Metric learning for sequences in relational LVQ[*]

Bassam Mokbel [°], Benjamin Paassen [°],
Frank-Michael Schleif [•], and Barbara Hammer [°]

(°) CITEC centre of excellence, Bielefeld University
Inspiration 1, 33619 Bielefeld, Germany

(•) University of Birmingham, School of Computer Science
Birmingham B15 2TT, UK

(This is a preprint of the publication [26], as provided by the authors.)

Abstract

Metric learning constitutes a well-investigated field for vectorial data with successful applications, e.g. in computer vision, information retrieval, or bioinformatics. One particularly promising approach is offered by low-rank metric adaptation integrated into modern variants of learning vector quantization (LVQ). This technique is scalable with respect to both, data dimensionality and the number of data points, and it can be accompanied by strong guarantees of learning theory. Recent extensions of LVQ to general (dis-)similarity data have paved the way towards LVQ classifiers for non-vectorial, possibly discrete, structured objects such as sequences, which are addressed by classical alignment in bioinformatics applications. In this context, the choice of metric parameters plays a crucial role for the result, just as it does in the vectorial setting. In this contribution, we propose a metric learning scheme which allows for an autonomous learning of parameters (such as the underlying scoring matrix in sequence alignments) according to a given discriminative task in relational LVQ. Besides facilitating the often crucial and problematic choice of the scoring parameters in applications, this extension offers an increased interpretability of the results by pointing out structural invariances for the given task.

[*]Funding by the DFG priority programme 1527 *"autonomous learning"* under grant number HA 2719/6-1, and by the CITEC center of excellence, as well as the Marie Curie Intra-European Fellowship (IEF): FP7-PEOPLE-2012-IEF (FP7-327791-ProMoS) is gratefully acknowledged.

1 Introduction

1.1 Motivation and related work

Similarity-based classification or clustering constitutes a well-investigated field of research, two of the most popular methods, the k-nearest-neighbor classifier and k-means classification, falling into this category [8, 22]. One striking property of these techniques is that they can be extended easily to general metric structures, by substituting the Euclidean metric with a more general choice, such as alignment distances or structure kernels. Due to their crucial dependency on the metric, however, these techniques fail if the choice of the metric or its parameterization are not suited for the given task. This observation motivated research about metric adaptation strategies based on given training data: today, several highly efficient metric learners are readily available for the vectorial setting, and the area constitutes a well-established field of research, see e.g. the excellent overview articles [2, 21].

In the vectorial setting, metric learning generally aims at an automatic adaptation of the Euclidean metric towards a more general (possibly local) quadratic form based on auxiliary information. Most strategies act solely upon the metric and are not interlinked with the subsequent classification or clustering method. This has the advantage that efficient, usually convex optimization schemes can be derived. However, no such technique currently offers an adaptation which is efficient with respect to data size and dimensionality, which can deal with local metrics, and which can be accompanied by guarantees of learning theory.

By linking metric adaptation to the subsequent classification tool, the property of a convex cost function is lost, depending on the considered classifier. However, metric learning can be integrated efficiently into the classification scheme, and results from learning theory can be derived by referring to the resulting function class. This has been demonstrated in the context of learning vector quantization (LVQ), where metric learning opened the way towards efficient state-of-the-art results in various areas, including biomedical data analysis, robotic vision, and spectral analysis [4, 19, 1]. Because of the intuitive definition of models in terms of prototypical representatives, prototype-based methods like LVQ enjoy a wide popularity in application domains, particularly if human inspection and interaction are necessary, or life-long model adaptation is considered [29, 20, 18]. Modern LVQ schemes are accompanied by mathematical guarantees about their convergence behavior and generalization ability [30, 31]. Metric adaptation techniques in LVQ do not not only enhance the representational power of the classifier, but also facilitate interpretability by means of an attention focus regarding the input features and possible direct data visualization in case of low-rank matrices [30, 6].

Most classical LVQ approaches can process vectorial data only, limiting the suitability of these methods regarding complex data structures, such as sequences, trees or graph structures, for which a direct vectorial representation is often not available. Recent developments offer possible extensions of LVQ towards more general data, which are represented in terms of (dis)similarities only: kernel LVQ, relational LVQ, or generalizations thereof [13]. These techniques provide competitive results to modern kernel classifiers, see [13], but they are based on cost functions which relate to the distance of data to prototypes in a possibly complex structure space. An underlying implicit pseudo-Euclidean embedding opens the possibility of smooth prototype updates, even for discrete data structures. In this contribution, we focus on one variant which is suitable for data described by a general dissimilarity matrix, so-called *relational LVQ*.

Relational LVQ shares the sensitivity of LVQ with respect to a correct metric parameterization. For structure metrics, such as sequence alignment, metric parameters correspond to the choice of the underlying scoring matrix in case of symbolic sequences over a discrete alphabet, or the choice of relevance weights for the sequence entries in case of sequences of numeric vectors. Note that there exist ad hoc techniques how to pick a suitable scoring function e.g. in the biomedical domain: prime examples are given by the PAM or BLOSUM matrices often used for aligning DNA sequences, which rely on simple evolutionary models and corresponding data sets [14, 32]. It is, however, not clear in how far these scoring matrices are suitable for a given classification task. Thus, the question arises, how to extend metric learning strategies to the case of structure metrics.

It has been pointed out in a recent survey [2] that structure metric learning constitutes a novel, challenging area of research with high relevance, and only a few approaches exist particularly in the context of sequence alignment. Sequence alignment plays a major role in the biomedical domain, for processing time series data, or for string comparisons. Its optimum computation is usually based on dynamic programming or even more efficient approximations thereof. The question of how to infer an optimal scoring matrix from aligned sequences has been investigated under the umbrella term of 'inverse alignment'. Several promising approaches have been proposed in this context. While the resulting techniques can be accompanied by theoretical guarantees in simple settings, more complex approaches often rely on heuristics, see e.g. [12, 34, 3]. A popular platform which combines various adaptation methods for scoring functions is offered by SEDiL, for example [5].

In our scenario, however, we are dealing with the different question of how to infer structure metric parameters, given a classification task. Hence, optimal alignments are not known, rather data are separated into given classes, and metric parameters should be adapted such that sequences within one class are considered similar by the alignment. Eventually, this question aims at the identification of structural invariances for the given classification task at hand: which structural substitutions do not deteriorate the classification result? In this contribution, we will investigate in how far structure metric learning can be introduced into relational LVQ in a similar way as for its vectorial counterparts. For this purpose, we approximate discrete alignment by a differentiable function, and show that metric learning is possible based on the relational LVQ cost function and gradient mechanisms.

1.2 Scientific contributions and structure of the article

The paper presents the following key contributions:

- A novel approach for metric learning is proposed, driven by the cost function of the *relational LVQ* classification technique, in order to adapt parameters of a dissimilarity measure for structured data, in particular symbolic sequences. Metric adaptation is performed in conjunction with the classifier's own optimization procedure, providing a seamless integration.
- The proposed learning scheme is realized and demonstrated in particular for sequence alignment, where the complex choice of the underlying scoring parameters is inferred from the data. Practical experiments show how metric adaptation does not only facilitate class-discrimination, but also increases the interpretability of the classifier model.
- Several approximation techniques are investigated, in order to compensate for the inherent high computational cost of the metric learning algorithm.

The remainder of the paper is structured as follows: In Section 2, we will recall relational LVQ and its rationale shortly, before focusing on the considered metric, in our case sequence alignment. We will explain its objective and efficient computation via dynamic programming. By approximating the alignment with a smooth function, derivatives become well-defined, and metric adaptation can be integrated into the relational LVQ update rules. In this context, we introduce efficient approximations that warrant the feasibility of the algorithm. In Section 3, we demonstrate the behavior of our method in simple mock-up scenarios, where ground truth for the metric parameters is available, and the resulting cost surfaces can be inspected directly. Afterwards, in Section 4, we investigate the efficiency and effectiveness of the technique in two real-world examples, one dealing with discrete sequences from bioinformatics, where the scoring matrix is adapted, the other originating from the domain of educational tutoring systems, where metric parameters correspond to the relevance of multi-dimensional sequence entries. Finally, we discuss additional approximations to tackle large data sets in Section 5: on the one hand, alignment paths with small contribution can be ignored; on the other hand, general-purpose approximations, such as the Nyström technique, can be integrated easily into the workflow to reduce the number of necessary distance calculations. We briefly underline the validity of these techniques in one of our example scenarios, before closing with a conclusion and an outlook regarding future work in Section 6. We will occasionally refer to additional information in the Appendix Section 7.

2 Adaptive metric parameters in relational LVQ

2.1 Learning vector quantization for dissimilarity data

LVQ models aim at the classification of given data into a fixed number of classes. Assume data are assembled in a set $\mathcal{A}$. Then, an LVQ classifier is characterized by a fixed number of prototypes $\vec{w}^1, \ldots, \vec{w}^M \in \mathcal{A}$ which are equipped with labels $c(\vec{w}^j) \in \{1, \ldots, C\}$, where C is the number of classes. Classification is based on a distance measure $d : \mathcal{A} \times \mathcal{A} \to \mathbb{R}$ by a winner-takes-all scheme: a data point $\vec{a} \in \mathcal{A}$ is classified according to its closest prototype $\vec{a} \mapsto c(\vec{w}^i)$ where $d(\vec{a}, \vec{w}^i)$ is minimum. In classical LVQ variants, Euclidean data $\mathcal{A} = \mathbb{R}^n$ are usually considered and the dissimilarity measure d is provided by the squared Euclidean distance [20]. Metric learning schemes use a more general quadratic form d_λ, whereby metric parameters λ are optimized during LVQ training [30, 6].

Original LVQ variants are based on heuristic adaptation rules. We will rely on *generalized LVQ* (GLVQ) and its extension to relational data [13]. For training, labeled data are given of the form $(\vec{a}^i, c(\vec{a}^i))$, $i = 1, \ldots, N$. GLVQ minimizes the error term

$$E_{\text{GLVQ}} = \sum_{i=1}^{N} E^i_{\text{GLVQ}} = \sum_{i=1}^{N} \Phi\left(\frac{d^+(\vec{a}^i) - d^-(\vec{a}^i)}{d^+(\vec{a}^i) + d^-(\vec{a}^i)}\right) \tag{1}$$

where Φ is a monotonic function like the sigmoidal function, d^+ is the squared distance of $\vec{a}^i$ to the closest prototype with a matching label, and d^- refers to a non-matching label [30]. Since a data point is classified correctly iff d^- is larger than d^+, this cost function constitutes a reasonable choice. It has been shown that the difference $d^+ - d^-$ can be related to the so-called hypothesis margin of LVQ classifiers, a quantity which directly regulates the generalization ability of the resulting classifier. For numerical reasons, this numerator is normalized to the interval $[-1, 1]$ to prevent divergence of the prototypes. In the vectorial setting, prototype updates can directly be derived from this cost function via gradients, as detailed in [30], for example. Interestingly, for any differentiable metric such as a general quadratic form, metric parameters can be adapted simultaneously to the prototypes by simple gradient-based optimization techniques.

We are interested in the setting where $\mathcal{A}$ is a structure space such as sequential data and d is a structure metric, such as a pairwise alignment measure. Here, the problem occurs that a vectorial embedding of the training data is not fixed a priori. A general framework how to treat dissimilarity data was proposed in [27], and was transferred to LVQ techniques in [13]. Assume dissimilarity data are described by a symmetric matrix $\mathbf{D}$ with entries d_{ij} characterizing the dissimilarity $d_{ij} = d(\vec{a}^i, \vec{a}^j)$. We assume that dissimilarities are symmetric $d_{ij} = d_{ji}$, and have zero diagonal $d_{ii} = 0$. Under these conditions, it has been shown in [27] that an implicit, so-called pseudo-Euclidean embedding of data always exists, such that the dissimilarities are induced by a suitable symmetric bilinear form in this vector space. In general, the quadratic form can have negative Eigenvectors; it is positive semidefinite only iff the metric is in fact Euclidean, which is usually not the case for common structure metrics.

GLVQ can be extended to this setting by means of an implicit reference to this pseudo-Euclidean embedding, resulting in *relational LVQ* (RGLVQ) variants, see [27, 13]. It is assumed that vectors $\vec{a}^i$ induce the matrix entries d_{ij} via the pseudo-Euclidean embedding [27]. Prototypes are restricted to convex combinations of data in this pseudo-Euclidean space: $\vec{w}^j = \sum_i^N \alpha_i^j \vec{a}^i$ with $\sum_i^N \alpha_i^j = 1$. Then, dissimilarities can be computed as

$$d(\vec{a}^i, \vec{w}^j) = \sum_l^N \alpha_l^j d_{il} - 0.5 \sum_{ll'}^N \alpha_l^j \alpha_{l'}^j d_{ll'} \,.$$

This distance calculation is based on the coefficients $\vec{\alpha}^j$ and dissimilarities $\mathbf{D}$ only, without explicitly referring to the vectors $\vec{a}^i$, see [13]. Hence, classification of data is possible without computing the pseudo-Euclidean embedding of the data itself. Inserting these distance calculations into the error term of GLVQ results in a valid error function $E_{\mathrm{RGLVQ}} = \sum_{i=1}^N E_{\mathrm{RGLVQ}}^i$ for RGLVQ, which depends on the coefficients $\vec{\alpha}^j$ and dissimilarities $\mathbf{D}$ only. Learning rules for the prototypes can be derived, by a stochastic gradient descent with respect to the coefficients $\vec{\alpha}^j$: In every update step, given the sample number i, coefficients $\vec{\alpha}^+$ or $\vec{\alpha}^-$ of the closest correct or incorrect prototype are adapted as follows:

$$\begin{array}{lcl} \Delta\alpha_l^+ & \sim & -\,\Phi' \cdot \frac{2d^-(\vec{a}^i)}{(d^+(\vec{a}^i)+d^-(\vec{a}^i))^2} \cdot \left(d_{il} - \sum_{l'}^N \alpha_l^+ d_{ll'}\right) \\ \Delta\alpha_l^- & \sim & +\,\Phi' \cdot \frac{2d^+(\vec{a}^i)}{(d^+(\vec{a}^i)+d^-(\vec{a}^i))^2} \cdot \left(d_{il} - \sum_{l'}^N \alpha_l^- d_{ll'}\right) \end{array} \tag{2}$$

where Φ is evaluated at the position $\big(d^+(\vec{a}^i) - d^-(\vec{a}^i)\big)/\big(d^+(\vec{a}^i) + d^-(\vec{a}^i)\big)$.

As demonstrated in [13], RGLVQ provides state-of-the-art results in comparison to alternative classifiers such as support vector machines. It can be used directly for any data set described by a symmetric dissimilarity matrix, which is not required to be positive semidefinite, due to its reference to the coefficients in pseudo-Euclidean space rather than a direct kernelization of the update rules. However, RGLVQ has been proposed for a fixed dissimilarity measure $\mathbf{D}$, and it does not yet incorporate the adaptation of metric parameters in its current form.

2.2 Sequence alignment

We are interested in possibilities to extend RGLVQ by automatic metric adaptation schemes, aiming at a twofold goal: to improve the accuracy and generalization ability of the resulting prototype model, and to enhance its interpretability by learning explicit structural invariances in terms of metric parameters. In the following, we will consider one particularly relevant type of structured data and corresponding metric, namely sequential data and sequence alignment. Note that the proposed rationale can be extended to alternative structure metrics, as long as they are differentiable with respect to metric parameters.

Assume an alphabet Σ is given, which can be discrete or continuous. We denote sequences with entries $a_I \in \Sigma$ as $\bar{a} = (a_1, \ldots, a_I, \ldots, a_{|\bar{a}|})$. Thereby, their length $|\bar{a}|$ can vary. The set of all sequences is denoted as $\mathcal{A} = \Sigma^*$. We assume that a symmetric dissimilarity measure $d_\lambda : \Sigma \times \Sigma \to \mathbb{R}$, with zero self-dissimilarities, is given to quantify the dissimilarity between single elements of the alphabet. This measure involves parameters λ which we would like to adapt by means of metric learning. Common choices of the dissimilarity measure are, for example:

- A *scoring matrix* for discrete alphabets $|\Sigma| < \infty$:
  Let $k = a_I \in \Sigma,\ \ m = b_J \in \Sigma$ be symbols from the respective sequences $\bar{a}, \bar{b}$. Then, the dissimilarity $d_\lambda(a_I, b_J) = \lambda_{km} \geq 0$ specifies the substitution costs if symbol k is aligned with symbol m.
- A *relevance weighting* for vectorial sequence entries:
  Let $\vec{a}_I, \vec{b}_J \in \Sigma = \mathbb{R}^n$ be vectorial elements from the respective sequences $\bar{a}, \bar{b}$. The notation a_I^r refers to the r-th entry in the vector $\vec{a}_I = (a_I^1, \dots, a_I^n)$. Then, $d_\lambda(\vec{a}_I, \vec{b}_J) = \sum_{r=1}^{n} \lambda_r \cdot d_r(a_I^r, b_J^r)$ is a weighted sum of appropriate non-negative and symmetric dissimilarity measures d_r for each dimension. Therefore, the value $\lambda_r \geq 0$ specifies the 'relevance' of the r-th dimension for all sequence elements w.r.t. the given task.

Alignment incorporates the possibility of deletions and insertions to be able to compare two sequences of different lengths. For this purpose, the alphabet Σ is extended by a specific symbol, the gap "$-$". Similarly, the dissimilarity measure is extended to incorporate gaps, using the same symbol for simplicity:

$$d_\lambda : (\Sigma \cup \{-\})^2 \to \mathbb{R}$$

specifying the gap costs

$$d_\lambda(a_I, -) = d_\lambda(-, a_I) \geq 0\,.$$

We exclude the case of two gaps being aligned, by the choice $d_\lambda(-,-) = \infty$.

Based on these definitions, a dissimilarity measure for sequences can be defined via alignment: A (global) *alignment* of sequences $\bar{a}$ and $\bar{b}$ consists of extensions $\bar{a}^* \in (\Sigma \cup \{-\})^*$ and $\bar{b}^* \in (\Sigma \cup \{-\})^*$ by gaps such that $|\bar{a}^*| = |\bar{b}^*|$. The overall costs of a fixed alignment is comprised of the sum of pairwise local distances $d(a_i^*, b_i^*)$. The optimal *alignment costs* (which we also refer to as *alignment dissimilarity*) are given by the minimal achievable costs

$$d^*(\bar{a}, \bar{b}) = \min \left\{ \sum_{i=1}^{|\bar{a}^*|} d_\lambda(a_i^*, b_i^*) \;\middle|\; (\bar{a}^*, \bar{b}^*) \text{ is alignment of } (\bar{a}, \bar{b}) \right\}\,. \tag{3}$$

Although this definition inherently considers all possible arrangements (which is an exponential number), these costs can be computed efficiently based on the following dynamic programming (DP) scheme. We use the shorthand notation $\bar{a}(I) = (a_1, \dots, a_I)$ and $\bar{b}(J) = (b_1, \dots, b_J)$ to denote the first I or J components of a sequence. Then, the following Bellman equality holds for the alignment costs of the parts $\bar{a}(I)$ and $\bar{b}(J)$:

$$\begin{aligned} d^*(\bar{a}(0), \bar{b}(0)) &= 0 \ , \\ d^*(\bar{a}(0), \bar{b}(J)) &= \sum_{j=1}^{J} d_\lambda(-, b_j) \ , \\ d^*(\bar{a}(I), \bar{b}(0)) &= \sum_{i=1}^{I} d_\lambda(a_i, -) \ , \\ d^*(\bar{a}(I+1), \bar{b}(J+1)) &= \min \Big\{ A_{\texttt{Rep}} := d^*(\bar{a}(I), \bar{b}(J)) + d_\lambda(a_{I+1}, b_{J+1}), \\ &\qquad A_{\texttt{Ins}} := d^*(\bar{a}(I+1), \bar{b}(J)) + d_\lambda(-, b_{J+1}), \\ &\qquad A_{\texttt{Del}} := d^*(\bar{a}(I), \bar{b}(J+1)) + d_\lambda(a_{I+1}, -) \Big\} \ . \end{aligned} \tag{4}$$

Note that the three terms $A_{\texttt{Rep}}, A_{\texttt{Ins}}, A_{\texttt{Del}}$, respectively, refer to the cases

- *replacement*: symbols a_{I+1}, b_{J+1} are aligned (called *match* if $a_{I+1}{=}b_{J+1}$),
- *insertion*: symbol b_{J+1} is aligned with a gap,
- *deletion*: symbol a_{I+1} is aligned with a gap.

This recursive scheme can be computed efficiently in time and memory $\mathcal{O}(|\bar{a}| \cdot |\bar{b}|)$ based on DP.

2.3 Learning scoring parameters from labeled data

Sequence alignment crucially depends on the local dissimilarities d_λ, which in turn are determined by the parameters λ. For a discrete alphabet, these parameters correspond to the scoring matrix which quantifies the costs of substituting a symbol by another one (i.e. for symbolic replacements, insertions, or deletions). Based on the preliminary work in [25], we propose an adaptation of λ based on the RGLVQ error function, given labeled training data. This provides a way to automatically learn a suitable parameterization of the alignment dissimilarity for a given task.

We transfer the basic idea that was precedented for vectorial LVQ in [30]: simultaneously to prototype updates, the alignment parameters are optimized by means of a gradient descent based on the RGLVQ error. Thus, we consider the derivative of the summand E^i_{RGLVQ} corresponding to a sequence $\bar{a}^i$ with respect to one parameter λ_q in λ:

$$\begin{aligned} \frac{\partial E^i_{\text{RGLVQ}}}{\partial \lambda_q} = &\quad \Phi' \cdot \frac{2d^-(\bar{a}^i)}{(d^+(\bar{a}^i) + d^-(\bar{a}^i))^2} \cdot \frac{\partial d^+(\bar{a}^i)}{\partial \lambda_q} \\ & - \Phi' \cdot \frac{2d^+(\bar{a}^i)}{(d^+(\bar{a}^i) + d^-(\bar{a}^i))^2} \cdot \frac{\partial d^-(\bar{a}^i)}{\partial \lambda_q} \end{aligned} \tag{5}$$

with

$$\frac{\partial d(\bar{a}^i, \bar{w}^j)}{\partial \lambda_q} = \sum_l \alpha_l^j \partial d_{il}^* / \partial \lambda_q - 0.5 \sum_{ll'} \alpha_l^j \alpha_{l'}^j \partial d_{ll'}^* / \partial \lambda_q \tag{6}$$

where d_{il}^* refers to the alignment dissimilarity of sequences i and l. An alignment $d^*(\bar{a}, \bar{b})$ as introduced above is not differentiable. Therefore, we consider an approximation, which we call *soft alignment*. We substitute min by

$$\text{softmin}(x_1, \dots, x_n) = \sum_i x_i \cdot \frac{\exp(-\beta x_i)}{\sum_j \exp(-\beta x_j)}$$

with the derivative

$$\text{softmin}'(x_i) = \big(1 - \beta \cdot (x_i - \text{softmin}(x_1, \dots, x_n))\big) \cdot \frac{\exp(-\beta x_i)}{\sum_j \exp(-\beta x_j)} \ .$$

The derivative $\partial d^*(\bar{a}, \bar{b}) / \partial \lambda_q$ can be computed in a DP scheme analog to the alignment:

$$\begin{aligned}
\frac{\partial d^*(\bar{a}(0), \bar{b}(0))}{\partial \lambda_q} &= 0 \ , \qquad (7) \\
\frac{\partial d^*(\bar{a}(0), \bar{b}(J))}{\partial \lambda_q} &= \sum_{j=1}^{J} \frac{\partial d_\lambda(-, b_j)}{\partial \lambda_q} \ , \\
\frac{\partial d^*(\bar{a}(I), \bar{b}(0))}{\partial \lambda_q} &= \sum_{i=1}^{I} \frac{\partial d_\lambda(a_i, -)}{\partial \lambda_q} \ , \\
\frac{\partial d^*(\bar{a}(I+1), \bar{b}(J+1))}{\partial \lambda_q} &= \text{softmin}'(A_{\texttt{Rep}}) \cdot \left(\frac{\partial d^*(\bar{a}(I), \bar{b}(J))}{\partial \lambda_q} + \frac{\partial d_\lambda(a_{I+1}, b_{J+1})}{\partial \lambda_q} \right) \\
&+ \text{softmin}'(A_{\texttt{Ins}}) \cdot \left(\frac{\partial d^*(\bar{a}(I+1), \bar{b}(J))}{\partial \lambda_q} + \frac{\partial d_\lambda(-, b_{J+1})}{\partial \lambda_q} \right) \\
&+ \text{softmin}'(A_{\texttt{Del}}) \cdot \left(\frac{\partial d^*(\bar{a}(I), \bar{b}(J+1))}{\partial \lambda_q} + \frac{\partial d_\lambda(a_{I+1}, -)}{\partial \lambda_q} \right)
\end{aligned}$$

The full derivation of Equation 7 is specified in the Appendix Section 7.1.

The derivative $\partial d_\lambda\big(\bar{a}(I), \bar{b}(J)\big) / \partial \lambda_q$ depends on the choice of the dissimilarity measure d_λ. For the two particularly interesting cases of discrete symbolic, and vectorial sequence entries, we get:

- Dissimilarities for a discrete alphabet $d_\lambda(a_I, b_J)$, with scoring parameters λ_{km}:

$$\begin{aligned}
\frac{\partial d_\lambda(a_I, b_J)}{\partial \lambda_{km}} &= \delta(a_I, k) \cdot \delta(b_J, m) \\
\frac{\partial d_\lambda(a_I, -)}{\partial \lambda_{km}} &= \delta(a_I, k) \cdot \delta(-, m) \\
\frac{\partial d_\lambda(-, b_J)}{\partial \lambda_{km}} &= \delta(-, k) \cdot \delta(b_J, m), \ \text{with Kronecker-}\delta
\end{aligned}$$

- Dissimilarities for a vector alphabet $d_\lambda(\vec{a}_I, \vec{b}_J) = \sum_{r=1}^{n} \lambda_r \cdot d_r(a_I^r, b_J^r)$, with relevance weights λ_r:

$$\frac{\partial d_\lambda(a_I, b_J)}{\partial \lambda_r} = d_r(a_I^r, a_J^r)$$
$$\frac{\partial d_\lambda(a_I, -)}{\partial \lambda_r} = d_r(a_I^r, -)$$
$$\frac{\partial d_\lambda(-, b_J)}{\partial \lambda_r} = d_r(-, a_J^r)$$

  where, in the latter case, parameterized gap costs are considered as a suitable extension of d_r. For real numbers, this can be chosen as $d_r(a^r, \psi)$ for some constant $\psi \in \mathbb{R}$ such as $\psi = 0$, for example.

The costs of computing the derivative $\partial d^*(\bar{a}, \bar{b})$ are $\mathcal{O}(|\bar{a}| \cdot |\bar{b}|)$, as for alignment itself. This, however, has to be performed for every possible parameter. Further, due to the implicit prototype representation as a convex combination, it has to be computed for all pairs of sequences to achieve a single update step, see Eq. 6. Hence, costs amount to $\mathcal{O}\big(\,|\lambda| \cdot N^2 \cdot \max\big\{|\bar{a}| \,\big|\, \bar{a} \text{ is sequence in the training set}\big\}^2\big)$ for an update, where N denotes the number of training sequences, which is infeasible. Therefore, we will present an efficient approximation in the following, where every prototype is substituted by a convex combination over a fixed number of `k` data instances only.

Approximation of prototypes by closest exemplars Equation 6 contains two sums which both refer to *all* sequences $\bar{a}^l$ in the given set, weighted by a corresponding coefficient α_l^j. Therefore, computing the update for one sample $\bar{a}^i$ requires the derivatives for all sequences $\bar{a}^l,\ l \in \{1, \dots, N\}$.

To avoid this, we transfer an approximation principle from existing literature, and thereby restrict the dependency of metric updates to only a few 'exemplar' sequences per prototype. In [13], the authors have shown that positional updates of RGLVQ prototypes can be realized by a so-called `k`-*approximation* of the convex combination. This assumes that sparsity can be imposed on the weight vectors $\vec{\alpha}^j$ by restricting them to their largest `k` components, without loosing too much precision. Empirical results indicate that it works well for real data distributions, even when choosing $\mathtt{k} \ll N$.

Transferring this approximation to the representation of prototypes for metric adaptation, we calculate the derivative $\partial d(\bar{a}^i, \bar{w}^j)/\partial \lambda_q$ based only on a subset of sequences, namely the prototype's *exemplars* $\bar{a}^l,\ l \in \mathcal{E}_j$ where $\mathcal{E}_j$ is a set of indices with fixed size $\mathtt{k} = |\mathcal{E}_j|$. The indices $\mathcal{E}_j$ refer to the `k` largest components in the respective weight vector $\vec{\alpha}^j$. Therefore, the number of exemplars `k` is a meta-parameter in our method, which will be discussed further in Section 3.2. For the minimal choice $\mathtt{k} = 1$, the derivative reduces to the single term $\partial d^*_{il}/\partial \lambda_q$, i.e. a soft alignment derivative between the sample sequence $\bar{a}^i$ and only one exemplar $\bar{a}^l$. Even this coarse approximation seems to work well for practical data, as will be shown in later experiments. This approximation makes updates feasible, and allows for a user-controlled compromise between precision and speed of the metric adaptation. The complexity of a single update therefore reduces severely to $\mathcal{O}\big(\,|\lambda| \cdot \mathtt{k}^2 \cdot \max\big\{|\bar{a}| \;\big|\; \bar{a} \text{ is sequence in the training set}\big\}^2\big)$.

Hebbian learning as a limit case Finally, we want to point out that, in a limit case, the derived update rules strongly resemble Hebbian learning, hence the metric adaptation follows intuitive learning steps. We consider the limit where every prototype can be approximated by one data point, i.e. α_l^j is 0 for all but one l, so the approximation by $\mathtt{k} = 1$ is exact. Then, the derivative in Equation 6 is dominated by only one summand, namely the derivative of the alignment distance between a given training sequence and the corresponding prototype's single exemplar sequence. Further, the considered limit case refers to a crisp instead of a soft minimum, i.e. a softmin function with $\beta \to \infty$. Hence, only one path, the optimal alignment path, is relevant in the computation of the alignment dissimilarity. On this path, the contribution of a considered parameter is measured, as follows:

- for a specified pair of symbols, in case of a discrete alphabet, it is the number of the alignments of this pair on an optimal alignment path,
- for a given dimension in case of vectorial sequence entries, it is the optimal alignment distance restricted to the dimension in question.

A more formal demonstration is given in the Appendix Section 7.1.

For both settings, this number represents the learning stimulus, which (i) decreases the corresponding metric parameter if the labeling is correct, and (ii) increases the corresponding metric parameter if the labeling is incorrect. In general, normalization can take place, since the number of parameters $|\lambda|$ is fixed. Hence:

- For a discrete alphabet, in the limit, symbolic replacements are marked as costly if they contribute to a wrong labeling, while they become inexpensive if the labeling is correct.
- For vectorial alphabets, those vector dimensions are marked as relevant where the small values indicate a closeness to a correctly labeled prototype, while dimensions are marked as irrelevant otherwise.

3 Practical implementation

In this section, we will discuss the practical realization of the proposed metric learning strategy. First, we describe how the actual learning algorithm is implemented, followed by a discussion about meta-parameters and their influence. Thereafter, we investigate the algorithm's performance for artificial data in a first proof-of-concept evaluation and exemplify general characteristics of the error function.

3.1 Algorithm overview

To summarize our method, we provide pseudo-code in Algorithm 1 for the case of a discrete symbolic alphabet, i.e. the result of metric learning is a scoring matrix λ with entries λ_{km}. The algorithm works in a similar fashion for vectorial sequence entries. Since a learning step for the metric terms is more costly than an update of the prototypes, the former requiring alignment calculations, we always perform several prototype updates before addressing the metric parameters. We refer to this as a batch update since, typically, a batch of data points is considered. Similarly, metric parameter updates are performed in batches to avoid the necessity of recurring alignments for sequences in the batch.

As an initial solution for λ, see Line 1, a simple configuration is applied, in the following referred to as *equal costs*: we set $\lambda_{km} = 1/|\Sigma|$ for all pairs $(k,m) \in (\Sigma \cup \{-\})^2, k \neq m$, and add small random noise to break ties in the initial alignments. Only symbolic matches require no costs: $\lambda_{kk} = 0$. During the adaptation (see Line 12), small or negative values $\lambda_{km} < \epsilon = 0.005/|\Sigma|$ are reset to ϵ in order to keep $\mathbf{D}$ non-negative, and to ensure that an alignment always favors matches (k,k) over the trivial alternative of a deletion $(k,-)$ directly followed by an insertion $(-,k)$ or similar unnecessary replacements. RGLVQ requires symmetric dissimilarities in $\mathbf{D}$, which is ensured if the scoring matrix λ is itself symmetric. Therefore, we enforce the symmetry of λ after every update, in Line 13. We will refer to the part from Line 5 to 14 as one *epoch* of learning.

Algorithm 1: RGLVQ with metric adaptation

```
   Data: a set of sequences $\{\bar{a}^1, \dots, \bar{a}^N\} = \mathcal{S} \ni \bar{a}^i$ over an alphabet $\Sigma$
   Result: a set of prototypes $\{\vec{\alpha}^1, \dots, \vec{\alpha}^M\} \ni \vec{\alpha}^j$, a scoring matrix $\lambda$
1  initialize parameters $\lambda \in \mathbb{R}^{(|\Sigma|+1)^2}$, e.g. with equal costs as in Sec. 3.1
2  calculate all dissimilarities $\mathbf{D}$ according to $\lambda$
3  initialize prototypes $\vec{\alpha}^j$ near the center of the corresponding class
4  for number of epochs do
       // classic RGLVQ update:
5      perform (batch) update of prototypes $\vec{\alpha}^j$ acc. to Equation 2
       // find representative sequences for each prototype:
6      for $j = 1$ to $M$ do
7          determine k exemplar sequences $\bar{a}^l \in \mathcal{S}$ with indices $l \in \mathcal{E}_j$ for
           prototype $\vec{\alpha}^j$, as the k largest entries $\alpha_l^j$
       // update of metric parameters:
8      for $i = 1$ to $N$ do
9          foreach pair of symbols $(k, m) \in (\Sigma \cup \{-\})^2, k \neq m$ do
10             gradient descent step: $\lambda_{km} := \lambda_{km} - \eta \cdot \frac{\partial E^i_{\text{RGLVQ}}}{\partial \lambda_{km}}$
11             if $\lambda_{km} < \epsilon$ then
12                 enforce small positive costs by setting: $\lambda_{km} := \epsilon$
13         symmetrize: $\lambda := (\lambda^\top + \lambda) \,/\, 2$
14     re-calculate dissimilarities $\mathbf{D}$ according to new $\lambda$
```

3.2 Meta-parameters

Since our metric adaptation scheme optimizes the RGLVQ error function via a stochastic gradient descent, there are several meta-parameters that influence this learning process:

(I) RGLVQ meta-parameters

(II) the learning rate η

(III) the number of exemplars `k`

(IV) the ‘crispness’ β in the *softmin* function

(I) The *RGLVQ meta-parameters* are comprised of the number of training epochs, the prototype learning rate, and the number of prototypes. It has been observed in experiments with RGLVQ, that the algorithm is not sensitive to its meta-parameters: few prototypes often yield excellent results, and there is a small risk of overfitting even when a large number of prototypes is considered [13].

The necessary number of epochs and prototype learning rate are correlated, requiring a higher number of epochs when a smaller learning rate is chosen, and vice versa. In all our experiments, the number of epochs was fixed to 10. This choice is well justified, since a plausible convergence was achieved within the given time frame: during the last training epoch, the absolute error changes by less than 2% of the final error value, in every experiment.

The number of prototypes is crucial to determine the complexity of classification boundaries in RGLVQ, as is generally the case in prototype-based classifiers. For multimodal classes, too few prototypes lead to a high classification error. However, in particular in the light of an adaptive and hence very flexible metric, a good starting point is to train the classifier in the most simplistic setting with only one prototype per class, and increasing the number when necessary. To automatically adjust the number of prototypes, quite a few incremental variants of LVQ have been proposed, see e.g. [9, 17, 40]. Interestingly, for a complex image segmentation task, only few prototypes (3-4 per class, on average) were generated, supporting the claim that rather small LVQ networks already show representative behavior in particular in the context of an adaptive metric [9].

In our experiments, we will generally focus our discussion on the choice of one prototype per class, which allows us to emphasize the capability of adding sufficient complexity to the classifier model via metric adaptation only. For comparison, we will report the classifier performance using more prototypes, in addition to the highlighted results.

(II) The *learning rate* η for metric parameters is, in contrast to the prototype learning rate, a sensitive meta-parameter for the optimization via stochastic gradient descent. Considering parameters for alignment scoring in particular, changes in the gap costs (i.e. for deletions λ_{k-} and insertions λ_{-m}) have a stronger influence on the overall alignment than single pairwise replacement costs λ_{km}. Therefore, it can be advisable to assign separate learning rates $\eta_{\texttt{Gap}}$ and $\eta_{\texttt{Rep}}$ for the respective costs, similar to previous vectorial metric adaptation in the context of LVQ [30]. In this way, it is also possible to restrict the adaptation to parameters of interest, and limit the degrees of freedom for learning. In our experiments, we will not use this separation and generally maintain the simpler case of a single η.

(III) The *number of exemplars* $\texttt{k}$ determines by how many real sequences a prototype $\vec{w}^j$ is represented in the update rule for metric parameter learning. As described in the end of Section 2.3, this is an approximation of the precise theoretical update where $\texttt{k} = N$. While a lower number could hypothetically decrease precision, it has shown to work well in practice, even for choices $\texttt{k} \ll N$, for example $\texttt{k} = 1$. Since the approximation strongly influences the computational demand of a single update step, the parameter has an immense impact on the overall runtime. The minimum choice of $\texttt{k} = 1$ yields the fastest update calculation, and usually provides sufficiently accurate results from our practical experience. In fact, all experiments presented in this article rely on this setting, and we could achieve no considerable improvement in these cases, by choosing a larger number of exemplars $\mathtt{k} > 1$.

(IV) The *crispness in the softmin function* β influences the classifier training progress. In the following Sections 3.3 and 3.5, its direct effect on the convergence characteristics are discussed in artificial data scenarios. In Figure 1, we can see how a lower crispness (e.g. for $\beta = 2$) generally slows down the adaptation, while higher values seem to facilitate a faster convergence, sometimes at the expense of robustness (see $\beta = 80$ in Figure 1b). Generally, we can observe that β directly affects the convergence characteristics, with an optimal value lying in a medium range.

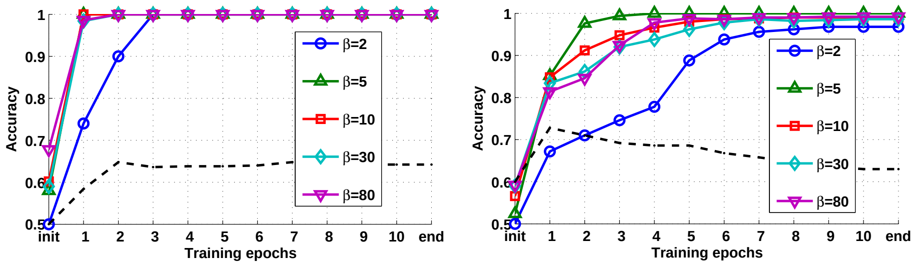
3.3 Proof-of-concept with artificial data

We designed two artificial data sets with class structures that demonstrate the method's ability to adequately adapt (i) replacement costs and (ii) gap costs for the case of discrete sequence entries. Both data sets contain random sequences which follow deliberate structural patterns, such that a specific parameter configuration in the scoring matrix λ leads to a perfect class separation, while a naive choice of costs λ causes severe overlaps between classes.

Replacement data: In this data set, all strings have 12 symbols, randomly generated from the alphabet $\Sigma = \{\mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{D}\}$ according to the regular expressions: $(\mathsf{A|B})^5$ $(\mathsf{A|B})$ $(\mathsf{C|D})$ $(\mathsf{C|D})^5$ for the first, and $(\mathsf{A|B})^5$ $(\mathsf{C|D})$ $(\mathsf{A|B})$ $(\mathsf{C|D})^5$ for the second class. Hence, replacements of A or B by C or D are discriminative, while replacements A with B, and C with D are not. After the training of λ, we expect high costs for discriminative replacements, while other replacement costs in λ are close to zero. Also, we expect positive gap costs, since gaps could otherwise circumvent the alignment of the discriminative middle parts.

Gap data: The second data set focuses on gap scoring. Strings in the first class are random sequences $\bar{a}^i \in \Sigma^{10}$ of length 10, whereas strings $\bar{a}^l \in \Sigma^{12}$ in the second class are longer by 2 symbols. Therefore, replacements of letters are not discriminative, while the introduction of any gaps discriminates classes. Thus, gap costs are expected to become high, while any other replacements should cost less.

Evaluation: For each data set, we created $N = 100$ sequences (50 for each class) and evaluated the average classifier performance in a 5-fold cross-validation with 5 repeats. RGLVQ was trained using one prototype per class, for 10 epochs. The learning rate for the adaptation of λ_{km} was set to $\eta = 1/N$, and the number of exemplars $\mathtt{k} = 1$. We use the aforementioned *equal costs* for the initial alignment parameters λ. Several settings of the 'crispness' β in the *softmin* function have been evaluated, but for now let us consider the intermediate setting of $\beta = 5$. We will discuss the influence of this meta-parameter later in this Section, and in Section 3.5.



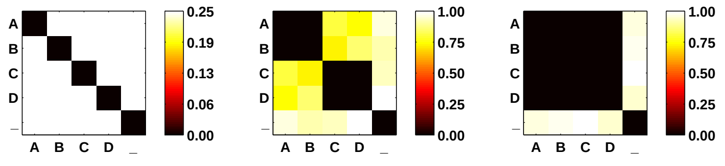
(a) *Replacement* data (b) *Gap* data

Figure 1: The figures show the average test accuracy achieved during 10 epochs of RGLVQ training in a 5-fold cross-validation with 5 repeats on artificial data sets. The dashed black line represents the training without adapting λ, and serves as a baseline in which the classifier remains close to random guessing. The other curves show the accuracies achieved with the proposed metric adaptation scheme, for different settings of the 'crispness' parameter β. The adaptation yields nearly perfect results in all settings, while the convergence characteristics are slightly affected by β.

The experimental results in Figure 1 show a drastically increased accuracy when adapting λ, for example, with $\beta = 5$ a perfect average test accuracy of 100% (with 0 deviation) was achieved after the 4th epoch. Consequently, the adapted λ represent ideal scoring matrices for both data sets, which exactly fulfill our aforementioned expectations: Figure 2 exemplarily shows the respective λ matrices before and after training from the last respective cross-validation run. For comparison, we trained RGLVQ in the classical fashion, based on fixed dissimilarities $\mathbf{D}$, without adapting the underlying scoring parameters. In this case, λ refers to the initial *equal costs*, which does not emphasize class-discrimination. As expected, classification remains close to random guessing in this setting, see the baseline in Figure 1: the average test accuracy after training was 64% for the *Replacement* data and 61% for the *Gap* data.

Figure 1 shows the progression of accuracy during training, for different values of the 'crispness' β. For lower settings (e.g. $\beta = 2$), we can see that the final level of accuracy is often achieved in later epochs, which indicates that the metric adaptation is slower. In contrast, higher values facilitate a faster adaptation, sometimes at the expense of robustness (see $\beta = 80$ in Figure 1b). In Section 3.5, we will demonstrate the influence of β in a soft alignment, implying its impact on the metric adaptation process and convergence characteristics.

From the proof-of-concept we can conclude that the proposed supervised metric adaptation strategy is indeed able to single-out discriminative parameters, which leads to a clear class separation and enables the training of a robust classifier model in our examples. The training arrives at the expected results, even for $\mathtt{k} = 1$, the most efficient approximation where each (virtual) prototype is represented by only one (tangible) exemplar sequence. In the following, we



(a) Standard/init. λ (b) *Replacement* data (c) *Gap* data

Figure 2: Visualizations of the scoring matrix λ, where color/intensity encodes the values. On the left is a standard choice of λ as *equal costs* which serves as the initial state for the training, the middle and right show the final state of λ after adaptation, fulfilling the expectations for the respective artificial data set.

will first observe the characteristics of the RGLVQ error function w.r.t. metric parameters in our toy scenario, and thereafter, take a closer look on the crispness in a soft alignment.

3.4 RGLVQ error function surface

To get an impression of the characteristics of the RGLVQ error function with regard to metric parameters, we visualize its values for varying parameter settings as a 3-D surface. Therefore, we simplify our artificial data sets even further, to restrict to only a few degrees of freedom in the parameters λ. We obtained an adapted λ, as well as prototype positions $\vec{\alpha}^1, \vec{\alpha}^2$ from a single training run of 10 epochs ($\beta = 10$, $\eta = 0.07/N$). To evaluate various configurations of λ, a pair of entries $(\lambda_{km}, \lambda_{qr})$ will be iterated over all combinations, while keeping the others fixed to their final state after training. Given the prototypes, we can visualize E_{RGLVQ} as a surface for all combinations $(\lambda_{km}, \lambda_{qr})$.

The *simplified Gap* data consists of random sequences over the two-letter alphabet $\Sigma = \{\mathsf{A}, \mathsf{B}\}$, as before with length 10 in the first, and length 12 in the second class, and $N = 100$. Again, the introduction of any gaps is crucial for class-discrimination, so a minimum of the error surface is expected for settings where both costs $\lambda_{\mathsf{A}-}$ and $\lambda_{\mathsf{B}-}$, become high. Figure 3 shows E_{RGLVQ} for configurations $(\lambda_{\mathsf{A}-}, \lambda_{\mathsf{B}-})$ in increasing steps of 0.1 over the interval $[0, 1]$. The remaining third parameter in λ is fixed to the final value after training, in this case it is close to the small constant $\lambda_{\mathsf{AB}} \approx \epsilon$. As expected, the error surface drops smoothly to a low plateau, when both gap costs are increased.

For the *simplified Replacement* data, we now use the three-letter alphabet $\Sigma = \{\mathsf{A}, \mathsf{B}, \mathsf{C}\}$, and regular expressions $(\mathsf{A}|\mathsf{B})^5$ B C $(\mathsf{B}|\mathsf{C})^5$ and $(\mathsf{A}|\mathsf{B})^5$ C B $(\mathsf{B}|\mathsf{C})^5$ to generate sequences in the first and second class, respectively. E_{RGLVQ} is then evaluated for all combinations of λ_{AB} and λ_{AC} (see Figure 4a), as well as λ_{AB} and λ_{BC} (Figure 4b). The respective remaining parameters in λ are constant at their final value from training, with low $\lambda_{\mathsf{AC}} \approx \epsilon$, and high $\lambda_{\mathsf{BC}}, \lambda_{\mathsf{A}-}$, $\lambda_{\mathsf{B}-}$, $\lambda_{\mathsf{C}-} > 0.7$. Since only replacements (B, C) and (C, B) are relevant for class-discrimination, we expect the error function to approach its minimum when λ_{AB} as well as λ_{AC} are low, and λ_{BC} is high. The surfaces in Figure 4 meet these expectations, with a monotonic decrease of error toward the optimum.

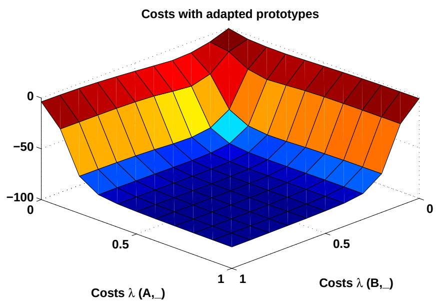


Figure 3: The error E_{RGLVQ} for the *simplified Gap* data, evaluated with all parameter combinations $(\lambda_{\mathsf{A}-}, \lambda_{\mathsf{B}-})$ in steps of 0.1 over the interval $[0, 1]$, while replacement costs are at a low constant $\lambda_{\mathsf{AB}} \approx \epsilon$. As expected, the error surface drops smoothly to a low plateau, when both gap cost parameters are increased.

In a realistic scenario, the number of metric parameters is likely to be much higher. For sequence alignments with a scoring matrix for discrete alphabets (where we assume symmetry and a zero diagonal in λ), the number of free parameters is $(|\Sigma|^2 + |\Sigma|)/2$, i.e. it grows quadratically with the size of the alphabet. Their influence on the RGLVQ error can be rather complex, including intricate dependencies among the parameters themselves. Therefore, we can expect the error function to exhibit several local optima w.r.t. changes of metric parameters in a real data scenario.

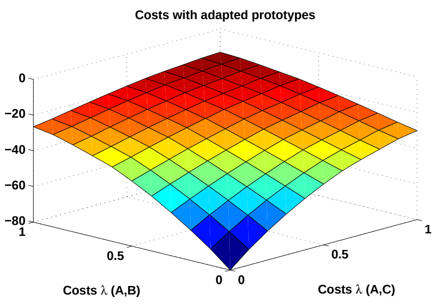
3.5 Influence of crispness on the alignment

In this paragraph, we demonstrate, on a small example, how soft alignment (with its crucial parameter β) compares to classical sequence alignment (which is the limit case of soft alignment, for $\beta \to \infty$). Here, we address only the calculation of the alignment distance, not the learning of parameters. As described in Section 2.2, page 8, the alignment of two sequences can be calculated by DP via a recursive algorithm, see Equation 5. All different possibilities to partially align the sequences and accumulate costs can be assembled in a DP matrix:

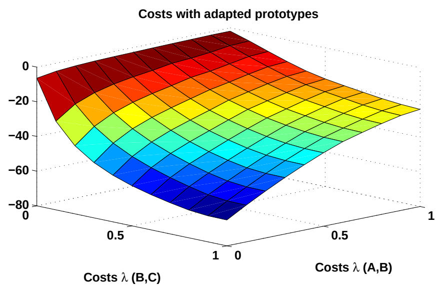
$$\left[\mathbf{M}\right]_{(I,J)} = M_{(I,J)} = d^*\left(\bar{a}(I), \bar{b}(J)\right) \;\; \forall\, 0 \leq I \leq |\bar{a}|,\; 0 \leq J \leq |\bar{b}|$$

The upper left entry $M_{(0,0)} = d^*\left(\bar{a}(0), \bar{b}(0)\right) = 0$ represents the initialization of the recursive calculation, while the bottom right entry contains the final accumulated costs for the full alignment $M_{(|\bar{a}|,|\bar{b}|)} = d^*(\bar{a}, \bar{b})$.

In a crisp alignment (where $\beta \to \infty$), the accumulated cost at a position $M_{(I+1,J+1)}$ is determined by selecting the discrete minimum among the choices $\{A_{\texttt{Rep}}, A_{\texttt{Ins}}, A_{\texttt{Del}}\}$, see Equation 5. This means that every value $M_{(I+1,J+1)}$ depends on only one of the preceding entries $\{M_{(I,J)}, M_{(I+1,J)}, M_{(I,J+1)}\}$. In contrast, using a *softmin* function (with smaller β) means that all choices contribute to the result to a certain extent. Therefore, $M_{(I+1,J+1)}$ depends on



(a) E_{RGLVQ} for λ_{AB} vs. λ_{AC}



(b) E_{RGLVQ} for λ_{BC} vs. λ_{AB}

Figure 4: Surfaces of E_{RGLVQ} for the *simplified Replacement* data, evaluated with parameter combinations λ_{AB} and λ_{AC} (left), as well as λ_{AB} and λ_{BC} (right) in steps of 0.1 over the interval $[0, 1]$. The respective remaining parameters are constant at the final value after training. As expected, the error approaches its minimum for $\lambda_{\mathsf{AB}} = \lambda_{\mathsf{AC}} = 0$ and $\lambda_{\mathsf{BC}} = 1$.

several preceding entries in the DP matrix. Accordingly, sub-optimal alignment choices have an increasing influence on the accumulated cost if β is decreased.

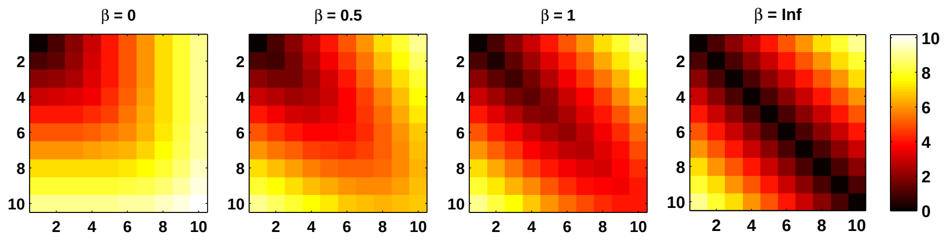
To demonstrate the impact of parameter β, we investigate the characteristics of $\mathbf{M}$ in a simple example. Consider the alignment of a sequence $\bar{a} = (\mathsf{AAAAAAAAA})$ with itself (i.e. $\bar{a} = \bar{b}$), using the simple scoring scheme $\lambda_{\mathsf{AA}} = 0$ and $\lambda_{\mathsf{A}-} = \lambda_{-\mathsf{A}} = 1$. Obviously, in a crisp sequence alignment, the optimal alignment path would match all symbols $\left(\bar{a}_I, \bar{b}_I\right) = (\mathsf{A}, \mathsf{A})$ without making use of deletions or insertions. This corresponds to the diagonal of $\mathbf{M}$, ending at a total cost of zero. Since only insertions or deletions can increase the accumulated cost in this case, the optimal alignment path (along the diagonal, using only matches) remains zero in every step, as can be seen in Figure 5d.

The three images on the left (Figures 5a-5c) show the corresponding DP matrix for values $\beta \in \{0, \frac{1}{2}, 1\}$: when increasing β from zero to one, the optimal path becomes more pronounced and stands out with significantly lower costs. Accordingly, the accumulated cost of the entire alignment drops for higher β.

For $\beta = 5$, the alignment approaches the *de facto* crisp condition. With $\lambda_{\mathsf{AA}} = 0, \lambda_{\mathsf{A}-} = \lambda_{-\mathsf{A}} = 1$, the weight by which a match operation $A_{\mathtt{Rep}}$ for (A, A) contributes to the *softmin* choice is

$$\mathrm{softmin}(A_{\mathtt{Rep}}, A_{\mathtt{Ins}}, A_{\mathtt{Del}}) = \frac{e^{(-5\cdot 0)}}{e^{(-5\cdot 1)} + e^{(-5\cdot 1)} + e^{(-5\cdot 0)}} = \frac{1}{2\, e^{(-5)+1}} \approx 0.99\,.$$

Therefore, insertions and deletions only contribute 1% to the total soft alignment in this case. For other scoring schemes, a higher β might be required to achieve the de facto crisp alignment. It is therefore helpful to evaluate *softmin* values exemplarily, given a scoring λ, to assess the impact of a certain β setting.



(a) $\beta = 0$ (b) $\beta = \frac{1}{2}$ (c) $\beta = 1$ (d) $\beta \to \infty$

Figure 5: The images demonstrate the impact of different choices of ‘crispness’ β on the DP matrix $\mathbf{M}$ for a trivial alignment of $\bar{a} = (\mathsf{AAAAAAAAA})$ with itself, using the simple scoring scheme of $\lambda_{\mathsf{AA}} = 0$ and $\lambda_{\mathsf{A}-} = \lambda_{-\mathsf{A}} = 1$. Each figure shows a color-coded view of values in $\mathbf{M}$ for a setting $\beta \in \{0, \frac{1}{2}, 1, \infty\}$. While the diagonal is the optimal alignment path in all four settings, it becomes more distinguished as a low-cost path when β is high. With lower β values, sub-optimal alignment operations (in this case off-diagonal entries) get a higher contribution to the accumulated cost in the optimal path along the diagonal.

4 Experiments with real-world data

In this section, we investigate the classification performance of our method on two real-world data sets. Additionally, we will take a look at general class-separation in the original and the adapted data dissimilarities, as well as interpretable aspects of the resulting adapted metric parameters.

4.1 Experimental procedure

Our experimental procedure, applied for both data sets, is summarized in the following. As before, the accuracy of an RGLVQ classifier with fixed metric parameters serves as a baseline, and is compared to the accuracy achieved via the proposed adaptation of metric parameters during RGLVQ training. This comparison directly reflects benefits which the classifier gains from metric adaptation. We report the respective average training and test accuracies (along with their standard deviation) obtained in a 5-fold cross-validation with 10 repeats.

To assess the overall class-separation without relying specifically on RGLVQ, we further evaluate the corresponding data dissimilarities before and after the metric is adapted. In the latter case, we use the adapted metric parameters resulting from the last respective cross-validation run of RGLVQ.

First, we report the average test accuracy of a *support vector machine* classifier (SVM), along with its corresponding average number of support vectors (#SV). The quantity of support vectors reflects the complexity of an SVM's classification boundary, where a lower number suggests that class-separation is easier in the given data, while higher values (up to the total number of given training data) indicate overlapping classes. In our practical implementation, we use the Open Source software *LIBSVM*[1] 3.18, and perform a 5-fold cross-validation with 10 repeats, based on the original, as well as the adapted metric. However, in order to apply SVM correctly, we need to obtain a valid kernel matrix from given dissimilarities d_{ij} in the matrix $\mathbf{D}$. Therefore, we first use Torgerson's *double centering*, see [35, p.258] to get similarities, as:

$$[\mathbf{S}]_{(i,j)} = s_{ij} = -\frac{1}{2} \cdot \left(d_{ij}^2 - \vec{c}_j^{\,2} - \vec{r}_i^{\,2} + o^2 \right)$$

where $\vec{c}_j, \vec{r}_i, o$ are the mean of the j-th column, of the i-th row, and of all values in $\mathbf{D}$, respectively. Thereafter, a kernel matrix $\mathbf{K}$ is created from $\mathbf{S}$ by correcting possible non-metric aspects in the given similarities, via 'flipping' negative Eigenvalues of $\mathbf{S}$, as described in [13].

Further, the accuracy of a simple *k-nearest-neighbor* classifier (k-NN) is evaluated, using $k = 5$ neighbors. Obviously, k-NN and SVM are expected to achieve a higher accuracy in general, since the model complexity in the sparse RGLVQ classifier is highly restricted by using only one prototype per class. For these evaluation models, we will therefore focus on differences between fixed and adapted dissimilarities, instead of comparing the sparse RGLVQ model with SVM or k-NN classification in terms of accuracy.

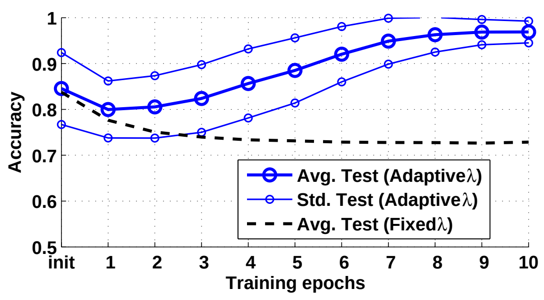
As an additional quantitative indicator, independent of any particular classification model, we measure the ratio of mean intra-class distances to mean inter-class distances, in the following referred to as *separation ratio*. Here, smaller values indicate a clearer class-separation in general, which is an expected result from the metric adaptation procedure.

4.2 Copenhagen Chromosomes

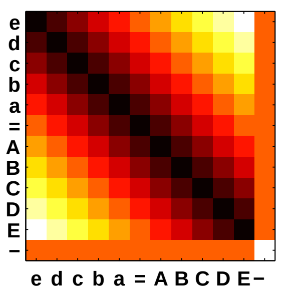
The sequences in this set represent band patterns from the Copenhagen Chromosomes database [23]. Every sequence encodes the differential succession of density levels observed in gray-scale images of a human chromosome. Since 7 levels of density are distinguished, a 13-letter alphabet $\Sigma = \{\mathsf{f}, \ldots, \mathsf{a}, =, \mathsf{A}, \ldots, \mathsf{F}\}$ represents a difference coding of successive positions, where upper and lower case letters mark positive and negative changes respectively, and "=" means no change[2]. Table 6 on page 39 in the Appendix section 7.2 lists all symbols with their associated difference levels, and the number of occurrences in the considered data set. From the database we use the *"CopChromTwo"* subset for binary classification, containing classes 4 and 5 with 200 sequences each ($N = 400$). In the literature, these two classes have been reported to yield a lower recognition rate than the others, see [10]. The authors in [10] used an organized ensemble of multilayer perceptrons to classify all 22 chromosomes in the Copenhagen database, and list the classification accuracies for individual classes. For the chromosomes 4 and 5, they report 91% and 89% accuracy on the test set, respectively, whereas the overall average is 95.86%. However, since every class is

[1] `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`

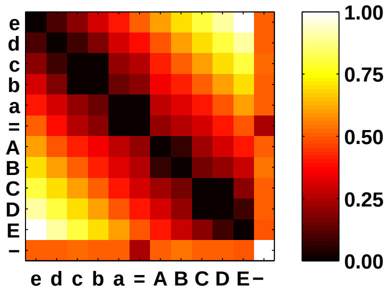
[2] For details, see `http://algoval.essex.ac.uk/data/sequence/copchrom/`



(a) Classification accuracies



(b) Initial λ



(c) Adapted λ

Figure 6: Metric parameters and results for the *Chromosomes* data set. The bottom left shows the initial alignment scoring pattern in the parameters λ according to a weighting scheme from [15]. The bottom right shows an adaptation of λ where costs for an insertion or deletion of the most frequent symbol "=" are strongly reduced, and replacements of neighboring symbols are decreased slightly. The graphs in the top figure show that the adaptation improves classification accuracy over 10 learning epochs in a repeated 5-fold cross-validation.

addressed by a one-versus-all classifier, these values are not directly comparable to the binary classification task on which we will focus in the following. To handle the full 22-class database, a local scoring matrix λ^j for every prototype $\vec{a}^j$ would be necessary, which is the subject of ongoing work, see Section 6.

For the Copenhagen Chromosomes data, assumptions about a meaningful parameterization of the metric are available in [15]. The authors propose a weighting scheme for the edit distance, where replacement costs are the absolute difference of corresponding density changes: for example, $\lambda_{\mathsf{a}\,\mathsf{e}} = |-1-(-5)| = 4$, and $\lambda_{\mathsf{f}\,\mathsf{F}} = |-6-6| = 12$. The introduction of any gaps requires half the maximum of replacement costs, i.e. $\lambda_{k-} = \lambda_{-m} = 6$ for all $k, m \in \Sigma$. See Figure 6b for the full cost matrix[3]. Therefore, we compared two different options to initialize metric parameters λ in our experiment: (i) using the cost pattern from [15], and (ii) using the simple initialization with *equal costs*. For both cases,

[3]Symbols f, F did not occur in the *CopChromTwo* subset and were thus not considered.

Method	***Init.***	**Train (Std)**	**Test (Std)**	**SVM (#SV)**	**5-NN**	**Sep.**
Fixed λ	*prior*	74% (2%)	73% (6%)	97% (174)	95%	0.93
Adapted λ	*prior*	97% (2%)	97% (2%)	98% (129)	98%	0.91
Fixed λ	*equal*	89% (3%)	89% (6%)	96% (210)	97%	0.99
Adapted λ	*equal*	95% (1%)	95% (3%)	97% (139)	97%	0.93

Table 1: Performance of RGLVQ for the *Chromosomes* data set, comparing fixed vs. adaptive metric parameters λ, measured by the average **Train**ing and **Test** accuracies and their standard deviation (**Std**) in a 5-fold cross-validation with 10 repeats. From the respective last run, dissimilarities induced by λ are evaluated by **5-NN** classification accuracy, the separation ratio (**Sep.**, where smaller is better), as well as the average test accuracy of **SVM** (and number of support vectors **#SV**) from a repeated 5-fold cross-validation. All results are reported for two initialization methods for λ: *equal* costs, and a weighting scheme from [15] using *prior* knowledge.

we compare the classification performance of RGLVQ with and without metric adaptation, as before. In the latter case, RGLVQ training uses fixed dissimilarities based on the respective initial costs λ. The choice of meta-parameters was optimized w.r.t. the data in a 5-fold cross-validation with 10 repeats, setting crispness $\beta = 20$, and learning rate $\eta = 0.45/N$ to adapt λ. In order to minimize the computational effort, we chose $\mathtt{k} = 1$, which prove to be sufficient.

First, we consider the initialization of λ according to the weighting scheme from [15]. The results are displayed in Figure 6, and Table 1 (the two top rows). With fixed metric parameters, the final classification accuracies are rather low with 73% average test accuracy, see the baseline in Figure 6a. This is expectable, since the number of prototypes for RGLVQ was deliberately chosen to be only one per class, which implies minimal complexity of the classification boundary. Using more prototypes yields some improvements: with 3, 5, and 7 prototypes per class, 79%, 79%, and 81% average test accuracy is achieved, respectively.

However, metric adaptation with only one prototype enables a nearly perfect average accuracy of 97% for training and test sets. This demonstrates how a problem-adapted metric can alleviate the given classification task, even without a complex classifier model. (We observed no considerable benefit, when more prototypes are used.) Interestingly, this major improvement was achieved by subtle changes in λ from the initial scoring (see Figure 6b) to the final state after training (see Figure 6c, taken from the last cross-validation run). The adaptation mainly changes the replacement costs for neighboring scales of difference: many values on the first off-diagonals become nearly zero, signifying that these symbols are interchangeable within classes. At the same time, the gap costs for the symbol “=” become lower, which can be attributed to the fact that it is the most frequent symbol in the set (see Table 6 in the Appendix 7.2).

Comparing the class-separation in the fixed vs. the adapted dissimilarities, we observe that the separation ratio drops and the 5-NN accuracy improves, as reported in Table 1. Also, the average number of support vectors used in the trained SVM classifiers decreases drastically, which indicates a less complex classification boundary. This underlines our hypothesis, that metric learning can greatly facilitate class-discrimination, especially when the parameterization of the underlying dissimilarity measure is complex.

In the next step, we consider the case of initializing λ with *equal costs* (meta-parameters set to $\beta = 7$, $\eta = 0.45/N$, $\mathtt{k} = 1$). Surprisingly, this very simple setting of λ yields higher accuracies than the cost pattern proposed in [15], see the bottom two rows in Table 1. Without any adaptation of the alignment costs, RGLVQ achieves 89% average test accuracy for one prototype per class. This can be improved by increasing the model complexity: 93%, 95%, and 95% average test accuracy are achieved with 3, 5, and 7 prototypes per class, respectively. Learning the metric parameters provides the same level of improvement to 95%, however, with only one prototype, since the underlying data representation is tuned according to the classification task. Thereby, the resulting model is very sparse, and it offers the possibility to inspect and interpret the adapted metric parameterization. In this case, increasing the number of prototypes also results in a slightly better classification performance: for example, 5 prototypes per class yield 96% average test accuracy. Although the 5-NN accuracy is not increased, the separation ratio is improved by the metric adaptation, and the average number of support vectors for SVM drops, again supporting our claim.

4.3 Intelligent tutoring systems for Java programming

In the context of educational technology, *intelligent tutoring systems* (ITSs) have greatly advanced in recent years. The goal of these systems is to provide intelligent, one-on-one, computer-based support to students, as they learn to solve problems in a type of instruction that is often not available because of scarce (human) resources [36, 7, 37]. One approach to facilitate ITSs is based on the automatic clustering and classification of student solutions, see [11, 24]. Therefore, a crucial ingredient is a reliable (dis)similarity measure for pairs of solutions [24]. While a solution could be represented in many forms, we will focus here on a representation as a (multi-dimensional) sequence. In this experiment, we will consider the dissimilarity between Java programs, as an example pointing towards the idea of adaptive metrics in an ITS for Java programming, as described in [24]. Given the complexity of syntactic structures, we demonstrate how the parameterization of such a dissimilarity measure can be adapted to facilitate a classification task.

To properly model Java programs as sequential data, we no longer consider discrete symbolic sequences as before, but instead refer to sequences with more complex, multi-modal entries. Each entry, in the following called a *node*, holds a collection of *properties*, where the number of properties K is fixed a priori for the given data set. For every single property, either a finite discrete symbolic alphabet, or a numeric domain is defined a priori. Given the property number $\kappa \in \{1, \dots, K\}$, we refer to its designated alphabet or domain as Σ_κ, i.e. the set of all possible values for that property. A multi-modal sequence is denoted by: $\widetilde{a} = (a_1, \dots, a_I, \dots, a_{|\widetilde{a}|})$ where a_I is a node, and a_I^κ refers to the (symbolic or numeric) content of property number κ in this node.

In the case of Java, the nodes represent syntactic building blocks of a Java program, which were extracted from the abstract syntax tree via the official Oracle Java Compiler API. Properties are, for example, the node's position in the source code file (`codePosition`, an array of integers indicating the starting and ending line and column), the `type` of this node (e.g. a method declaration, a variable declaration, an assignment, etc.), or more specific properties like the `name` of a variable, method or class that is declared.

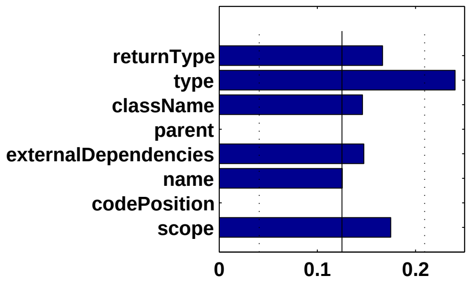
To define a parameterized alignment measure for such multi-modal sequences, we will use a generalization of the two alignment scenarios described in Section 2.2, on page 6. We adopt the *relevance weighting* from vectorial sequence alignment, but loose the restriction to numerical vectors containing real-valued entries, in favor of a more general notion of nodes containing multiple properties. Therefore, we replace the 'inner' dissimilarity measures for each vector dimension with a measure for each property: assume a symmetric dissimilarity $d^\kappa : \Sigma_\kappa \times \Sigma_\kappa \to [0,1]$ for each property κ. This measure can be parameterized by some λ (denoted as d^κ_λ), but if this is not specified, we simply assume $d^\kappa(s,t) = 0 \Leftrightarrow s = t$, and $d^\kappa(s,t) = 1 \Leftrightarrow s \neq t$ for all symbols or values $s, t \in \Sigma_\kappa \cup \{-\}$ in the corresponding symbolic alphabet or numeric domain. This means that costs for replacements, insertions, and deletions can be specified, which corresponds to the case of a *scoring matrix* in Section 2.2, but now individually for every property κ. Then, we redefine the 'outer' dissimilarity between single nodes as:

$$d_{\vec{g}}(a_I, b_J) = \sum_{\kappa=1}^{K} g_\kappa \cdot d^\kappa_\lambda(a^\kappa_I, b^\kappa_J) \ .$$

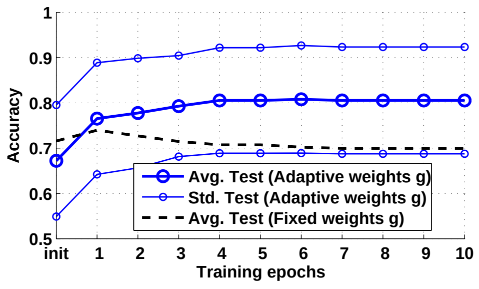
The vector $\vec{g} = (g_1, \dots, g_K)$ contains the *relevance weights* g_κ for each property κ, which lie in the interval $[0,1]$ and are normalized to sum up to one. Thus, for $g_\kappa = 1$, no other property is considered, but κ. If $g_\kappa = 0$, the property κ does not contribute to the alignment at all.

In this experiment, our goal is to learn the metric parameters of both, the outer and inner dissimilarity, i.e. g_κ and λ, respectively. Therefore, we use a two-stage consecutive process: first, we fix the 'inner' parameterization λ for all properties κ, and adapt the 'outer' parameters g_κ. Thereafter, the resulting weights g_κ remain fixed, and the 'inner' parameters λ are adapted for the property with the highest relevance, given by $\arg\max_{\kappa \in \{1,\dots,K\}}(g_\kappa)$. For the adaptation in each stage, we can directly transfer the respective learning scheme for *relevance weights* and *scoring matrices*, as described in Section 2.3.

The *Sorting* data set consists of a collection of Java programs, which are freely available in online code-sharing platforms. The programs implement two different algorithms to sort sets of integer numbers: we collected $N = 78$ programs in total, of which 44 implement the *BubbleSort* algorithm, and 34 realize *InsertionSort*. From each program, the sequence of syntactic nodes was extracted by a parser, where 8 properties are defined for every node. A list of the properties is given in Figure 7a.



(a) Adapted weights g_κ



(b) Classification accuracies

Figure 7: Results for the *Sorting* data set, in which the (semantically sound) adaptation of weights g_κ for properties κ (left) yields an improvement of 11% in average test accuracy (right) in a 5-fold cross-validation with 5 repeats.

Initially, every property $\kappa \in \{1, \ldots, 8\}$ is weighted equally, with $g_\kappa = 1/8$. First, we aim to learn a configuration of weights g_κ to facilitate class discrimination. The RGLVQ classifier was trained, with and without metric adaptation, for 10 epochs with one prototype per class, in a 5-fold cross-validation with 5 repeats. The meta-parameters for metric learning were set to: $\eta = 0.002/N$, $\mathtt{k} = 1$, and $\beta = 200$ (i.e. using a de facto crisp alignment).

The results in Figure 7 and Table 2 (the two top rows) show that RGLVQ with metric adaptation improves the average test accuracy by 11%, compared to the default metric with all equal weights g_κ. Under the initial metric parameterization, even a higher number of prototypes in RGLVQ does not yield comparable performance: for 7 prototypes per class, the average accuracy is 72% on the test set. The 5-NN accuracy and separation ratio are also improved by the metric adaptation, and the average number of support vectors for SVM decreases, indicating a simpler classification boundary.

The resulting weights are reported in the bar graph in Figure 7a and can be interpreted as semantically sound for the classification task: `type` is weighted as the most relevant property, while `parent` and `codePosition` are deactivated entirely. This is justified, since `type` holds the most important semantic information by specifying one out of 29 possible categorial values encoding the basic functionality of the respective syntax part. (The alphabet for this property refers to token types in the abstract syntax tree of a program, a full list of symbols and corresponding Java code examples can be found in Section 7.3 in the Appendix.) In contrast, (i) `parent` and (ii) `codePosition` clearly introduce noise w.r.t. classification, since they encode (i) the index of the previous node in the syntax tree, and (ii) the position in the raw source code file, both of which can drastically change from minor alterations in the program and are thus not discriminative. The intermediate weights for the remaining properties like `className` and `returnType` are also justified, since they convey valuable information about the semantics, like the class of (return) variables, such as

Method	***Init.***	**Train (Std)**	**Test (Std)**	**SVM (#SV)**	**5-NN**	**Sep.**
Fixed g	λ *equal*	75% (3%)	70% (14%)	75% (58)	82%	0.93
Adapted g	λ *equal*	81% (3%)	81% (12%)	87% (49)	87%	0.81
Adapted λ	g *prior*	83% (2%)	82% (9%)	87% (49)	87%	0.81

Table 2: Performance of RGLVQ for the *Sorting* data set, comparing fixed vs. adaptive metric parameters (λ and g), measured by the average **Train**ing and **Test** accuracies and their standard deviation (**Std**) in a 5-fold cross-validation with 5 repeats. From the respective last run, dissimilarities (induced by λ or g) are evaluated with the **5-NN** classification accuracy, the separation ratio (**Sep.**, where smaller is better), as well as the average test accuracy of **SVM** (and number of support vectors **#SV**) from a repeated 5-fold cross-validation. The results refer to a two-stage learning process: the adaptation of property weights g (in the two top rows) where λ is set to *equal costs*, and the subsequent adaptation of costs λ (in the bottom row) where g is fixed to the result of the previous learning procedure.

Integer or *String*. However, since they are empty for many nodes, the lower relevance, as compared to `type`, can be explained. Interestingly, the property `name` refers to textual definitions for variable, class, and function names, freely chosen by the programmer. Of course, such names are not guaranteed to be meaningful for class-discrimination, and could potentially introduce noise in the data. However, since our set contains programs from an educational context, these names are likely to be defined in an explanatory fashion, which justifies the intermediate weighting.

To facilitate the classification further, we assume the trained weights g_κ as fixed, and subsequently adapt the metric again, now by learning the parameters d^κ_λ for the most relevant property `type`. Thus, κ refers to the index of property `type` in the following, and we focus on the alphabet Σ_κ of 29 symbols, as listed in the Appendix Section 7.3. We therefore return to the learning scheme for alignment scoring parameters d^κ_λ, in the following denoted as λ_{km} for symbols $k, m \in \Sigma_\kappa \cup \{-\}$. The subsequent adaptation improves the results again, by 1% for the average test accuracy. While this is only a moderate quantitative enhancement, it can be seen as a refinement of the metric, since the standard deviation of training and test accuracies was reduced, suggesting a higher robustness, see Table 2 (the bottom row).

5 Reducing computational demand

Besides an approximation of prototypes by its `k` most prominent exemplars, a variety of further approximation techniques can be integrated to enhance the computational performance. While these methods are not mandatory for standard data sets, they become necessary as soon as larger data sets, e.g. $N > 500$ are addressed: The complexity of RGLVQ scales quadratically with the number of data points, so that it becomes infeasible for large data sets. We will discuss two options for speedup, which address two different bottlenecks of the computational load:

- Section 5.1 addresses the computation of derivatives with respect to metric parameters: soft alignment requires the consideration of the full alignment path for every metric parameter, while crisp alignment reduces to contributions of the optimal path only. We will consider an approximation scheme, which disregards small contributions of the alignment paths, enabling a computation of the derivatives in linear time with respect to sequence length in the best case, as compared to squared complexity. This approximation is particularly relevant for long input sequences.
- Section 5.2 addresses the computation of dissimilarities by means of a reference to the full dissimilarity matrix **D**: the full dissimilarity matrix scales quadratically with the number of data. We can approximate **D** by a low rank matrix via the popular Nyström approximation. Since **D** is used in matrix vector operations only, a low rank approximation speeds up these operations to linear instead of quadratic time with respect to the number of sequences. This approximation is particularly relevant if a large number of training sequences is considered.

5.1 Approximated alignment derivative

As before, we exemplarily consider the setting of a discrete alphabet and the adaptation of the scoring matrix, parameterized by λ_{km}. The overall runtime for online learning of metric parameters is strongly affected by the computational effort to calculate a single alignment derivative: given a set of sample sequences $\mathcal{S}$ and the set of exemplars $\mathcal{E}_j$ for one prototype $\vec{\alpha}^j$ in one learning epoch, the derivative $\partial d^*(\bar{a}^i, \bar{w}^l)/\partial d_\lambda$ is calculated for all pairs of samples $\bar{a}^i \in \mathcal{S}$ and exemplars $\bar{w}^l \in \mathcal{E}_j$, i.e. it is done $|\mathcal{S}| \times |\mathcal{E}_j| = N \cdot \mathtt{k}$ times for the update w.r.t. one prototype in one epoch alone.

Therefore, we empirically evaluate the speedup gained from dropping alignment paths with a small contribution, as follows: In the limit $\beta \to \infty$, contributions restrict to the best alignment path, hence derivatives $\partial d^*(\bar{a}, \bar{b})/\partial \lambda_{km}$ for all λ_{km} can be computed in time $\mathcal{O}(|\bar{a}| + |\bar{b}|)$ based on the DP matrix. In general, derivatives are weighted sums corresponding to alignments of the symbols k and m at some position (I, J) of the matrix. Weighting takes into account all possible paths which include this pair according to the path eligibility measured by $\mathrm{softmin}'(A_i)$ for actions $A_i \in \{A_{\mathtt{Rep}}, A_{\mathtt{Ins}}, A_{\mathtt{Del}}\}$ on the path. The worst case complexity is $\mathcal{O}(|\bar{a}| \cdot |\bar{b}| \cdot |\Sigma|^2)$, using backtracing in the alignment matrix. We propose an approximation based on the observation that a small $\mathrm{softmin}'(A_i)$ leads to a small weight of paths including A_i. Hence, we store the 3 terms $A_{\mathtt{Rep}}$, $A_{\mathtt{Del}}$, $A_{\mathtt{Ins}}$ together with the distances $\mathrm{softmin}(A_{\mathtt{Rep}}, A_{\mathtt{Del}}, A_{\mathtt{Ins}})$ in the data structure of the DP matrix, and we cut all values $\mathrm{softmin}'(A_i) < \theta$ for a fixed threshold $\theta \geq 0$. Backtracing depends on the nonzero values only, so that a speedup to linear complexity is possible in the best case.

The threshold θ therefore determines that values $\text{softmin}'(A_i) < \theta$ are ignored in the backtracing of alignment paths. Since the impact of θ depends on the alphabet size and sequence length, it should be tuned according to good classification results for the given data set. Typical values lie in the interval $\theta \in [0.01, 0.2]$. As a simple test scenario, we created several sets of random sequences, each consisting of 10 sequences $\bar{a}^i \in \Sigma^L$ with $\Sigma = \{\mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{D}\}$, with four different choices of length L. For various thresholds θ, we tracked the runtime of calculating alignment derivatives for all 100 sequence pairs on a standard laptop computer with an *Intel Core i7* processor (4 cores at 2.9 GHz, and calculations done in parallel). The results in Table 3 clearly show how increasing θ drastically reduces the computational effort, especially for longer sequences.

To demonstrate that reasonable approximations do not impede classifier performance in practice, we performed single training runs on the *Chromosomes* data from Section 4.2, using a random split of 80% training and 20% test data, with various settings of θ. The training was performed on a server computer with two *Intel Xeon X5690* processors (each with 6 cores at 3.47 GHz), using the same meta-parameter settings as in the original experiment. Table 4 lists the achieved test accuracies and runtimes, in comparison with the original result from Section 4.2. The values show that a slight approximation with $\theta = 0.1$ already reduces the average runtime for one learning epoch by 21%, without decreasing the classification accuracy. More crude degrees of approximation yield further, but marginal speedup, which stagnates for settings $\theta \geq 0.2$, while the accuracy drops continuously to 90% for the extreme setting of $\theta = 0.65$. In general, choices $\theta > 1/3$ carry the risk of loosing potentially valuable learning stimuli, since all three values $A_{\texttt{Rep}}, A_{\texttt{Ins}}, A_{\texttt{Del}}$ could be lower than θ for certain steps in the soft alignment, and therefore this entire alignment path would be ignored.

Table 3: Runtimes (in seconds) to calculate the alignment derivatives for all pairs of random strings $\bar{a}^i \in \Sigma^L$, $i \in \{1 \dots 10\}$, using different thresholds θ and $\beta = 10$. (Note, that this is not a classification task to discriminate labeled data, but a plain runtime test using all pairs of sequences. Therefore, no classification accuracies are reported.)

Sequence length L	100	150	200	250
Runtime for $\theta = 0$	7.2	24.6	87.6	426
Runtime for $\theta = 0.15$	4.2	13.2	31.8	98.4
Runtime for $\theta = 0.2$	1.8	6.6	13.8	27.0
Runtime for $\theta = 0.25$	1.2	3.6	7.2	13.2

5.2 Nyström approximation

In our algorithm, metric parameters λ are adapted in every epoch. They induce a different dissimilarity measure, thus $\mathbf{D}$ needs to be re-calculated according to this new parameterization, see Line 14 in Algorithm 1 on page 13. To avoid

Table 4: RGLVQ with metric adaptation, evaluated in single training runs on the *Chromosomes* data from Section 4.2, using an approximation technique to calculate the alignment derivative. The degree of approximation is controlled via the threshold θ by which marginally contributing alignment paths are neglected in the derivative calculation, i.e. with higher θ the approximation becomes more crude. For each setting of θ, the final test accuracy is reported, along with the average runtime for one learning epoch (in seconds). For small settings of θ, the approximation yields speedup without sacrificing accuracy.

Degree of approximation	no approx.	$\theta = 0.1$	$\theta = 0.2$	$\theta = 0.35$	$\theta = 0.65$
Epoch runtime in s (%)	303 (100%)	240 (79%)	225 (74%)	228 (75%)	228 (75%)
Test accuracy	avg. 95%	97%	95%	94%	90%

the repeated alignment between *all* sequence pairs, we refer to the *Nyström* technique to approximate the full matrix as $\mathbf{D}^{\nu} \approx \mathbf{D}$.

The Nyström approximation, as introduced by Williams and Seeger in [38], allows for an efficient approximation of a kernel matrix by a low-rank matrix. This approximation can be directly transferred to dissimilarity data, see [28]. The basic principle is to pick a number of V representative landmarks, and consider the rectangular sub-matrix $\mathbf{D}_{V,N}$ of dissimilarities between landmarks and all data points (in our case sequences). This matrix is of linear size, assuming that V is fixed. The full matrix can be approximated in an optimal way, in the form $\mathbf{D} \approx \mathbf{D}^{\nu} = \mathbf{D}_{V,N}^{\top}\mathbf{D}_{V,V}^{-1}\mathbf{D}_{V,N}$ where $\mathbf{D}_{V,V}$ is the square sub-matrix of $\mathbf{D}$, and $\mathbf{D}_{V,V}^{-1}$ refers to its pseudo-inverse. While calculating the full pairwise dissimilarity matrix $\mathbf{D}$ takes $\mathcal{O}(N^2)$ time, the complexity to produce its approximated counterpart $\mathbf{D}^{\nu}$ is dominated by: the calculation of $\mathbf{D}_{V,N}$ in $\mathcal{O}(V \cdot N)$ steps, and the pseudo-inverse $\mathbf{D}_{V,V}^{-1}$, which is $\mathcal{O}(V^3)$. This results in an overall complexity of $\mathcal{O}(V^2 \cdot N)$, which becomes profitable when N is increasing while V is assumed to be constant. The resulting approximation is exact, if V corresponds to the rank of matrix $\mathbf{D}$.

To demonstrate the suitability of Nyström approximation for our method, we replaced the corresponding dissimilarity calculations in our algorithm, and consider single training runs on the *Chromosomes* data from Section 4.2. Using this data set, with $N = 400$ sequences, we can showcase the validity of the Nyström technique in principle. However, since the approximation trades the $\mathcal{O}(N^2)$ complexity for $\mathcal{O}(V^2 \cdot N)$ based on the chosen number of landmarks V, the benefits become more apparent in large-scale scenarios, where the number of sequences is very high, e.g. $N > 1000$, and a choice $V \ll N$ is justified. For the selection of appropriate landmark sequences, we use a random sample of the data in the corresponding epoch. This simple strategy relies on no assumptions about the data structure, and has shown to work well in our experiment. However, more informed selection plans can be found in the literature, see [39] for example.

The results in Table 5 show that the average runtime for one training epoch is decreased by 17%, when 70% of the data are chosen as landmarks, i.e.

$V = \lfloor 0.7 \cdot N \rfloor$. However, this causes a slight drop in test accuracy, from 95% to 89%. Interestingly, we observe that the accuracy does not decrease monotonically with the crudeness of the approximation: using 80% of all sequences as landmarks yields a better accuracy than using 90%. In the context of Nyström approximation, this effect has been observed in the literature for the Copenhagen Chromosomes data, see [41]. A plausible explanation would be that noise in the data representation is suppressed at a certain level of approximation. Our experiment shows, that the Nyström approximation is a valid technique to decrease the computational effort of the proposed algorithm. A more elaborate evaluation is the subject of ongoing work, since a realistic application scenario would involve larger data sets.

Table 5: RGLVQ with metric adaptation, evaluated in single training runs on the *Chromosomes* data, using the Nyström approximation to (repeatedly) calculate the dissimilarity matrix $\mathbf{D}^{\nu} \approx \mathbf{D}$. The degree of approximation is controlled via the number of landmarks V, in relation to the total number of sequences N, i.e. with lower V the approximation becomes more crude. For each setting of V, the final test accuracy is reported, along with the average runtime for one learning epoch (in seconds).

Degree of approximation	no approx.	$V = \lfloor 0.9 \cdot N \rfloor$	$V = \lfloor 0.8 \cdot N \rfloor$	$V = \lfloor 0.7 \cdot N \rfloor$
Epoch runtime in s (%)	303 (100%)	274 (91%)	261 (87%)	251 (83%)
Test accuracy	95%	91%	93%	89%

6 Discussion and future work

In this article, we have extended relational LVQ by the powerful concept of metric learning for sequential data, enabling an automatic adaptation of crucial metric parameters to the given classification task. Unlike alternative approaches for sequence metric learning [2], we have combined the metric adaptation strategy with the classifier itself. This enables a very straightforward adaptation by means of cost function optimization, and an easy evaluation of the performance by a reference to the resulting classification error. Besides an improved classification accuracy and generalization ability of the models, this method can enhance the interpretability of the resulting classifiers, by pointing out the relevance of single edit operations. We have demonstrated this effect on several examples: an inspection of metric parameters enables us to interpret the relevance of certain data components for the overall learning task. Thus, the proposed metric adaptation transfers the principle of relevance learning to the structural domain.

Although this article focuses on sequence alignment, the approach opens the way towards efficient metric adaptation schemes for distance-based methods in other discrete structure spaces, such as trees or graph structures. A similar metric learning becomes possible, provided the metric is differentiable with respect to its significant parameters. Note that, unlike the approach [16], we do not assume differentiability of the dissimilarity measure with respect to the data structures itself, but differentiability with respect to the adaptive metric parameters only. Hence, discrete structure spaces are covered by our proposed technique.

So far, our method relies on a global metric with one set of parameters λ. This can be problematic, if relevant structural constituents change, depending on their position in the data space, as is common e.g. for classification problems with more than two classes. In this context, it could be beneficial to use class-specific parameter sets λ^j associated with every prototype $\vec{w}^j$. For vectorial LVQ, this 'local' metric learning has been proposed in [30, 13], and it could be transferred to the relational setting. However, its computational efficiency becomes problematic, due to the computational demand for calculating individual dissimilarity matrices for every parameter set λ^j. In this context, it might be worthwhile to investigate efficient low-rank metric approximations only.

Another important question arises in the context of the interpretability of metric parameters: it is not clear whether parameter configurations are unique, and whether invariances exist, caused e.g. by structural invariances. In the latter case, metric parameters do not necessarily relate to the true relevance of these structural constituents, rather random effects can occur. This property has recently been observed in the vectorial setting, when dealing with very high data dimensionalities. In this case, high relevance can be related to correlations of data dimensions in some cases, falsely suggesting a high feature relevance if interpreted directly [33]. Therefore, before relying on the interpretation of metric parameters, a normalization of the representation with respect to structural invariances is mandatory. For structural data, similar effects can be expected. Therefore, it is the subject of ongoing work to exactly identify these invariances for a given metric, and to devise unique representatives of the resulting equivalence classes for valid interpretation.

References

References

[1] A. Backhaus and U. Seiffert. Classification in high-dimensional spectral data: Accuracy vs. interpretability vs. model size. *Neurocomputing*, 131:15–22, 2014.

[2] A. Bellet, A. Habrard, and M. Sebban. A survey on metric learning for feature vectors and structured data. *CoRR*, abs/1306.6709, 2013.

[3] M. Bernard, L. Boyer, A. Habrard, and M. Sebban. Learning probabilistic models of tree edit distance. *Pattern Recognition*, 41(8):2611–2629, 2008.

[4] M. Biehl, K. Bunte, and P. Schneider. Analysis of flow cytometry data by matrix relevance learning vector quantization. *Plos One*, 8(3), 2013.

[5] L. Boyer, Y. Esposito, A. Habrard, J. Oncina, and M. Sebban. Sedil: Software for edit distance learning. *Lect. Notes Comput. Sci.*, 5212 LNAI(2):672–677, 2008.

[6] K. Bunte, P. Schneider, B. Hammer, F.-M. Schleif, T. Villmann, and M. Biehl. Limited rank matrix learning, discriminative dimension reduction and visualization. *Neural Networks*, 26:159–173, 2012.

[7] A. T. Corbett, K. R. Koedinger, and J. R. Anderson. Intelligent tutoring systems. In M. G. Helander, T. K. Landauer, and P. Prabhu, editors, *Handbook of Human-Computer Interaction*, pages 859–874. The Netherlands: Elsevier Science, 1997.

[8] T. M. Cover and P. E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[9] A. Denecke, H. Wersing, J. J. Steil, and E. Körner. Online figure-ground segmentation with adaptive metrics in generalized LVQ. *Neurocomputing*, 72(7-9):1470–1482, 2009.

[10] S. Gagula-Palalic and M. Can. An organized committee of artificial neural networks in the classification of human chromosomes. *International Journal of Computer Applications*, 80(8):38–41, 2013.

[11] S. Gross, X. Zhu, B. Hammer, and N. Pinkwart. Cluster based feedback provision strategies in intelligent tutoring systems. In *Proceedings of the 11th international conference on Intelligent Tutoring Systems*, ITS'12, pages 699–700, Berlin, Heidelberg, 2012. Springer-Verlag.

[12] A. Habrard, J. Iñesta, D. Rizo, and M. Sebban. Melody recognition with learned edit distances. *Lect. Notes Comput. Sci.*, 5342 LNCS:86–96, 2008.

[13] B. Hammer, D. Hofmann, F.-M. Schleif, and X. Zhu. Learning vector quantization for (dis-)similarities. *Neurocomputing*, 131:43–51, 2014.

[14] S. Henikoff and J. G. Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences*, 89(22):10915–10919, 1992.

[15] A. Juan and E. Vidal. On the use of normalized edit distances and an efficient k-nn search technique (k-aesa) for fast and accurate string classification. In *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, volume 2, pages 676–679 vol.2, 2000.

[16] M. Kästner, D. Nebel, M. Riedel, M. Biehl, and T. Villmann. Differentiable kernels in generalized matrix learning vector quantization. In *ICMLA, p. 132-137*, 2012.

[17] T. C. Kietzmann, S. Lange, and M. A. Riedmiller. Computational object recognition: a biologically motivated approach. *Biological Cybernetics*, 100(1):59–79, 2009.

[18] S. Kirstein, A. Denecke, S. Hasler, H. Wersing, H.-M. Gross, and E. Körner. A vision architecture for unconstrained and incremental learning of multiple categories. *Memetic Computing*, 1(4):291–304, 2009.

[19] S. Kirstein, H. Wersing, H.-M. Gross, and E. Körner. A life-long learning vector quantization approach for interactive learning of multiple categories. *Neural Networks*, 28:90–105, 2012.

[20] T. Kohonen. *Self-organizing maps.* Springer, 1995.

[21] B. Kulis. Metric learning: A survey. *Foundations and Trends in Machine Learning*, 5(4):287–364, 2013.

[22] S. P. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–136, 1982.

[23] C. Lundsteen, J. Phillip, and E. Granum. Quantitative analysis of 6985 digitized trypsin G-banded human metaphase chromosomes. *Clin. Genet.*, 18:355–370, 1980.

[24] B. Mokbel, S. Gross, B. Paassen, N. Pinkwart, and B. Hammer. Domain-independent proximity measures in intelligent tutoring systems. *Proceedings of the 6th International Conference on Educational Data Mining (EDM)*, pages 334–335, 2013.

[25] B. Mokbel, B. Paassen, and B. Hammer. Adaptive distance measures for sequential data. In M. Verleysen, editor, *ESANN*, pages 265–270. i6doc.com, 2014.

[26] B. Mokbel, B. Paassen, F.-M. Schleif, and B. Hammer. Metric learning for sequences in relational LVQ. *Neurocomputing*, (accepted/in press), 2015.

[27] E. Pekalska and B. Duin. *The Dissimilarity Representation for Pattern Recognition. Foundations and Applications.* World Scientific, 2005.

[28] F.-M. Schleif and A. Gisbrecht. Data analysis of (non-)metric proximities at linear costs. In E. R. Hancock and M. Pelillo, editors, *SIMBAD*, volume 7953 of *Lecture Notes in Computer Science*, pages 59–74. Springer, 2013.

[29] F.-M. Schleif, B. Hammer, M. Kostrzewa, and T. Villmann. Exploration of mass-spectrometric data in clinical proteomics using learning vector quantization methods. *Briefings in Bioinformatics*, 9(2):129–143, 2008.

[30] P. Schneider, M. Biehl, and B. Hammer. Adaptive relevance matrices in learning vector quantization. *Neural Computation*, 21(12):3532–3561, 2009.

[31] S. Seo and K. Obermayer. Soft learning vector quantization. *Neural Computation*, 15(7):1589–1604, 2003.

[32] V. Sperschneider. *Bioinformatics.* Springer, 2008.

[33] M. Strickert, B. Hammer, T. Villmann, and M. Biehl. Regularization and improved interpretation of linear data mappings and adaptive distance measures. In *IEEE SSCI CIDM 2013*, pages 10–17. IEEE Computational Intelligence Society, 2013.

[34] A. Takasu, D. Fukagawa, and T. Akutsu. Statistical learning algorithm for tree similarity. In *IEEE Int. Conf. on Data Mining, ICDM*, pages 667–672, 2007.

[35] W. Torgerson. *Theory and methods of scaling.* Wiley, 1958.

[36] K. Vanlehn. The behavior of tutoring systems. *International Journal of Artificial Intelligence in Education*, 16:227–265, August 2006.

[37] E. Wenger. *Artificial intelligence and tutoring systems: computational and cognitive approaches to the communication of knowledge.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1987.

[38] C. K. I. Williams and M. Seeger. Using the nyström method to speed up kernel machines. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *NIPS*, pages 682–688. MIT Press, 2000.

[39] K. Zhang and J. T. Kwok. Clustered nyström method for large scale manifold learning and dimension reduction. *IEEE Transactions on Neural Networks*, pages 1576–1587, 2010.

[40] X. Zhu. *Adaptive prototype-based dissimilarity learning.* PhD thesis, Bielefeld University, Faculty of Technology, Bielefeld, Germany, 2014 (submitted).

[41] X. Zhu, A. Gisbrecht, F.-M. Schleif, and B. Hammer. Approximation techniques for clustering dissimilarity data. *Neurocomputing*, 90:72–84, 2012.

7 Appendix

7.1 Derivative of soft alignment

Recall the definition of the soft minimum:

$$\text{softmin}(x_1, \ldots, x_n) = \frac{1}{Z} \sum_i p_i \cdot x_i$$

where

$$\begin{aligned} p_i &= \exp(-\beta \cdot x_i) \\ Z &= \sum_j p_j \end{aligned}$$

The derivative of the *soft sequence alignment dissimilarity* with respect to the parameter λ_q is given as:

$$\begin{aligned} \frac{\partial}{\partial \lambda_q} \text{softmin}(x_1, \ldots, x_n) &= \frac{\partial}{\partial \lambda_q} \frac{1}{Z} \sum_i p_i \cdot x_i \\ &= \frac{1}{Z^2} \left[\left(\sum_i \frac{\partial}{\partial \lambda_q} p_i \cdot x_i \right) \cdot Z - \left(\sum_i p_i \cdot x_i \right) \cdot \left(\frac{\partial}{\partial \lambda_q} Z \right) \right] \\ &= \frac{1}{Z} \left[\sum_i \frac{\partial}{\partial \lambda_q} p_i \cdot x_i - \text{softmin}(x_1, \ldots, x_n) \cdot \frac{\partial}{\partial \lambda_q} Z \right] (*) \end{aligned}$$

Furthermore:

$$\begin{aligned} \frac{\partial}{\partial \lambda_q} p_i \cdot x_i &= \left(\frac{\partial}{\partial \lambda_q} p_i \right) \cdot x_i + p_i \cdot \left(\frac{\partial}{\partial \lambda_q} x_i \right) \\ &= p_i \cdot (-\beta) \cdot \left(\frac{\partial}{\partial \lambda_q} x_i \right) \cdot x_i + p_i \cdot \left(\frac{\partial}{\partial \lambda_q} x_i \right) \\ &= p_i \cdot \left(\frac{\partial}{\partial \lambda_q} x_i \right) \cdot (-\beta \cdot x_i + 1) \end{aligned}$$

and:

$$\begin{aligned} \frac{\partial}{\partial \lambda_q} Z &= \sum_j \frac{\partial}{\partial \lambda_q} p_j \\ &= \sum_j p_j \cdot (-\beta) \cdot \frac{\partial}{\partial \lambda_q} x_j \end{aligned}$$

It follows:

$$
\begin{aligned}
(*) &= \frac{1}{Z}\left[\sum_i p_i \cdot \left(\frac{\partial}{\partial \lambda_q} x_i\right) \cdot (-\beta \cdot x_i + 1) - \operatorname{softmin}(x_1, \ldots, x_n) \cdot \left(\sum_j p_j \cdot (-\beta) \cdot \frac{\partial}{\partial \lambda_q} x_j\right)\right] \\
&= \frac{1}{Z}\left[\sum_i p_i \cdot \left(\frac{\partial}{\partial \lambda_q} x_i\right) \cdot (-\beta \cdot x_i + 1) + \sum_i \operatorname{softmin}(x_1, \ldots, x_n) \cdot \left(p_i \cdot \beta \cdot \frac{\partial}{\partial \lambda_q} x_i\right)\right] \\
&= \frac{1}{Z}\left[\sum_i p_i \cdot \left(\frac{\partial}{\partial \lambda_q} x_i\right) \cdot (-\beta \cdot x_i + 1) + \operatorname{softmin}(x_1, \ldots, x_n) \cdot \left(p_i \cdot \beta \cdot \frac{\partial}{\partial \lambda_q} x_i\right)\right] \\
&= \frac{1}{Z}\sum_i p_i \cdot \left(\frac{\partial}{\partial \lambda_q} x_i\right) \cdot [-\beta \cdot x_i + 1 + \operatorname{softmin}(x_1, \ldots, x_n) \cdot \beta] \\
&= \frac{1}{Z}\sum_i p_i \cdot \left(\frac{\partial}{\partial \lambda_q} x_i\right) \cdot [1 - \beta \cdot (x_i - \operatorname{softmin}(x_1, \ldots, x_n))] \\
&= \sum_i \left(\frac{\partial}{\partial \lambda_q} x_i\right) \cdot \operatorname{softmin}'(x_i)
\end{aligned}
$$

with

$$\operatorname{softmin}'(x_i) = \frac{p_i}{Z} \cdot [1 - \beta \cdot (x_i - \operatorname{softmin}(x_1, \ldots, x_n))]$$

This directly leads to Equation 7.

Hebbian learning as a limit case The derivative has a particular nice interpretation for $\beta \to \infty$. Consider:

$$
\begin{aligned}
\frac{p_i}{Z} &= \frac{\exp(-\beta \cdot x_i)}{\sum_j \exp(-\beta \cdot x_j)} \\
&= \frac{\exp(-\beta \cdot x_i) \cdot \exp(\beta \cdot \min(x_1, \ldots, x_n))}{\sum_j \exp(-\beta \cdot x_j) \cdot \exp(\beta \cdot \min(x_1, \ldots, x_n))} \\
&= \frac{\exp[-\beta \cdot (x_i - \min(x_1, \ldots, x_n))]}{\sum_j \exp[-\beta \cdot (x_j - \min(x_1, \ldots, x_n))]}
\end{aligned}
$$

Consider two distinct cases for x_j:

- $x_j = \min(x_1, \ldots, x_n)$. Then:

$$\exp[-\beta \cdot (x_j - \min(x_1, \ldots, x_n))] = \exp[-\beta \cdot 0] = 1$$

- $x_j > \min(x_1, \ldots, x_n)$. Then (using $\beta \to \infty$):

$$\exp[-\beta \cdot (x_j - \min(x_1, \ldots, x_n))] \approx 0$$

Let $i_1, \ldots, i_T$ be the indices, for which $x_{i_t} = \min(x_1, \ldots, x_n)$. Then it follows:

$$\sum_j \exp[-\beta \cdot (x_j - \min(x_1, \ldots, x_n)) \approx \sum_{t=1}^{T} \exp[-\beta \cdot (x_{i_t} - \min(x_1, \ldots, x_n)) = \sum_{t=1}^{T} 1 = T$$

which in turn leads to:

$$\frac{p_i}{Z} \approx \delta_{\min}(x_i)$$

where

$$\delta_{\min}(x_i) := \begin{cases} \frac{1}{T} & \text{if } x_i = \min(x_1, \ldots, x_n) \\ 0 & \text{otherwise} \end{cases}$$

Now it is obvious that softmin does indeed approach min for large β:

$$\text{softmin}(x_1, \ldots, x_n) = \frac{1}{Z} \sum_i p_i \cdot x_i \approx \sum_{t=1}^{T} \frac{1}{T} \cdot \min(x_1, \ldots, x_n) = \min(x_1, \ldots, x_n)$$

For $\text{softmin}'(x_{i_t})$ we get:

$$\begin{aligned} \text{softmin}'(x_{i_t}) &= \frac{p_i}{Z} \cdot [1 - \beta \cdot (x_i - \text{softmin}(x_1, \ldots, x_n))] \\ &\approx \frac{1}{T} \cdot [1 - \beta \cdot (x_{i_t} - \min(x_1, \ldots, x_n))] \\ &= \frac{1}{T} \cdot [1 - \beta \cdot 0] \\ &= \frac{1}{T} \end{aligned}$$

For all other x_j with $x_j > \min(x_1, \ldots, x_n)$:

$$\begin{aligned} \text{softmin}'(x_j) &= \frac{p_i}{Z} \cdot [1 - \beta \cdot (x_j - \text{softmin}(x_1, \ldots, x_n))] \\ &\approx 0 \cdot [1 - \beta \cdot (x_j - \min(x_1, \ldots, x_n))] \\ &= 0 \end{aligned}$$

Therefore: $\text{softmin}'(x_i) = \delta_{\min}(x_i)$. Consider Equation 7 again, and plug in that result:

$$\begin{aligned} \frac{\partial d^*(\bar{a}(I+1), \bar{b}(J+1))}{\partial \lambda_q} &= \delta_{\min}(A_{\texttt{Rep}}) \cdot \left(\frac{\partial d^*\left(\bar{a}(I), \bar{b}(J)\right)}{\partial \lambda_q} + \frac{\partial d_\lambda(a_{I+1}, b_{J+1})}{\partial \lambda_q} \right) \\ &+ \delta_{\min}(A_{\texttt{Ins}}) \cdot \left(\frac{\partial d^*\left(\bar{a}(I+1), \bar{b}(J)\right)}{\partial \lambda_q} + \frac{\partial d_\lambda(-, b_{J+1})}{\partial \lambda_q} \right) \\ &+ \delta_{\min}(A_{\texttt{Del}}) \cdot \left(\frac{\partial d^*\left(\bar{a}(I), \bar{b}(J+1)\right)}{\partial \lambda_q} + \frac{\partial d_\lambda(a_{I+1}, -)}{\partial \lambda_q} \right) \end{aligned}$$

Recall Equation 3 and consider the optimal extensions $\bar{a}^*$ and $\bar{b}^*$ using arg min instead of min. Using a simple inductive argument it clearly follows:

- For the case of symbolic sequences:

$$\frac{\partial}{\partial \lambda_{km}} d(\bar{a}, \bar{b}) = \sum_{i=1}^{|\bar{a}^*|} \delta(a_i^*, k) \cdot \delta(b_i^*, m)$$

- For the case of vectorial sequences:

$$\frac{\partial}{\partial \lambda_r} d(\bar{a}, \bar{b}) = \sum_{i=1}^{|\bar{a}^*|} d_r({a_i^*}^r, {b_i^*}^r)$$

  where ${a_i^*}^r = \psi$ if $a_i^* = -$ and ${b_i^*}^r = \psi$ if $b_i^* = -$.

This strongly resembles Hebbian learning as argued in Section 2.3.

7.2 Information about the Chromosomes data set

Σ	Δ	# occ.
f	-6	0
e	-5	32
d	-4	149
c	-3	468
b	-2	770
a	-1	2542
=	0	17675
A	+1	2746
B	+2	596
C	+3	318
D	+4	195
E	+5	114
F	+6	0

Table 6: The differential encoding for sequences from the *Chromosomes* data, described in Section 4.2: each symbol of the alphabet Σ represents a level of difference Δ in the density along a banded chromosome. The number of occurrences (# occ.) of each symbol are reported for the *"CopChromTwo"* set, which is a subset of the original Copenhagen Chromosomes database, see [23, 15].

7.3 Information about the Sorting data set

Σ_κ for property '`type`'	Example
`array access`	`arr[4]`
`array type`	`int[]`
`assignment`	`tmp = arr[i]`
`binary`	`arr[i] > arr[i + 1]`
`block`	`{ ... }`
`break`	`break;`
`class`	`public class MyClass { ... }`
`compilation unit`	The entire program file
`compound assignment`	`a += 2`
`do while loop`	`do{ i++;} while(i < 10);`
`expression statement`	`tmp = arr[i];`
`for loop`	`for(i = 1; i < 10; i++) { ... }`
`identifier`	`i`
`if`	`if(i > 10){ ... }`
`import`	`import java.util.HashMap;`
`literal`	`5`
`member select`	`arr.length`
`method`	`int my_fun(int i) { ... }`
`method invocation`	`bubble(A, l, r)`
`modifiers`	`public`
`new array`	`new arr[4]`
`new class`	`new ArrayList<Integer>()`
`parameterized type`	`new ArrayList<Integer>`
`paranthesized`	`(arr[i] > arr[i + 1])`
`primitive type`	`int`
`return`	`return arr;`
`unary`	`i++`
`variable`	`int i = 0;`
`while loop`	`while (swapped) { ... }`

Table 7: The alphabet Σ_κ for property `type` used in the *Sorting* data set, in Section 4.3: every symbol of the alphabet (left), with an example Java code snippet illustrating the respective type (right).