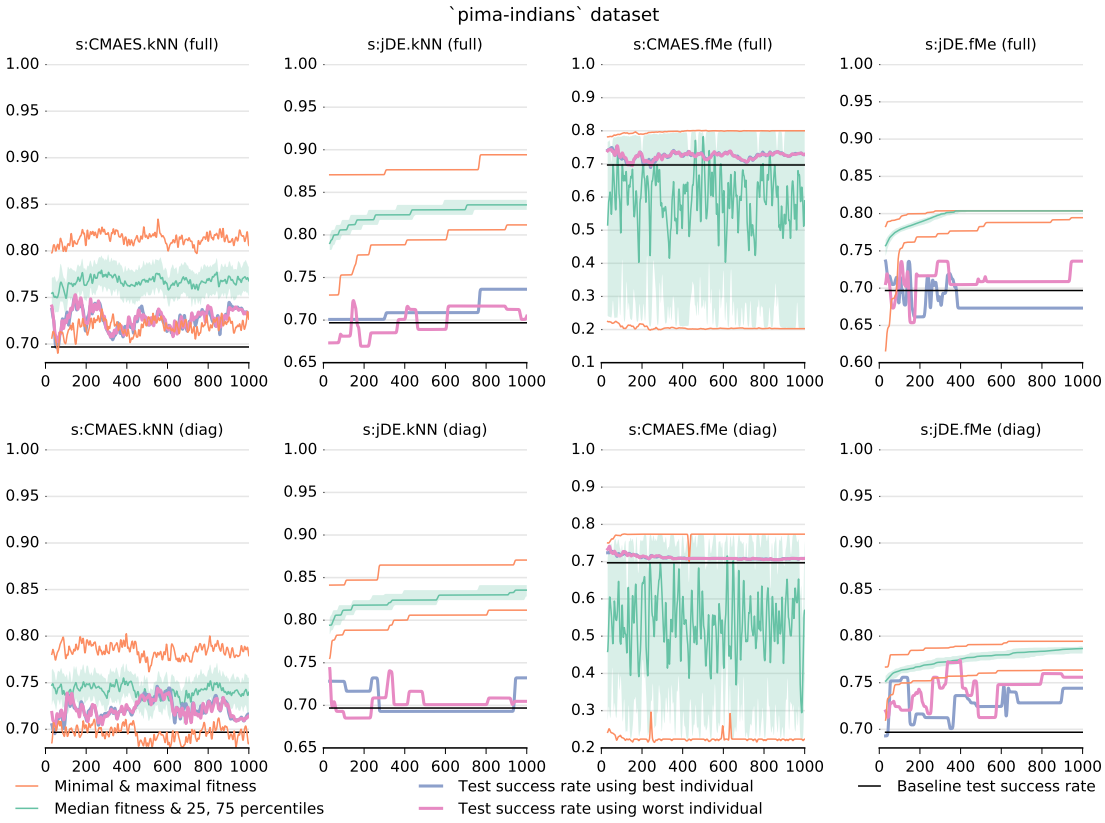

Figure 4.12: Fitness evolution and generalisation on *pima-indians* dataset

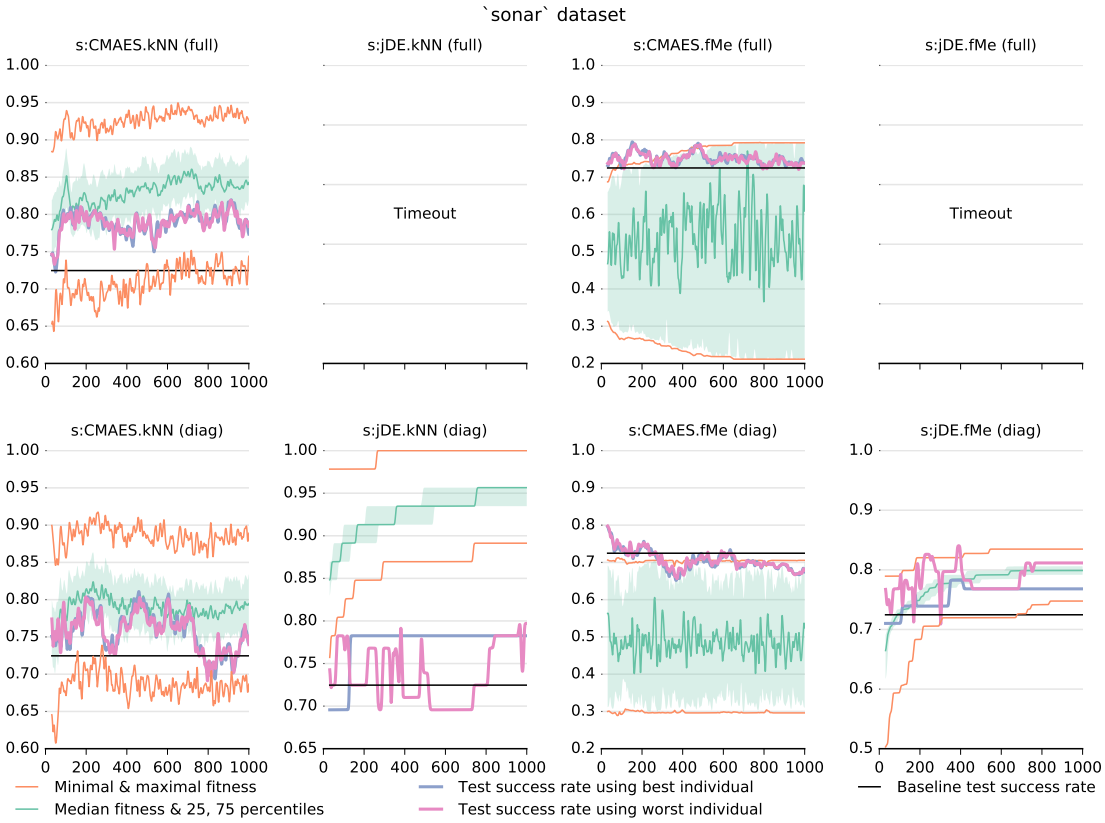

Figure 4.13: Fitness evolution and generalisation on *sonar* dataset

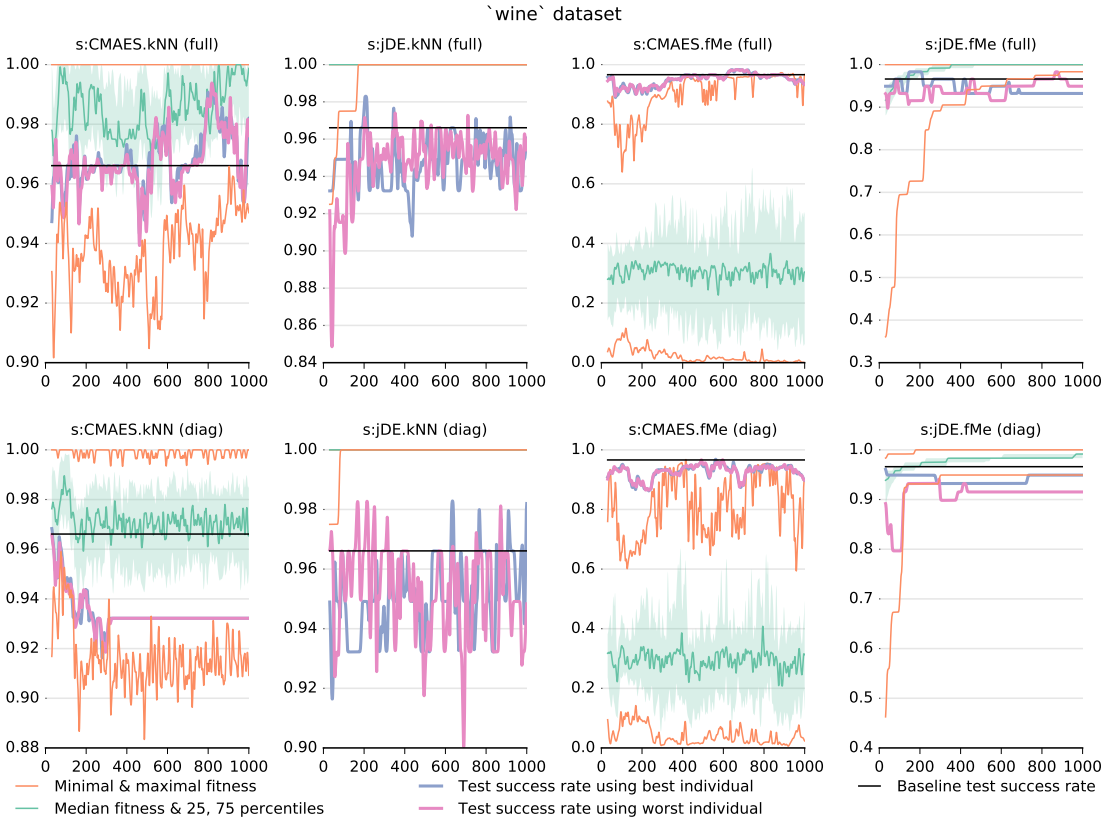

Figure 4.14: Fitness evolution and generalisation on *wine* dataset

4.4 Experiment: jDE with log population

In previous experiments, different population sizes for CMA-ES and jDE were used: $4 + 3\log d^2$ and $10d^2$, respectively (d is the dimensionality of the problem, a full matrix was evolved). These population sizes for CMA-ES and jDE are suggested in [Hansen and Ostermeier, 2001] and [Brest et al., 2006], respectively. The size of the population directly translates into the performance of the evolutionary algorithm, which has better exploration capabilities with a larger population and is also more robust.

In experiments in Sections 4.2 and 4.3, CMA-ES outperforms jDE [Fukui et al., 2013] even with imbalanced population sizes. The main drawback of a larger population is increased computational time, especially when the evaluation of the fitness function is expensive.

This experiment is designed to test whether such a large population for jDE is necessary and the generalisation performance of jDE with both population sizes, $4 + 3\log d^2$ and $10d^2$, is tested. The setting of this experiment is the same as for the experiment in Section 4.3.

4.4.1 Results

Figures 4.15—4.18 use the same type of graphs as in the previous experiment. Each figure shows minimal, mean and maximal fitnesses for each generation and also test success rates using best and worst individuals in each generation. We chose to show only datasets where the differences between the population sizes are the most prominent.

Figures 4.15 and 4.17 show that jDE with smaller population "(log)jDE" has lower maximal fitness and also lower test success rates, which suggests the algorithm is not able to explore the space as good as with a larger population size. In Figures 4.16 and 4.18, the test success rates of (log)jDE with the k-NN fitness oscillate and drop in multiple generations, which suggests that the algorithm overfits.

We conclude that the population size has a big influence on the performance of an evolutionary algorithm. jDE with logarithmic population size performs worse on some datasets than the same algorithm with a population size linear to the dimensionality of the individuals. However, CMA-ES with logarithmic population beats jDE even with a larger population as could be seen in the previous experiments.

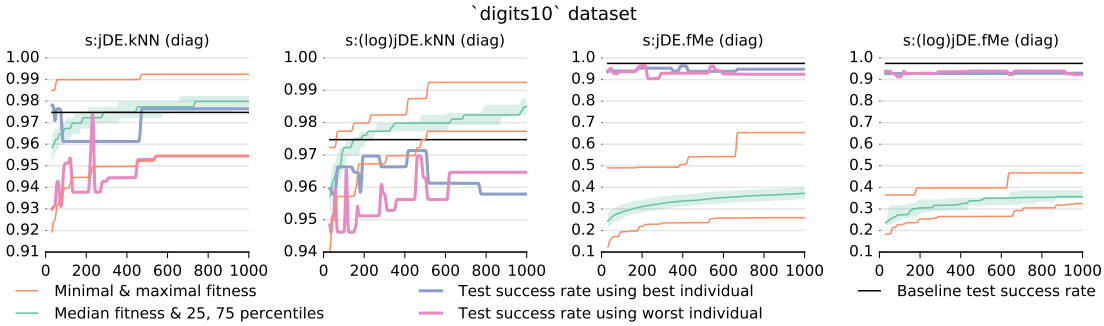

Figure 4.15: jDE with linear and log population size on *digits10* dataset

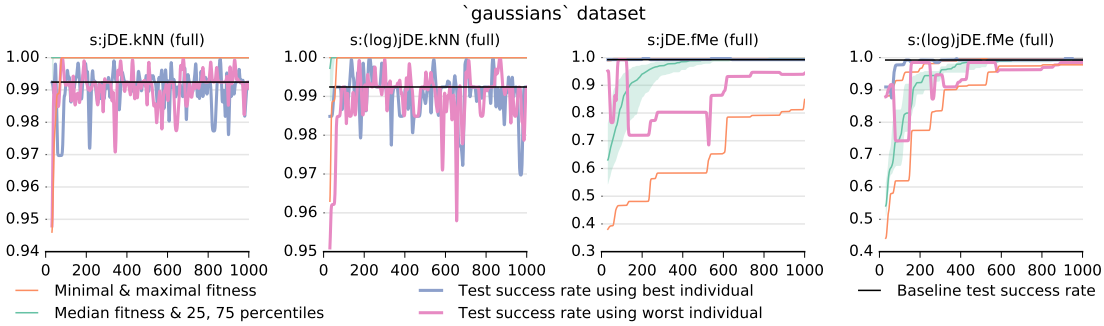

Figure 4.16: jDE with linear and log population size on *gaussians* dataset

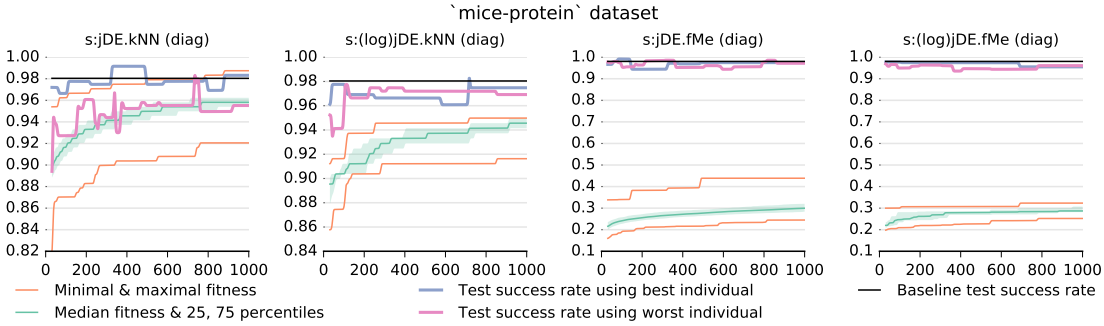

Figure 4.17: jDE with linear and log population size on *mice-protein* dataset

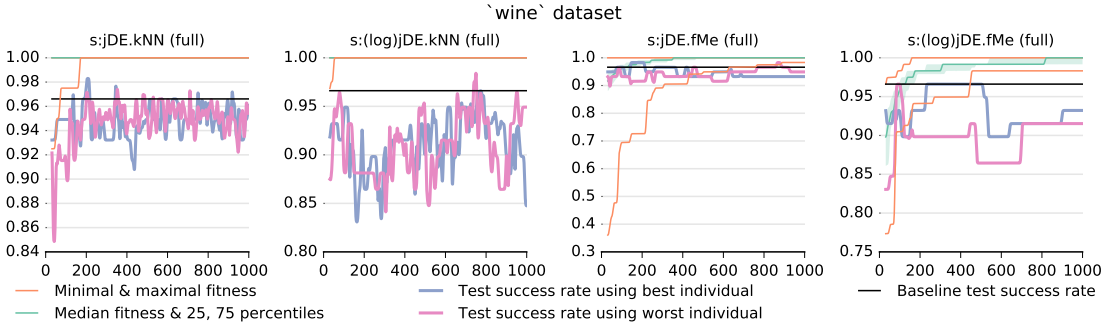

Figure 4.18: jDE with linear and log population size on *wine* dataset

4.5 Experiment: Learning time

Learning time of each method is an important factor. Some methods are not scalable to large datasets as already discussed in Chapter 2. Moreover, some of the methods can still end up in a local minima thus multiple runs may be necessary for optimal results.

In this experiment the learning times of all the methods from previous experiments were measured: Covariance, LMNN, NCA, LFDA and both evolution strategies with both evolution fitnesses: CMAES.kNN, CMAES.fMe, jDE.kNN and jDE.fMe. For the evolutionary methods, learning only a diagonal transformation matrix was also measured. All methods were run 10 times on every dataset with a set of the hyperparameters which corresponds to best error rate for given dataset from the first experiment in Section 4.2. Euclidean distance does not need any training and is therefore omitted from this experiment altogether. The experiment was performed on a single computer so that the results are comparable. Having this restriction, the maximal time for the method to finish was set to six hours; after this period the experiment was interrupted. The methods that did not finish are marked as "Timeout".

4.5.1 Results

The results are in Table 4.5, where the learning times shown are averaged across all 10 runs. Calculating the Covariance metric is the fastest, closely followed by LFDA. Some of the datasets were so small that these two methods would take a less than a hundredth of a second. On the other hand, LMNN and NCA methods and all the evolutionary methods take a lot of time to learn. Generally, jDE should be much faster compared to CMA-ES strategy because calculating a covariance matrix inside CMA-ES is expensive, however, jDE needs a much larger population and the fitness evaluation is very expensive in our case as discussed in Section 2.6. Therefore, jDE is clearly much slower compared to CMA-ES strategy. It is also clear that F-measure fitness is much slower compared to k-NN fitness as discussed in Section 2.2 when CMAES.kNN is compared with CMAES.fMe and jDE.kNN with jDE.fMe. This makes the method jDE.fMe described in Section 2.6 the slowest method in all of the benchmarked methods and is even several times slower than NCA. CMAES.kNN performs comparable to LMNN in terms of learning time. The restricted versions of the evolutionary algorithms to a diagonal transformation matrix learn much faster thanks to smaller individuals, which directly encode the Mahalanobis matrix and also the smaller populations as also discussed in Chapter 3. What is important to note, however, is that differences

between learning a full and a diagonal matrices using CMA-ES are much smaller than the differences between learning a full and a diagonal matrices using jDE, where for CMA-ES the difference is about 2 times, but for jDE the difference is much more significant, even 10 times slower on some datasets.

	balance-scale	breast-cancer	digits6	digits10	gaussians
s:Covariance	**0.01**	**0.00**	**0.00**	**0.01**	**0.00**
s:LMNN	2.78	10.78	49.75	858.98	1.77
s:NCA	71.37	176.38	Timeout	Timeout	38.04
s:LFDA	0.02	0.03	0.02	0.03	0.01
s:CMAES.kNN (full)	5.76	9.20	7601.11	5490.68	5.02
s:CMAES.fMe (full)	17.73	15.80	7082.51	7037.36	15.43
s:jDE.fMe (full)	206.34	648.13	Timeout	Timeout	166.43
s:jDE.kNN (full)	77.31	495.96	Timeout	Timeout	56.14
s:CMAES.kNN (diag)	3.81	5.35	26.86	61.63	3.12
s:CMAES.fMe (diag)	8.04	10.01	42.44	116.62	9.73
s:jDE.fMe (diag)	30.64	69.18	1577.14	2951.39	35.63
s:jDE.kNN (diag)	16.48	52.55	2230.37	4578.65	13.47
	iris	mice-protein	pima-indians	sonar	wine
s:Covariance	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
s:LMNN	0.09	57.50	12.99	7.87	2.34
s:NCA	10.47	Timeout	164.51	880.07	21.90
s:LFDA	**0.00**	0.20	0.02	0.04	0.01
s:CMAES.kNN (full)	4.67	11835.03	7.56	3165.69	8.46
s:CMAES.fMe (full)	9.83	11758.96	18.78	3481.85	18.54
s:jDE.fMe (full)	104.10	Timeout	644.15	Timeout	1404.01
s:jDE.kNN (full)	46.97	Timeout	380.90	Timeout	696.73
s:CMAES.kNN (diag)	2.80	37.04	4.54	6.48	4.02
s:CMAES.fMe (diag)	6.59	65.79	12.94	17.05	10.29
s:jDE.fMe (diag)	23.15	2619.58	78.49	549.09	92.20
s:jDE.kNN (diag)	11.27	3561.06	41.83	285.22	40.19

Table 4.5: Learning times (in seconds) for each method on every dataset

To further investigate how well metric learning methods scale, two additional experiments were also designed to investigate the learning times. In the first experiment the dimensionality of the data was fixed to 5 and the learning times were measured for the number of samples in the range from 100 to 1500 with 100 increments. In the second experiment the number of samples was fixed to 500 and the learning times were measured for all methods with the increasing dimension in

the range from 2 to 10 with an increment of 1. All these measurements were done using *digits10* dataset from which the required number of samples was sampled using the stratified sampling and then the dimension of the samples was reduced using PCA to a required dimension.

Both experiments are represented in Figures 4.19 and 4.20. Most of the methods merge together in the graph so two graphs are showed for each experiment: one with linear and one with logarithmic scale applied on y axis. Covariance and LFDA are not included in these two figures because they perform so fast on this experiment that the results are not interpretable due to random fluctuations in the measurements.

From Figure 4.19 it is clear that jDE performs the worst and does not scale well with an increasing number of samples with both k-NN and F-measure fitnesses. NCA performs also very bad with an increasing number of samples. On the other hand the rest of the methods perform relatively well and are indistinguishable on the linearly scaled axis. LMNN and CMA-ES have a similar performance, however LMNN also performs much slower with more and more instances. In contrast, CMA-ES scales very well and the number of samples hardly influences the performance. Looking at the evolutionary methods restricted to a diagonal matrix, it is clear that CMAES.kNN (diag) is the fastest among all of them, however, it is not much slower compared to its unrestricted version CMAES.kNN (full). Restricted version of jDE is much faster compared to its unrestricted counterpart, but even the restricted version is slower and scales worse than CMAES.kNN (full).

Similarly, jDE scales very poorly with an increasing dimensionality of the dataset as can be seen from Figure 4.20. The order of the methods in the graphs remained the same as in the previous experiment, however, there are some notable changes in the shapes of the curves. LFDA scales very well with an increasing dimensionality; NCA and LMNN are not as steep as in the previous experiment, but they still scale worse compared to CMA-ES. It is clear that restricted versions of all evolutionary algorithms scale much better in regards of the increasing dimensionality because the dimensionality directly reflects in the individuals size: for the restricted version the individuals have size D, but for the unrestricted version the whole matrix has D^2 parameters. The dimensionality of the individuals directly reflect in the population size as discussed. Therefore the restricted versions scale better with the increasing dimension and their graphs are less steep. When the steepness of CMA-ES and jDE is directly compared, jDE scales worse even in the restricted version. That is due to the large population size that jDE uses by default, compared to CMA-ES, which only has a population of a logarithmic size as discussed in Chapter 3.

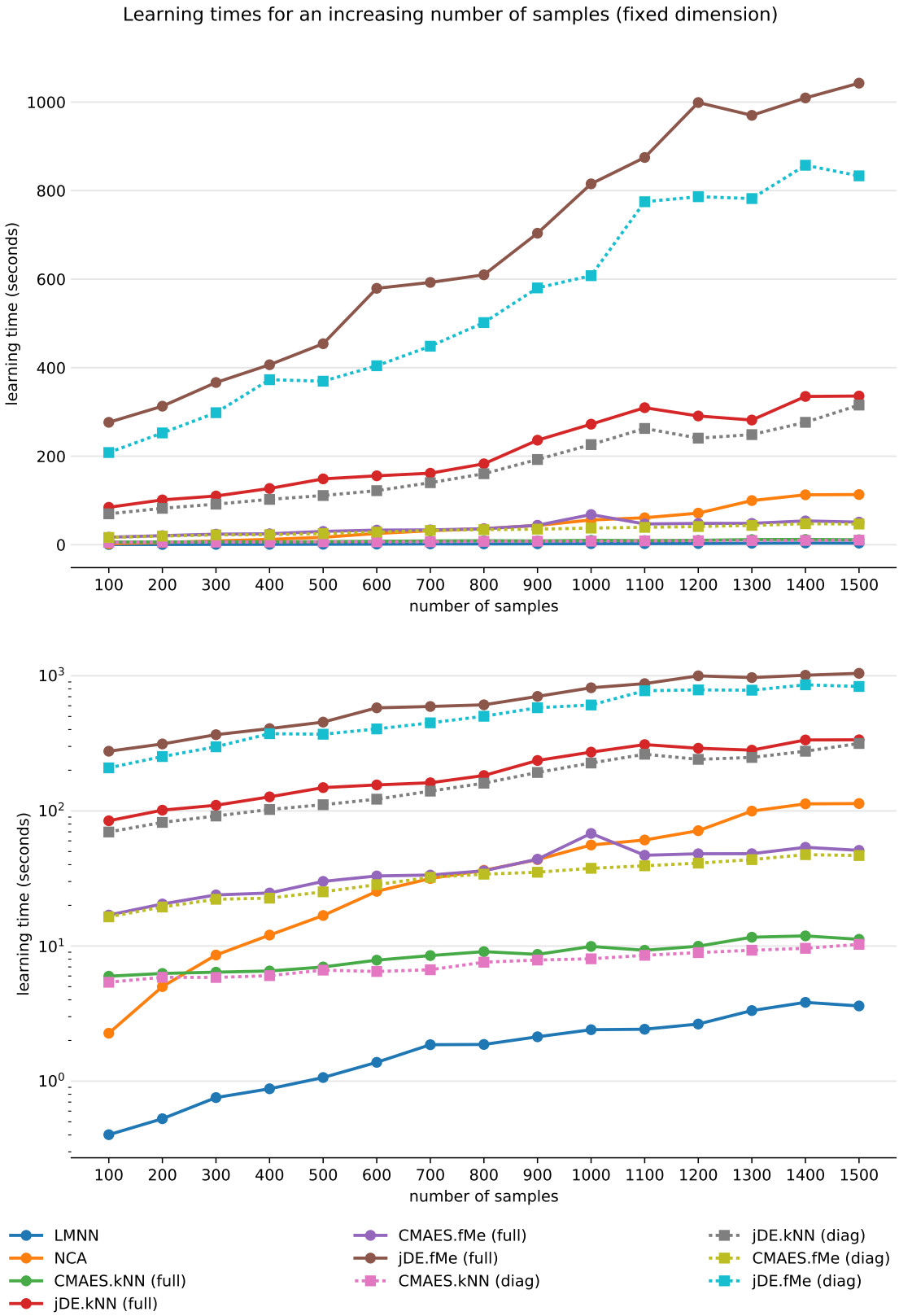

Figure 4.19: Learning times for increasing number of samples with a fixed dimension (linear and log scale)

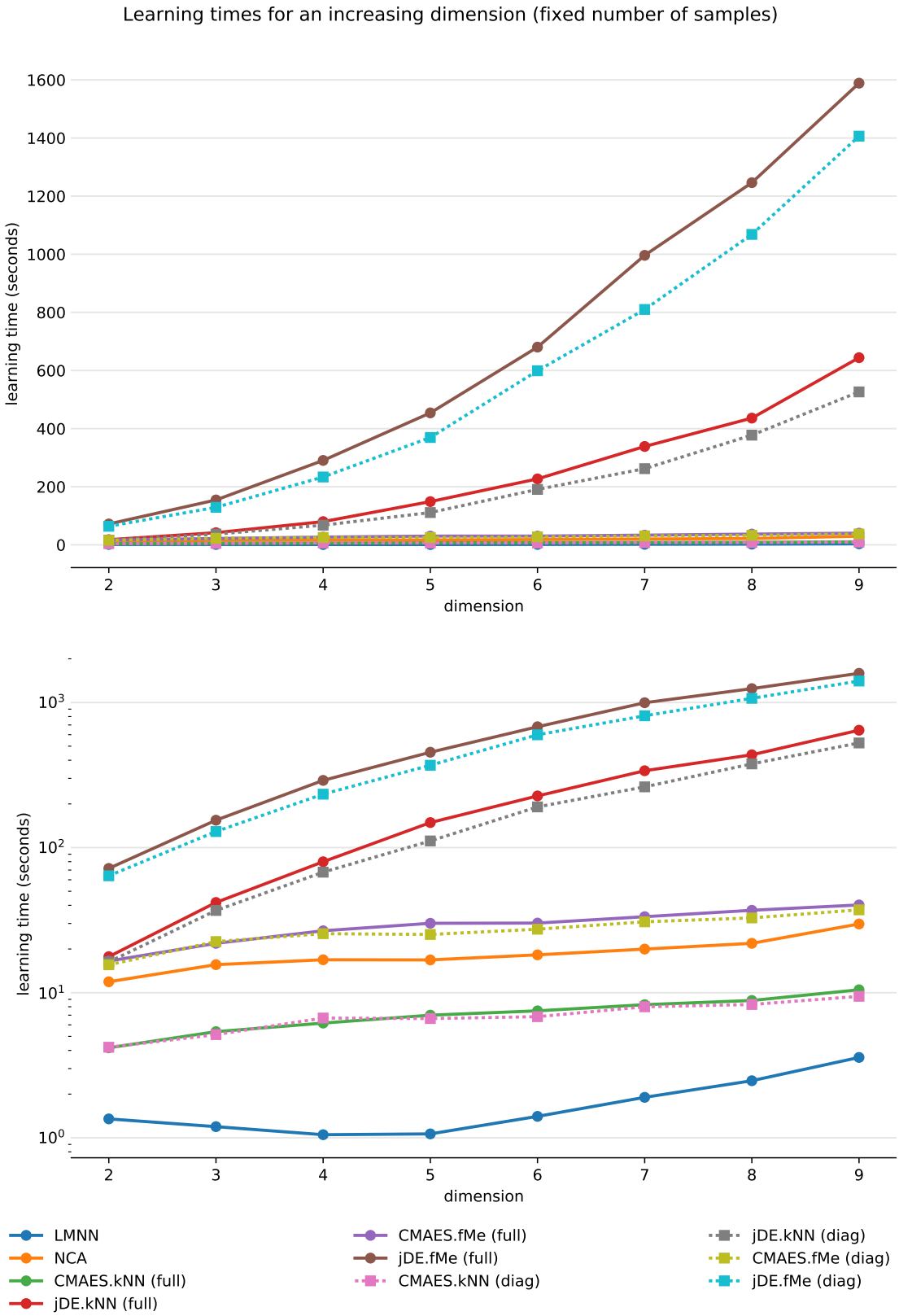

Figure 4.20: Learning times for increasing dimension with a fixed number of samples (linear and log scale)

4.6 Experiment: Dimensionality reduction

In this experiment, PCA and t-SNE methods are compared with DML methods supporting dimensionality reduction: NCA, LFDA and CMAES.kNN. Apart from evolving a transformation matrix, we also experimented with evolving fully-connected neural networks for dimensionality reduction using CMAES.kNN. Six different neural networks were evolved differing in depth (one or five hidden layers) and their activation functions (sigmoid, tanh or ReLU). The shallow networks have one hidden layer with 4 neurons and the deep networks have 16-12-8-6-4 neurons in the hidden layers. To reduce the dimension into 2-dimensional space, all networks always end with an output layer with 2 neurons. Therefore, the shallow network has $4d + 8$ free parameters and the deeper network has $16d + 354$ free parameters. No biases were used.

PCA, LFDA, and NCA methods are linear and so the dimensionality reduction works by projecting the data onto the 2-dimensional space by a matrix multiplication. The methods only differ in how to find this transformation matrix.

The same datasets, listed in Table 4.1, are used in this experiments. The data are standardised as described in Section 4.1.2. The 10-fold cross validation is used to split the dataset to train and test sets. After training the algorithms, they are used to transform both train and test sets into two dimensions. To compare the dimensionality reduction, the k-NN classifier is fit using this 2-dimensional training set and calculate a success rate on the test set. The idea is that the k-NN classifier has a large accuracy if instances from the same class are close together, which indicates a good dimensionality reduction.

t-SNE is also included in the visualisations, however, as it is a transductive method, it cannot be used to classify new data. Therefore in this case, the k-NN classifier is fit on all the data and the numbers may be overly optimistic. The main reason for including t-SNE is to show how the visualisation from CMAES.kNN compare to this popular visualisation method.

We were also interested in whether CMA-ES would help t-SNE make a better visualisation if used as a preprocessing and we call this method CMAES.kNN+t-SNE, where CMAES.kNN is trained first and data are transformed using the evolved transformation matrix keeping the original dimensionality, and only then t-SNE is used to transform the preprocessed data from CMA-ES into two dimensions.

4.6.1 Results

The results are presented in Table 4.6. The error rates and standard deviations are shown for each method on each dataset from the 10-fold cross validation. The

best inductive method (i.e. of those that do not use t-SNE) for every dataset is in a bold font. The best method among methods that have a transformation function (all except t-SNE) is emphasised in italics.

Generally, CMAES.kNN works very well for dimensionality reduction and also very fast because only a $d \times 2$ matrix is evolved. CMAES.kNN has the lowest error only on *pima-indians*, however the difference between this method and the best method on each dataset is never statistically significant. Again, the statistical significance tables for this experiment are provided in Attachments (Tables 6.11—6.20). t-SNE works the best on large datasets, namely *digits6* and *digits10* and the preprocessing with CMA-ES helps t-SNE on the majority of the datasets. LFDA and NCA have both also very good results, but NCA runs out of memory on three datasets (marked with "Memory" in the results table).

	balance-scale	breast-cancer	digits6	digits10	gaussians
s:PCA	44.26 ± 13.07	***3.43 ± 2.73***	24.06 ± 3.07	46.12 ± 2.91	9.75 ± 2.61
s:NCA	11.65 ± 3.25	3.58 ± 2.33	Memory	Memory	0.50 ± 1.00
s:LFDA	12.11 ± 2.52	4.58 ± 2.48	38.02 ± 10.99	57.07 ± 6.76	***0.25 ± 0.75***
s:CMAES.kNN	*9.39 ± 3.78*	4.16 ± 2.90	16.74 ± 3.94	41.22 ± 3.46	1.25 ± 2.02
s:t-SNE	22.32 ± 3.22	4.44 ± 2.26	**1.39 ± 1.12**	**2.90 ± 1.13**	3.25 ± 2.97
s:CMAES.kNN+t-SNE	**7.48 ± 3.01**	4.59 ± 2.48	1.57 ± 1.02	3.67 ± 1.98	2.25 ± 2.84
s:NN.sigm (shallow)	11.32 ± 3.18	4.30 ± 3.22	17.36 ± 4.61	*37.93 ± 4.16*	2.50 ± 1.94
s:NN.tanh (shallow)	*9.39 ± 2.49*	4.01 ± 2.70	*15.68 ± 3.35*	39.07 ± 4.64	3.25 ± 3.17
s:NN.ReLU (shallow)	12.74 ± 3.18	4.30 ± 2.32	18.10 ± 3.98	40.68 ± 4.35	3.00 ± 2.45
s:NN.sigm (deep)	14.32 ± 3.86	4.73 ± 2.88	37.33 ± 4.60	58.62 ± 4.60	7.75 ± 2.61
s:NN.tanh (deep)	18.00 ± 3.65	4.43 ± 2.51	38.59 ± 5.19	57.55 ± 4.84	13.25 ± 7.08
s:NN.ReLU (deep)	14.81 ± 2.46	5.01 ± 3.52	35.75 ± 6.89	56.83 ± 3.83	10.00 ± 6.02

	iris	mice-protein	pima-indians	sonar	wine
s:PCA	10.00 ± 4.47	67.80 ± 3.43	31.50 ± 4.39	39.43 ± 6.61	3.99 ± 3.72
s:NCA	***2.67 ± 3.27***	Memory	26.44 ± 2.87	***22.02 ± 7.87***	***1.70 ± 2.60***
s:LFDA	6.67 ± 5.96	64.75 ± 12.90	27.87 ± 2.61	22.61 ± 4.33	1.74 ± 2.66
s:CMAES.kNN	6.00 ± 5.54	*36.09 ± 6.29*	***26.18 ± 3.94***	31.35 ± 6.61	9.59 ± 7.92
s:t-SNE	24.67 ± 14.62	4.18 ± 1.80	28.65 ± 4.55	45.82 ± 15.06	33.30 ± 12.25
s:CMAES.kNN+t-SNE	19.33 ± 11.33	**2.59 ± 1.70**	26.70 ± 3.11	35.20 ± 9.89	40.58 ± 14.55
s:NN.sigm (shallow)	6.67 ± 5.96	46.19 ± 6.54	28.79 ± 3.19	23.51 ± 7.98	6.12 ± 5.81
s:NN.tanh (shallow)	8.00 ± 8.33	47.03 ± 5.53	29.82 ± 4.23	33.16 ± 12.78	7.23 ± 6.12
s:NN.ReLU (shallow)	6.67 ± 5.16	45.54 ± 6.90	27.35 ± 3.42	30.80 ± 9.69	9.17 ± 5.60
s:NN.sigm (deep)	12.00 ± 9.80	56.97 ± 3.97	32.94 ± 4.43	42.42 ± 9.81	13.29 ± 9.06
s:NN.tanh (deep)	16.67 ± 6.83	61.31 ± 4.87	32.94 ± 6.01	35.40 ± 13.63	15.80 ± 8.62
s:NN.ReLU (deep)	5.33 ± 4.00	58.56 ± 5.91	29.68 ± 4.10	29.62 ± 10.75	10.14 ± 9.73

Table 4.6: Error rates (in %) for the dimensionality reduction experiment

The success rate, however, is only one of the factors that imply a good visualisation and therefore we also check the projections visually by plotting both the training and testing sets using a scatter plot. The following criteria are used to evaluate the visualisation: the instances with the same label are together; the clusters are well separated; clusters form small blobs and they are not spread out;

all instances with the same class are in one large cluster. In the title of each plot there is a name of the method used together with the success rate on the test set for the given run that is being visualised.

In each scatter plot the instances are coloured according to their label. The training instances are highly opaque without any border. The testing instances are less opaque and they have a black or red border. Black border signifies that the instance was classified correctly using the k-NN classifier and red border means the instance was classified incorrectly. The instances that were classified incorrectly are also visualised as slightly larger. In the title of each plot there is a name of the method used together with the success rate on the test set for the given run that is being visualised. The success rates of t-SNE are prefixed with a tilde to signify it is an optimistic value.

The results of the dimensionality reduction for all methods are in Figures 4.21—4.30. One figure represents one dataset and each figure contains a scatter plot for each method. PCA is also plotted in all figures and acts as the baseline method that other methods are compared to.

On *balance-scale* (Figure 4.21) dataset PCA finds a good projection of the data and it is possible to see some intrinsic relationships between the classes. Nevertheless, NCA, LFDA and CMAES.kNN all find much better projection in which the blue class forms a barrier between the orange and the green classes. t-SNE by itself completely hides these relationships between the data: the green class is split into two clusters and the blue class is scattered around. Preprocessing the data using CMAES.kNN first before using t-SNE fixes both these problems, the green class stays clustered together and the blue class forms a cluster. The evolved neural networks are also able to separate the classes from each other, but the deep neural network with tanh activation makes two well separated clusters, but spreads out the third class.

On *breast-cancer* (Figure 4.22) dataset NCA and CMAES.kNN separate the clusters more strictly. t-SNE works well by itself, but preprocessing the data with CMA-ES first improves the visualisation again. The neural networks also separate the classes very well. The networks with sigmoid and tanh activations tend to keep the instances in long chains, but ReLU activation function tends to cluster one class together and spreads out the other one.

When looking at *digits6* and *digits10* datasets (Figures 4.23 and 4.24), t-SNE visualisations outperforms other methods. Nevertheless, CMAES.kNN+t-SNE still improves the visualisation by joining some clusters together on both datasets. PCA finds some clusters, but they are merged together and not well separated. CMAES.kNN works better compared to PCA, it manages to separate the clusters more and the clusters themselves are also more uniform. LFDA method fails on

this dataset completely, it spreads out just few data instances, but keeps the rest together in one dense cluster. NCA method runs out of memory on these datasets.

All methods perform well on *gaussians* dataset (Figure 4.25). The supervised methods, however, find more compact clustering than PCA. t-SNE splits the blue class into two clusters and this is where CMAES.kNN+t-SNE improves on again. Very similar observation can be made about *iris* dataset (Figure 4.26). The neural networks again visualise and cluster the data almost as chains.

Both *mice-protein* (Figure 4.27) and *pima-indians* datasets (Figure 4.28) are hard to visualise. Regarding *mice-protein* dataset, PCA does not find any nice intrinsic properties and just makes one large heterogeneous cluster. NCA ran out of memory and LFDA only separated out the grey class. CMAES.kNN was able to separate out the most classes, but some classes are still mixed together and they are not well-separated. The best visualisation, at least in regards to success rate, was achieved using t-SNE, but it was not able to capture the classes into homogeneous clusters, but it created many small fragmented clusters. On *pima-indians* there is not a clear winner, but NCA, LFDA, CMAES.kNN and CMAES.kNN+t-SNE were the best at separating the two classes from each other.

Datasets *sonar* (Figure 4.29) and *wine* (Figure 4.30) have very few data instances and that might be the reason why t-SNE and even CMAES.kNN+t-SNE failed completely. *sonar* is a high-dimensional dataset and PCA failed to separate the clusters as well on this dataset. On the other hand NCA, LFDA and CMAES.kNN were able to separate the classes into a homogeneous clusters on both datasets. The neural network methods have high success rate which signifies that the dataset probably has some non-linear intrinsic structure.

Interestingly, the neural networks with sigmoid and tanh activations tend to create interesting, almost cube-like structures, meanwhile ReLU activation creates more of a light torch structure, where many instances are clustered in a centre and the rest is spread out in one major direction. We also found out that sometimes the simpler models (shallow networks, on only a linear mapping) often provide better results than the deeper neural networks. One of the reasons may be the deeper network has more parameters which means it would take longer to tune them. Increasing the number of generations may improve the performance of the deeper models, however, the training would also take considerably longer.

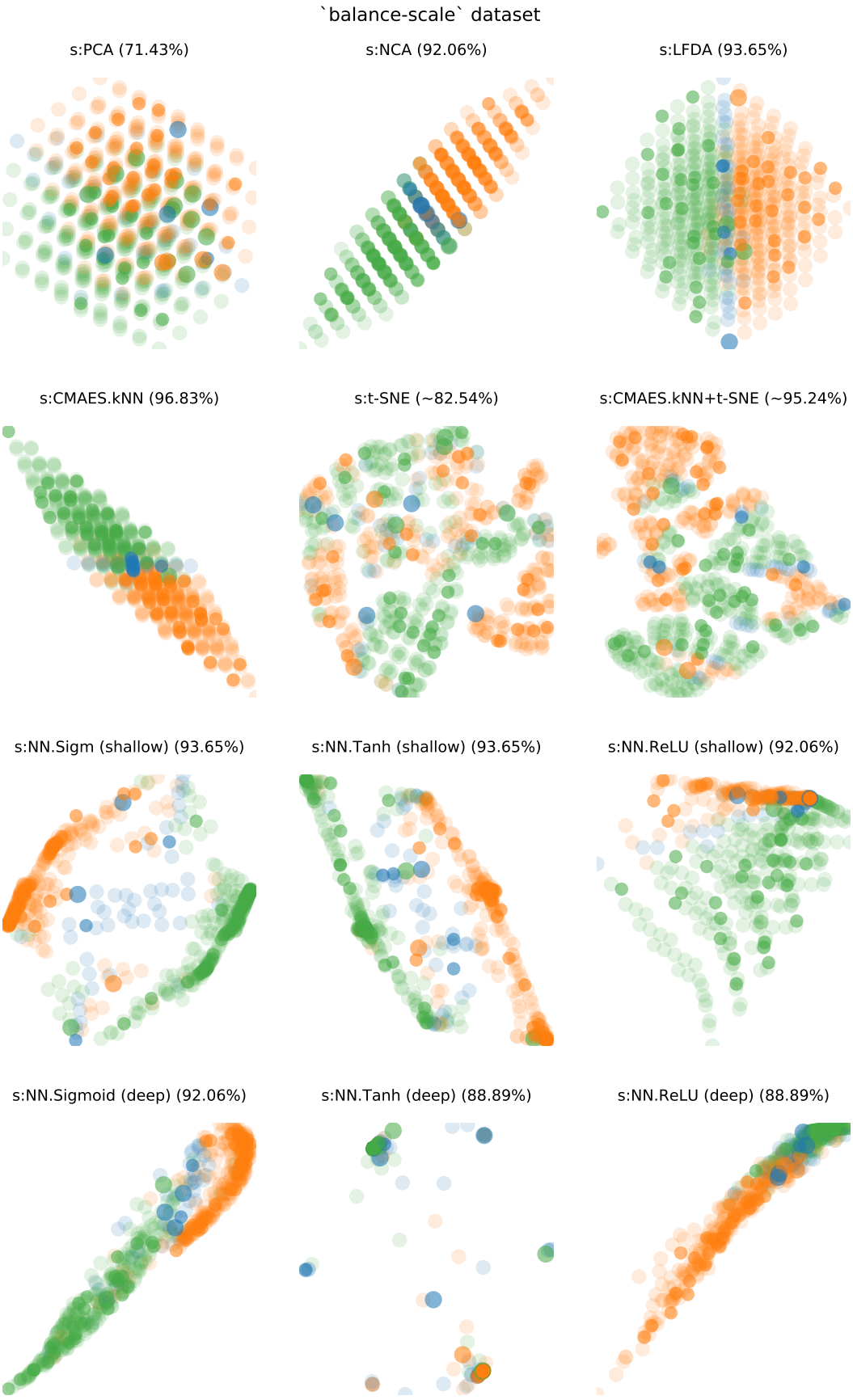

Figure 4.21: Visualisation of *balance-scale* dataset with DML methods

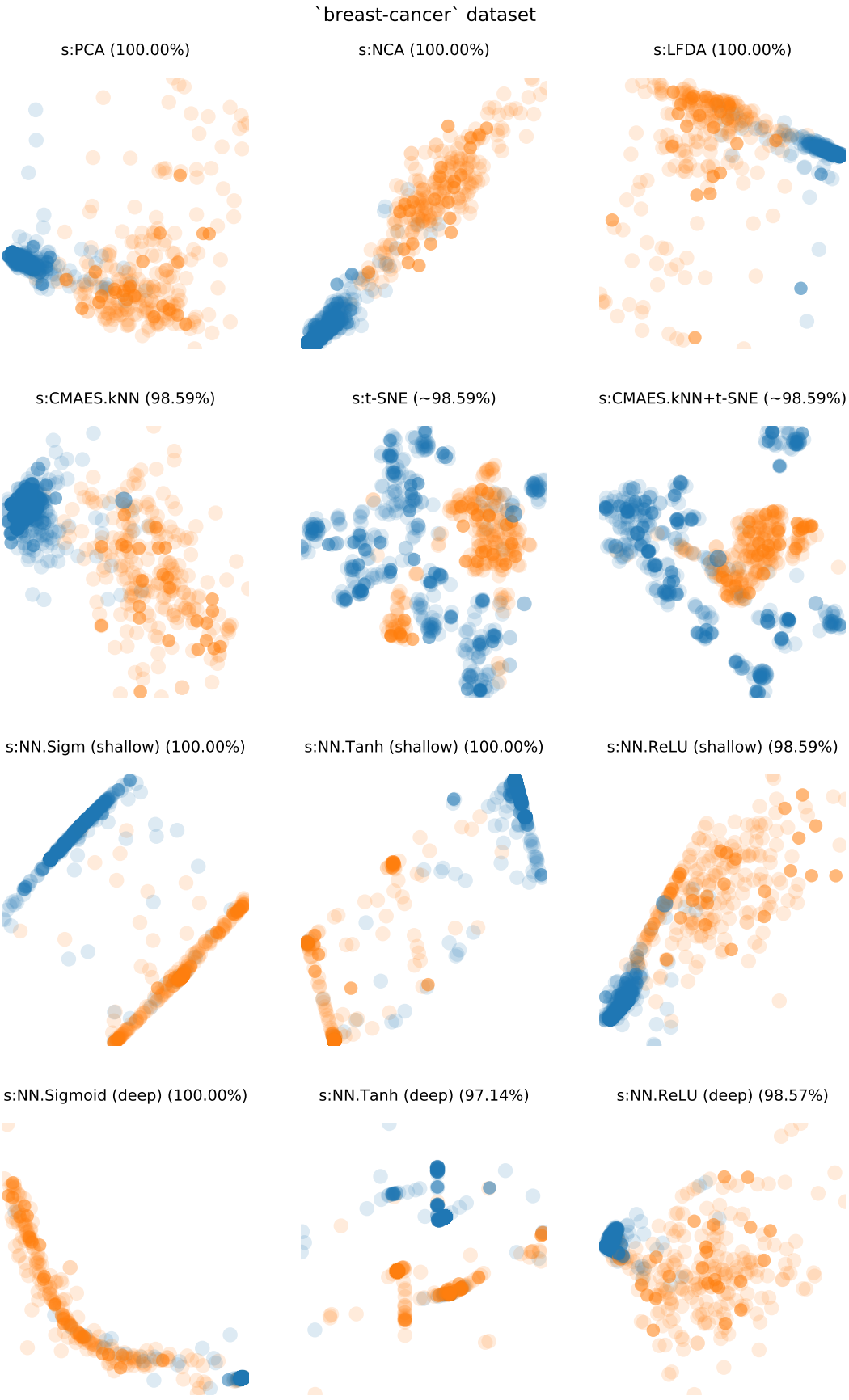

Figure 4.22: Visualisation of *breast-cancer* dataset with DML methods

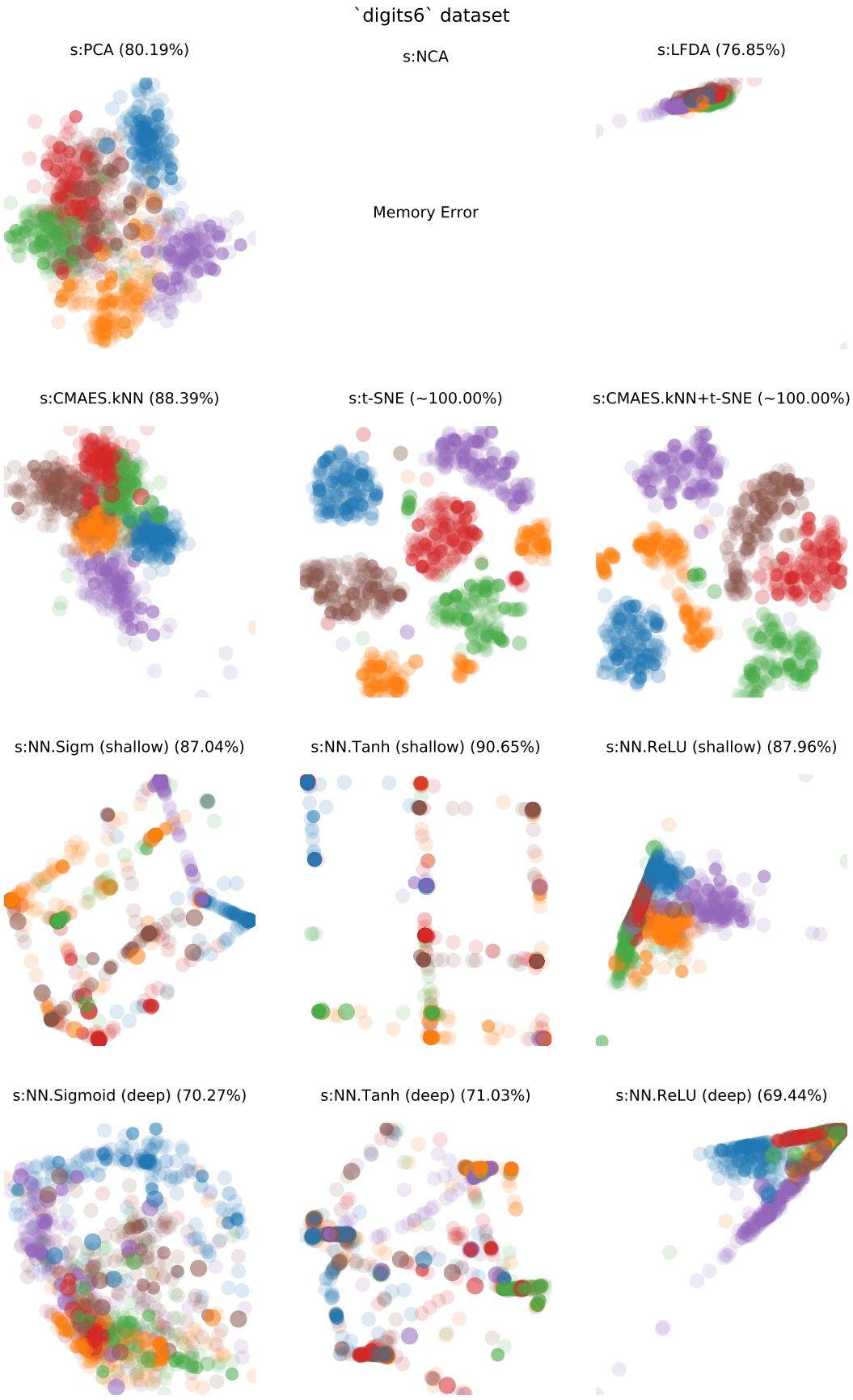

Figure 4.23: Visualisation of *digits6* dataset with DML methods

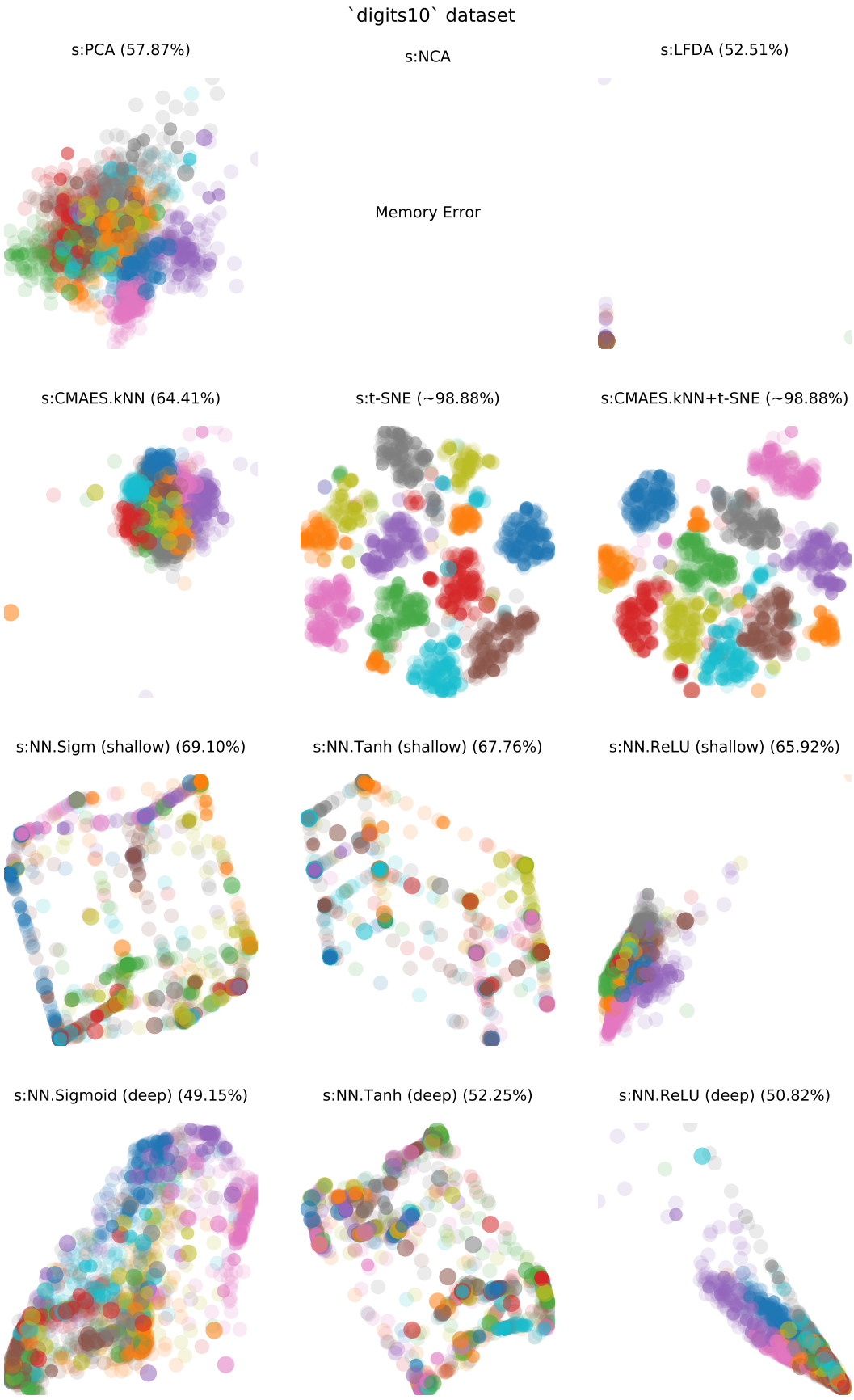

Figure 4.24: Visualisation of *digits10* dataset with DML methods

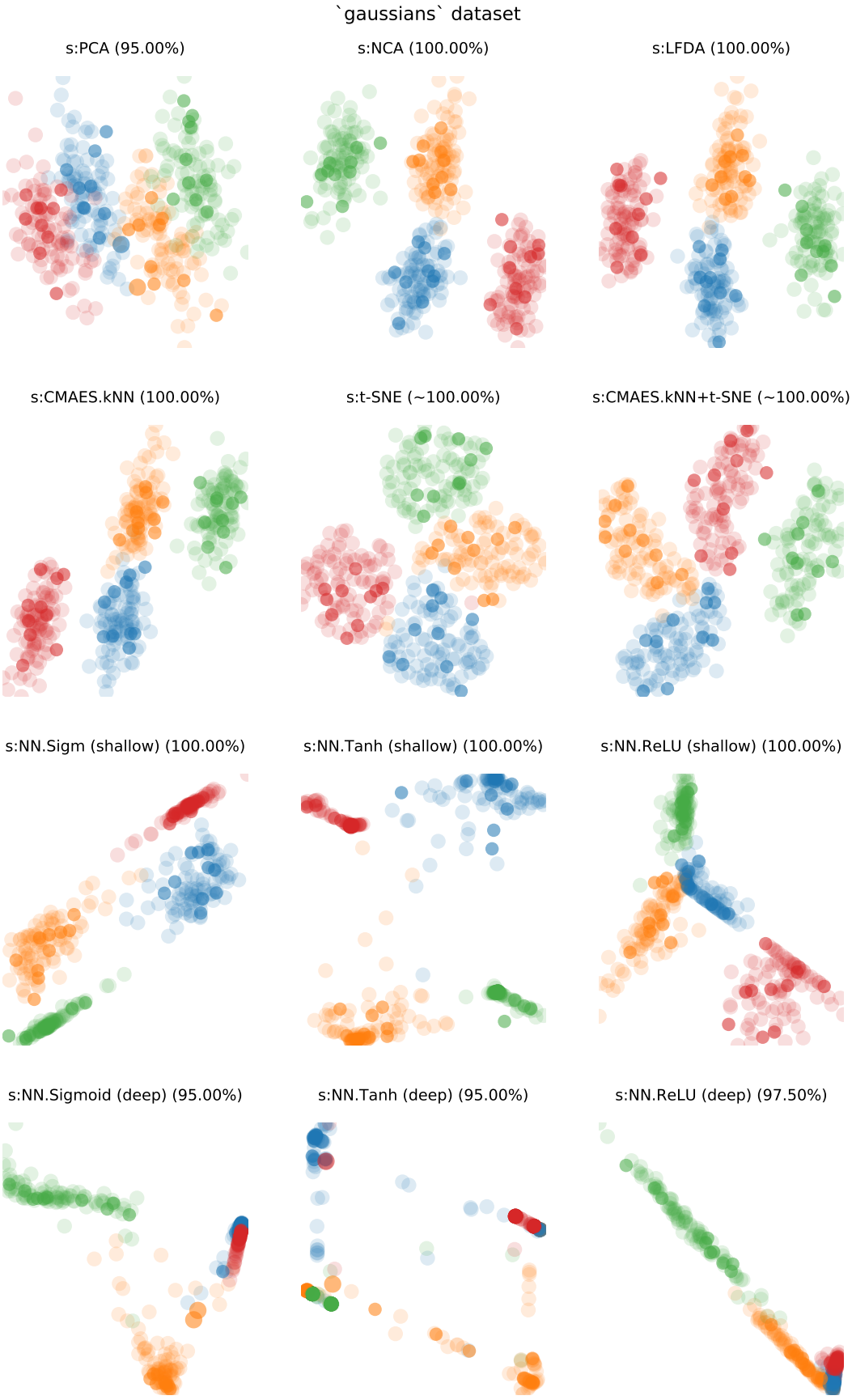

Figure 4.25: Visualisation of *gaussians* dataset with DML methods

Figure 4.26: Visualisation of *iris* dataset with DML methods

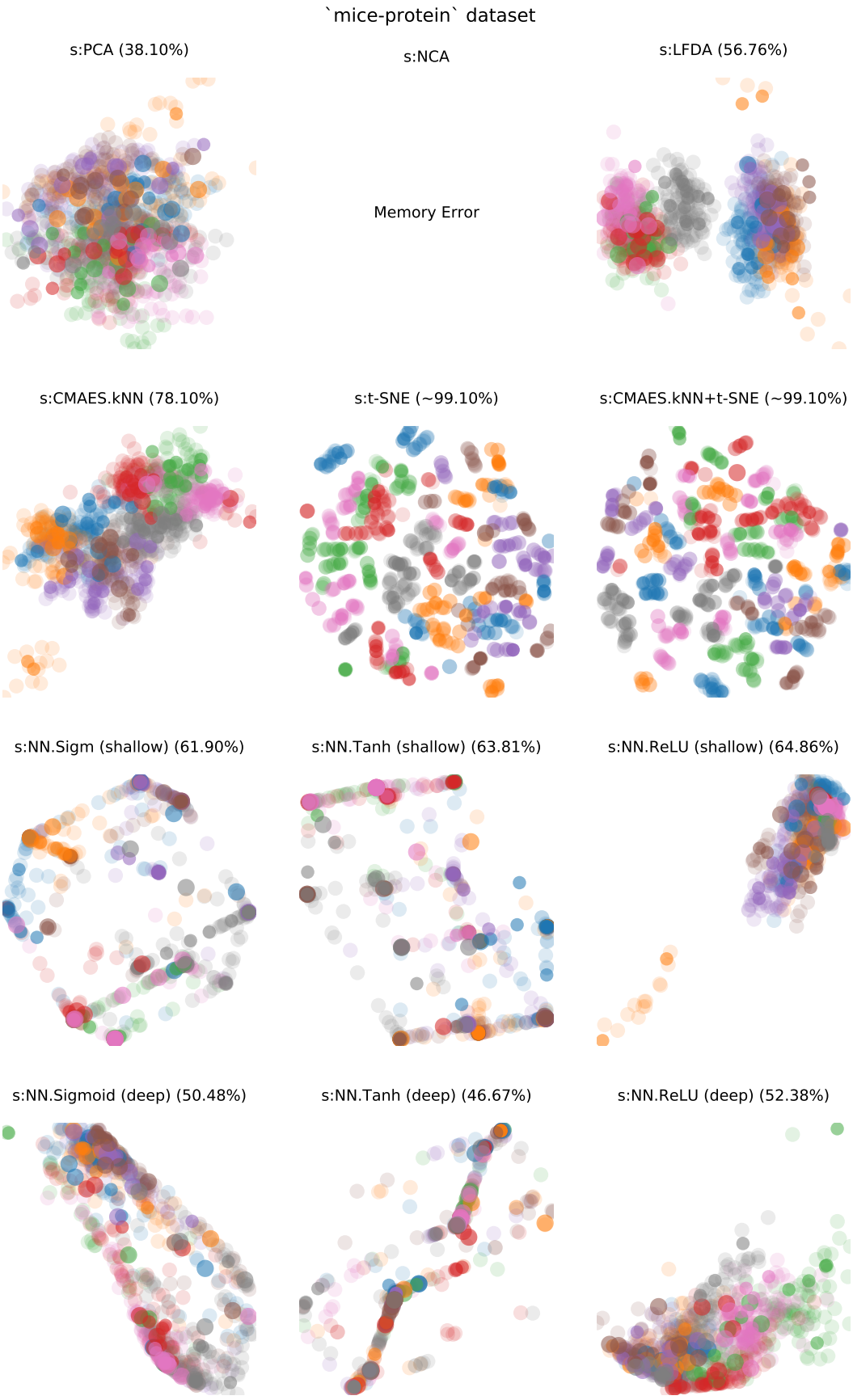

Figure 4.27: Visualisation of *mice-protein* dataset with DML methods

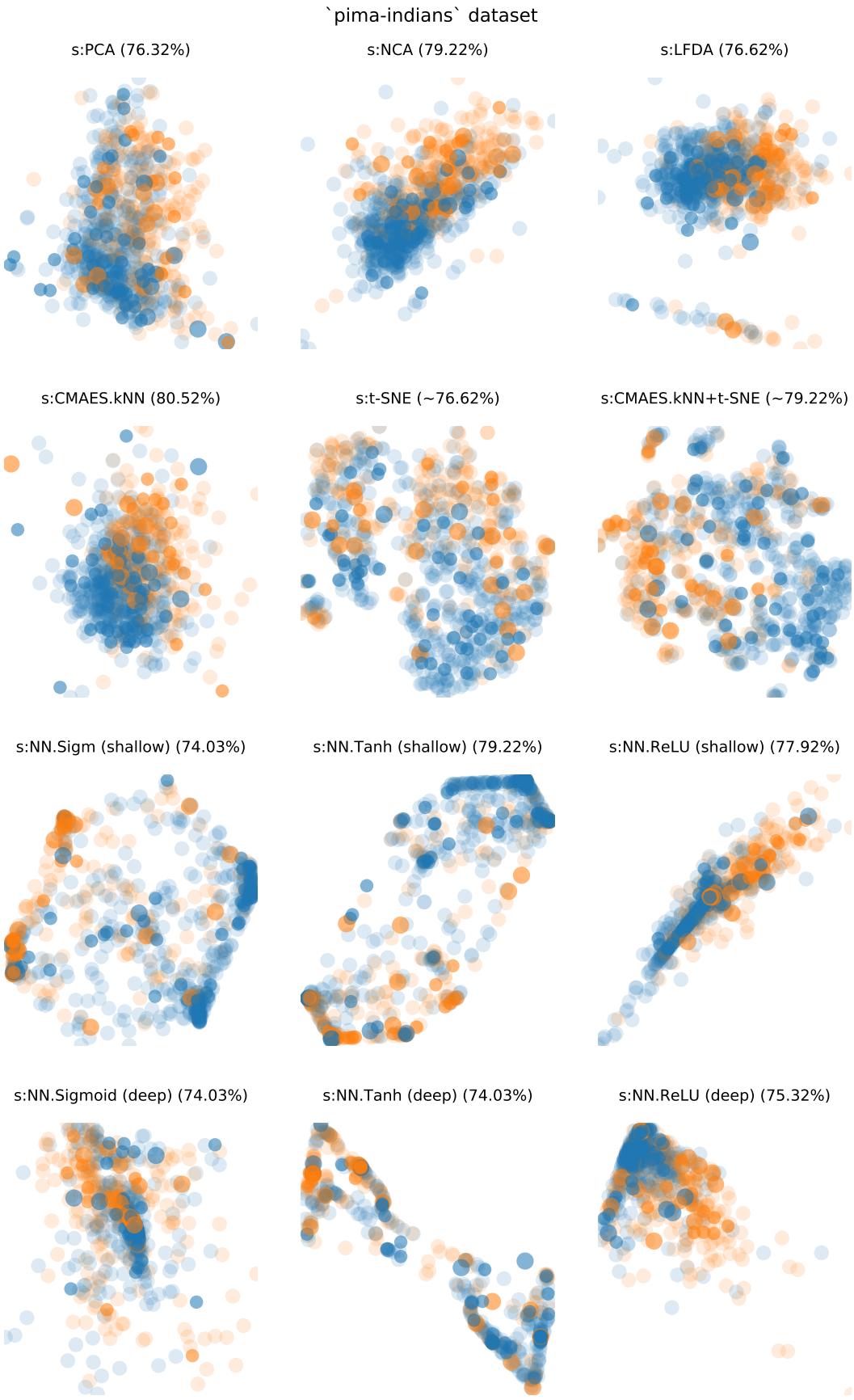

Figure 4.28: Visualisation of *pima-indians* dataset with DML methods

Figure 4.29: Visualisation of *sonar* dataset with DML methods

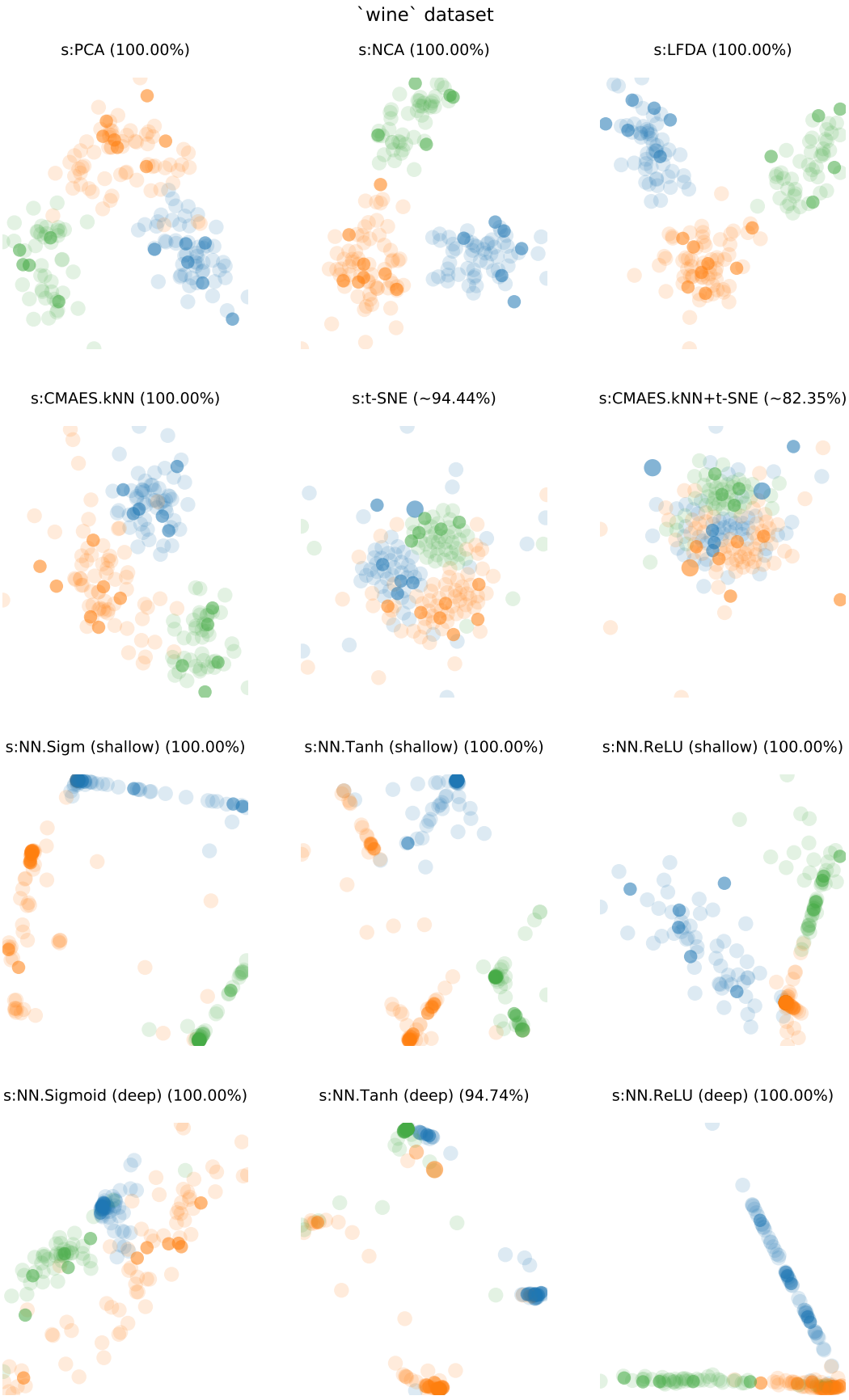

Figure 4.30: Visualisation of *wine* dataset with DML methods

5. Conclusion

We presented an evolutionary algorithm for metric learning and dimensionality reduction with a focus on visualisation. The algorithm extends previous ideas of [Fukui et al., 2013] and improves on its results. The main idea of the algorithm is to use a k-NN-based fitness function and CMA-ES in order to find distance metrics and mappings to lower-dimensional spaces. We have further extended the idea by replacing the usual linear transformation with a more complex non-linear mapping represented as neural networks.

Our algorithm was compared to standard Euclidean distance and state-of-the-art DML algorithms: LMNN, NCA, LFDA and jDE.fMe. In the metric learning experiment focused on improving classification, our algorithm was able to find metrics better than other algorithms in four out of the ten cases, while in the remaining cases the results were comparable to those obtained by the best algorithms. The k-NN classifier used as a fitness function performs better than using F-measure index.

In the dimensionality reduction experiment, our algorithm obtained the best results on five out of ten datasets. On the rest of the datasets, the other methods (especially NCA) obtained better results. However, these methods sometimes failed completely due to their complexity or numerical instability, while our proposed method does not suffer from these limitations.

Lastly, an experiment was performed to measure the generalisation performance of the evolutionary algorithms and their fitnesses. We found that jDE needs a much larger population to perform well, compared to CMA-ES. We also found that CMA-ES generalises well, but can overfit in some cases.

We also measured the scalability and time-performance of all algorithms and found that the proposed algorithm CMAES.kNN scales very well with number of samples and also scales well with an increasing dimensionality of the dataset. CMAES.kNN is $10 - 20\times$ faster than the competing evolutionary jDE.fMe algorithm, described by [Fukui et al., 2013], due to the population size. When compared to non-evolutionary methods, CMAES.kNN also performs several times faster than NCA, nevertheless, LMNN and LFDA are faster.

6. Future work

In spite of our best effort, not everything was finished and there is space for further extensions and research. We recommend focusing on these areas:

- Evolutionary algorithms are stochastic and multiple runs might improve the results in both classification and dimensionality reduction experiments.
- Our evolutionary method is sometimes overfitting the metric and not generalising well. This problem might be mitigated using an early stopping technique.
- Another approach to overfitting might be adding a regularising technique to the fitness function, such as Frobenius Norm [Schultz and Joachims, 2003] or LogDet divergence [Davis et al., 2007].
- Our work focused mainly on learning linear metrics, but further extensions to non-linear metrics are possible using the kernel trick [Chatpatanasiri et al., 2010] or multiple local metrics as discussed in [Bellet et al., 2013]. We also experimented with fully connected neural networks, however, more experiments can be conducted using other neural network architectures.
- Instead of choosing a single individual with the best fitness across all generations for the final transformation, multiple individuals can be combined together to form the final transformation.
- Additional objectives can be added to the fitness function together with the k-NN classifier to improve the generalisation. Some clustering indexes (e.g. purity, homogeneity) could perform well with k-NN in describing the final embedding. CMA-ES can be naturally extended to handle multi-objective optimisation [Igel et al., 2007].
- In our work we evolved a transformation matrix that could be used to form Mahalanobis distance. We focused on evolving a full and a diagonal matrix, however, evolving a triangular matrix could be a good compromise between model performance and computation feasibility.
- It might be possible to evaluate the fitness on subsamples of the dataset to speed up the evolution, similarly to the way how batches are used in stochastic gradient descend to improve the speed of the algorithm.

Bibliography

M Ehsan Abbasnejad, Dhanesh Ramachandram, and Rajeswari Mandava. A survey of the state of the art in learning the kernels. *Knowledge and information systems*, 31(2):193–221, 2012.

Aurélien Bellet, Amaury Habrard, and Marc Sebban. A survey on metric learning for feature vectors and structured data. *arXiv preprint arXiv:1306.6709*, 2013.

Xianye Ben, Weixiao Meng, Rui Yan, and Kejun Wang. An improved biometrics technique based on metric learning approach. *Neurocomputing*, 97:44–51, 2012.

Janez Brest, Sao Greiner, Borko Boskovic, Marjan Mernik, and Viljem Zumer. Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *IEEE transactions on evolutionary computation*, 10(6):646–657, 2006.

Serhat S Bucak, Rong Jin, and Anil K Jain. Multiple kernel learning for visual object recognition: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(7):1354–1369, 2014.

Ratthachat Chatpatanasiri, Teesid Korsrilabutr, Pasakorn Tangchanachaianan, and Boonserm Kijsirikul. A new kernelization framework for mahalanobis distance learning algorithms. *Neurocomputing*, 73(10):1570–1579, 2010.

Lin Chen, Lixin Duan, and Dong Xu. Event recognition in videos by learning from heterogeneous web sources. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2666–2673, 2013.

Pierre Comon. Independent component analysis, a new concept? *Signal processing*, 36(3):287–314, 1994.

Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

Jason V Davis, Brian Kulis, Prateek Jain, Suvrit Sra, and Inderjit S Dhillon. Information-theoretic metric learning. In *Proceedings of the 24th international conference on Machine learning*, pages 209–216. ACM, 2007.

Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188, 1936.

Imola K Fodor. A survey of dimension reduction techniques. *Center for Applied Scientific Computing, Lawrence Livermore National Laboratory*, 9:1–18, 2002.

Andrea Frome, Yoram Singer, Fei Sha, and Jitendra Malik. Learning globally-consistent local distance functions for shape-based image retrieval and classification. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, 2007.

Ken-Ichi Fukui, Satoshi Ono, Taishi Megano, and Masayuki Numao. Evolutionary distance metric learning approach to semi-supervised clustering with neighbor relations. In *Tools with Artificial Intelligence (ICTAI), 2013 IEEE 25th International Conference on*, pages 398–403. IEEE, 2013.

Mehmet Gönen and Ethem Alpaydın. Multiple kernel learning algorithms. *Journal of Machine Learning Research*, 12(Jul):2211–2268, 2011.

Matthieu Guillaumin, Thomas Mensink, Jakob Verbeek, and Cordelia Schmid. Tagprop: Discriminative metric learning in nearest neighbor models for image auto-annotation. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 309–316. IEEE, 2009a.

Matthieu Guillaumin, Jakob Verbeek, and Cordelia Schmid. Is that you? metric learning approaches for face identification. In *Computer Vision, 2009 IEEE 12th international conference on*, pages 498–505. IEEE, 2009b.

Nikolaus Hansen. The cma evolution strategy: a comparing review. *Towards a new evolutionary computation*, pages 75–102, 2006.

Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

Xiaofei He and Partha Niyogi. Locality preserving projections. In *NIPS*, volume 16, 2003.

Christian Igel, Nikolaus Hansen, and Stefan Roth. Covariance matrix adaptation for multi-objective optimization. *Evolutionary computation*, 15(1):1–28, 2007.

Sam Roweis Jacob Goldberger and Ruslan Salakhutdinov Geoff Hinton. Neighbourhood components analysis. *NIPS'04*, 2004.

Nan Jiang, Wenyu Liu, and Ying Wu. Order determination and sparsity-regularized metric learning adaptive visual tracking. In *Computer Vision and*

Pattern Recognition (CVPR), 2012 IEEE Conference on, pages 1956–1963. IEEE, 2012.

Ian Jolliffe. Principal component analysis. *Wiley StatsRef: Statistics Reference Online*, 2002.

IT Jolliffe. Principal component analysis. 1986. *Spring-verlag, New York*, 1986.

Tsuyoshi Kato and Nozomi Nagano. Metric learning for enzyme active-site search. *Bioinformatics*, 26(21):2698–2704, 2010.

Dor Kedem, Stephen Tyree, Fei Sha, Gert R Lanckriet, and Kilian Q Weinberger. Non-linear metric learning. In *Advances in Neural Information Processing Systems*, pages 2573–2581, 2012.

David Koloseni, Jouni Lampinen, and Pasi Luukka. Optimized distance metrics for differential evolution based nearest prototype classifier. *Expert Systems With Applications*, 39(12):10564–10570, 2012.

Joseph B Kruskal and Myron Wish. *Multidimensional scaling*, volume 11. Sage, 1978.

Marc Teva Law, Carlos Sureda Gutierrez, Nicolas Thome, and Stóphane Gançarski. Structural and visual similarity learning for web page archiving. In *Content-Based Multimedia Indexing (CBMI), 2012 10th International Workshop on*, pages 1–6. IEEE, 2012.

Guy Lebanon. Metric learning for text documents. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(4):497–508, 2006.

Jung-Eun Lee, Rong Jin, and Anil K Jain. Rank-based distance metric learning: An application to image retrieval. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

Alex Lemon, Anthony Man-Cho So, Yinyu Ye, et al. Low-rank semidefinite programming: Theory and applications. *Foundations and Trends® in Optimization*, 2(1-2):1–156, 2016.

Xi Li, Chunhua Shen, Qinfeng Shi, Anthony Dick, and Anton van den Hengel. Non-sparse linear representations for visual tracking with online reservoir metric learning. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 1760–1767. IEEE, 2012.

Zhaohui Liang, Gang Zhang, Li Jiang, and Wenbin Fu. Learning a consistent pro-outcomes metric through kcca for an efficacy assessing model of acupuncture. *Journal of Chinese Medicine Research and Development (JCMRD)*, 1(3):79–88, 2012.

Daryl Lim, Gert RG Lanckriet, and Brian McFee. Robust structural metric learning. *ICML (1)*, 28:615–623, 2013.

Jiwen Lu, Xiuzhuang Zhou, Yap-Pen Tan, Yuanyuan Shang, and Jie Zhou. Neighborhood repulsed metric learning for kinship verification. *IEEE transactions on pattern analysis and machine intelligence*, 36(2):331–345, 2014.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

Prasanta Chandra Mahalanobis. On the generalized distance in statistics. *Proceedings of the National Institute of Sciences (Calcutta)*, 2:49–55, 1936.

Brian McFee and Gert R Lanckriet. Metric learning to rank. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 775–782, 2010.

Brian McFee, Luke Barrington, and Gert Lanckriet. Learning content similarity for music recommendation. *IEEE transactions on audio, speech, and language processing*, 20(8):2207–2218, 2012.

Thomas Mensink, Jakob Verbeek, Florent Perronnin, and Gabriela Csurka. Metric learning for large scale image classification: Generalizing to new classes at near-zero cost. *Computer Vision–ECCV 2012*, pages 488–501, 2012.

Shibin Parameswaran and Kilian Q Weinberger. Large margin multi-task metric learning. In *Advances in neural information processing systems*, pages 1867–1875, 2010.

Qi Qian, Rong Jin, Jinfeng Yi, Lijun Zhang, and Shenghuo Zhu. Efficient distance metric learning by adaptive sampling and mini-batch stochastic gradient descent (sgd). *Machine Learning*, 99(3):353–372, 2015.

Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.

Hiroto Saigo, Jean-Philippe Vert, and Tatsuya Akutsu. Optimizing amino acid substitution matrices with a local alignment kernel. *BMC bioinformatics*, 7(1): 246, 2006.

Matthew Schultz and Thorsten Joachims. Learning a distance metric from relative comparisons. In *NIPS*, volume 1, page 2, 2003.

Blake Shaw, Bert Huang, and Tony Jebara. Learning a distance metric from a network. In *Advances in Neural Information Processing Systems*, pages 1899–1907, 2011.

R Short and Keinosuke Fukunaga. The optimal distance measure for nearest neighbor classification. *IEEE transactions on Information Theory*, 27(5):622–627, 1981.

Masashi Sugiyama. Dimensionality reduction of multimodal labeled data by local fisher discriminant analysis. *Journal of machine learning research*, 8(May): 1027–1061, 2007.

Matthew E Taylor, Brian Kulis, and Fei Sha. Metric learning for reinforcement learning agents. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 777–784. International Foundation for Autonomous Agents and Multiagent Systems, 2011.

Laurens Van Der Maaten, Eric Postma, and Jaap Van den Herik. Dimensionality reduction: a comparative. *J Mach Learn Res*, 10:66–71, 2009.

Nakul Verma, Dhruv Mahajan, Sundararajan Sellamanickam, and Vinod Nair. Learning hierarchical similarity metrics. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2280–2287. IEEE, 2012.

Jingyan Wang, Xin Gao, Quanquan Wang, and Yongping Li. Prodis-contshc: learning protein dissimilarity measures and hierarchical context coherently for protein-protein comparison in protein database retrieval. *BMC bioinformatics*, 13(7):S2, 2012.

Kilian Q Weinberger and Lawrence K Saul. Fast solvers and efficient implementations for distance metric learning. In *Proceedings of the 25th international conference on Machine learning*, pages 1160–1167. ACM, 2008.

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

Eric P Xing, Andrew Y Ng, Michael I Jordan, and Stuart Russell. Distance metric learning with application to clustering with side-information. In *NIPS*, volume 15, page 12, 2002.

Huilin Xiong and Xue-wen Chen. Kernel-based distance metric learning for microarray data classification. *BMC bioinformatics*, 7(1):299, 2006.

Jun Yu, Meng Wang, and Dacheng Tao. Semisupervised multiview distance metric learning for cartoon synthesis. *IEEE Transactions on Image Processing*, 21(11):4636–4648, 2012.

Shi Yu, Tillmann Falck, Anneleen Daemen, Leon-Charles Tranchevent, Johan AK Suykens, Bart De Moor, and Yves Moreau. L 2-norm multiple kernel learning and its application to biomedical data fusion. *BMC bioinformatics*, 11(1):309, 2010.

List of Figures

1.1 Properties of metric learning algorithms. This work focuses on the highlighted categories. 5
1.2 Correlated data centred around origin 9
1.3 Uncorrelated data using Cholesky decomposition of covariance matrix 9

2.1 Illustration of the input space before and after training using LMNN 17
2.2 Schema of an evolutionary algorithm 19

4.1 Success rates for classification experiment on *balance-scale*, *breast-cancer*, *digits6*, *digits10*, *gaussians*, and *iris* datasets 36
4.2 Success rates for classification experiment on *mice-protein*, *pima-indians*, *sonar*, and *wine* datasets 37
4.3 Success rates of DML algorithms while fixing k in the final k-NN classifier 39
4.4 Success rates of DML algorithms when fixing given hyperparameters 41
4.5 Fitness evolution and generalisation on *balance-scale* dataset . . . 45
4.6 Fitness evolution and generalisation on *breast-cancer* dataset . . . 45
4.7 Fitness evolution and generalisation on *digits6* dataset 46
4.8 Fitness evolution and generalisation on *digits10* dataset 46
4.9 Fitness evolution and generalisation on *gaussians* dataset 47
4.10 Fitness evolution and generalisation on *iris* dataset 47
4.11 Fitness evolution and generalisation on *mice-protein* dataset . . . 48
4.12 Fitness evolution and generalisation on *pima-indians* dataset . . . 48
4.13 Fitness evolution and generalisation on *sonar* dataset 49
4.14 Fitness evolution and generalisation on *wine* dataset 49
4.15 jDE with linear and log population size on *digits10* dataset 51
4.16 jDE with linear and log population size on *gaussians* dataset . . . 51
4.17 jDE with linear and log population size on *mice-protein* dataset . 51
4.18 jDE with linear and log population size on *wine* dataset 51
4.19 Learning times for increasing number of samples with a fixed dimension (linear and log scale) 55
4.20 Learning times for increasing dimension with a fixed number of samples (linear and log scale) 56
4.21 Visualisation of *balance-scale* dataset with DML methods 61
4.22 Visualisation of *breast-cancer* dataset with DML methods 62
4.23 Visualisation of *digits6* dataset with DML methods 63

4.24 Visualisation of *digits10* dataset with DML methods 64
4.25 Visualisation of *gaussians* dataset with DML methods 65
4.26 Visualisation of *iris* dataset with DML methods 66
4.27 Visualisation of *mice-protein* dataset with DML methods 67
4.28 Visualisation of *pima-indians* dataset with DML methods 68
4.29 Visualisation of *sonar* dataset with DML methods 69
4.30 Visualisation of *wine* dataset with DML methods 70

List of Algorithms

1 Pseudocode for classification using k-NN 16
2 Pseudocode for differential evolution 20
3 Evolving a mapping using an evolutionary algorithm 25
4 k-NN as a fitness function in an evolutionary algorithm 26

List of Tables

2.1 Summary of related metric learning methods 23

4.1 Datasets overall summary 30
4.2 Datasets samples statistics 30
4.3 Values of hyperparameters used for each algorithm 32
4.4 Error rates (in %) for the classification experiment 35
4.5 Learning times (in seconds) for each method on every dataset . . 53
4.6 Error rates (in %) for the dimensionality reduction experiment . . 58

6.1 Stat. significance for the classification on *balance-scale* dataset . . 85
6.2 Stat. significance for the classification on *breast-cancer* dataset . . 85
6.3 Stat. significance for the classification on *digits6* dataset 86
6.4 Stat. significance for the classification on *digits10* dataset 86
6.5 Stat. significance for the classification on *gaussians* dataset 86
6.6 Stat. significance for the classification on *iris* dataset 87
6.7 Stat. significance for the classification on *mice-protein* dataset . . 87
6.8 Stat. significance for the classification on *pima-indians* dataset . . 87
6.9 Stat. significance for the classification on *sonar* dataset 88
6.10 Stat. significance for the classification on *wine* dataset 88
6.11 Stat. significance for the dim. reduction on *balance-scale* dataset . 89
6.12 Stat. significance for the dim. reduction on *breast-cancer* dataset 89
6.13 Stat. significance for the dim. reduction on *digits6* dataset 90
6.14 Stat. significance for the dim. reduction on *digits10* dataset . . . 90
6.15 Stat. significance for the dim. reduction on *gaussians* dataset . . 91
6.16 Stat. significance for the dim. reduction on *iris* dataset 91
6.17 Stat. significance for the dim. reduction on *mice-protein* dataset . 92
6.18 Stat. significance for the dim. reduction on *pima-indians* dataset . 92
6.19 Stat. significance for the dim. reduction on *sonar* dataset 93
6.20 Stat. significance for the dim. reduction on *wine* dataset 93

List of Abbreviations

CMA-ES Covariance Matrix Adaptation Evolution Strategy

CMAES.fMe CMA-ES with the F-measure fitness

CMAES.kNN CMA-ES with the kNN fitness

DML distance metric learning

HOF hall of fame

ICA Independent Component Analysis

jDE Self-adaptive Differential Evolution

jDE.fMe jDE with the F-measure fitness

jDE.kNN jDE with the kNN fitness

***k*-NN** k-Nearest Neighbour

LDA Linear Discriminant Analysis

LFDA Local Fisher Discriminant Analysis

LLE Locally linear embedding

LMNN Large Margin Nearest Neighbor

LPP Locality-Preserving Projection

MDS Multi-dimensional Scaling

MKL Multiple Kernel Learning

NCA Neighbourhood Components Analysis

PCA Principal Component Analysis

PD positive definite

PSD positive semi-definite

ReLU rectified linear unit

tanh hyperbolic tangent

t-SNE t-distributed Stochastic Neighbor Embedding

List of Notations

$\mathbb{R}$	Set of real numbers
$\mathbb{R}^d$	Set of d-dimensional real-valued vectors
$\mathbb{R}^{c \times d}$	Set of c×d real-valued matrices
$\mathbb{S}_+^d$	Cone of symmetric PSD $d \times d$ real-valued matrices
$\mathbf{x}$	An arbitrary vector
$\boldsymbol{M}$	An arbitrary matrix
$\boldsymbol{I}$	Identity matrix
$\boldsymbol{M} \succ 0$	PD matrix M
$\boldsymbol{M} \succeq 0$	PSD matrix M
$\|\cdot\|_p$	p-norm
$\|\cdot\|_{\mathcal{F}}$	Frobenius norm
$\|\cdot\|_*$	Nuclear norm
$\mathrm{tr}(\boldsymbol{M})$	Trace of matrix M
$\boldsymbol{M}^\top$	Transposition of matrix M
$[t]_+ = max(0, 1-t)$	Hinge loss function
ξ	Slack variable
Σ	Finite alphabet
$\mathcal{X} \subseteq \mathbb{R}^d$	Input (instance) space
$\mathcal{Y} = \{1, \ldots, c\}$	Set of c output labels
$\boldsymbol{z} = (\boldsymbol{x}, y) \in \mathcal{X} \times \mathcal{Y}$	An arbitrary labelled instance
$\mathcal{Z} = \{\boldsymbol{z}_i = (\boldsymbol{x}_i, y_i)\}_{i=1}^n$	Set of n labelled training instances
$\mathcal{S} = \{(x_i, x_j) \mid y_i = y_j\}$	Set of must-link constraints
$\mathcal{D} = \{(x_i, x_j) \mid y_i \neq y_j\}$	Set of cannot-link constraints
$d_{\mathbf{2}}$	Euclidean distance
$d_{\boldsymbol{M}}$	Mahalanobis distance parametrized by $\boldsymbol{M}$

Attachments

The tables showing statistical significance between individual methods are attached below, one table per dataset. Tables 6.1—6.10 correspond to experiment from Section 4.2 and Tables 6.11—6.20 correspond to experiment from Section 4.6. All experiments were performed using a 10-fold cross validation and then the t-test was used to calculate the p-value. The p-values are not shown in the tables directly, but they are encoded in asterisks: the smaller the p-value, the larger the significance between the models. More asterisks signify larger significance.

	s:Euclidean	s:Covariance	s:LMNN	s:NCA	s:LFDA	s:CMAES.kNN	s:CMAES.fMe	s:jDE.fMe	s:jDE.kNN
s:Euclidean	.	.	.	.	.	.	.	.	.
s:Covariance	.	.	.	.	.	.	.	.	.
s:LMNN	****	****	.	.	.	.	**	*	.
s:NCA	****	****	.	.	**	.	***	**	.
s:LFDA	**	**	.	.	.	.	.	.	.
s:CMAES.kNN	****	****	.	.	**	.	***	***	.
s:CMAES.fMe	.	.	.	.	.	.	.	.	.
s:jDE.fMe	.	.	.	.	.	.	.	.	.
s:jDE.kNN	****	****	.	.	**	.	***	***	.

**** $p < 0.0001$, *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$

Table 6.1: Stat. significance for the classification on *balance-scale* dataset

	s:Euclidean	s:Covariance	s:LMNN	s:NCA	s:LFDA	s:CMAES.kNN	s:CMAES.fMe	s:jDE.fMe	s:jDE.kNN
s:Euclidean	.	*	.	.	.	.	.	.	.
s:Covariance	.	.	.	.	.	.	.	.	.
s:LMNN	.	*	.	.	.	.	.	.	.
s:NCA	.	*	.	.	.	.	.	.	.
s:LFDA	.	**	.	.	.	.	.	.	.
s:CMAES.kNN	.	**	.	.	.	.	.	.	.
s:CMAES.fMe	.	*	.	****	.	.	.	.	.
s:jDE.fMe	.	*	**	.	.	.	.	.	.
s:jDE.kNN	.	**	.	.	.	.	.	.	.

**** $p < 0.0001$, *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$

Table 6.2: Stat. significance for the classification on *breast-cancer* dataset

	s:Euclidean	s:Covariance	s:LMNN	s:NCA	s:LFDA	s:CMAES.kNN	s:CMAES.fMe	s:jDE.fMe	s:jDE.kNN
s:Euclidean	.	?	.	?	?	.	.	.	.
s:Covariance	?	?	?	?	?	?	?	?	?
s:LMNN	.	?	.	?	?	.	*	.	.
s:NCA	?	?	?	?	?	?	?	?	?
s:LFDA	?	?	?	?	?	?	?	?	?
s:CMAES.kNN	.	?	.	?	?	.	.	.	.
s:CMAES.fMe	.	?	.	?	?	.	.	.	.
s:jDE.fMe	.	?	.	?	?	.	.	.	.
s:jDE.kNN	.	?	.	?	?	.	.	.	.

**** $p < 0.0001$, *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$

Table 6.3: Stat. significance for the classification on *digits6* dataset

	s:Euclidean	s:Covariance	s:LMNN	s:NCA	s:LFDA	s:CMAES.kNN	s:CMAES.fMe	s:jDE.fMe	s:jDE.kNN
s:Euclidean	.	?	.	?	?	.	.	.	.
s:Covariance	?	?	?	?	?	?	?	?	?
s:LMNN	*	?	.	?	?	.	***	***	**
s:NCA	?	?	?	?	?	?	?	?	?
s:LFDA	?	?	?	?	?	?	?	?	?
s:CMAES.kNN	.	?	.	?	?	.	*	*	.
s:CMAES.fMe	.	?	.	?	?	.	.	.	.
s:jDE.fMe	.	?	.	?	?	.	.	.	.
s:jDE.kNN	.	?	.	?	?	.	.	.	.

**** $p < 0.0001$, *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$

Table 6.4: Stat. significance for the classification on *digits10* dataset

	s:Euclidean	s:Covariance	s:LMNN	s:NCA	s:LFDA	s:CMAES.kNN	s:CMAES.fMe	s:jDE.fMe	s:jDE.kNN
s:Euclidean	.	.	.	.	.	.	.	.	.
s:Covariance	.	.	.	.	.	.	.	.	.
s:LMNN	.	.	.	.	.	.	.	.	.
s:NCA	.	.	.	.	.	.	.	.	.
s:LFDA	.	.	.	.	.	.	.	.	.
s:CMAES.kNN	.	.	.	.	.	.	.	.	.
s:CMAES.fMe	.	.	.	.	.	.	.	.	.
s:jDE.fMe	.	.	.	.	.	.	.	.	.
s:jDE.kNN	.	.	.	.	.	.	.	.	.

**** $p < 0.0001$, *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$

Table 6.5: Stat. significance for the classification on *gaussians* dataset

	s:Euclidean	s:Covariance	s:LMNN	s:NCA	s:LFDA	s:CMAES.kNN	s:CMAES.fMe	s:jDE.fMe	s:jDE.kNN
s:Euclidean	.	.	.	.	.	.	.	.	.
s:Covariance	.	.	.	.	.	.	.	.	.
s:LMNN	.	*	.	.	.	.	.	.	.
s:NCA	.	*	.	.	.	.	.	.	.
s:LFDA	.	*	.	.	.	.	.	.	.
s:CMAES.kNN	.	*	.	.	.	.	.	.	.
s:CMAES.fMe	.	*	.	.	.	.	.	.	.
s:jDE.fMe	.	.	.	.	.	.	.	.	.
s:jDE.kNN	.	*	.	.	.	.	.	.	.

**** $p < 0.0001$, *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$

Table 6.6: Stat. significance for the classification on *iris* dataset

	s:Euclidean	s:Covariance	s:LMNN	s:NCA	s:LFDA	s:CMAES.kNN	s:CMAES.fMe	s:jDE.fMe	s:jDE.kNN
s:Euclidean	.	*	.	?	?	.	.	.	.
s:Covariance	.	.	.	?	?	.	.	.	.
s:LMNN	.	*	.	?	?	.	.	.	.
s:NCA	?	?	?	?	?	?	?	?	?
s:LFDA	?	?	?	?	?	?	?	?	?
s:CMAES.kNN	.	*	.	?	?	.	.	.	.
s:CMAES.fMe	.	*	.	?	?	.	.	.	.
s:jDE.fMe	*	*	.	?	?	.	.	.	.
s:jDE.kNN	.	*	.	?	?	.	.	.	.

**** $p < 0.0001$, *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$

Table 6.7: Stat. significance for the classification on *mice-protein* dataset

	s:Euclidean	s:Covariance	s:LMNN	s:NCA	s:LFDA	s:CMAES.kNN	s:CMAES.fMe	s:jDE.fMe	s:jDE.kNN
s:Euclidean	.	.	.	.	.	.	.	.	.
s:Covariance	.	.	.	.	.	.	.	.	.
s:LMNN	.	.	.	.	.	.	.	.	.
s:NCA	.	.	.	.	.	.	.	*	.
s:LFDA	.	.	.	.	.	.	.	.	.
s:CMAES.kNN	.	**	.	.	.	.	.	.	.
s:CMAES.fMe	.	*	.	.	.	.	.	.	.
s:jDE.fMe	.	.	.	.	.	.	.	.	.
s:jDE.kNN	.	.	.	.	*	.	.	.	.

**** $p < 0.0001$, *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$

Table 6.8: Stat. significance for the classification on *pima-indians* dataset

	s:Euclidean	s:Covariance	s:LMNN	s:NCA	s:LFDA	s:CMAES.kNN	s:CMAES.fMe	s:jDE.fMe	s:jDE.kNN
s:Euclidean	.	.	.	.	.	.	.	.	.
s:Covariance	.	.	.	.	.	.	.	.	.
s:LMNN	.	*	.	.	.	.	.	*	.
s:NCA	.	.	.	.	.	.	.	.	.
s:LFDA	.	.	.	.	.	.	.	.	.
s:CMAES.kNN	.	*	.	.	.	.	*	*	.
s:CMAES.fMe	.	*	.	.	.	.	.	*	.
s:jDE.fMe	.	.	.	.	.	.	.	.	.
s:jDE.kNN	.	.	.	.	.	.	.	.	.

**** $p < 0.0001$, *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$

Table 6.9: Stat. significance for the classification on *sonar* dataset

	s:Euclidean	s:Covariance	s:LMNN	s:NCA	s:LFDA	s:CMAES.kNN	s:CMAES.fMe	s:jDE.fMe	s:jDE.kNN
s:Euclidean	.	.	.	.	.	.	.	.	.
s:Covariance	.	.	.	.	.	.	.	.	.
s:LMNN	.	*	.	.	.	.	.	.	.
s:NCA	.	.	.	.	.	*	.	*	.
s:LFDA	.	*	.	.	.	.	.	.	.
s:CMAES.kNN	.	.	.	.	.	.	.	*	.
s:CMAES.fMe	.	.	.	.	.	.	.	.	**
s:jDE.fMe	.	.	.	.	.	.	.	.	.
s:jDE.kNN	.	.	.	.	.	.	.	.	.

**** $p < 0.0001$, *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$

Table 6.10: Stat. significance for the classification on *wine* dataset

	s:PCA	s:NCA	s:LFDA	s:CMAES.kNN	s:t-SNE	s:CMAES.kNN+t-SNE	s:NN.sigm (shallow)	s:NN.tanh (shallow)	s:NN.ReLU (shallow)	s:NN.sigm (deep)	s:NN.tanh (deep)	s:NN.ReLU (deep)
s:PCA	.	.	.	.	.	.	.	.	.	.	.	.
s:NCA	****	.	.	.	****	.	.	.	.	.	**	*
s:LFDA	****	.	.	.	****	.	.	.	.	.	***	*
s:CMAES.kNN	****	.	.	.	****	.	.	.	.	*	***	**
s:t-SNE	***	.	.	.	.	.	.	.	.	.	.	.
s:CMAES.kNN+t-SNE	****	*	**	.	****	.	*	.	**	***	****	****
s:NN.sigm (shallow)	****	.	.	.	****	.	.	.	.	.	***	*
s:NN.tanh (shallow)	****	.	*	.	****	.	.	.	*	**	****	***
s:NN.ReLU (shallow)	****	.	.	.	****	.	.	.	.	.	**	.
s:NN.sigm (deep)	****	.	.	.	***	.	.	.	.	.	.	.
s:NN.tanh (deep)	****	.	.	.	*	.	.	.	.	.	.	.
s:NN.ReLU (deep)	****	.	.	.	****	.	.	.	.	.	*	.

**** $p < 0.0001$, *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$

Table 6.11: Stat. significance for the dim. reduction on *balance-scale* dataset

	s:PCA	s:NCA	s:LFDA	s:CMAES.kNN	s:t-SNE	s:CMAES.kNN+t-SNE	s:NN.sigm (shallow)	s:NN.tanh (shallow)	s:NN.ReLU (shallow)	s:NN.sigm (deep)	s:NN.tanh (deep)	s:NN.ReLU (deep)
s:PCA	.	.	.	.	.	.	.	.	.	.	.	.
s:NCA	.	.	.	.	.	.	.	.	.	.	.	.
s:LFDA	.	.	.	.	.	.	.	.	.	.	.	.
s:CMAES.kNN	.	.	.	.	.	.	.	.	.	.	.	.
s:t-SNE	.	.	.	.	.	.	.	.	.	.	**	.
s:CMAES.kNN+t-SNE	.	.	**	.	.	.	.	.	.	.	.	.
s:NN.sigm (shallow)	.	.	.	.	.	.	.	.	**	.	.	.
s:NN.tanh (shallow)	.	.	.	.	.	.	.	.	.	.	.	.
s:NN.ReLU (shallow)	.	.	.	.	.	.	.	.	.	.	.	.
s:NN.sigm (deep)	.	.	.	.	.	.	.	.	.	.	.	.
s:NN.tanh (deep)	.	.	.	.	.	.	.	.	.	.	.	.
s:NN.ReLU (deep)	.	.	.	.	.	.	.	.	.	.	.	.

**** $p < 0.0001$, *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$

Table 6.12: Stat. significance for the dim. reduction on *breast-cancer* dataset

	s:PCA	s:NCA	s:LFDA	s:CMAES.kNN	s:t-SNE	s:CMAES.kNN+t-SNE	s:NN.sigm (shallow)	s:NN.tanh (shallow)	s:NN.ReLU (shallow)	s:NN.sigm (deep)	s:NN.tanh (deep)	s:NN.ReLU (deep)
s:PCA	.	?	**	.	.	.	.	.	.	****	****	***
s:NCA	?	?	?	?	?	?	?	?	?	?	?	?
s:LFDA	.	?	.	.	.	.	.	.	.	.	.	.
s:CMAES.kNN	***	?	****	.	.	.	.	.	.	****	****	****
s:t-SNE	****	?	****	****	.	.	****	****	****	****	****	****
s:CMAES.kNN+t-SNE	****	?	****	****	.	.	****	****	****	****	****	****
s:NN.sigm (shallow)	**	?	****	.	.	.	.	.	.	****	****	****
s:NN.tanh (shallow)	****	?	****	.	.	.	.	.	.	****	****	****
s:NN.ReLU (shallow)	**	?	****	.	.	.	.	.	.	****	****	****
s:NN.sigm (deep)	.	?	.	.	.	.	.	.	.	.	.	.
s:NN.tanh (deep)	.	?	.	.	.	.	.	.	.	.	.	.
s:NN.ReLU (deep)	.	?	.	.	.	.	.	.	.	.	.	.

**** $p < 0.0001$, *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$

Table 6.13: Stat. significance for the dim. reduction on *digits6* dataset

	s:PCA	s:NCA	s:LFDA	s:CMAES.kNN	s:t-SNE	s:CMAES.kNN+t-SNE	s:NN.sigm (shallow)	s:NN.tanh (shallow)	s:NN.ReLU (shallow)	s:NN.sigm (deep)	s:NN.tanh (deep)	s:NN.ReLU (deep)
s:PCA	.	?	***	.	.	.	.	.	.	****	****	****
s:NCA	?	?	?	?	?	?	?	?	?	?	?	?
s:LFDA	.	?	.	.	.	.	.	.	.	.	.	.
s:CMAES.kNN	**	?	****	.	.	.	.	.	.	****	****	****
s:t-SNE	****	?	****	****	.	.	****	****	****	****	****	****
s:CMAES.kNN+t-SNE	****	?	****	****	.	.	****	****	****	****	****	****
s:NN.sigm (shallow)	***	?	****	.	.	.	.	.	.	****	****	****
s:NN.tanh (shallow)	**	?	****	.	.	.	.	.	.	****	****	****
s:NN.ReLU (shallow)	**	?	****	.	.	.	.	.	.	****	****	****
s:NN.sigm (deep)	.	?	.	.	.	.	.	.	.	.	.	.
s:NN.tanh (deep)	.	?	.	.	.	.	.	.	.	.	.	.
s:NN.ReLU (deep)	.	?	.	.	.	.	.	.	.	.	.	.

**** $p < 0.0001$, *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$

Table 6.14: Stat. significance for the dim. reduction on *digits10* dataset

	s:PCA	s:NCA	s:LFDA	s:CMAES.kNN	s:t-SNE	s:CMAES.kNN+t-SNE	s:NN.sigm (shallow)	s:NN.tanh (shallow)	s:NN.ReLU (shallow)	s:NN.sigm (deep)	s:NN.tanh (deep)	s:NN.ReLU (deep)
s:PCA	.	.	.	.	.	.	.	.	.	.	.	.
s:NCA	****	.	.	.	*	.	*	*	*	****	****	***
s:LFDA	****	.	.	.	**	.	**	*	**	****	****	***
s:CMAES.kNN	****	.	.	.	.	.	.	.	.	****	***	***
s:t-SNE	***	.	.	.	.	.	.	.	.	**	**	**
s:CMAES.kNN+t-SNE	****	.	.	.	.	.	.	.	.	***	***	**
s:NN.sigm (shallow)	****	.	.	.	.	.	.	.	.	***	***	**
s:NN.tanh (shallow)	***	.	.	.	.	.	.	.	.	**	**	**
s:NN.ReLU (shallow)	****	.	.	.	.	.	.	.	.	***	***	**
s:NN.sigm (deep)	.	.	.	.	.	.	.	.	.	.	*	.
s:NN.tanh (deep)	.	.	.	.	.	.	.	.	.	.	.	.
s:NN.ReLU (deep)	.	.	.	.	.	.	.	.	.	.	.	.

**** $p < 0.0001$, *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$

Table 6.15: Stat. significance for the dim. reduction on *gaussians* dataset

	s:PCA	s:NCA	s:LFDA	s:CMAES.kNN	s:t-SNE	s:CMAES.kNN+t-SNE	s:NN.sigm (shallow)	s:NN.tanh (shallow)	s:NN.ReLU (shallow)	s:NN.sigm (deep)	s:NN.tanh (deep)	s:NN.ReLU (deep)
s:PCA	.	.	.	.	*	*	.	.	.	.	*	.
s:NCA	***	.	.	.	***	***	.	.	.	*	****	.
s:LFDA	.	.	.	.	**	**	.	.	.	.	**	.
s:CMAES.kNN	.	.	.	.	**	**	.	.	.	.	**	.
s:t-SNE	.	.	.	.	.	.	.	.	.	.	.	.
s:CMAES.kNN+t-SNE	.	.	.	.	.	.	.	.	.	.	.	.
s:NN.sigm (shallow)	.	.	.	.	**	**	.	.	.	.	**	.
s:NN.tanh (shallow)	.	.	.	.	**	*	.	.	.	.	*	.
s:NN.ReLU (shallow)	.	.	.	.	**	**	.	.	.	.	**	.
s:NN.sigm (deep)	.	.	.	.	*	.	.	.	.	.	.	.
s:NN.tanh (deep)	.	.	.	.	.	.	.	.	.	.	.	.
s:NN.ReLU (deep)	*	.	.	.	**	**	.	.	.	.	***	.

**** $p < 0.0001$, *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$

Table 6.16: Stat. significance for the dim. reduction on *iris* dataset

	s:PCA	s:NCA	s:LFDA	s:CMAES.kNN	s:t-SNE	s:CMAES.kNN+t-SNE	s:NN.sigm (shallow)	s:NN.tanh (shallow)	s:NN.ReLU (shallow)	s:NN.sigm (deep)	s:NN.tanh (deep)	s:NN.ReLU (deep)
s:PCA	.	?	.	.	.	.	.	.	.	.	.	.
s:NCA	?	?	?	?	?	?	?	?	?	?	?	?
s:LFDA	.	?	.	.	.	.	.	.	.	.	.	.
s:CMAES.kNN	****	?	****	.	.	.	**	**	**	****	****	****
s:t-SNE	****	?	****	****	.	.	****	****	****	****	****	****
s:CMAES.kNN+t-SNE	****	?	****	****	.	.	****	****	****	****	****	****
s:NN.sigm (shallow)	****	?	**	.	.	.	.	.	.	***	****	***
s:NN.tanh (shallow)	****	?	**	.	.	.	.	.	.	***	****	***
s:NN.ReLU (shallow)	****	?	***	.	.	.	.	.	.	***	****	***
s:NN.sigm (deep)	****	?	.	.	.	.	.	.	.	.	.	.
s:NN.tanh (deep)	**	?	.	.	.	.	.	.	.	.	.	.
s:NN.ReLU (deep)	***	?	.	.	.	.	.	.	.	.	.	.

**** $p < 0.0001$, *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$

Table 6.17: Stat. significance for the dim. reduction on *mice-protein* dataset

	s:PCA	s:NCA	s:LFDA	s:CMAES.kNN	s:t-SNE	s:CMAES.kNN+t-SNE	s:NN.sigm (shallow)	s:NN.tanh (shallow)	s:NN.ReLU (shallow)	s:NN.sigm (deep)	s:NN.tanh (deep)	s:NN.ReLU (deep)
s:PCA	.	.	.	.	.	.	.	.	.	.	.	.
s:NCA	**	.	.	.	.	.	.	.	.	**	**	.
s:LFDA	*	.	.	.	.	.	.	.	.	**	*	.
s:CMAES.kNN	*	.	.	.	.	.	.	.	.	**	*	.
s:t-SNE	.	.	.	.	.	.	.	.	.	.	.	.
s:CMAES.kNN+t-SNE	*	.	.	.	.	.	.	.	.	**	*	.
s:NN.sigm (shallow)	.	.	.	.	.	.	.	.	.	*	.	.
s:NN.tanh (shallow)	.	.	.	.	.	.	.	.	.	.	.	.
s:NN.ReLU (shallow)	*	.	.	.	.	.	.	.	.	**	*	.
s:NN.sigm (deep)	.	.	.	.	.	.	.	.	.	.	.	.
s:NN.tanh (deep)	.	.	.	.	.	.	.	.	.	.	.	.
s:NN.ReLU (deep)	.	.	.	.	.	.	.	.	.	.	.	.

**** $p < 0.0001$, *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$

Table 6.18: Stat. significance for the dim. reduction on *pima-indians* dataset

	s:PCA	s:NCA	s:LFDA	s:CMAES.kNN	s:t-SNE	s:CMAES.kNN+t-SNE	s:NN.sigm (shallow)	s:NN.tanh (shallow)	s:NN.ReLU (shallow)	s:NN.sigm (deep)	s:NN.tanh (deep)	s:NN.ReLU (deep)
s:PCA	.	.	.	.	.	.	.	.	.	.	.	.
s:NCA	****	.	.	*	***	**	.	*	*	***	*	.
s:LFDA	****	.	.	**	***	**	.	*	*	****	*	.
s:CMAES.kNN	*	.	.	.	*	.	.	.	.	*	.	.
s:t-SNE	.	.	.	.	.	.	.	.	.	.	.	.
s:CMAES.kNN+t-SNE	.	.	.	.	.	.	.	.	.	.	.	.
s:NN.sigm (shallow)	***	.	.	*	***	*	.	.	.	***	*	.
s:NN.tanh (shallow)	.	.	.	.	.	.	.	.	.	.	.	.
s:NN.ReLU (shallow)	*	.	.	.	*	.	.	.	.	*	.	.
s:NN.sigm (deep)	.	.	.	.	.	.	.	.	.	.	.	.
s:NN.tanh (deep)	.	.	.	.	.	*	.	.	.	.	.	.
s:NN.ReLU (deep)	*	.	.	.	*	.	.	.	.	*	.	.

**** $p < 0.0001$, *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$

Table 6.19: Stat. significance for the dim. reduction on *sonar* dataset

	s:PCA	s:NCA	s:LFDA	s:CMAES.kNN	s:t-SNE	s:CMAES.kNN+t-SNE	s:NN.sigm (shallow)	s:NN.tanh (shallow)	s:NN.ReLU (shallow)	s:NN.sigm (deep)	s:NN.tanh (deep)	s:NN.ReLU (deep)
s:PCA	.	.	.	.	****	****	.	.	*	*	**	.
s:NCA	.	.	.	*	****	****	.	*	**	**	***	*
s:LFDA	.	*	.	*	****	****	.	*	**	**	***	*
s:CMAES.kNN	.	.	.	.	***	****	.	.	.	.	.	.
s:t-SNE	.	.	.	.	.	.	.	.	.	.	.	.
s:CMAES.kNN+t-SNE	.	.	.	.	.	.	.	.	.	.	.	.
s:NN.sigm (shallow)	.	.	.	.	****	****	.	.	.	.	*	.
s:NN.tanh (shallow)	.	.	.	.	****	****	.	.	.	.	*	.
s:NN.ReLU (shallow)	.	.	.	.	****	****	.	.	.	.	.	.
s:NN.sigm (deep)	.	.	.	.	***	***	.	.	.	.	.	.
s:NN.tanh (deep)	.	.	.	.	**	***	.	.	.	.	.	.
s:NN.ReLU (deep)	.	.	.	.	***	****	.	.	.	.	.	.

**** $p < 0.0001$, *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$

Table 6.20: Stat. significance for the dim. reduction on *wine* dataset