EBERHARD-KARLS-UNIVERSITÄT TÜBINGEN
Wilhelm-Schickard-Institut für Informatik
Lehrstuhl Rechnerarchitektur

Studienarbeit

Using label metrics for Distance Metric Learning

Michael Schober

Betreuer:	Prof. Dr. rer. nat. Andreas Zell Wilhelm-Schickard-Institut für Informatik
Begonnen am:	July 6, 2009
Beendet am:	December 11, 2009

Erklärung

Hiermit versichere ich, diese Arbeit selbstständig verfasst und nur die angegebenen Quellen benutzt zu haben.

Tübingen am December 11, 2009

_____

Michael Schober

Abstract. In this research thesis I want to investigate the usage of label metrics for Distance Metric Learning. The question is: given a specific label metric on an input space, can we learn a metric on this input space that respects the metric on its label set? I am going to examine whether the input can be transformed in such a way that we can use existing machine learning algorithms to learn the desired distance function. Furthermore, I am going to ask which label metrics lead to a successful input space metric and if and how useful label metrics can be generated.

Acknowledgments

I want to thank my advisor Hannes Planatscher for giving me this unique opportunity to work on such an interesting topic. Without his help, his patience and continuing support this would have never been possible or thinkable.

Contents

1. Introduction

Years and experience in software development have revealed many applications for meaningful similarity measurements, also called distance metrics. Web-shops want to offer their clients similar products, private users search for similar documents on their personal computers or on the web and the essentially same task in computer vision has grown so popular that it received its own name: content-based image retrieval or short CBIR. Generally, these tasks are called cluster analysis or simply clustering.

It has been shown that a good distance metric can improve the quality of clustering results significantly [XNJR03]. Another very important application is the k-nearest neighbor (KNN) machine learning algorithm for classification and regression. KNN is one of the oldest and simplest machine learning algorithms but still yields comparably good results. This algorithm assumes in classification that a given instance to predict shares the same class as its k-nearest neighbors or the average of its neighbors' values in a regression setting.

Although the importance of a good distance metric has been recognized, it is still considerably hard to create a meaningful metric for any specific task and setting. To declare such a metric manually, the developer has to consider both the data available in the input-space and the designated application. If the input-space is represented as a subset of the n-dimensional real vector-space, one can choose a standard metric such as Euclidean distance or Mahalanobis distance. However, this rarely solves the problem at hand. For example, Euclidean distance implicitly weights every component of the input-space equally important and does not combine different dimensions to calculate the distance which often leads to meaningless or even counterproductive metrics. One can suggest that the developer could think of a way to transform the input-space in such a manner that one of these metrics becomes meaningful. But this basically leads to the same difficult and complex problem as before. (This thesis will show in section 2.2 that this problem is not only *basically* but the *exact* same problem in a certain sense.)

Another possible solution to this problem is to find such a metric computationally. This lead to an increasing interest in *Distance Metric Learning* in the Machine Learning community. But Distance Metric Learning is still a very recent technique and as [Yan07] shows in her current overview that most of the research is not older than ten years.

Since then, many different approaches to learn a meaningful distance metric have been proposed and applied. Most of these algorithms share simple intuitions and algorithms. Many of them heavily relate on standard techniques in linear algebra, specially matrix manipulation and eigenproblems. Therefore, these techniques will automatically improve with further advances in numerical analysis. Although DML has a fast growing research knowledge, there are still many methods and applications to be discovered.

Additionally, the result of these techniques are often still interpretable, which makes it even more useful to a human user than traditional methods, e.g. support vector machines or neural networks. However, these algorithms suffer from a common drawback: up to this date, non of these algorithms learn a specific metric representing a distance that a human user would assign to two different objects. Therefore, assigned distances can only be interpreted as a relative measurement, not as absolute values. Even if we assume, that one of these algorithms could perfectly learn a distance metric on a given problem, an user could not determine, whether the

inter-distances grow linear, logarithmic or even exponentially. That means: given three samples x, y and z of some problem setting, he could probably order them linearly, e.g. $x \leq y \leq z$, but he wouldn't know if z is twice as much away from x as y is or maybe even ten times.

Therefore, in this research thesis it will be examined whether a specific distance metric can be learned by declaring a distance metric on the instance labels. A new general algorithm will be proposed which tries to learn and use the specific distances between samples. Additionally, distance metric learning is applied on machine learning tasks that have not been covered by DML previously, namely to regression analysis and more general transduction settings. Specifically, the new algorithm is applied to a *Quantitative structure-activity relationship (QSAR)* problem. The goal is to predict biological or chemical reactivity according to a chemical structure. In the problem setting on hand, the reactivity of enzymes on three different proteins are predicted simultaneously. Other current Distance Metric Learning algorithms are not capable of such a prediction. Further, it is analyzed how different metrics will influence the prediction results.

The rest of this thesis is organized as followed: in chapter 2 this thesis covers some of the theoretical background, analyzes the problem setting in a little more detail and presents related work. Chapter 3 introduces the new algorithm and describes a sample implementation as a Java framework. In chapter 4, the new algorithm is evaluated in a series of different test settings on benchmark data sets. Chapter 5 discusses the findings of the empirical results and chapter 6 comes to a conclusion and gives prospects of future work.

2. Background

First, a short introduction to machine learning topics in general and distance metric learning topics specifically is given. Afterwards, this thesis gives an overview of related work.

2.1. General topics in Machine Learning

Machine Learning or *Pattern Recognition* is the algorithmic approach to implement automatically a rule-based decision progress for some problem setting. The goal is that these rule-based decisions are the same decisions a human observer would come to. Typical questions arise naturally in statistical evaluations or other sensor data information. Applications include the automatic evaluation of large tests series, where much performance data is available, for example in biology or medicine or the recognition of some typical buying behavior based on collected data in databases. Another important application is the automatic recognition of sensor *content*, which means not the actual color data of every pixel or the frequencies of some audio source, but the information *what exactly* we see or hear, for example a person or an airplane etc.

Among the first algorithms that were applied was the *Least squares method* developed by Carl Friedrich Gauss around 1800 to predict celestial orbits. Since then, many different algorithms and heuristics have been applied. A good introduction on current techniques is given in [Bis06].

Two algorithm types that will be mentioned in this thesis are *supervised learning* and *unsupervised learning*. The goal of supervised learning is to generalize a function that maps some input data on some desired output data, called *labels*. In this learning setting the machine learning algorithm is given a set of examples with according labels, which is called the *training set*. Common input data and used throughout this thesis are real-valued vectors $x = (x_1, \ldots, x_n)^T \in \mathbb{R}^n$. Typical label spaces are classes, denoted by natural numbers $\mathscr{L} = \{1, \ldots, J\}$. This task is called *classification*. If the labels are continuous real values $\mathscr{L} \subset \mathbb{R}$, then the task is called *regression analysis* or simply *regression*. These two are the most common supervised learning settings.

In one case, the new presented algorithm will also try to predict three target values simultaneously, i.e. the label set will be a subset of $\mathbb{R}^3$. Such a general prediction task is called a *transductive task*.

In supervised learning, a so-called *model* is used to represent the target function. During the training phase, the training set is presented to the machine learning algorithm, which either creates a new model or adapts parameters of a pre-selected model. For example, if it's already known, that the training data will represent a simple one-dimensional linear transformation, a good choice for a pre-selected model would be

$$f(x) = \alpha x + \beta \tag{2.1}$$

with adaptive parameters α and β. In binary classification tasks (that is, a classification task with exact two different classes) the model is often a representation of some hyperplane with the vectors of each class on each side.

In contrast to supervised learning, in unsupervised learning the algorithms are not given additional label information, the training set $X = \{x_1, \ldots, x_m\}$ is simply a subset of $\mathbb{R}^n$. In unsupervised learning the goal is not to learn an explicit label mapping as in supervised learning, but to retrieve information of some underlying structure or patterns. The most common task is clustering, where each input vector is assigned to a class, similar to classification. However, it is not necessarily known how many classes exists or by which rule they should by grouped. The algorithms have to find meaningful groupings on their own.

A common way to evaluate the quality of a new algorithm or method is called *k-fold cross-validation* on benchmark data sets where the labels of all instances are already known. This data set is randomly divided into k groups. The algorithm is then applied k times with $k-1$ groups to learn from and 1 group to predict. The performance is measured in every run, each time leaving out another group. Finally, the results over all performances are averaged. For a thorough analysis this setting can be repeated a few times with different groupings.

2.1.1. Support Vector Machines for Regression

This section introduces an important Machine Learning algorithm which will be used later on in this thesis. Additionally, this section presents important concepts of current Machine Learning techniques which will also be used by some algorithms presented in section 2.3.

Let $\mathscr{X} \times \mathbb{R} = \{(x_1, y_1), \ldots, (x_m, y_m)\}$, $\mathscr{X} \subset \mathbb{R}^n$ denote a training set for a regression problem. The goal is to find a function f which approximates the training data very well and is as flat as possible at the same time. Approximating the training data very well in this context means that the prediction error is not greater than a given ε. Initially it is assumed that f is a linear function

$$f(x) = \langle w, x \rangle + b \qquad w \in \mathbb{R}^n, b \in \mathbb{R} \tag{2.2}$$

where $\langle \cdot, \cdot \rangle$ denotes the regular dot-product. To gain a flat function in this formula is the same as demanding $||w||_2^2$ to be small. For samples in the training set that cannot be interpolated without an error less then ε additional variables are introduced called *slack variables* which represent the additional error. Therefore, the optimization problem for this problem setting is:

$$\begin{aligned} \min \ & \frac{1}{2}||w||_2^2 + C\sum_{i=1}^{m}(\xi_i + \xi_i^*) \\ \text{subject to (s.t.) } & y_i - \langle w, x_i \rangle - b \leq \varepsilon + \xi_i \\ & \langle w, x_i \rangle + b - y_i \leq \varepsilon + \xi_i^* \\ & \xi_i, \xi_i^* \geq 0 \end{aligned} \tag{2.3}$$

The constant C determines the trade-off between the flatness of the function f and the tolerance for greater interpolation errors then ε. The advantage of formula (2.3) is that this formula is convex. Research in numerical analysis have shown that there exist efficient algorithms that solve convex optimization problems. Furthermore, these algorithms find the global optimum, if it exists. Summarizingly this means that it is guaranteed to find the best possible approximation in a considerable time. A good introduction into solving convex optimization problems can be found in [BV04].

Formula (2.3) can be rewritten as a single optimization problem by transforming it into its corresponding Lagrangre function. In convex programming this is also called the *dual objective* and the Lagrange multipliers are called *dual variables*. The original formula is called *primal*

objective. The dual objective of formula (2.3) is given by:

$$\begin{aligned} L := \frac{1}{2}||w||_2^2 + C\sum_{i=1}^{m}(\xi_i + \xi_i^*) - \sum_{i=1}^{l}(\eta_i\xi_i + \eta_i^*\xi_i^*) \\ - \sum_{i=1}^{l}\alpha_i(\varepsilon + \xi_i - y_i + \langle w_i, x\rangle + b) - \sum_{i=1}^{l}\alpha_i^*(\varepsilon + \xi_i^* + y_i - \langle w_i, x\rangle - b) \end{aligned} \tag{2.4}$$

where $\eta_i, \eta_i^*, \alpha_i, \alpha_i^*$ are the Lagrange multipliers.

In the global optimum of the primal function, (2.4) has a saddle point, therefore the first-order partial derivatives of L with respect to the primal variables have to vanish.

$$\partial_b L = \sum_{i=1}^{m}(\alpha_i^* - \alpha_i) \stackrel{!}{=} 0 \qquad \partial_w L = w - \sum_{i=1}^{m}(\alpha_i - \alpha_i^*)x_i \stackrel{!}{=} 0 \tag{2.5}$$

$$\partial_{\xi_i} L = C - \alpha_i - \eta_i \stackrel{!}{=} 0 \qquad \partial_{\xi_i^*} L = C - \alpha_i^* - \eta_i^* \stackrel{!}{=} 0 \tag{2.6}$$

Substituting (2.5) and (2.6) into (2.4) leads to the following dual optimization problem:

$$\begin{aligned} \max\ & -\frac{1}{2}\sum_{i,j=1}^{m}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)\langle x_i, x_j\rangle \\ & - \varepsilon\sum_{i=1}^{m}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{m} y_i(\alpha_i - \alpha_i^*) \\ \text{s.t. } & \sum_{i=1}^{m}(\alpha_i - \alpha_i^*) = 0 \\ & 0 \le \alpha_i, \alpha_i^* \le C \end{aligned} \tag{2.7}$$

From (2.5) also follows that $w = \sum_{i=1}^{m}(\alpha_i - \alpha_i^*)x_i$, therefore f can be rewritten as

$$f(x) = \sum_{i=1}^{m}(\alpha_i - \alpha_i^*)\langle x_i, x\rangle + b \tag{2.8}$$

This formula is called the *Support Vector expansion*, because f can now be completely described as a linear combination of the samples in the training set. Further analysis shows that only samples which lie on the edge or outside the ε-tube have non-zero weights. Therefore, only these samples are needed to compute the value f for some input x, making the representation sparse and reducing computational cost. These samples are called *Support Vectors* of f giving the algorithm its name. Figure 2.1 presents an example application of this algorithm. Note that the bigger the value for ε is chosen the broader the tube gets and the less samples receive a non-zero weight. On the other hand, if the original function f is not very smooth from the beginning or the training data includes a lot wrong or compromised data the number of support vectors will rise again. Therefore, the number of support vectors remaining after optimization is an indicator how good the function f can be represented and learned. This fact will be needed again later in the analysis of the newly presented algorithm.

It can be further noted that this reformulation also makes it possible to extend f into a non-linear function. This can be achieved by mapping the training samples from the original input space in some higher-dimensional feature-space $\mathscr{F}$ via $\phi : \mathscr{X} \to \mathscr{F}$ and applying the SVR algorithm in the feature-space. For example, let $\mathscr{X} \subset \mathbb{R}^2$ and $\phi(x_1, x_2) = (x_1^2, \sqrt{2}x_1x_2, x_2^2) \in \mathbb{R}^3$. But this could be very expansive computationally. However, from (2.8) follows that only inner-products are needed in the feature-space $\mathscr{F}$. Finding a function $k(x_i, x_j) = \langle\phi(x_i), \phi(x_j)\rangle$ is

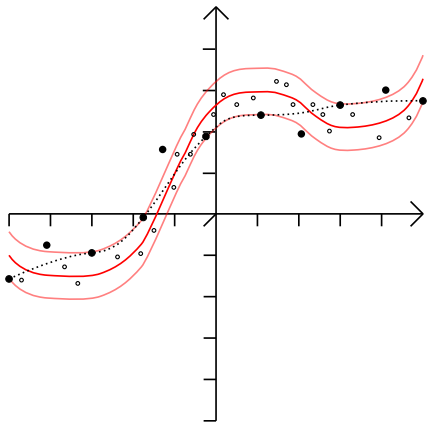


Figure 2.1.: Example for Support Vector Regression.
This figure illustrates the application of the Support Vector Regression algorithm. The true function (red) and the ε-insensitive tube around it. The dotted line is the regression function with its support vectors (**filled dots**). Training samples that lie inside the ε-tube and are not needed to compute the regressions are empty dots.

therefore sufficient to solve the problem at hand. Such a function is called a *kernel* and the idea of implicitly transforming the input-space into a high-dimensional space is called the *kernel trick*. The theory of kernels has been thoroughly studied and methods are known how to create valid kernels. The following kernel functions have been tested in this thesis:

$$\textbf{Dotproduct kernel:} \qquad k_1(x_i, x_j) = \langle x_i, x_j \rangle \tag{2.9a}$$

$$\textbf{RBF kernel:} \qquad k_2(x_i, x_j) = \exp\left(\frac{-||x_i - x_j||_2}{2\sigma^2}\right) \tag{2.9b}$$

$$\textbf{Rational quadratic kernel:} \qquad k_3(x_i, x_j) = \left(1 + \frac{||x_i - x_j||_2^2}{2\alpha\sigma^2}\right)^{-\alpha} \tag{2.9c}$$

Formula (2.9a) is the standard dot-product in $\mathbb{R}^n$. Therefore, this kernel function does not induce non-linearity. The following two kernels are motivated by probability theory and do induce non-linearity to the target function f. The function (2.9b) is called *Radial Basis Function (RBF)* or *Gaussian* kernel with parameter σ. The best value for the parameter σ has to be optimized for each training set separately and cannot be found with the SVR algorithm. Often this is done through a series of cross-validations with different values for σ. The last kernel is called *rational quadratic* kernel with parameters σ and α and can be interpreted as sum of different RBF kernels. Again, the parameters have to be adjusted independently from the original optimization algorithm.

Currently the general approach to Support Vector Regression is to guess and try different choices for the kernel function with the three presented above being popular and often successful choices. Usually an user starts with the simplest kernel and slowly progresses to more complex ones if necessary. From the given list the Rational Quadratic kernel can be said to be the most complex kernel, implicitly because of its interpretation as sum of RBF kernels and explicitly because of its additional number of parameters. This also will be an important observation in later analysis.

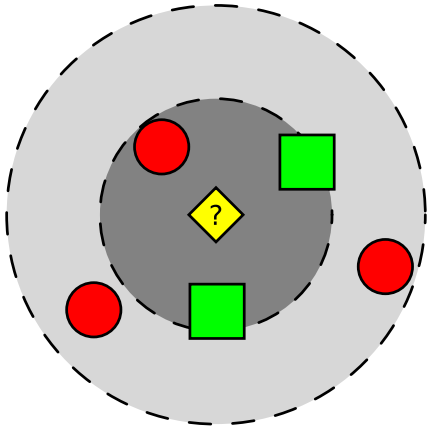


Figure 2.2.: Example for a kNN classification.
An example for the classification process in the kNN algorithm. The yellow square is the unknown test sample. Depending on the size of k, the yellow square is either assigned to the red circles' class ($k = 1$ or $k = 5$) or to the green squares' class ($k = 3$).

2.1.2. The k-Nearest-Neighbor algorithm

A very important machine learning algorithm in the context of Distance Metric Learning is the *k-nearest neighbor (KNN)* algorithm. KNN will widely be applied throughout this thesis. The KNN algorithm for classification assumes that the training set $X \times \mathscr{L}$ is generated by an unknown distribution $P(x,l)$. Let $p(j|x)$ denote the probability that an unknown vector x belongs to class j. This is called the *posterior distribution* in dependency of x, i.e. the probability depends specifically on the input vector. To classify a new sample $\tilde{x}$ the class posterior probabilities $p(j|\tilde{x})_{j=1}^{J}$ are assumed as follows.

The class posterior probabilities are unlikely to change dramatically in a region near $\tilde{x}$, i.e. $p(j|\tilde{x}) \approx p(j|\tilde{x}+\delta r)$, where r is a random vector and δ is a factor close to 0. This is called the *smooth neighborhood assumption*. Therefore, $p(j|\tilde{x})$ can be expressed as

$$p(j|\tilde{x}) \approx \frac{\sum_{x \in N(\tilde{x})} p(j|x)}{|N(\tilde{x})|} \tag{2.10}$$

where $N(\tilde{x})$ is the *neighborhood* of $\tilde{x}$, consisting of the k-nearest vectors in the training set. The simplest estimate of this probability is a frequentist approach, meaning that the probability is the number of vectors with label j in the neighborhood divided by the number of all vectors in the neighborhood, or as formula:

$$p(j|\tilde{x}) \approx \frac{|\{(x,l)|\, x \in N(\tilde{x}), l = j\}|}{k} \tag{2.11}$$

$\tilde{x}$ is then assigned the class with the highest posterior probability. Figure 2.2 illustrates this operation method.

The KNN algorithm for regression assumes that the label of each vector can approximately be expressed as a linear combination of the values of the vectors nearby:

$$f(\tilde{x}) \approx \frac{1}{k} \sum_{x \in N(\tilde{x})} f(x) = \frac{1}{k} \sum_{(x,l) \in N(\tilde{x})} l \tag{2.12}$$

The function can be smoothed by weighting the samples in the neighborhood by their distance.

A common approach is for example

$$f(\tilde{x}) \approx \frac{1}{S_{\tilde{x}}} \sum_{x \in N(\tilde{x})} \exp(-d(\tilde{x},x)) f(x) \tag{2.13}$$

where $S_{\tilde{x}} = \sum_{x \in N(\tilde{x})} \exp(-d(\tilde{x},x))$. Since our algorithm will produce meaningful distances between samples, it suggests the usage of the later formula.

At this point, the interpretation of the parameter k is also noteworthy, because it will play a role in later analysis . In the context of a probability analysis, the *k*NN algorithm can be thought of as an estimator for the underlying distribution the data was generated with. There are generally very different ways to estimate the original distribution for a data source with *k*NN belonging to the so-called *non-parametric* methods, because no general distribution model was to be chosen before applying this algorithm. The parameter k in the nearest-neighbor algorithm controls the sensitivity to local data peaks. If k is close to 1, the distribution will be estimated very noise and rather unsmooth. As k gets larger the algorithm will smooth out local irregularities and also start to smooth out global distributions. Finally, as k approaches the size of the training data set, the k nearest neighbors will almost always be the same input samples, therefore always predicting the most occurring label in the training set.

These algorithms share common advantages and disadvantages. Some advantages are:

- The theory is simple and demonstrative, yet also convincing.
- The implementation is straight forward and very simple.
- The algorithm does not need to compute any prior model before it is able to predict, making it very fast at learn-time.
- The algorithm is not restricted to a specific learning task, e.g. classification or regression. It can be applied to any retrieval task, if some kind of comprise or mean can be declared on the label space. As last resort, 1NN can be applied to really any task. Thus, it is very flexible. An universal machine learning algorithm in a manner of speaking.

However, there are also some disadvantages:

- The computation is very expensive at predicting-time, when the size of the training set is very large, since it has to compute all the distances to the instances in the training set.
- However, if the number of instances is very small, the neighborhood might not contain actual nearby neighbors, but also very far neighbors, inferring with our assumption of smoothness. Even worse, the training set should best be equally distributed over the input space, which is not always the case.
- Additionally, with a linear increasing number of dimensions an exponentially increasing number of instances is needed to maintain the same density of the instance distribution. This is called the *curse of dimensionality*.
- Also, the label changes will increase more significantly near decision boundaries or along the gradient of the target function, further imbalancing the assumption.

To overcome the problems handling large data sets some authors [LMGY04, FBF77, BKL06] have proposed special tree-based data structures that speed-up the *k*NN search. The idea is to partition the input space into smaller regions. Then during look-up a whole region can be cut

off, if the query point is nearer to one region than to another, which reduces the number of comparisons and calculated distances. However, this methods were not applied in this thesis. The usage of a machine learning algorithm for distance prediction implies that the calculated distances are not necessarily correct. Therefore, such a data structure could wrongly cut off a whole region, leading to worse results. Additionally, computationally efficiency was not a question of debate for this research thesis. We first need to know *if* this method works anyway before we think of how we could possibly make it faster.

The last three bullet points share an underlying theme: the KNN algorithm is heavily influenced by the positions of the training vectors in the vector-space, potentially disturbing the smooth neighborhood assumption . These shortcomings can thereby be tackled with the same approach. The important idea is to declare neighborhood with a metric adjusted to the problem setting at hand. This is exactly the goal of *Distance Metric Learning*. In the next chapters, I will first present some mathematical background on distance metrics and related topics in $\mathbb{R}^n$, then I will give an overview of previous and current research related to DML and then I will present the background to my approach.

2.2. Metrics and Machine Learning algorithms on $\mathbb{R}^n$

In mathematics, a *metric* is a generalization of our understanding of distance measurement. It is defined as follows:

Definition 1. Let $\mathscr{L} \neq \emptyset$ be a set. A *metric* is a function $d : \mathscr{L} \times \mathscr{L} \to \mathbb{R}_{\geq 0}$ with

$$\forall x, y \in \mathscr{L} : \qquad d(x,y) = d(y,x) \tag{2.14a}$$

$$\forall x, y, z \in \mathscr{L} : \qquad d(x,z) \leq d(x,y) + d(y,z) \tag{2.14b}$$

$$\forall x, y \in \mathscr{L} : \qquad d(x,y) = 0 \Leftrightarrow x = y \tag{2.14c}$$

If d satisfies all but (2.14c), d is called a *pseudo-metric*.

Distance Metric Learning is the process of finding such a function d computationally. The input data of the analyzed problems will all be in form of a n-dimensional real-vector, therefore it is useful to investigate distance measurement on the $\mathbb{R}^n$ vector-space.

The most intuitive distance metric is the generalization of the Pythagorean theorem. It declares the distance of two vectors x, y as

$$d(x,y) := \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2} = ||x - y||_2 \tag{2.15}$$

This can also be written as

$$d(x,y) = \sqrt{\langle x - y, x - y \rangle} \tag{2.16}$$

which suggests that a distance metric can be generally obtained by an inner product on $\mathbb{R}^n$. Results from linear algebra proof that this assumption is true. Additionally it has been shown, that every symmetric positive definite matrix $\mathbf{M}$ declares such an inner product by

$$\langle x, y \rangle_{\mathbf{M}} := x^T \mathbf{M} y \tag{2.17}$$

Since we will often be satisfied with a pseudo-metric that does not distinguish between different vectors but only between vectors with different labels, the positive definite constraint on the matrix can be relaxed to a positive semi-definite (p.s.d.) constraint.

Although many other metrics are also imaginable on $\mathbb{R}^n$, this type of metric has some advantage towards Distance Metric Learning:

1. The model that has to be learned is obviously the p.s.d. matrix $\mathbf{M}$ which eliminates the difficulty of finding an appropriate model.

2. Every p.s.d. matrix induces also a *norm* on $\mathbb{R}^n$, preserving the linear structures of the vector space.

3. These metrics can be interpreted very concretely, thus giving access to a better understanding of the found solution and the original data.

The last point of this enumeration might need some clarification. Consider for a moment Euclidean Distance. This metric is induced by the identity matrix $\mathbf{I}$. This matrix is characterized by its diagonality and its equal entries. We can derive two properties, namely that this metric will not combine features from different dimensions and every dimension will be weighted equally important (compare this to page 1). If our matrix $\mathbf{M}$ has non-zero elements outside the main diagonal for some value $a_{ij} \neq 0$, then the i-th feature and the j-th feature will be combined somehow to calculate the distance between two input vectors. After applying a machine learning algorithm on the problem data set, one can interpret the learned model in this way and consider its implications, for example, whether a result like this was expected or makes sense.

Another way to interpret the result is the following: every p.s.d. matrix can be written as product of a lower-triangular matrix $\mathbf{L}$ and its transpose:

$$\mathbf{M} = \mathbf{L}^T\mathbf{L} \tag{2.18}$$

Then the metric can also be written as

$$d_{\mathbf{M}}(x,y) = \sqrt{(x-y)^T\mathbf{L}^T\mathbf{L}(x-y)} = \sqrt{||\mathbf{L}(x-y)||_2} = \sqrt{||\mathbf{L}x-\mathbf{L}y||_2} \tag{2.19}$$

which leads to another interesting interpretation: first the input space is linear transformed with the transformation $\mathbf{L}$, in which then a Euclidean Distance is applied. Even more, for every matrix $\mathbf{L} \in \mathbb{R}^{n\times n}$, the matrix $\mathbf{M}$ computed as in 2.18 is a positive semi-definite matrix. Thus, we could interpret Distance Metric Learning also as some kind of *information maximizing transformation learning*.

2.3. Related work

A good overview of previous work on Distance Metric Learning give [YJ06, Yan07]. Generally speaking, Distance Metric Learning can be divided into two categories: *unsupervised DML* and *supervised DML*.

2.3.1. Unsupervised Distance Metric Learning

Unsupervised distance metric learning, or also called manifold learning, tries to find a dimensionality reduction that respects the geometric relationships, such as the distance or local neighborhood. Such a structure is called *manifold* in mathematics. There are several approaches using different techniques and targeting at different aims. Generally speaking, these algorithms can be classified whether they preserve local or global information and wether they work linear or nonlinear. Table 2.1 summarizes the most important methods. Since this thesis is mainly related to supervised DML, only the most important methods are mentioned and their basic ideas summarized.

	Linear	Nonlinear
Global	PCA [GW92], MDS [Cox00]	ISOMAP [TSL00]
Local	NPE [HCYZ05], LPP [HN03]	LLE [RS00], LE [BN03]

Table 2.1.: Several unsupervised DML algorithms, classified whether they preserve local or global information and whether they work linear or nonlinear.

Principal Component Analysis (PCA) [GW92] tries to find the subspace that best preserves the variance of the data. Therefore, it rotates the first axis towards the direction of the most variance (called the first principal component), the second axis towards of the second most variance and so on. Additionally, after the transformation different components are unrelated.

Multidimensional Scaling (MDS) [Cox00] finds a m-dimensional projection with m defined by the user that preserves pairwise distances for any two input samples. The information of the pairwise distances is given as a matrix

$$\Delta := (d(x_i, x_j))_{i,j=1,\dots,m} \tag{2.20}$$

This has the advantage that the distance does not need to be Euclidean distance but can be another useful similarity measurement. MDS is closely related to PCA in that sense that the computed transformation only differs in a further multiplication with the eigenvalues of the input variance matrix. The computation of this transformation uses amongst others spectral decomposition.

ISOMAP [TSL00] is a nonlinear dimension reduction algorithm that tries to preserve geodesic distances. It constructs a weighted graph by computing the distances to each input point's nearest neighbors. The distance of any two points in the dataset is then declared as the weight of shortest path in the graph between these points. This distance measurement is then used as the distance matrix for the MDS algorithm in formula (2.20).

Locally Linear Embedding (LLE) [RS00] is a nonlinear dimension reduction that tries to preserve global manifolds like MDS and ISOMAP. However, LLE achieves this goal through local linear information. The idea is that each data point and its neighbors lie on a local linear patch, i.e. they lie locally on a common hyperplane. Therefore, each point can be reconstructed via a linear combination of its neighbors.

$$\begin{aligned} &\min_{W \in R^{m \times m}} \sum_{i=1}^{m} ||x_i - \sum_{j=1}^{m} W_{ij} x_j|| \\ &\text{s.t.} \sum_{j=1}^{m} W_{ij} = 1, \quad \forall\, i = 1, \dots, m \\ &\qquad W_{ij} = 0, \quad \text{if } x_j \text{ is not a neighbor of } x_i \end{aligned} \tag{2.21}$$

In (2.21) the constraint form the geometric intuition and will later contain the geodesic information. The next step is to find lower-dimensional points y_i that preserve the linear patches, calculated in 2.21.

$$\min_{y_1,\dots,y_m} \sum_{i=1}^{m} ||y_i - \sum_{j=1}^{m} W_{ij} y_j|| \tag{2.22}$$

Notice, that in (2.21) W is optimized whereas in (2.22) W is fix and the points get optimized.

Neighborhood Preserving Embedding (NPE) [HCYZ05] is a linear approximation to the LLE algorithm. It similarly computes the weight matrix W. However, to preserve the linear reduction the following eigenvector problem is solved:

$$XMX^T a = \lambda XX^T a \tag{2.23}$$

where

$$\begin{aligned} X &= (x_1, \ldots, x_m) \\ M &= (I-W)^T (I-W) \\ I &= \text{diag}(1, \ldots, 1) \end{aligned}$$

The first d column vectors $a_0, \ldots, a_{d-1}$ sorted by their eigenvalues $\lambda_0 \leq \cdots \lambda_{d-1}$ from equation 2.23 constitute the transformation matrix A:

$$y_i := Ax_i \tag{2.24}$$

Laplacian Eigenmap (LE) [BN03] looks for an embedding manifold that preserves local neighbor structure. Therefore, LE creates a weighted graph similar to ISOMAP. It then solves an eigen decomposition problem and uses the leading eigenvectors to transform the instances. It has been shown that there is a strong connection between LLE and LE. *Locality Preserving Projections (LPP)* [HN03] is the linear approximation to LE.

2.3.2. Supervised Distance Metric Learning

Supervised Distance Metric Learning uses additional information to learn a meaningful metric on the original input space. Despite their name, most of these algorithms are actually *semi-supervised* learning algorithms, which means, that most of them do not need access to all label information. Usually these methods are given sets of similarity and dissimilarity constraints as follows:

$$\mathscr{S} = \{(x_i, x_j) | x_i \text{ and } x_j \text{ are similar}\} \qquad \mathscr{D} = \{(x_i, x_j) | x_i \text{ and } x_j \text{ are dissimilar}\} \tag{2.25}$$

Existing methods can be classified by some extend by the amount of side-information they use. Additionally, as in unsupervised DML, the algorithms can further be classified, whether they find a global or a local metric.

The most general approach makes [XNJR03]. They formulate the problem of finding a good distance metric as convex optimization problem as follows:

$$\begin{aligned} \min_{\mathbf{M} \in \mathbb{R}^{n \times n}} & \sum_{(x_i, x_j) \in \mathscr{S}} d_{\mathbf{M}}(x_i, x_j)^2 \\ \text{s.t.} & \sum_{(x_i, x_j) \in \mathscr{D}} d_{\mathbf{M}}(x_i, x_j)^2 \geq 1, \\ & M \succeq 0 \end{aligned} \tag{2.26}$$

The positive semi-definite constraint ($\mathbf{M} \succeq 0$) ensures, that a valid metric is learned. The optimization tries to bring similar points as close together as possible by ensuring a minimum distance of 1 for dissimilar points. Note also, that the first constraint is needed so that the optimization problem doesn't find the trivial solution by collapsing all points into a single one.

[KT03] extends this algorithm to the nonlinear case with the introduction of kernels. [SJ04] uses feedback of the form "A is closer to B than to C". They apply a SVM-algorithm to DML.

While these techniques try to find a meaningful distance metric in a general sense, many algorithms specifically try to find a metric that improves KNN or clustering performance. The former are therefore classified as *Global Distance Metric Learning* whereas the latter are referred to as *Local Distance Metric Learning*. Local Distance Metric Learning specifically targets the two major drawbacks for the KNN algorithm as mentioned above (see p. 9).

Neighborhood Components Analysis (NCA) [GRHS05] learns a linear transformation $\mathbf{A}$ for the input-space, that tries to maximize the right classification rate. In NCA, neighbors are selected through a probability measurement. The probability, that x_j is chosen as a neighbor for x_i is given by

$$p_{ij} = p^{\mathbf{A}}(j|i) = \frac{\exp(-||\mathbf{A}(x_i - x_j)||^2)}{\sum_{k \neq i} \exp(-||\mathbf{A}(x_i - x_k)||^2)} \tag{2.27}$$

Let the set of points that share the same class with x_i denoted by $L_i = \{j | l_i = l_j\}$. Then, the chance of classifying x_i correctly is given by $\sum_{j \in L_i} p_{ij}$ and the leave-one-out estimated error-rate is given by

$$f(\mathbf{A}) = \sum_{i=1}^{n} \log(\sum_{j \in L_i} p_{ij}) \tag{2.28}$$

This function can by maximized by using a gradient based optimizer. However, since the objective function is not convex, these maximizing techniques might get stuck on local optima.

Maximally Collapsing Metric Learning (MCML) [GR06] essentially use the same distribution (2.27), however they use a slightly different learning algorithm. The main intuition is that points sharing the same class should ideally be collapsed into a single point, whereas other points should be infinitely far away. This perfect distribution would be represented via

$$p_0(j|i) \propto \begin{cases} 1 & y_i = y_j \\ 0 & y_i \neq y_j \end{cases} \tag{2.29}$$

To match the distributions (2.27) and (2.29), the authors minimize the Kullback-Leibler divergence:

$$\begin{aligned} &\min_{\mathbf{A} \in \mathbb{R}^{n \times n}} \sum_i KL[p_0(j|i) | p^{\mathbf{A}}(j|i)] \\ &\text{s.t. } \mathbf{A} \succeq 0 \end{aligned} \tag{2.30}$$

This is a convex optimization problem and thus the global minimum can efficiently be computed. Additionally, the authors proposed a kernel extension.

The *Local Distance Metric (LDM)* algorithm [YJSL06] tries to learn a metric that is capable of handling multi modal distributions. They also apply a probabilistic framework on the leave-one-out evaluation. For any point x in the training set, let $\phi_{\mathscr{S}}(x) = \{x_i | (x, x_i) \in \mathscr{S}\}$ and $\phi_{\mathscr{D}}(x) = \{x_i | (x, x_i) \in \mathscr{D}\}$ denote the set of points that are declared similar to x and dissimilar respectively. According to the kernel-based KNN classifier, the probability of making the right prediction for x is given by

$$p_{\mathbf{A}}(+|x) = \frac{\sum_{x_i \in \phi_{\mathscr{S}}(x)} \exp(-||\mathbf{A}(x - x_i)||^2)}{\sum_{x_i \in \phi_{\mathscr{S}}(x)} \exp(-||\mathbf{A}(x - x_i)||^2) + \sum_{x_j \in \phi_{\mathscr{D}}(x)} \exp(-||\mathbf{A}(x - x_i)||^2)} \tag{2.31}$$

The leave-one-out log likelihood can then be optimized via

$$\begin{aligned} &\max_{\mathbf{A}\in\mathbb{R}^{n\times n}} \sum_{x\in\mathscr{T}} p_{\mathbf{A}}(+|x) \\ &\text{s.t. } A \succeq 0 \end{aligned} \tag{2.32}$$

where $\mathscr{T}$ is the set of all points included in $\mathscr{S}$ or $\mathscr{D}$. They propose an algorithm that employs eigenvector analysis and bound optimization to optimize this non-convex function.

Bayesian Distance Metric Learning (BDML) [YJS07] estimates the distance metric through a full Bayesian treatment. Additionally, their approach is extensible to support *active learning*, that is to present unlabeled example pairs with the greatest uncertainty of their relative distance.

Local Linear Discriminative Analysis (LLDA) [HT96] applies linear discriminative analysis to neighborhood of each testing point x. In this sense, LLDA is globally nonlinear.

Relevant Component Analysis (RCA) [SHWP02] learns a global linear transformation that weights the importance of the involved dimensions. It therefore uses information in form of *chunklets*. A chunklet is a subset of points belonging to the same class. RCA computes the covariance matrix of all points as the average of all covariance matrix of the chunklets:

$$\widehat{\mathbf{C}} = \frac{1}{p}\sum_{j=1}^{k}\sum_{i=1}^{n_j}(x_{ji}-\widehat{m}_j)(x_{ji}-\widehat{m}_j)^T \tag{2.33}$$

where p is the number of all points, k is the number of chunklets $\{x_{ji}\}_{i=1}^{n_j}$ and $\widehat{m}_j$ is the mean of chunklet j. The global linear transformation then is $x_{new} = \widehat{\mathbf{C}}^{-\frac{1}{2}}x$. [BHHSW03] uses an information theoretic approach for the same task.

Discriminant Component Analysis (DCA) [HLLM06] extends RCA by additionally using negative constraints among the chunklets. They also introduce the kernelized version of DCA for learning nonlinear metrics.

The *Large Margin Nearest Neighbor (LMNN)* classification algorithm [WBS06] introduces the margin based learning to DML. For every point in the training-set, the k nearest points are calculated using Euclidean Distance, here called *target neighbors*. Points x_j that have different labels from point x_i are called *impostors*. A linear distance metric is now calculated that brings the target neighbors with the same class near together and that tries to separate the impostors with a large margin. These goals are formulated as a semi-definite programming instance:

$$\begin{aligned} \min_{\mathbf{M}\in\mathbb{R}^{n\times n}} &(1-\mu)\sum_{i,j\rightsquigarrow i}(x_i-x_j)\mathbf{M}(x_i-x_j)+\mu\sum_{i,j\rightsquigarrow i,l}(1-y_{il})\xi_{ijl} \\ \text{s.t. } &(x_i-x_l)\mathbf{M}(x_i-x_l)-(x_i-x_j)\mathbf{M}(x_i-x_j)\geq 1-\xi_{ijl} \\ &\xi_{ijl}\geq 0 \\ &\mathbf{M}\succeq 0 \end{aligned} \tag{2.34}$$

where y_{il} is 1 if x_i and x_l have the same class labels and 0 otherwise. The parameter μ regulates the tradeoff between bringing target neighbors close together and enforcing the large margin. Similar to support vector machines, the first set of constraints in formula (2.34) introduces slack-variables for constraints that could not be kept otherwise.

3. Methods and materials

This chapter describes the proposed new Distance Metric Learning algorithm, its theoretical background and its realization as a *Java* software package.

3.1. The *Label Metric Learning* algorithm

In this section, the usage and the application of label metrics is motivated and discussed. Additionally, further metrics on the label set are presented.

3.1.1. Using label metrics for Distance Metric Learning

Section 2.3.2 showed that most current *Distance Metric Learning* algorithms try to improve the prediction quality of the KNN algorithm. These algorithms share a common paradigm: the technical background and problems of the algorithm is analyzed and tried to improve, thus leading to improved predictions or, in a short term, *improved technique results in an implicitly better semantic content*.

Here a new algorithm is suggested, in which explicitly the *semantic of the algorithm* is tried to improved by constructing such a semantic. The target is not longer to improve the existing distance relationship but enforcing a self-chosen distance metric. If such a metric could be declared and learned, more information would be available for assigning a label to an unknown instance. For example, we would not only know which are the nearest neighbors, but how far they are actually away. With this additional information it could be possible to defer how accurate is the predicted value and, in a classification problem, maybe even if the unknown sample is an actual instance of a class or if it maybe belongs to a previously unknown class.

These considerations lead to the question how such a metric could be found and declared. A property that the new metric should certainly posses is that it should preserve the semantic relationship between the samples according to the target value, i. e. the metric should represent a distance measurement on the target value that the user thinks is informative. The following proposition will proof that this is possible in a canonical way, if the user provides us with a metric on the label space:

Proposition and definition 2. *Let $X = \{x_1, \ldots, x_m\} \subset \mathbb{R}^n$ be an input-space, $\mathscr{L} \neq \emptyset$ a label-space and $\varphi : X \to \mathscr{L}$ its label mapping. Let further $d_{\mathscr{L}} : \mathscr{L} \times \mathscr{L} \to \mathbb{R}_{\geq 0}$ be a (pseudo-) metric on $\mathscr{L}$.*

Then $d_{\varphi, d_{\mathscr{L}}} : X \times X \to \mathbb{R}_{\geq 0}$ declares a pseudo-metric on $\mathscr{X}$ via

$$d_{\varphi, d_{\mathscr{L}}}(x_i, x_j) := d_{\mathscr{L}}(\varphi(x_i), \varphi(x_j))$$

We call $d_{\varphi, d_{\mathscr{L}}}$ the label-induced metric *or simply* induced metric *and write d where it's clear from the context.*

Proof. We state that (2.14c) is not required and (2.14a) is clear. We only need to show (2.14b): Let $x, y, z \in X$. Then

$$\begin{aligned} d(x,z) &= d_{\mathscr{L}}(\varphi(x), \varphi(z)) \\ &\leq d_{\mathscr{L}}(\varphi(x), \varphi(y)) + d_{\mathscr{L}}(\varphi(y), \varphi(z)) \\ &= d(x,y) + d(y,z) \end{aligned}$$ □

The fact that the induced metric is only a pseudo-metric is a blessing rather than a problem in this case. Not-identical samples that share the same label should in fact not be treated as two separate instances but recognized as the same value in manners of prediction.

The next step in this problem setting is to chose an appropriate metric. The next section will present some different metrics on different learning settings.

3.1.2. Metrics on the label set $\mathscr{L}$

Luckily there is a trivial metric which can be applied to any set $S \neq \emptyset$. It is defined as follows:

$$d_S : S \times S \to \mathbb{R}_{\geq 0} \quad d(x,y) = \begin{cases} 0 & \text{if } x = y \\ 1 & \text{if } x \neq y \end{cases} \tag{3.1}$$

This metric is called the *discrete metric*. In a classifier setting, this metric induces a 1-against-the-rest classifying on the input-space. Additionally, this metric also declares the same similar and dissimilar pairs like the information previous research uses via

$$\mathscr{S} = \{(x_i, x_j) | d_S(x_i, x_j) = 0\} \qquad \mathscr{D} = \{(x_i, x_j) | d_S(x_i, x_j) = 1\} \tag{3.2}$$

This thesis will specifically target regression problems which have not been target with prior Distance Metric Learning algorithms. As a starting point, one can use Euclidean Distance on $\mathbb{R}$:

$$d_{ED}(x,y) := |x - y| \quad \forall x, y \in \mathbb{R} \tag{3.3}$$

One can construct further metrics out of existing metrics by applying monotonically increasing functions to a given metric. Thus, given an existing metric d,

$$d_1(x,y) = \alpha d(x,y) + \beta \bar{d}(x,y) \quad \forall \alpha, \beta > 0 \tag{3.4a}$$
$$d_2(x,y) = d(x,y)^{\gamma} \quad \forall \gamma > 0 \tag{3.4b}$$
$$d_3(x,y) = \exp(d(x,y)) - 1 \tag{3.4c}$$
$$d_4(x,y) = \ln(d(x,y) + 1) \tag{3.4d}$$
$$d_5(x,y) = \min(d(x,y), 1) \tag{3.4e}$$

are all valid metrics. One can also *blend* several metrics $d_1, \ldots, d_l$ into one via

$$d_{\text{blended}}(x,y) := \sum_{i=1}^{l} \lambda_i d_i(x,y) \quad \text{where } \lambda_i \geq 0, \sum_{i=1}^{l} \lambda_i = 1 \tag{3.5}$$

If d is a metric on $\mathbb{R}$, then

$$d_B(x,y) := \frac{d(x,y)}{1 + d(x,y)} \tag{3.6}$$

is also a metric on $\mathbb{R}$. Since $d_B(\mathbb{R}, \mathbb{R}) = [0, 1)$, I call d_B the *bounded metric*.

A point of investigation of this special research thesis will be which metrics on $\mathbb{R}$ can be learned better or worse and which lead to better results for regression problems.

Next I want to discuss the possibilities of the KNN-algorithm for a QSAR problem setting. In QSAR problems, the input-space consists of numerical representations for chemical structures such as proteins. Typical target values are chemical activity with other chemical substances. The QSAR approach assumes that these target values can be expressed as a function of the chemical structure. This suggests the usage of Machine Learning algorithms to predict unknown target values.

Scientists are often interested in not only one chemical activity for each structure but on different activities on different substances. This task could be interpreted as a manifold regression task. However, this approach does not meet the goal of the task completely. By learning each activity on its own, the method does not take into account the pairwise relative structure of the problem. Moreover, results cannot be interpreted in a comprehensive view, since the learning task broke up the explicitly underlying connection.

In this case, a regular KNN approach is much more feasible, since a scientist can compare results and draw better conclusions. But again, as in the problem-setting before, the available data is not necessarily in the format, that simple Euclidean Distance is an appropriate choice. By declaring a meaningful relative distance measurement on $\mathbb{R}^n$, it might be possible to improve KNN prediction and therefore the overall results. Additionally, from the result which samples actually are the k nearest neighbors according to their chemical activity, scientists might also uncover previously unknown relationships among the input samples and gain a even further understanding of the problem at hand.

Therefore, metrics on $\mathbb{R}^n$ are needed, which help an user of Label Metric Learning to draw meaningful conclusions out of the distance predictions. These are not necessarily algebraically motivated metrics, but rather analytically motivated ones.

$$d(x,y) = \sqrt{|x_1 - y_1|^2 + \cdots + |x_n - y_n|^2} = (|x_1 - y_1|^2 + \cdots + |x_n - y_n|^2)^{\frac{1}{2}} \tag{3.7}$$

Analytical results show, that this metric can be generalized to the so-called *p-norm*:

$$d_p(x,y) := (\sum_{i=1}^{n} |x_i - y_i|^p)^{\frac{1}{p}} \tag{3.8}$$

One can proof, that these metrics converge against the so-called *maximum-norm* as p goes to infinity:

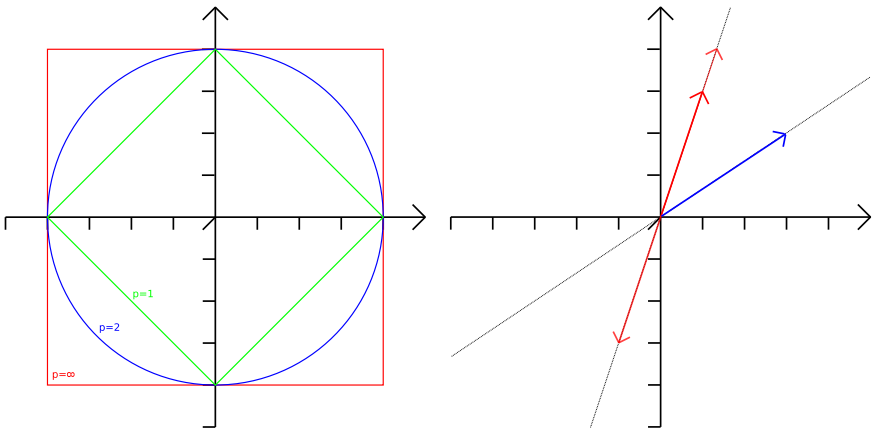
$$d_{max}(x,y) := \max_{i=1}^{n} |x_i - y_i| = \lim_{p \to \infty} (\sum_{i=1}^{n} |x_i - y_i|^p)^{\frac{1}{p}} \tag{3.9}$$

All these metrics have the desirable property, that they're also norms on $\mathbb{R}^n$. This means, they also fulfill the following property:

$$||\lambda(x-y)||_p = |\lambda| \cdot ||x-y||_p \tag{3.10}$$

which means, that, if the distance between two given points is λ-times as big, the assigned distance by the metric will also be λ-times as big.

Figure 3.1(a) shows the unit circle as defined by three different metrics. One can note, that the closer p is to 1 all components are equally weighted to compute the distance, whereas the closer p is to ∞, the absolute bigger values get more important. Therefore, a scientist can chose an appropriate metric according to her interest in the activity. If she is only interested on which



(a) Different p-norms (b) Illustration of the SNCF metric

Figure 3.1.: Illustration of different metrics on $\mathbb{R}^n$

Figure 3.1(a): The unit circle as defined by different p-norms. For $p = 1$ it is defined by the green square and every component is weighted equally important. For $p = 2$ an actual circle appears. As $p \to \infty$ only the biggest component is important and the unit *circle* is extended towards a big square.
Figure 3.1(b): Illustration of the SNCF metric. If two vectors are linear dependent, i.e. lie on the same line, then the distance is measured directly. If they do not lie on the same line, the distance between them is measured as the sum of the distances to the origin. Here: the bright red vector is linear dependent with the light red vectors, but not with the blue vector.

protein the enzyme reacts the most, she should chose a rather high value for p, whereas she is interested in all values, she should chose a value for p close to one.

Another important metric for our problem at hand might be the *SNCF-metric*. Given some metric d on $\mathbb{R}^n$, it is defined as follows:

$$d_{SNCF}(x,y) = \begin{cases} d(x,y) & \text{if } x = \alpha y \text{ for a } \alpha \in \mathbb{R} \\ d(x,0) + d(0,y) & \text{else, where } 0 = (0,\ldots,0)^T \end{cases} \tag{3.11}$$

This implies that this metric assigns smaller distances to vectors which share the same relative relationships between their components. Applied to our problem, this metric would favor samples that share the same relative biological reactions, specially the same ranking. This might be useful to the scientist, which is not so much interest in the actual amount of reaction, but only in their comparative relationships. This information might be useful e.g. in drug design.

Although the SNCF-metric seems very appropriate to our problem at hand, it also suffers from an severe drawback. It is highly doubtful that the SNCF-metric can be expressed as a linear or even as an easy non-linear function over the input space. Therefore, more distance functions which specifically consider relative comparison must be sought.

If the absolute values of the problem setting are far less important than the relative values, it might be sensible to ignore the absolute values completely and only measure relationship properties. One way to do so is by taking the *angle* between two samples as distance measurement.

$$d(x,y) = \cos^{-1}\left(\frac{\langle x,y\rangle}{||x||_2||y||_2}\right) \tag{3.12}$$

where $\cos^{-1}$ is the inverted cosine function. Although this distance measurement is not a metric on $\mathbb{R}^n$, it is a valid pseudo-metric and therefore also induces a valid pseudo-metric on the input-space.

Taking this approach even one step further, one could merely be interested in the pairwise *rank correlation*. To measure this kind of similarity, I'm going to apply the following formula:

$$d(x,y) = 1 - \frac{\rho(x,y)+1}{2} \tag{3.13}$$

where

$$\rho(x,y) = \frac{n(\sum x_i y_i) - (\sum x_i)(\sum y_i)}{\sqrt{n(\sum x_i^2) - (\sum x_i)^2}\sqrt{n(\sum y_i^2) - (\sum y_i)^2}} \tag{3.14}$$

describes the *Spearman's rank correlation coefficient*, also called *Spearman's* ρ. This function returns values in the range $[-1,1]$ with 1 if the rankings are identical and -1 if they are perfectly inverted to each other. Formula (3.13) scales (3.14) to $[0,1]$ and subtracts the result from 1, so that $d(x,y) = 0$, if the rankings are identical.

However, one must also consider that (3.14) also returns 1 if for any argument of ρ every component has the same value. This could bias the outcome of a KNN search, because any combination of target values would be considered practically identical with $0 = (0,\ldots,0)$. In the QSAR problem setting this might be a protein which does not react with any of the target substances. If such an sample would be in the input-space, this could influence all results negatively. Therefore an user would have to take special precautions.

3.1.3. Learning a chosen metric

Another very important question in this context is the way how this induced metric should be learned on the training set. Let p denote the approximating model. The function to be optimized by the learning algorithm is then:

$$\min \sum_{i,j=1}^{m} |d(x_i,x_j) - p(x_i,x_j)| \tag{3.15}$$

Compared to many other optimizing functions presented in this thesis like (2.7) and (2.26) it can be noted that this function is not constrained. There are two reasons for this fact:

1. Unlike the other presented optimization problems, no concrete model is given for the function p. There are no constraints on the function, because it is not stated how this function is build up at all.

2. Although an obvious model might be a linear transformation as in 2.2, this does not need to be the case. The induced label metric is not necessarily approximatable through a linear expression, specially if the chosen metric is mainly motivated rather analytically then algebraically.

Therefore, this approach really represents a paradigm shift as stated in section 3.1.1. As long there is no indication given by the chosen metric or the data set, it is probably better not to assume a specific model that might not be suitable. Without prior knowledge it is therefore a wise decision to use a general purpose solver. An additional supporting argument for this approach lies in the reduced research and development work. The entire existing knowledge of current algorithms can be used. In this research thesis, a Support Vector Regression algorithm as presented in section 2.1.1 was applied as general purpose solver.

But there is also a downside of this procedure: current general purpose solving algorithms expect and learn functions consuming one argument. The distance function however is a function consuming two arguments. We therefore have to transform our training-set in a manner

that existing algorithms can deal with the presented problem as usual, i.e. taking one argument and predicting one target value. To clarify further discussion, I define the following term:

Definition 3. Let $\mathscr{X} \times \mathscr{L}$ be an input-space and $d : \mathscr{L} \to \mathbb{R}_{\geq 0}$ be a metric on $\mathscr{L}$. We call

$$\mathscr{X}_2 \times d(\mathscr{L}_2) := \{(x \cdot y, d(x,y)) | x, y \in \mathscr{X}\}$$

the *second-order set* or *second-order space* and elements $(x \cdot y, d(x,y))$ in the second-order set we call *second-order samples*, where $x \cdot y$ is some combination of x and y. Generally we will call this interpretation of the data as *second-order view* and the original interpretation the *first-order view*.

Some possible combinations of x and y are

Normal combination $x \cdot y := (x_1, \ldots, x_n, y_1, \ldots, y_n) \in \mathbb{R}^{2n}$

Small combination $x \cdot y := ((x_1 - y_1)^2, \ldots, (x_n - y_n)^2) \in \mathbb{R}^n$

Dotproduct combination $x \cdot y := (x_1 y_1, \ldots, x_n y_n) \in \mathbb{R}^n$

Matrix combination $x \cdot y := (x_1 y_1, x_1 y_2, \ldots, x_1 y_n, x_2 y_1, \ldots, x_n y_n) \in \mathbb{R}^{n^2}$

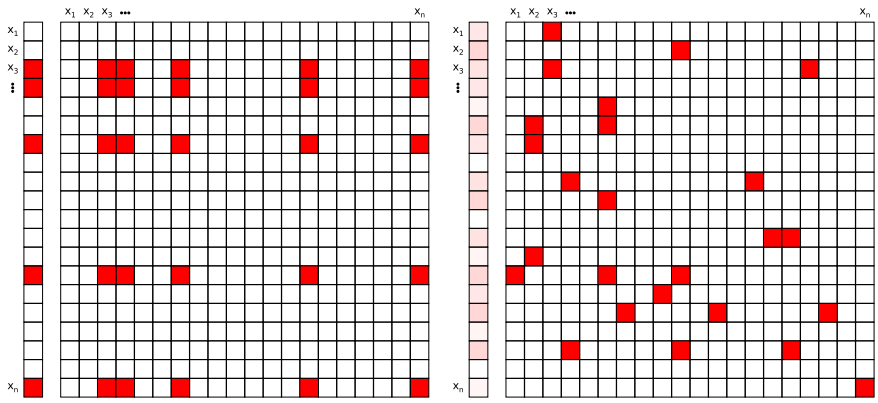
There are different advantages and disadvantages of each description. The small and the dot-product combination do not suffer from the curse of dimensionality, however much information might get lost through the transformation. The matrix combination offers high dimensionality, thus giving a learning algorithm better opportunity to tweak parameters, but challenges through a high complexity. The normal combination offers the best original information. However, this combination might get biased through the arrangement of the data within the vector, i. e. with this second-order view $p(x,y) \neq p(y,x)$ could be predicted which is not expected from a valid metric.

Another critical point of this approach is the size of the input space . Let $m := |\mathscr{X}|$ be the size of the original input setting. The size of the second-order space then is m^2. For very small m this is actually an advantage, because we enlarge the number of training-data immensely. But most of the time this will be a serious problem, because the number of training-data will blow up to a size that most Machine Learning Algorithm won't handle or need lots of time. We therefore need to reduce our second-order training-set to a appropriate size. But which second-order samples are to be chosen as training-data?

There are several possibilities, when and how to reduce the size of the training-set. One can either reduce the training-set *before* transforming them to second-order samples or *after* transforming them to second-order samples. Both techniques have advantages and disadvantages.

The advantage of reducing the set-size before transforming is that you have control which samples and labels are chosen from the original set. Therefore, you can chose a subset of the original training-data, that shares its characteristics. But since the pre-transformed set has to be very small, you might only have the possibility to chose one instance out of each label-value or maybe not even each label-value, if $|\mathscr{L}| \approx |\mathscr{X}|$. Additionally there might be a lot of instance combinations for the same label-value, which all you might have to leave out, which may lead to poor generalization on the second-order set.

On the other hand, reducing the training-set after transforming it to second-order view, you can make sure that your subset shares the same characteristics of the full second-order set, but you don't longer have the control over the instance/label pairs. This might lead to an implicitly strange selection of the first-order set.



(a) Reduction before transforming (b) Reduction after transforming

Figure 3.2.: Illustration of the first- (bar) and second-order (square) set and various reduction techniques.

Illustration of transforming the original input space (on line of samples on the left) to second-order view (big square). Figure 3.2(a) illustrates the reduction before transforming to second-order view. Samples are chosen randomly in the first-order set, but they form patterns on the second-order set and do not capture the distribution of the second-order set well.

Figure 3.2(b) shows a reduction after transforming to second-order view. The distribution of the second-order set is better approximated through the random selection, however an individual sample is not selected completely in first-order view.

Summing it up, one could say that reducing *before* transforming leads to realistic image of the first-order set but leads to overfitting and poor generalization on the second-order set, whereas reducing *after* transforming leads to a good generalization of the second-order training-set with poor generalization on the first-order set. We will try to find a answer to question whether to reduce before or after transforming empirically.

Although we're actually learning a distance measure between samples of the input-space, our real goal is to assign a label to an unknown instance $\tilde{x}$. We therefore wanted to assign $\tilde{x}$ a combination of the labels of its k nearest neighbors (see p. 7). But since we only learned a metric on a subset of $\mathscr{X}$ and we don't know, whether this metric will be a good generalization on the complete set, we might consider calculating only the k nearest neighbors of the subset. I call this subset the *assigning set*. We will see, which role the assigning set plays for the overall outcome.

The above mentioned considerations all addressed the issue of learning the distance function between two samples. Before presenting the new Distance Metric Learning framework another important component in this algorithm has to be discussed: the prediction of the actual label. Up to this point of the algorithm only the *distances* to known samples are computed for an unknown input $\tilde{x}$. Although this is in fact a very useful information, the main purpose still lies in automated prediction of the unknown *label*. The question is how the distances could be used to assign a label. A straightforward way to do so is to apply a KNN algorithm as presented in section 2.1.2. But since the distance predictions actually gives absolute values it might also be possible not only to estimate the target value but also to calculate the precise value. To see this assertion, let's assume n target values that could be represented as vector $\in \mathbb{R}^n$. A pair of a known target vector and a distance creates a hyperball with the predicted distance as radius where the sought target vector could be located. If the predicted distances are correct and the intersection of $n+1$ hyperballs is calculated there remains exactly one possible value solution,

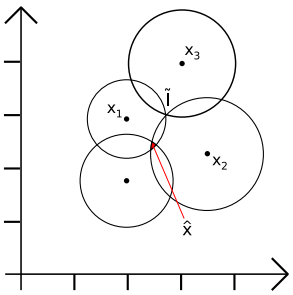


Figure 3.3.: Illustration of an explicit label calculation with two target values.
The figure illustrates the algorithm for calculating two target values out of three known target values and the predicted distances. In this example the perfect predicted distances between the unknown sample $\tilde{x}$ and the known instances x_1, x_2 and x_3 suffice to calculate $\tilde{l}$. However, for sample $\hat{x}$ the distance predictions do not match perfectly and the label cannot be calculated uniquely.

Algorithm 1 The *Label Metric Learning* algorithm

Input: Training set $\mathcal{X} \times \mathcal{L}$, test sample $\tilde{x}$,
metric d, regression algorithm p, label assignment strategy s
Output: Label prediction $\tilde{l}$

1: $\mathcal{X}_2 \times d(\mathcal{L}_2) \leftarrow createSecondOrderSet(\mathcal{X} \times \mathcal{L}, d)$
2: $p \leftarrow trainModel(\mathcal{X}_2 \times d(\mathcal{L}_2))$
3: $i \leftarrow 0, distances[(\mathcal{X} \times \mathcal{L}).size]$
4: **for all** $y \in \mathcal{X}$ **do**
5: $\quad distances[i++] \leftarrow predict(p, \tilde{x} \cdot y)$
6: **end for**
7: $\tilde{l} \leftarrow assign(s, distances, \mathcal{L})$
8: **return** $\tilde{l}$

thus the unknown target value could be predicted *exactly*. Figure 3.3 illustrates this method.

Alternatively, if a regular KNN prediction is regarded, it is important to note the new role of the parameter k: since we have *absolut* distance values, there is actually no need to estimate the local target value distribution according to coordinate values, because we know the target value distribution. We therefore can draw other conclusions of the predicted distances. For example, if the target value is continuous and the predicted distances grow as the nearest target value neighbors get less, the unknown label is probably even bigger than the closest neighbor. Therefore it would be unreasonable to predict an average of the nearest neighbors' targets. Or if the average predicted distance or the standard deviation for the predicted distances is rather high, it is more likely that the assigned label will be more error prone. Additionally, with correct distance values according to the target value, an increased value for k would not smooth the prediction but draw the prediction away from the nearest neighbors more to the more general labels.

However, the possibility of this (sub-)algorithms and quality predictions depends on the used metric and the quality of the distance predictions. For the moment it is only important that such alternate strategies exist and that they have to be considered on applying the new algorithm. I call this component the *Label Assignment* component.

Concluding, algorithm 1 presents the newly proposed algorithm. The following parts can be identified:

1. The training of the distance model (lines 1 and 2).

2. The prediction of the distances (lines 3 to 6).

3. The application of the distances to assign a label (lines 7 and 8).

Note that if more than one unknown sample is to be predicted only the lines 3 to 8 have to be repeated. Additionally note that assigned labels could also be used to relearn and possibly improve the distance model. However this is not necessary and not recommended if the quality of the label assignment is not guaranteed to be good or if adding the information to the learning algorithm makes it necessary to recompute the whole optimization (offline learning).

Finally note that the presented algorithm also suffers form important drawbacks: the user has to chose an appropriate metric, a learning algorithm he thinks can generalize the metric well and a fit label assignment strategy. One could argue that this actually enlarges the number of needed parameters rather then reducing them. This thesis will show in the next chapter that very basic choices already lead to reasonable good label predictions.

3.2. Implementation and Design

To test this new approach to Distance Metric Learning, a *Java* framework called *JDistLearner* was developed. The architecture of the framework was mainly motivated by two major issues:

- The Label Metric Learning algorithm is capable of any label format as long as valid metric can be constructed on the label space. The framework should support this advantage.

- As mentioned in 3.1.3, the main bottle-neck of Label Metric Learning will rather be memory than computation capacity. Although computation time might get an issue, memory usage is the foremost problem. Therefore, design decisions should always favor effective memory usage over computational complexity.

Considering the first issue, the label space is represented using Java *Generics*. Generics make it possible to abstract over types. Declaring a class, variable or parameter generic tells the compiler that the type of a reference is not necessarily known at compile time, but can be restricted over different parts of code. For example, to create a valid second-order set the type of the label does not have to be known, but the used distance metric must be of the same type as the label type. Therefore, the usage of generics enforces the training-set and the chosen distance metric to be parameterized over the same type. A good introduction to Java Generics is given in [Bra04].

To solve the memory problem at hand, the design was inspired by the *Model-View-Controller* pattern. Applied to the *JDistLearner* framework, this means, that the training-set and the assigning-sets are not saved explicitly, but each input sample is only saved once. Set and instance manipulating code only safes the reference to a given object. Any data that will be transformed into some other representation will be created on-the-fly for each query. However, if enough memory is available, it is easy to safe the transformed data.

Additionally, the training-set and the label-set are saved separately. Rather than storing the label for each instance, both the training-set and the label-set are enumerated, making it possible to represent the label mapping as an array of `int`s, i.e. for the i-th input sample the number of the according label is stored in the i-th array position. For training-sets in which many instances share the same target value and a label consumes more memory space then an `int`, this reduces memory usage even more. As a nice side-effect it is possible to transform the label-space on its own and much more efficiently.

The framework also enables sparse data representation. This means that zero-values of input-vectors are not stored explicitly but are represented as a missing value.

Further design considerations included that the framework should be easy extensible and yet robust. To achieve these goals, all major components are represented with interfaces, making it easy to exchange parts according to own judgement and purposes yet restricting unauthorized data manipulation.

3.3. Summary

Theoretically it should be possible to learn a (pseudo-)distance metric d on an input-space $\mathscr{X}$ by declaring a distance metric $d_{\mathscr{L}}$ on the label-set $\mathscr{L}$ and inducing it via $d(x,y) = d_{\mathscr{L}}(\varphi(x), \varphi(y))$. Even in the absence of special purpose solvers we have several ways to gain a good generalization and prediction.

Our approach has several advantages over the previous approaches:

1. Instead of predicting the label $\tilde{l}$ of an unknown sample $\tilde{x}$ *directly*, we are predicting only the underlying structures in comparison to the other samples and then assign $\tilde{l}$ to $\tilde{x}$ with this information. If our assigning strategy is correct, we will be less error-prone because of we have more chances to predict correct values for $d(\tilde{x},x)$. Therefore we should rather overestimate $d(\tilde{x},x)$ then underestimate it. (Note that this advantage is shared by all Distance Metric Learning approaches.)

2. If our learned model is even good enough, we're also given information about *how* good the prediction will be. If the distances to the k nearest neighbors is very high or if we've got high variance between the predicted distances, we can assume that the prediction of this sample has failed and a learning-supervisor can try to classify it manually.

3. And finally, a human user can request more information, *why* the algorithm chose a certain label. The algorithm is not like a "magic box" that simply coughs up a miracle result, but we have numerous distance predictions which should give us a clearer picture of the unknown instance in comparison to the known instances. (Of course, we still have a "magic box" one step further, but we actually could take this problem one step further again by declaring a label metric on $d(\mathscr{L}_2)$, but this really doesn't solve anything.)

4. Experiments and results

This chapter describes the background of the experiment settings and evaluation criteria and lists the experiments that were conducted.

4.1. Evaluation settings

For the analysis of the *Label Metric Learning* framework, I used four different regression data-sets and one QSAR problem data-set.

The first regression data-set is referred to as *Abalone*. It is available from the *UCI Machine Learning Repository* [AN07]. It represents different physical measurements of abalones, e.g. size and weight. The goal is to predict the age of an abalone. It was originally created for a non-machine-learning study [NST+94]. The data-set contains of 4177 samples, each with seven training attributes and one target attribute. Six of the seven training attributes are continuous, one is nominal represented as integer $\in \{1,2,3\}$. The target attribute is also an integer. Figure 4.1(a) shows the distribution of the target attribute. Since there are many samples for each target value, this data-set is also used as classification benchmark data-set sometimes.

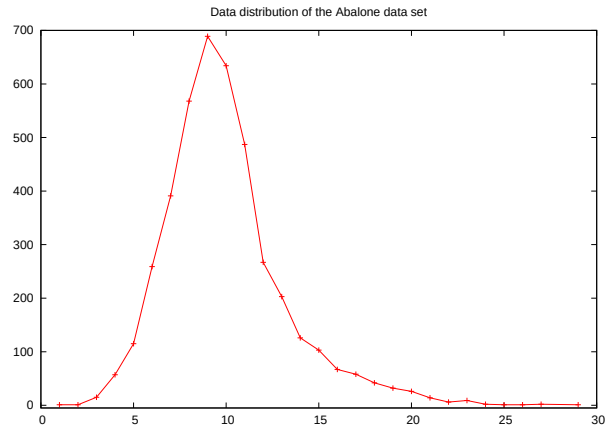
The second regression data-set is referred to as *Housing*. It also is available from [AN07]. It measures housing values in suburbs of Boston in dependency of different quality indicators. The data-set contains of 506 samples with 13 training attributes. Twelve attributes are continuous, one is categorical, represented as integer $\in \{0,1\}$. Figure 4.1(b) shows the distribution of the target-attribute.

Another data-set that was used to benchmark the regression performance was the so-called *MG* data-set. It was specifically created as benchmark data-set for regression problems by [FL01] and consists of 1385 samples with six training features and one target feature. The data models the **M**ackey-**G**lass equation given by

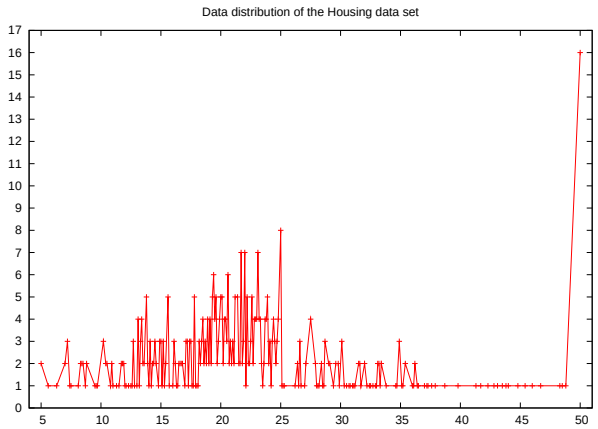
$$\frac{dx}{dt} = \frac{ax(t-\tau)}{1+x^{10}(t-\tau)} - bx(t) \tag{4.1}$$

The authors used $\tau = 17, a = 0.2, b = 0.1$ and $\Delta t = 1$. The problem consists of six inputs with delay value of six and forecasts made 60 time steps into the future. The distribution of the target value is shown in figure 4.1(c).

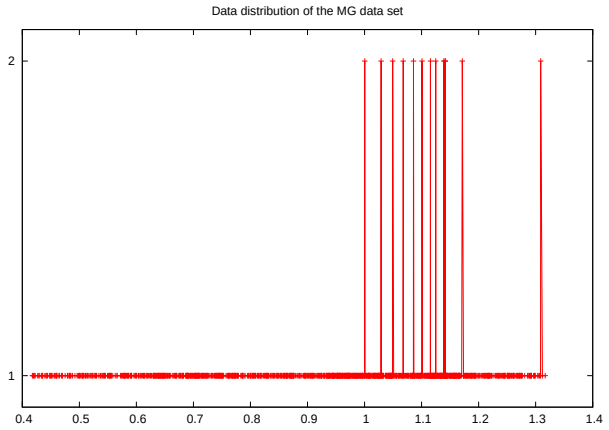
The final regression benchmark data-set is another data-set from [AN07] here referred to as *MPG*. The data represents city-cycle fuel consumption in **m**iles **p**er **g**allon. The target value is modeled in dependency of car-related attributes such as cylinders, horsepower, weight, acceleration and others. The data-set is composed of 392 samples with seven training features. Four training values are continuous, 3 are multi-valued discrete. The target value distribution for this data-set is shown in figure 4.1(d).



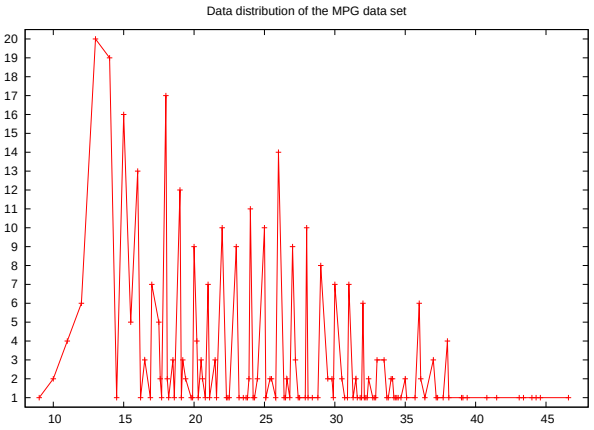
(a) Abalone data set



(b) Housing data set



(c) MG data set



(d) MPG data set

Figure 4.1.: Distributions of the regression data sets

All regression data-sets where downloaded from [CL01] where they were available with each training attribute already linearly scaled to $[0,1]$. The data was not preprocessed otherwise.

To evaluate the results of a experiment configuration I used two standardized evaluation criteria. The *Mean Squared Error (MSE)* measures the average of the squared error. Given two distributions X and Y, it is computed as follows:

$$MSE(X,Y) = \frac{1}{k}\sum_{i=1}^{k}(x_i - y_i)^2 \tag{4.2}$$

where k is the sample size. The other evaluation criteria is the correlation coefficient, defined by

$$Kor(X,Y) = \frac{\sum_{i=1}^{k}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{k}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{k}(y_i - \bar{x})}} \tag{4.3}$$

where $\bar{x} = \frac{1}{k}\sum_{i=1}^{k} x_i$ is the empirical mean value of distribution X ($\bar{y}$ analogous). It can be proven that the value of Kor for any two distribution lies within $[-1,1]$ with $Kor(X,Y) \in \{\pm 1\}$ if and only if $Y = \alpha X + \beta$ for some value of α and β. Since such a linear function would preserve to ranking of the samples for the kNN algorithm, (4.3) is slightly more important than (4.2) when evaluating the quality of the trained models. However, if the variance of the predicted distance values is rather small, (4.3) will also yield a high result, suggesting a successful model. But a small variance in the predicted distances will yield in bad results for the kNN algorithm. Therefore it is necessary to compare the correlation coefficient with the MSE.

When applied to a cross-validation, the correlation coefficient will be stated as Q^2. MSE and correlation coefficient were computed both for the distance predictions (the trained model) and the computed value of the kNN algorithm as defined in (2.13).

In the QSAR-problem setting the goal is to predict biological activity on three different serine endopeptidases, namely Thrombin, Trypsin and Factor Xa. Thrombin and Factor Xa play an important role in blood clotting, whereas Trypsin is part of the human digestive system. In medicine and industry, coagulation proteins are pharmaceutical target proteins. However, most enzymes that react with Thrombin and Factor Xa also react with Trypsin, causing unwanted side-effects. Therefore it is important to know the specific selectivity profile for a given enzyme.

The data-set that was available for benchmarking consists of 72 representations of enzymatic proteins. The numerical representation is described in [TC00]. Our data available consisted of a full description with 1465 features. Over all enzymes, 14 features included missing values, which were excluded from all representations. The size of the remaining 1451 features was not reduced otherwise. All training attributes were automatically linearly scaled to $[0,1]$ by the IO-interface.

Evaluating the QSAR-problem setting is a bit more difficult, since there are not standardized multi-dimensional evaluation criteria. This thesis uses the following to criteria.

To evaluate the error of the label assignment, a generalized mean squared error was computed as defined by

$$MSE(X,Y) = \frac{1}{k}\sum_{i=1}^{k}||x_i - y_i||_2^2 \tag{4.4}$$

where n is the dimension of the target vector, which was three in my benchmark data-set. To evaluate the quality of the pairwise relative relationships, the average Spearman rank correlation

coefficient was computed by

$$\bar{\rho}(X,Y) = \frac{1}{k}\sum_{i=1}^{k}\rho(x_i,y_i) \tag{4.5}$$

where ρ is defined as in (3.14).

4.2. Regression experiments

The difficulty in evaluating the performance of the *Label Metric Learning* algorithm lies in the big number of possible configuration settings. To further increase this difficulty, it is not necessary true that each configuration aspect can be optimized independent from all other aspects. However, it was assumed that independent optimization will improve the overall performance non the less. Therefore different test cycles were run to evaluate different parts of the overall configuration.

To test a possible configuration, a 10-fold cross-validation was run on each data set. Although this might not be a sufficient number of tests to indicate the true average performance, it still is a rather good approximation. However, it was not possible to run more tests due to computational complexity. My implementation used an interface to the *LibSVM* [CL01] ε-regression algorithm. In a first test cycle different kernels and parameters were tested. Best results were obtained with an rational quadratic kernel (2.9c) with parameters $\varepsilon = 10^{-3}$ and $C = 5$. These settings were kept throughout the rest of the test runs.

The next test cycle investigated the usage of different second-order views (see p. 20). For each cross-validation run the data was randomly reduced to the maximum capacity of *LibSVM* after transforming the training-set to second-order view. For each data set and second-order configuration two different metrics were applied:

1. Euclidean distance, linearly scaled to $[0,1]$ according to the maximum value of the label set (see (4.6))

2. Bounded metric with regular Euclidean distance as foundation (see (4.7))

The trained models were evaluated on the second-order test-set. Results are shown in table A.1.

Although there was no configuration that led to best results for each data-set and metric, the normal second-order view performed best in the most cases and always approximated the true best value very well. Therefore, in sufficient test cycles the normal second-order view was applied.

Since the *LibSVM* library was running out of memory very quickly, the following test cycle was concerned with the best reduction technique for the second-order set. Tests were run both with reduction before and after transformation to second-order view. The following three reduction algorithms were applied:

Random reduction [*no*] The training-set was reduced randomly to the size of *no* instances.

Equal reduction [*no*] Each label-value was equally often placed in the training-set. Remaining spaces were filled with the most frequent label-values.

Wide spread reduction [*no*] The training samples were sorted according to label value. Then, every $\frac{no}{|\mathscr{L}|}$th sample was selected.

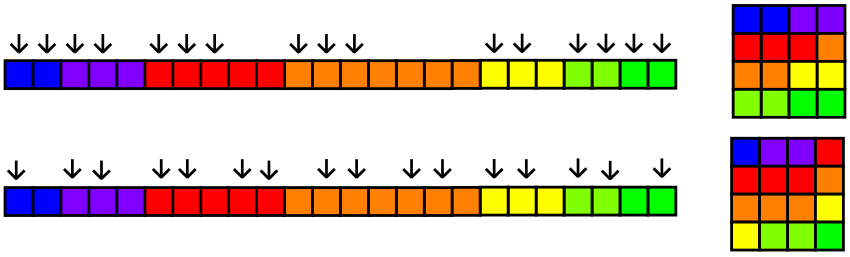


Figure 4.2.: Example for equal and wide spread reduction.
This figure illustrates the equal and wide spread reduction. The top figure shows the equal reduction technique. Every value (represented with according colors) is selected equally often, as long there are samples of each color left. Afterwards it is filled up with the most frequent (here: brown and red) values. The bottom figure shows the application of the wide spread reduction algorithm. The values are sorted according to label value. Then they are separated equally over the line, leading to a better estimation of the original distribution.

Figure 4.2 illustrates the latter two reduction heuristics. All other settings were as in the previous test setting. Tables A.2 and A.3 show the results of this test cycle.

The results show clearly, that reducing the training-set *after* transformation to second-order view outperforms reducing *before* transforming the samples. Additionally, the data suggests that performance of the model improves with the size of the training-set. Again no clear overall winner could be conducted, but Wide Spread Reduction and Random Reduction seem to perform very similar. This is not surprising, since both reductions approximate the true distribution of the second-order set quite good. In the following experiments, the Random Reduction after transformation was kept as configuration setting, because the Random Reduction was implemented with slightly less memory usage.

After that, the main test cycle was evaluated. In this test setting, different metrics were applied to all data sets. The following metrics were used :

1. Euclidean distance, linearly scaled to $[0,1]$ as in the first test cycle:

$$d_{ED}(x,y) = \frac{1}{\max d(\mathscr{L}_2)}|x-y| \tag{4.6}$$

2. Bounded metric:

$$d_B(x,y) = \frac{|x-y|}{1+|x-y|} \tag{4.7}$$

3. Bounded metric with (4.6) as basis metric

4. Bounded metric, linearly scaled to $[0,1]$:

$$d_{LSB}(x,y) = \frac{1}{\max d(\mathscr{L}_2)}\frac{|x-y|}{1+|x-y|} \tag{4.8}$$

5. Three different metrics with linear transformations of (4.6):

$$d_{LTE}(x,y) = \min(1, \alpha \cdot d_{ED}(x,y)) \quad \alpha \in \{1.25, 1.5, 1.75\} \tag{4.9}$$

6. Four different metrics that extract a root of (4.6):

$$d_{RED}(x,y) = \sqrt[\beta]{d_{ED}(x,y)} \quad \beta \in \{1.5, 2, 3, 4\} \tag{4.10}$$

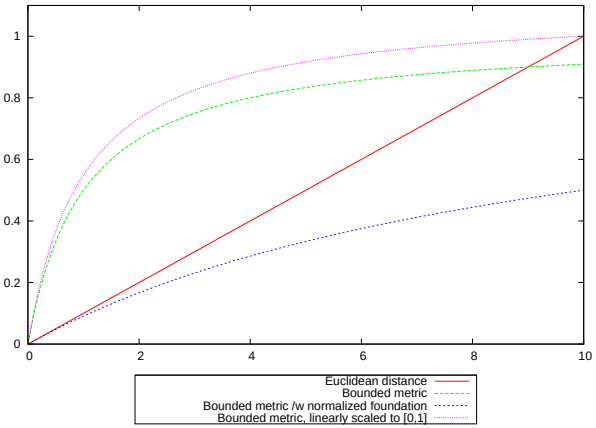
7. Three different metrics that raised (4.6) by some value:

$$d_{SED}(x,y) = d_{ED}(x,y)^{\gamma} \quad \gamma \in \{1.5, 2, 3\} \tag{4.11}$$

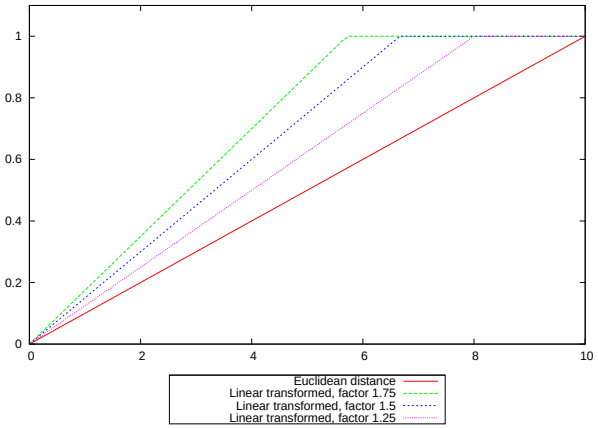
8. Finally, four different blends of (4.6) and (4.7):

$$d_{BEB}(x,y) = \lambda d_{ED}(x,y) + (1-\lambda) d_B(x,y) \quad \lambda \in \{0.2, 0.4, 0.6, 0.8\} \tag{4.12}$$

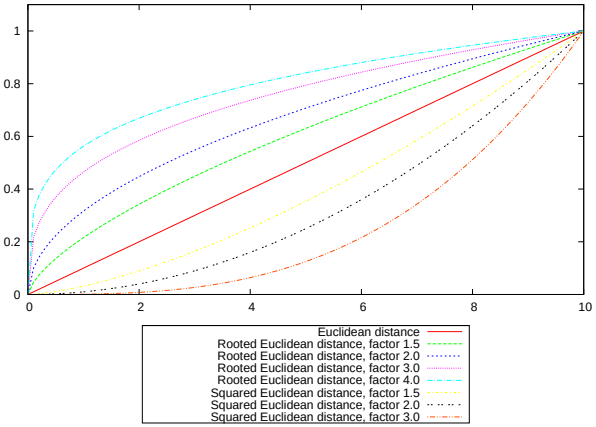
Figure 4.3 illustrate some of these metrices on an imaginary data set with a diameter of 10.



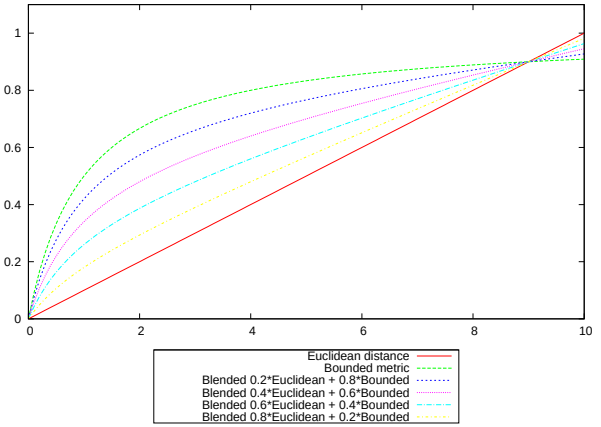
(a) Standard metrics



(b) Linear transformed metrics



(c) Exponentiated metrics



(d) Mixed metrics

Figure 4.3.: Graphs of different metric transformations

The results of the trained models are shown in table A.4. However, as previous test cycles had revealed, the quality of the kNN algorithm does not depend directly of the prediction quality. Therefore, all models were evaluated with a smooth 3NN prediction as in (2.13). These results are shown in table A.5.

It can be seen that the quality of the model improves with ratio of training set size and actual available second-order data. Further, the data suggests that some metrics can be learned essentially better than other metrics. This is surprising, because linear metrics were learned best, although the kernel that was used was non-linear. I did not find a explanation for this effect. The next possible observation concerns the relationship between the learned distance model and the label assignment. For each metric the quality of the label assignment improved with the quality of the distance model. However, this effect is not true if different metrics are compared to each other. Although some metrics lead to significantly worse prediction results, they produce much better 3NN predictions. A possible explanation can be derived from the gradient of the metric. Metrics with a high gradient for small distance values produced better results than other metrics. This is consistent with the functionality of the ε-SVR algorithm. By not penalting an error less than ε, the SVR algorithm blurs the well-ordering of the distances. This effect can be reduced by increasing the distance between two samples, which is equivalent to increase the gradient of the distance graph. However, if the gradient was increased to much, the worse prediction results would reduce the effect again. Therefore, the results suggest a trade-off between a steep distance function and the ability to generalize the function correctly.

Following the considerations of section 3.1.3, the next test configurations investigated whether the distance prediction quality and with it the label assignment quality could be improved if not all training data were compared to an input sample but only the data that were available after the training set reduction. However, it is much more difficult to decide which data should be considered in the first-order assigning set when the training set is reduced *after* transforming to second-order. This is not only a difficulty of implementation details, but also a technical one. If the training set is reduced in first-order view, it is absolutely clear which samples created the second-order set. But if the reduction occurred in second-order view and a sample x contributed only to one second-order sample $(x \cdot y, d(x,y))$, then how much did sample x actually contribute and more specifically was the contribution big enough that the generalization is expected to be positive influenced by it or not? Therefore this configuration was tested with a training-set reduction *before* transforming it to second-order.

The results can be seen in tables A.6 to A.9. As expected, it can be see that the quality of the distance predictions is significantly better on the reduced assigning set and so are the label predictions. However, compared to the results of the previous test cycle, the quality of the label predictions on the small data sets actually decreased. There are probably two factors to this effect:

1. The improvement of the distance prediction quality on the *before*-transformation-data set is not enough to compensate for the general better result on the *after*-transformation data set.

2. Additionally, the decreased number of comparison possibilities make the label assignment more sensitive to wrong distance predictions.

Both effects could be reduced with a bigger first-order set but with the available software and hardware the set size reached its limit.

The last test cycle analyzed the label assignment quality for different values for the parameter k. Tables A.10 and A.11 show the results. It can be seen that the quality of the distance

predictions are independent from specific values for k as expected. Additionally it can be seen that the label assignment quality improved in most cases with the size of k to the value of 7 but seems to degrade with $k = 10$. There are probably to forces that cause this effect:

1. The quality of the predicted distances is not very good. This disturbs the which samples are found as nearest neighbors in particular. Increasing the size of the neighborhood enlarges the chance of finding the actual nearest neighbors.

2. Wrong predicted distances could also be interpreted as random results. Therefore, the nearest neighbors will more likely be distributed like the original training set, favoring frequent labels. The label assignment will be biased towards frequent labels in the training set. But since cross-validation is used to generate the testing set, there is a high chance that the most frequent labels in the testing set and in the training set will be the same. That means that if all unknown instances are assigned with a frequent label, those instances having a frequent label will be assigned right in particular, thus increasing the label assignment quality.

The tables show the results for a reduced assigning set, although this effect could also be observed for a full assigning set.

Finally, standard techniques were applied to be able to compare the results with existing algorithms. Therefore, standard 3NN with smooth prediction and standard *LibSVM* ε-SVR with the same settings as above were applied. Table A.12 show the results.

4.3. QSAR experiments

Due to the small data set size, 5-fold cross-validation was applied to evaluate possible settings for the QSAR prediction task. Each run was repeated five times and results were averaged to compensate for the less number of folds. Again the interface to the *LibSVM* ε-regression algorithm was used. On this benchmark data set, the framework was able to handle the complete second-order set with a heap size of 8 GB. Early experiments with different parameters suggested best results with the same parameters as in the other regression tests.

In a first experiment setting, different p-norms (see (3.9)) were applied to see, which metrics could be learned best, but also to see, whether the characteristic of a metric will be represented in the results of a KNN application. Therefore, for each target attribute a smooth 3NN as in (2.13) was applied. For each value of p overall results were evaluated but also the correlation coefficient and mean squared error were calculated for each target attribute. Results are shown in table A.13. Note that unlike to section 4.2 the metrics were not scaled to the range $[0,1]$. Thus, the different values of the MSE are not directly comparable.

The desired outcome that different values for p would be represented in the label assignment on the different targets did not occur. Unfortunately the question if this is a general result for this algorithm can not be answered securely on this data basis. The investigated data source is to small.

In a further test run, alternative metrics which reflect the relative rank relationships of the target attributes better were tested. The following metrics were applied:

1. The *SNCF-metric*, as described in (3.11).

2. The rank pseudo-metric, as described in (3.13).

3. A slight variation of the rank pseudo-metric with higher gradient:

$$d_{\rho,3}(x,y) = 1 - \left(\frac{\rho(x,y)+1}{2}\right)^3 \tag{4.13}$$

4. The angle pseudo-metric (3.12) multiplied by factors 10 and 100 respectively:

$$d_{\alpha,\lambda}(x,y) = \lambda \cdot d_\alpha(x,y) \quad \lambda \in \{10,100\} \tag{4.14}$$

Table A.14 show the results of this test setting. As in the above experiments, the MSE values cannot be compared directly. The experiments demonstrate that the rank pseudo-metric and the angle pseudo-metric could be learned significantly worse. This outcome was expected because these metrics are in formula the most analytical ones in the list whereas the other metrics are a bit more similar to the metrics induced by positive semi-definite matrices. The results of the SNCF metric might be surprising at first glance but there exists a simple explanation. None of the labels in the data set are actually linear dependent. Therefore the SNCF metric is reduced to the simpler case

$$d_{\bar{\text{SNFC}}}(x,y) = d(x,0) + d(0,y) \tag{4.15}$$

which then can be computed efficiently. The samples with the smallest distance then were the samples with the smallest overall values and all samples were assigned the same label.

The results of the label assignment showed that the learned metrics produced higher MSE values but comparable ρ values which were expected to be higher. There are two possible explanations for this outcome. The straightforward explanation is the fact that the considered metrics were predicted significantly worse thus simply leading to wrong label assignment. The other explanation is that the labels of the data set are distributed with a strong bias. Almost all labels of the data set have the biggest value in the third component, the second most value in the second component and the smallest value in the first component. Predicting any label that does not follow this distribution is considerably harder thus much more error prone. Therefore the pretty constant value as ρ result could also be explained solely through the limitations of the *k*NN algorithm and the label distribution. As in the previous experiment there are no clear answers to be given.

In a next experiment it should be tested whether a combination of absolute and relative comparison improve the label assignment. Therefore, the best two metrics of the first test cycle were combined with the best two metrics of the second test cycle, each blended with a factor of 0.5 per metric. The results can be seen in table A.15.

The outcome shows that the combined metrics could be learned as good as the sole p-norms previously and the label assignment improved slightly. A possible interpretation is that the improved distance prediction effectively improved the relative prediction and therefore also leading to improved overall predictions but it remains uncertain how much this effect played a role.

Finally, different second-order views and values for k have been tested. Table A.16 shows the results. The matrix second-order view could not be applied to this data set, because the resulting vectors were to big. From the applied second-order views, metrics could be best generalized with the dot-product view and worst with the small view. Comparing to the results from the analogous tests in the regression setting in table A.1 it can be seen that the quality of the distance model does not depend on the size of the second-order vector. However, a more detailed analysis is difficult, because many descriptors do not represent continuous values and therefore a direct mathematical interpretation is not possible.

Regarding the number of nearest neighbors that were considered for assigning a label, similar observations as in the regression setting could be observed. With an increased value for k the quality of the assigned label improved. For a possible explanation see again page 33.

To set these evaluations in some context, the original KNN algorithm was applied to this data set. As in the above experiments a five times multi-run over five cross-validations were applied and averaged. The evaluation criteria were the same as for the label assignment in the Label Metric Learning algorithm. The results can be seen in table A.17.

5. Discussion

5.1. Summary of the found results

It has been shown that *Label Metric Learning* can be applied to regression tasks and more generally to any transductive task, here demonstrated by predicting a 3-dimensional target vector. The quality of the overall performance hereby is improved with the quality of the learned distance model, but this is no linear dependency, because wrong distance predictions rather randomize the results then corrupting them directly.

The quality on the distance model depends on several factors of which some can be stated generally and some depend on specific input data. The following influences have been discovered:

- the second-order view that has been applied
- the size of the data set and the capacity of the Machine Learning algorithm and hardware respectively
- the chosen label metric
- the reduction strategy if not enough memory is available

From this list, in the latter two general trends could be discovered. It was found that with the here applied Machine Learning algorithm approximately linear increasing metrics could be learned best. The best label assignments were achieved with metrics with a slightly steeper gradient then linear. A possible explanation for this effect is that the SVR algorithm tries to smooth out the target function, with it disturbing the well-ordering of the distances. In a professional application of the Label Metric Learning algorithm a trade-off between good distance prediction and high gradient has to be sought.

As a main problem memory has been discovered. By transforming to second-order view, the number of available training instances grows quadratically, thus leading to very big training data sets for the applied Machine Learning algorithm. The used SVR algorithm needs to store pairwise kernel values in a matrix, thereby quadrating once more the need for memory. In most cases, the training set had to be reduced that a model could be learned.

To remain a good distance model, two possible strategies have been found: Reducing the training-set after second-order view leads to a acceptable generalization on the whole data set. Alternatively, the training set can be reduced before transforming to second-order view. If then unknown instances are only compared to the reduced training set, the distance model still produces good results. It was found that if the ratio between the original set and the reduced set was not too big, the first method produces better results, because the increased quality of the distance model does not compensate for the missing chances to predict right distance values. If however many samples had to be removed from the training set, it is better to use only this subset as assigning set, because in this case the quality of the distance model increases significantly.

5.2. Comprehensive Evaluation

Comparing the results of the regression experiments with other state-of-the-art algorithms, it can be seen that other methods outperform *Label Metric Learning* at this moment. One reason for this outcome has been showed in the previous section, namely that the applied Support Vector Regression fails to generalize the distance function good enough. Comparing the correlation coefficients of the results of table A.12 with any distance function generalization, it can be seen that the distance function always gets learned significantly worse. Although it might be possible that the distance functions cannot be learned any better with any Machine Learning algorithm, it is highly doubtable.

Since it was additionally seen that the overall performance generally improved with the quality of the distance model, it seems promising to try to improve the quality of the learned distance function. Furthermore, it might even be possible to use more sophisticated methods to assign a label to an unknown instance with a better distance function, improving the overall performance even more. For an example, see again section 3.1.3. Additionally, it might be possible to fasten up the algorithm by successfully applying techniques such as metric-trees (see section 2.1.2).

For all these reasons, it is important to analyze to current short-comings of the Support Vector Algorithm to generalize the distance function. Three possible factors can be found:

1. Through the transformation to second-order view, the availability of training samples grows quadratically. The *UCI Machine Learning Repository* currently lists 14 regression data sets. The smallest data set consists of 23 instances which would already lead to 529 second-order instances. The next bigger consists of 103 instances which could maybe be handled with a very big heap space, but still would be a challenge for the *LibSVM* implementation. The biggest data sets consist of 9000 and 191779 instances which lead to $81 \cdot 10^6$ and $36 \cdot 10^9$ second-order instances. Current large-scale SVR implementations can handle about 20000 instances [CB01]. Therefore, more efficient algorithms must be thought of to handle these large-scale regression problems.

2. More generally, a detailed analysis of the problem setting and, even more important, suitable models are needed. In this thesis only Support Vector Regression was applied. However, it is questionable whether the Support Vector Regression is the best possible model for this regression task. This is not only an issue of computational cost, but also a more general question of how certain metrics can be represented. One aspect is the fact that Label Metric Learning needs models that handle two arguments, whereas Support Vector Regression only handles one argument. It is still not quite clear how this influences the prediction results. Additionally, as presented in section 2.3.2, current Distance Metric Learning algorithms try to learn the positive semi-definite matrix that represents the distance function. In this case though, only linear transformations can be learned. This restricts the power of the model significantly. Even for usual metrics, this constraint can be very exclusive, although many of the metrics presented in section 3.1.2 are derived from an analytical point of view. Therefore a more detailed research about possible connections between the analytical expressions of possible label metrics on one hand and suitable regression models on the other hand is needed.

3. But probably the biggest issue is the vast number of parameters that have to be chosen for Label Metric Learning. Figure 5.1 shows aspects which play an important role in Label Metric Learning. Even in model selection and label assigning has to be decided a

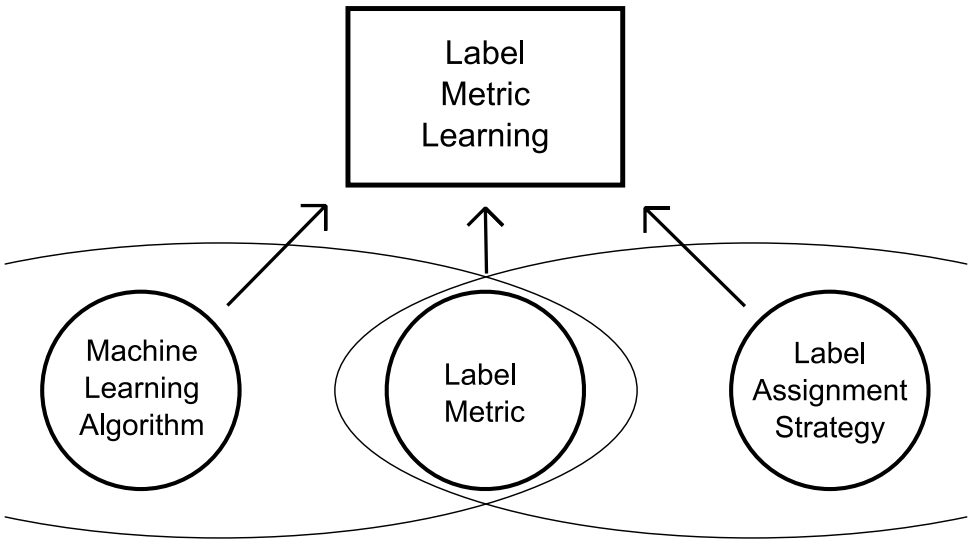


Figure 5.1.: Aspects of the *Label Metric Learning* algorithm and their parameters.
Components of the *Label Metric Learning* algorithm and some examples for possible parameters. The half circles indicate that the specific choice of a label metric could influence both the results of the Machine Learning algorithm and the results of the label assignment strategy (see section 5.3).

	Abalone	Housing	MG	MPG
Euclidean Distance	455	581	1028	446
Bounded metric	1337	1203	527	921

Table 5.1.: Average number of support vectors for the different data sets and metrics.

big number of parameters currently. And, as we have seen in section 3.1.2, there is an infinite possibilities of combining different metrics, many with parameters to be selected on their own. Because this aspect is very important, I want to discuss it in a little more detail in the next section.

5.3. Parameter selection for Label Metric Learning

The experiments have shown that different metrics on the label set could be learned with different quality and, depending on this quality, lead to different results. This effect could also be observed in the number of support vectors. Table 5.1 shows the average number of support vectors after the training of the model for the Euclidean distance and the bounded metric. It can be seen that in almost all benchmark data sets the number increased significantly in the case of the bounded metric which means that the function was much more difficult to learn for the support vector regression (see again section 2.1.1).

There exists several possible solutions to this problem. One of this solution is obviously to chose a metric which can be learned sufficiently good enough. However, this solution is not a good possibility. First of all, it is a priori not known which metrics will generalize well on a given data set, as show tables A.4 and 5.1. Secondly, as shows table A.5, there are also metrics which lead to reasonable label assignments, although the distance predictions are not as good, e.g. the bounded metric.

But it is also not possible to simply choose any metric and then hope for the best. As the above results or table A.14 show, some metrics simple could not be learned on the available data sets with the Support Vector Regression algorithm. Additionally, not every metric is suited for every problem and data set, especially if more sophisticated methods should be applied. For example, it might be possible to calculate an exact label with a p-norm (see section 3.1.3) but it is not possible with the discrete metric. That this is not only a hypothetical issue, compare the results for the SNCF metric in table A.14. Although the metric could be generalized reasonably good, the label assignment produced the worst results for the whole QSAR benchmark data set.

With the current available methods and knowledge, there is no other possibility to try several metrics and compare the results. However, this approach seems to be a direct contradiction to the original idea to Distance Metric Learning, because the main motivation to Distance Metric Learning was exactly to eliminate the need to provide the application with a specific metric.

This is certainly true in some sense. While only accounting for the opportunity to a better semantic model, the technical issues were not considered enough, this leading to the dilemma at hand. A very important goal of future research must be to combine the available methods: optimizing the process to help the user to find an appropriate metric on the label set and at the same time ensuring the possibility to learn this metric good enough. However, it must not be forgotten that *Label Metric Learning* improved the original problem after all. Instead of finding a metric on the input space, the user must provide a to him meaningful metric on the label space. If such a metric exists, they, as the label space itself, are interpretable by the user and task gets easier. For many problems, this thesis has already presented metrics in section 3.1.2 and it will also provide some more suggestions in the next chapter.

6. Conclusions

This research thesis has summarized current issues and algorithms in *Distance Metric Learning* and introduced a new algorithm called *Label Metric Learning*. It has compared *Label Metric Learning* to the other techniques and analyzed the mathematical background to some extend. The capability of this algorithm has been tested on standard benchmark data sets and also on a data set which couldn't been analyzed integratively by conventional methods before. Finally, the results and the implications have been discussed.

This thesis has shown that *Label Metric Learning* works in general. However, *Label Metric Learning* is still outperformed by current machine learning algorithms, not only in time and memory usage, but also in the corresponding results. At this time you have to invest more resources for LML to get worse results.

On the other hand, worse results also suggest there is space for improvements. In this case, the analysis at hand encourages this hypothesis firmly. Since *Label Metric Learning* is basically a combination of different *exchangable* parts, there is a huge number of possibilities to be investigated. This thesis merely scratched the surface of this analysis. Much more research has still to be done in the effects of the interaction and combination of different components. More important, *Label Metric Learning* also benefits of further research and improvements off the individual parts. With the availability of new general purpose machine learning algorithms, Label Metric Learning will improve automatically much like the other Distance Metric Learning algorithms will improve with the advancement of numerical analysis.

Additionally, many possible applications for *Label Metric Learning* have not been tested or even suggested yet. Most of the conventional Machine Learning algorithms target at predicting class membership of a previously known number of classes or at performing regression analysis, i.e. the target values (and most of the time it is only one) are either discrete or continuous numbers. But with Label Metric Learning it could be possible to predict much more complex structures, whenever a metric and a compromise of some sort can be declared on the target range. Possible targets therefore include:

1. Strings, using for example the *Hamming distance*.
2. Quadratic matrices, using any norm $||.||$ on $\mathbb{R}^n$ and declaring a norm on $\mathbb{R}^{n\times n}$ by
$$||A|| := \sup_{x\neq 0} \frac{||Ax||}{||x||}$$
3. Graphs, by identifying the graph with its adjacency matrix.
4. Functions, using one of many function norms.

This last usage might be particularly interesting in our context, since the main goal of machine learning generally is to approximate a function. Label Metric Learning could be used to predict target functions or data source distributions by comparing a unknown function with sample functions saved in some data base. However, the question if this is applicable is currently unknown and must be part of future research.

More generally, this research thesis must leave many problems and questions unanswered which include the following:

1. The most important problem is probably to learn a given distance function. For this issue, many possible solutions are thinkable. For example, in this thesis only one general problem solver was applied to find a suitable model. A next step could be to evaluate other Machine Learning algorithms. However, maybe even a better step would be to find algorithms which are specifically designed to model *any* distance function. Much empirical as well as theoretical analysis still lies ahead in this question.

2. In this research thesis, only the k-Nearest-Neighbor algorithm has been applied to predict the target value. Other possible strategies have already been introduced in this thesis, but could not be evaluated at this time. Therefore, it must be investigated which methods exist to make better use of the predicted distances and if they can be applied practically.

3. With the possibility to predict not only standardized target values, better evaluation methods have to be thought of. To illustrate this need it is sufficient to recall the evaluation of the QSAR benchmark data set. There are many possible goals a user could have, presented by choosing an appropriate metric: maybe he wants the exact value for only one target, maybe he is satisfied by predicting the correct relative quantities or even the correct ranking. If a learned model can represent this goals cannot be evaluated solemnly by the Mean Squared Error. And for other possible applications, these criteria might fail completely.

Although there are currently still shortcomings and pit-falls to overcome, the possible applications as well as the available results and interpretation possibilities should be appealing to use and further improve *Label Metric Learning*.

A. Evaluation results

Here are listed the tables containing the evaluation results. Fur a more detailed description see each table.

		Euclidean distance		Bounded metric	
		ø MSE	ø Q^2	ø MSE	ø Q^2
Abalone	Normal	9.71	46.6	60.0	34.0
	Small	9.58	45.6	60.1	33.4
	Dotproduct	**9.13**	**48.6**	**59.6**	**35.5**
	Matrix	10.6	42.5	65.3	28.4
Housing	Normal	**12.7**	**81.5**	30.8	47.2
	Small	17.6	72.0	30.8	41.4
	Dotproduct	24.0	60.1	32.1	36.1
	Matrix	19.1	74.2	**30.5**	**48.0**
MG	Normal	**30.1**	**60.1**	8.64	59.1
	Small	37.0	46.0	11.0	43.6
	Dotproduct	30.7	56.1	9.32	54.3
	Matrix	30.5	59.1	**8.58**	**59.2**
MPG	Normal	**9.15**	**84.2**	27.0	**56.2**
	Small	15.9	70.1	27.3	47.3
	Dotproduct	15.4	70.9	**26.5**	50.3
	Matrix	10.3	82.6	28.3	53.3

Table A.1.: Results for different second-order views.

Comparison of the prediction quality based on different second-order views (see section 3.1.3), MSE $\cdot 10^{-3}$ and correlation coefficient $\cdot 10^{-2}$. For each data-set a random 10-fold cross-validation was run and results averaged. The two different metrics that were applied were Euclidean distance, linearly scaled to $[0, 1]$ (based on the labels of each training set) and *bounded metric* (see section 3.1.2) with Euclidean Distance as foundation. Best result for each data set and each category are marked **bold**.

		Euclidean distance		Bounded metric	
		ø MSE	ø Q^2	ø MSE	ø Q^2
Abalone	Random [*1600*]	10.4	43.7	62.7	29.7
	Random [*2400*]	10.1	45.0	61.2	31.8
	Random [*3200*]	9.74	46.6	60.5	33.0
	Equal [*1600*]	12.0	21.0	111	2.37
	Equal [*2400*]	13.1	18.6	84.4	17.0
	Equal [*3200*]	12.0	10.8	94.3	13.0
	Wide spread [*1600*]	10.3	43.6	62.7	30.1
	Wide spread [*2400*]	9.97	45.6	61.2	32.1
	Wide spread [*3200*]	**9.69**	**46.6**	**60.5**	**33.1**
Housing	Random [*1600*]	16.3	75.9	31.9	42.9
	Random [*2400*]	14.1	79.2	31.5	45.4
	Random [*3200*]	**13.1**	**80.9**	**31.1**	**47.0**
	Equal [*1600*]	25.5	63.1	40.6	22.2
	Equal [*2400*]	21.4	66.8	36.7	27.6
	Equal [*3200*]	22.3	70.6	38.7	30.5
	Wide spread [*1600*]	16.4	75.7	32.3	42.3
	Wide spread [*2400*]	14.6	78.8	31.6	45.2
	Wide spread [*3200*]	13.5	80.4	31.1	46.7
MG	Random [*1600*]	31.4	57.7	8.91	57.1
	Random [*2400*]	30.7	59.3	8.81	57.9
	Random [*3200*]	**30.0**	**60.2**	**8.70**	**58.6**
	Equal [*1600*]	42.9	54.0	11.4	48.5
	Equal [*2400*]	56.4	51.8	11.9	46.3
	Equal [*3200*]	56.0	51.0	11.2	54.2
	Wide spread [*1600*]	31.4	57.7	8.89	57.2
	Wide spread [*2400*]	30.7	59.2	8.79	58.0
	Wide spread [*3200*]	30.1	59.7	8.72	58.5
MPG	Random [*1600*]	10.1	82.5	29.3	52.1
	Random [*2400*]	9.46	83.6	28.0	54.1
	Random [*3200*]	**9.12**	**84.4**	27.0	**55.8**
	Equal [*1600*]	15.0	76.0	31.0	46.3
	Equal [*2400*]	16.3	76.2	33.0	39.9
	Equal [*3200*]	15.5	76.2	31.1	49.4
	Wide spread [*1600*]	10.2	82.4	29.0	52.6
	Wide spread [*2400*]	9.67	83.2	27.3	54.7
	Wide spread [*3200*]	9.23	84.1	**26.9**	55.4

Table A.2.: Results for different reduction techniques *after* transforming to second-order. Results for different reduction techniques *after* transforming to second-order view (see section 3.1.3), MSE $\cdot 10^{-3}$ and correlation coefficient $\cdot 10^{-2}$. For each configuration a random 10-fold cross-validation was run and results averaged. The two different metrics that were applied were Euclidean distance, linearly scaled to $[0, 1]$ (based on the labels of each training set) and *bounded metric* (see section 3.1.2) with Euclidean Distance as foundation. Best result for each data set and each category are marked **bold**.

		Euclidean distance		Bounded metric	
		ø MSE	ø Q^2	ø MSE	ø Q^2
Abalone	Random [*30*]	15.4	28.4	79.2	20.7
	Random [*40*]	**12.5**	30.0	76.8	18.0
	Random [*50*]	13.1	29.3	74.5	21.6
	Equal [*30*]	34.6	31.4	83.6	19.1
	Equal [*40*]	37.9	29.4	77.9	**26.8**
	Equal [*50*]	26.3	**34.4**	78.4	23.4
	Wide spread [*30*]	23.8	19.8	75.3	22.4
	Wide spread [*40*]	22.1	24.7	73.7	21.5
	Wide spread [*50*]	18.0	29.8	**72.8**	21.0
Housing	Random [*30*]	26.3	56.5	39.1	34.5
	Random [*40*]	27.0	55.1	39.1	34.6
	Random [*50*]	21.9	64.3	37.1	38.0
	Equal [*30*]	29.7	46.9	61.6	27.2
	Equal [*40*]	29.7	55.4	67.8	21.1
	Equal [*50*]	32.4	50.9	52.3	30.9
	Wide spread [*30*]	28.8	51.0	37.6	32.7
	Wide spread [*40*]	24.4	60.3	35.9	37.4
	Wide spread [*50*]	**20.0**	**68.4**	**33.7**	**40.7**
MG	Random [*30*]	40.6	47.2	11.4	46.0
	Random [*40*]	39.7	51.2	11.5	43.9
	Random [*50*]	**38.7**	**50.4**	9.82	53.0
	Equal [*30*]	60.1	26.3	14.2	21.1
	Equal [*40*]	62.1	22.8	14.3	21.3
	Equal [*50*]	63.7	26.0	14.2	23.5
	Wide spread [*30*]	39.8	47.4	**9.63**	52.2
	Wide spread [*40*]	40.9	48.7	9.86	51.6
	Wide spread [*50*]	43.2	47.1	9.81	**53.4**
MPG	Random [*30*]	15.7	74.6	37.1	43.9
	Random [*40*]	13.6	75.9	33.6	46.8
	Random [*50*]	12.8	77.5	34.1	49.7
	Equal [*30*]	15.9	70.1	32.5	48.0
	Equal [*40*]	12.8	78.2	30.8	50.8
	Equal [*50*]	**12.7**	**78.5**	**29.5**	49.8
	Wide spread [*30*]	22.3	68.5	30.7	50.0
	Wide spread [*40*]	15.5	73.4	30.7	**49.9**
	Wide spread [*50*]	16.7	73.6	30.4	51.1

Table A.3.: Results for different reduction techniques *before* transforming to second-order. Results for different reduction techniques *before* transforming to second-order view (see section 3.1.3), MSE $\cdot 10^{-3}$ and correlation coefficient $\cdot 10^{-2}$. For each configuration a random 10-fold cross-validation was run and results averaged. The two different metrics that were applied were Euclidean distance, linearly scaled to $[0,1]$ (based on the labels of each training set) and *bounded metric* (see section 3.1.2) with Euclidean Distance as foundation. Best result for each data set and each category are marked **bold**.

	Abalone		Housing		MG		MPG	
	ø MSE	ø Q^2	ø MSE	ø Q^2	ø MSE	ø Q^2	ø MSE	ø Q^2
Euclid. dist.	9.83	46.5	13.7	80.0	29.6	60.5	9.20	84.2
Bounded dist.	60.4	33.3	31.3	46.3	**8.73**	58.3	26.8	55.9
Bound. scal. dist.	64.9	33.3	32.2	47.0	41.5	58.1	28.2	55.2
Bound. norm. dist.	**5.30**	44.2	**6.25**	76.4	9.98	58.2	**4.20**	81.2
Lin. t. Eucl. [1.25]	14.6	47.6	18.5	82.1	46.5	59.8	13.5	84.9
Lin. t. Eucl. [1.5]	20.6	**48.9**	24.0	**82.2**	61.4	59.6	18.3	85.1
Lin. t. Eucl. [1.75]	27.8	48.7	29.3	81.3	74.2	58.0	22.7	**85.3**
Rooted Eucl. [1.5]	16.7	46.5	15.5	79.6	33.7	57.8	11.4	83.3
Rooted Eucl. [2.0]	22.7	44.2	16.7	77.3	33.1	55.4	12.9	81.2
Rooted Eucl. [3.0]	31.5	38.2	16.0	73.5	27.5	52.5	14.2	76.4
Rooted Eucl. [4.0]	38.7	33.5	15.2	68.7	21.9	50.7	14.7	71.5
Squared Eucl. [1.5]	5.91	38.6	10.9	78.9	22.8	**60.9**	7.11	82.3
Squared Eucl. [2.0]	4.54	29.8	9.73	75.9	17.8	59.2	5.87	78.4
Squared Eucl. [3.0]	4.49	14.0	8.18	68.1	11.4	53.9	4.52	67.7
Blended dist. [0.2|0.8]	44.5	35.2	23.9	55.4	11.8	59.4	21.0	62.3
Blended dist. [0.4|0.6]	31.6	37.5	18.4	64.5	15.5	59.7	15.7	69.7
Blended dist. [0.6|0.4]	21.6	41.2	14.4	73.1	20.0	59.5	11.8	77.1
Blended dist. [0.8|0.2]	14.3	45.3	13.0	77.3	24.4	60.4	9.34	82.4

Table A.4.: Results for learning different metrics.

Results for different metrics, MSE $\cdot 10^{-3}$ and correlation coefficient $\cdot 10^{-2}$. For each configuration a random 10-fold cross-validation was run and results averaged. Best result for each data set and each category are marked **bold**. Please see section 4.2 for a detailed description of the different metrics that were applied.

	Abalone		Housing		MG		MPG	
	ø MSE	ø Q^2	ø MSE	ø Q^2	ø MSE	ø Q^2	ø MSE	ø Q^2
Euclid. dist.	8.10	49.9	35.0	80.8	2.52	71.9	11.0	90.5
Bounded dist.	8.26	51.9	34.8	78.0	2.62	71.1	13.9	88.3
Bound. scal. dist.	8.05	52.6	35.0	79.1	2.53	72.6	12.6	89.7
Bound. norm. dist.	8.43	46.6	37.5	81.7	2.70	69.8	11.7	89.7
Lin. t. Eucl. [1.25]	7.51	54.1	34.1	**83.3**	2.49	73.0	10.0	91.7
Lin. t. Eucl. [1.5]	7.48	53.8	36.1	78.1	2.54	72.0	10.9	91.1
Lin. t. Eucl. [1.75]	7.54	54.7	36.5	79.4	2.43	73.5	10.9	91.0
Rooted Eucl. [1.5]	**7.46**	55.3	**33.5**	82.8	2.52	72.8	10.1	92.0
Rooted Eucl. [2.0]	7.64	55.2	34.5	82.0	2.78	69.7	**9.67**	**92.1**
Rooted Eucl. [3.0]	8.05	53.3	33.9	78.8	2.70	70.6	11.9	90.2
Rooted Eucl. [4.0]	8.45	51.0	34.8	78.9	**2.37**	**74.6**	12.0	90.2
Squared Eucl. [1.5]	9.84	28.7	39.3	80.1	2.80	68.1	15.7	87.3
Squared Eucl. [2.0]	11.7	14.7	48.3	75.0	3.55	58.8	20.3	82.6
Squared Eucl. [3.0]	13.9	-9.0	67.4	67.9	4.60	42.6	58.0	35.2
Blended dist. [0.2|0.8]	8.19	53.6	38.0	77.2	2.79	68.6	13.3	89.7
Blended dist. [0.4|0.6]	8.15	52.4	40.4	75.6	2.54	71.4	13.6	89.2
Blended dist. [0.6|0.4]	7.74	53.3	38.8	76.5	2.62	70.9	11.0	90.4
Blended dist. [0.8|0.2]	7.56	**55.5**	35.1	79.6	2.67	70.2	10.2	91.4

Table A.5.: Results for label assignments with an 3NN assigner.

Results for label assignment with an smooth 3NN assigner as in (2.13), MSE as is except for the MG data-set with MSE $\cdot 10^{-2}$ and correlation coefficient $\cdot 10^{-2}$ for all data-sets. For each configuration a random 10-fold cross-validation was run and results averaged. Best result for each data set and each category are marked **bold**. Please see section 4.2 for a detailed description of the different metrics that were applied.

	Abalone		Housing		MG		MPG	
	ø MSE	ø Q^2	ø MSE	ø Q^2	ø MSE	ø Q^2	ø MSE	ø Q^2
Random[55]								
Euclid. dist.	**13.5**	30.8	22.9	62.3	41.6	50.7	**11.7**	79.2
Bounded dist.	69.6	24.9	34.9	41.4	**9.82**	**52.8**	32.0	50.5
Lin. t. Eucl. [1.25]	20.6	**34.5**	31.4	66.9	59.7	51.4	17.8	**79.7**
Rooted Eucl. [1.5]	25.0	30.0	**22.5**	**68.8**	44.0	49.4	15.1	78.3
Wide spread[55]								
Euclid. dist.	18.9	28.8	25.4	56.2	4.18	50.1	16.0	74.7
Bounded dist.	70.3	24.9	33.9	42.9	10.0	51.3	30.5	51.5
Lin. t. Eucl. [1.25]	30.0	28.8	31.0	67.7	61.9	50.0	26.6	73.0
Rooted Eucl. [1.5]	25.7	32.4	28.2	58.5	45.0	48.3	18.2	74.9

Table A.6.: Quality of learned metrics with a full assignment set

Results for different metrics, MSE $\cdot 10^{-3}$ and correlation coefficient $\cdot 10^{-2}$. For each configuration a random 10-fold cross-validation was run and results averaged. Best result for each data set and each category are marked **bold**. Please see section 4.2 for a detailed description of the different metrics that were applied.

	Abalone		Housing		MG		MPG	
	ø MSE	ø Q^2	ø MSE	ø Q^2	ø MSE	ø Q^2	ø MSE	ø Q^2
Random[55]								
Euclid. dist.	14.2	37.5	47.7	72.7	3.89	58.1	**11.8**	**90.1**
Bounded dist.	13.2	47.3	40.6	76.2	3.74	60.6	16.1	86.3
Lin. t. Eucl. [1.25]	**9.83**	37.7	42.0	75.4	4.20	57.6	16.5	87.7
Rooted Eucl. [1.5]	14.8	**47.4**	41.0	**78.1**	**3.35**	64.8	12.9	89.4
Wide spread[55]								
Euclid. dist.	16.8	31.5	48.7	68.6	4.01	53.1	23.4	80.4
Bounded dist.	9.88	45.6	39.6	75.2	3.48	63.6	15.2	86.9
Lin. t. Eucl. [1.25]	16.5	32.9	47.2	70.9	3.52	**66.4**	31.1	82.8
Rooted Eucl. [1.5]	10.3	45.2	**39.2**	77.1	3.83	58.6	17.1	86.4

Table A.7.: Quality of the label prediction with the full assignment set

Results for label assignment with an smooth 3NN assigner as in (2.13), MSE as is except for the MG data-set with MSE $\cdot 10^{-2}$ and correlation coefficient $\cdot 10^{-2}$ for all data-sets. For each configuration a random 10-fold cross-validation was run and results averaged. Best result for each data set and each category are marked **bold**. Please see section 4.2 for a detailed description of the different metrics that were applied.

	Abalone		Housing		MG		MPG	
	ø MSE	ø Q^2	ø MSE	ø Q^2	ø MSE	ø Q^2	ø MSE	ø Q^2
Random[*55*]								
Euclid. dist.	**9.15**	50.2	**13.2**	80.1	25.7	**68.8**	**9.92**	84.6
Bounded dist.	58.9	38.6	31.6	49.7	8.04	63.4	25.0	61.4
Lin. t. Eucl. [1.25]	13.7	53.9	20.2	**80.6**	39.5	66.9	12.0	**86.8**
Rooted Eucl. [1.5]	18.5	51.6	16.8	78.0	29.9	64.7	10.8	84.9
Wide spread[*55*]								
Euclid. dist.	11.9	61.7	15.8	73.4	26.8	68.0	14.4	76.2
Bounded dist.	56.7	41.7	30.3	49.1	**7.84**	65.2	24.4	60.7
Lin. t. Eucl. [1.25]	19.8	**61.9**	21.1	76.4	41.6	67.7	19.3	80.5
Rooted Eucl. [1.5]	18.1	59.7	18.7	72.1	29.3	66.9	12.5	82.3

Table A.8.: Quality of distance predictions with a reduced assignment set

Results for different metrics, MSE $\cdot 10^{-3}$ and correlation coefficient $\cdot 10^{-2}$. The distances have only been predicted between the unknown samples and samples that have been used in the second-order set. For each configuration a random 10-fold cross-validation was run and results averaged. Best result for each data set and each category are marked **bold**. Please see section 4.2 for a detailed description of the different metrics that were applied.

	Abalone		Housing		MG		MPG	
	ø MSE	ø Q^2	ø MSE	ø Q^2	ø MSE	ø Q^2	ø MSE	ø Q^2
Random[*55*]								
Euclid. dist.	7.39	54.7	**35.9**	77.7	2.40	74.7	14.2	89.5
Bounded dist.	7.91	55.9	49.9	67.4	2.55	71.7	11.6	**90.7**
Lin. t. Eucl. [1.25]	**7.14**	57.5	38.2	77.3	2.42	74.2	**11.4**	90.1
Rooted Eucl. [1.5]	7.50	57.1	42.5	72.6	2.65	72.7	11.6	90.3
Wide spread[*55*]								
Euclid. dist.	7.88	50.4	41.8	75.6	2.46	74.2	13.8	88.5
Bounded dist.	7.87	56.1	44.1	71.7	**2.35**	74.6	15.5	87.2
Lin. t. Eucl. [1.25]	7.77	53.2	42.0	75.3	2.44	74.0	12.7	90.3
Rooted Eucl. [1.5]	7.32	**58.5**	38.0	**79.0**	2.43	**75.1**	13.2	89.9
Random[*40*]								
Euclid. dist.	7.89	50.7	47.1	70.3	2.61	72.0	13.8	88.1
Bounded dist.	8.37	50.3	51.3	66.5	2.52	73.3	16.6	85.7
Lin. t. Eucl. [1.25]	8.10	50.0	41.1	74.0	2.63	71.6	13.9	88.2
Rooted Eucl. [1.5]	7.84	55.7	51.4	66.3	2.82	69.0	12.9	89.1
Wide spread[*40*]								
Euclid. dist.	8.05	51.3	38.6	75.2	2.33	75.4	13.2	89.0
Bounded dist.	8.03	53.0	48.1	71.3	2.64	72.0	16.4	87.0
Lin. t. Eucl. [1.25]	7.67	53.7	42.9	73.3	2.48	73.7	12.5	89.7
Rooted Eucl. [1.5]	7.74	54.6	46.0	71.6	2.67	72.2	13.6	88.5

Table A.9.: Quality of the label prediction with a reduced assignment set

Results for label assignment with an smooth 3NN assigner as in (2.13), MSE as is except for the MG data-set with MSE $\cdot 10^{-2}$ and correlation coefficient $\cdot 10^{-2}$ for all data-sets. To assign a label only the subset of samples has been used which were part of the training set. For each configuration a random 10-fold cross-validation was run and results averaged. Best result for each data set and each category are marked **bold**. Please see section 4.2 for a detailed description of the different metrics that were applied.

	Abalone		Housing		MG		MPG	
	ø MSE	ø Q^2	ø MSE	ø Q^2	ø MSE	ø Q^2	ø MSE	ø Q^2
$k = 3$								
Euclid. dist.	**7.76**	53.3	16.4	74.7	29.7	63.1	9.57	84.3
Bounded dist.	58.6	39.4	31.3	49.9	**7.74**	64.8	25.0	59.8
Lin. t. Eucl. [1.25]	15.1	51.1	20.3	79.8	43.2	**66.9**	11.6	**87.4**
Rooted Eucl. [1.5]	17.8	50.5	14.8	**81.5**	30.3	63.9	10.6	84.8
$k = 5$								
Euclid. dist.	8.84	50.3	14.1	79.9	26.1	64.9	**8.19**	85.4
Bounded dist.	58.9	40.0	30.6	49.5	7.95	64.4	26.7	59.4
Lin. t. Eucl. [1.25]	12.6	53.3	20.5	77.2	43.4	65.5	13.0	85.9
Rooted Eucl. [1.5]	16.4	53.1	16.8	78.0	30.4	63.7	11.1	84.5
$k = 7$								
Euclid. dist.	9.57	47.3	**13.8**	79.7	28.9	64.8	8.56	85.0
Bounded dist.	57.8	40.3	31.0	50.8	8.18	62.5	28.0	58.6
Lin. t. Eucl. [1.25]	13.5	**57.1**	22.1	78.4	42.0	67.5	12.2	86.5
Rooted Eucl. [1.5]	15.9	56.8	16.8	78.0	31.8	62.9	11.3	84.5
$k = 10$								
Euclid. dist.	8.48	51.6	14.8	75.9	26.4	66.8	8.82	84.5
Bounded dist.	58.5	37.7	30.5	50.3	7.86	64.3	27.1	57.3
Lin. t. Eucl. [1.25]	14.1	53.4	22.5	77.8	42.4	65.8	11.8	86.9
Rooted Eucl. [1.5]	15.6	54.7	17.6	77.4	28.4	66.0	11.2	84.3

Table A.10.: Quality of distance predictions for different k-values

Results for different metrics, MSE $\cdot 10^{-3}$ and correlation coefficient $\cdot 10^{-2}$. For each configuration a random 10-fold cross-validation was run and results averaged. Best result for each data set and each category are marked **bold**. Please see section 4.2 for a detailed description of the different metrics that were applied.

	Abalone		Housing		MG		MPG	
	ø MSE	ø Q^2	ø MSE	ø Q^2	ø MSE	ø Q^2	ø MSE	ø Q^2
$k = 3$								
Euclid. dist.	7.67	53.8	47.7	69.7	2.61	71.9	12.3	89.2
Bounded dist.	8.05	54.4	44.6	71.6	2.45	74.9	13.6	88.4
Lin. t. Eucl. [1.25]	7.62	56.1	41.0	74.3	2.38	**75.5**	11.4	90.6
Rooted Eucl. [1.5]	7.53	56.9	**34.8**	78.5	2.92	69.4	13.4	89.0
$k = 5$								
Euclid. dist.	7.42	55.7	36.3	78.6	2.37	73.6	11.6	90.1
Bounded dist.	8.18	55.5	44.9	72.5	2.40	73.7	13.1	88.5
Lin. t. Eucl. [1.25]	7.63	55.8	37.7	77.4	2.52	73.6	12.6	89.4
Rooted Eucl. [1.5]	7.21	58.7	40.1	75.3	2.47	73.3	12.4	89.5
$k = 7$								
Euclid. dist.	7.31	57.7	38.7	76.3	2.56	72.8	11.6	90.6
Bounded dist.	7.89	56.5	39.5	77.4	2.47	73.2	14.7	88.3
Lin. t. Eucl. [1.25]	7.39	57.1	41.4	77.1	2.38	75.4	**10.9**	90.9
Rooted Eucl. [1.5]	**7.05**	**61.2**	39.6	77.9	2.69	71.7	11.4	91.3
$k = 10$								
Euclid. dist.	7.40	57.8	39.5	**78.7**	2.41	73.9	12.2	90.2
Bounded dist.	7.89	55.5	39.6	77.0	2.44	75.0	13.1	89.4
Lin. t. Eucl. [1.25]	7.40	59.4	39.5	75.4	2.37	74.7	11.3	**91.6**
Rooted Eucl. [1.5]	7.21	59.5	41.0	75.9	**2.28**	75.4	13.5	89.1

Table A.11.: Quality of the label prediction for different k-values

Results for label assignment with an smooth KNN assigner as in (2.13) and different values for k, MSE as is except for the MG data-set with MSE $\cdot 10^{-2}$ and correlation coefficient $\cdot 10^{-2}$ for all data-sets. For each configuration a random 10-fold cross-validation was run and results averaged. Best result for each data set and each category are marked **bold**. Please see section 4.2 for a detailed description of the different metrics that were applied.

	3NN		*LibSVM* ε-SVR	
Data set	ø MSE	ø Q^2	ø MSE	ø Q^2
Abalone	4.60	75.8	5.69	68.3
Housing	18.8	89.4	18.2	88.8
MG	1.45	84.7	1.67	82.5
MPG	7.61	94.1	8.88	92.7

Table A.12.: Comparison of different algorithms for the benchmark regression data sets.

Comparison of different Machine Learning algorithms on the benchmark regression data sets, MSE as is except for the MG data-set with MSE $\cdot 10^{-2}$ and correlation coefficient $\cdot 10^{-2}$ for all data sets. For each configuration a random 10-fold cross-validation was run and results averaged. Please see section 4.2 for a detailed description.

	Model		Overall		Thrombin		Trypsin		Factor Xa	
p	ø MSE	ø Q^2	ø MSE	ø ρ	ø MSE	ø Q^2	ø MSE	ø Q^2	ø MSE	ø Q^2
0.6	8.53	58.9	2.18	88.7	1.04	29.5	0.73	50.6	0.41	7.3
0.8	3.15	62.0	1.98	89.5	0.97	36.5	0.62	**63.6**	0.4	12.8
1.0	1.78	63.3	1.90	89.5	0.94	38.3	0.57	61.7	0.39	22.4
1.5	0.89	**64.0**	1.86	**90.2**	0.93	39.6	0.56	61.3	**0.36**	**32.7**
2.0	0.67	63.6	1.87	89.8	**0.91**	**42.5**	0.59	57.2	0.37	31.9
3.0	0.52	61.2	1.86	89.3	0.91	41.4	0.57	55.8	0.37	30.5
4.0	**0.47**	61.9	**1.82**	89.8	0.92	41.2	**0.53**	62.2	0.37	32.0

Table A.13.: Results for different p-norms on the QSAR benchmark data set.
Results for different metrics and label assigning with a smooth 3NN assigner as in (2.13), ρ and correlation coefficient $\cdot 10^{-2}$ for all values. For each configuration a random 5-fold cross-validation was run and results averaged. Best result for each data set and each category are marked **bold**.

	Model		Assignment	
Metric	ø MSE	ø Q^2	ø MSE	ø ρ
SNCF	1.38	**79.5**	12.1	74.1
Rank [1]	**0.02**	53.3	2.68	89.0
Rank [3]	0.06	59.7	2.62	**90.0**
Angle [10]	0.02	50.1	2.37	89.9
Angle [100]	0.27	49.7	**2.26**	89.1

Table A.14.: Results for alternative metrics on the QSAR benchmark data set.
Results for different metrics and label assigning with a smooth 3NN assigner as in (2.13), ρ and correlation coefficient $\cdot 10^{-2}$ for all values. For each configuration a random 5-fold cross-validation was run and results averaged. Best result for each data set and each category are marked **bold**. Please see section 4.3 for a detailed description of the different metrics that were applied.

Metrics		Model		Assignment	
		ø MSE	ø Q^2	ø MSE	ø ρ
p-Norm [1.5]	Rank [3]	0.26	**63.3**	**1.77**	90.1
p-Norm [1.5]	Angle [10]	0.26	63.3	1.83	**90.3**
p-Norm [4.0]	Rank [3]	0.15	61.7	1.84	90.3
p-Norm [4.0]	Angle [10]	**0.15**	61.3	1.78	89.1

Table A.15.: Results for alternative metrics on the QSAR benchmark data set.
Results for different metrics and label assigning with a smooth 3NN assigner as in (2.13), ρ and correlation coefficient $\cdot 10^{-2}$ for all values. For each configuration a random 5-fold cross-validation was run and results averaged. Best result for each data set and each category are marked **bold**. Please see section 4.3 for a detailed description of the different metrics that were applied.

	p-Norm [1.5] / Rank [3]				p-Norm [1.5] / Angle [10]			
	Model		Assignment		Model		Assignment	
	ø MSE	ø Q^2	ø MSE	ø ρ	ø MSE	ø Q^2	ø MSE	ø ρ
Normal								
$k = 2$	0.25	65.3	1.86	90.1	0.26	62.8	1.79	90.3
$k = 3$	**0.26**	**63.7**	1.76	89.5	0.26	63.1	1.76	90.0
$k = 5$	0.25	63.7	1.76	90.5	0.26	63.3	1.80	90.1
$k = 7$	0.26	63.9	1.85	90.3	0.26	63.8	1.72	89.7
$k = 10$	0.26	64.2	1.76	89.2	0.25	64.3	1.76	90.0
Small								
$k = 2$	0.29	60.0	1.46	89.4	0.29	60.7	1.44	88.8
$k = 3$	0.29	60.0	**1.33**	**89.9**	0.29	60.4	**1.32**	**90.1**
$k = 5$	0.29	60.5	1.25	91.2	0.29	60.8	1.23	90.6
$k = 7$	0.29	60.7	1.25	90.3	0.29	60.1	1.22	90.8
$k = 10$	0.29	60.5	1.27	89.4	0.29	60.3	1.27	90.6
Dotproduct								
$k = 2$	0.26	64.9	1.52	90.1	0.26	64.3	1.44	90.1
$k = 3$	0.27	63.4	1.42	89.7	**0.25**	**64.9**	1.41	89.8
$k = 5$	0.26	65.7	1.35	89.7	0.26	63.0	1.35	90.7
$k = 7$	0.26	65.9	1.40	90.7	0.26	64.6	1.42	90.6
$k = 10$	0.25	66.1	1.42	90.3	0.25	64.9	1.41	91.1

Table A.16.: Results for different k-values on the QSAR benchmark data set.
Results for different metrics and label assigning with a smooth KNN assigner as in (2.13) and different values for k, ρ and correlation coefficient $\cdot 10^{-2}$ for all values. For each configuration a random 5-fold cross-validation was run and results averaged. Best result for each data set and each category are marked **bold**. Please see section 4.3 for a detailed description of the different metrics that were applied.

	Results	
	ø MSE	ø ρ
$k = 2$	1.91	88.7
$k = 3$	1.67	89.6
$k = 5$	1.52	90.1
$k = 7$	1.60	90.2
$k = 10$	1.42	90.2

Table A.17.: Results for the regular KNN algorithm on the QSAR benchmark data set. Results of a standard KNN algorithm with different values for k, $\rho \cdot 10^{-2}$. For each configuration a random 5-fold cross-validation was run and results averaged. Please see section 4.3 for a detailed description.

B. List of Abbreviations

DML	Distance Metric Learning
LML	Label Metric Learning
KNN	The K-nearest neighbor machine learning algorithm
QSAR	Quantitative structure-activity relationship
p.s.d.	positive semi-definite
s.t.	subject to (constraints in optimization formulas)

Bibliography

[AN07] A. Asuncion and D.J. Newman. UCI machine learning repository, 2007.

[BHHSW03] A. Bar-Hillel, T. Hertz, N. Shental, and D. Weinshall. Learning distance functions using equivalence relations. In *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*, volume 20, page 11, 2003.

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[BKL06] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *Proceedings of the 23rd international conference on Machine learning*, pages 97–104. ACM New York, NY, USA, 2006.

[BN03] M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396, 2003.

[Bra04] G. Bracha. Generics in the Java programming language. *Sun Microsystems, java. sun. com*, 2004.

[BV04] S.P. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[CB01] R. Collobert and S. Bengio. SVMTorch: Support vector machines for large-scale regression problems. *The Journal of Machine Learning Research*, 1:143–160, 2001.

[CL01] Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[Cox00] M.A.A. Cox. *Multidimensional scaling*. CRC Press, 2000.

[FBF77] J.H. Friedman, J.L. Bentley, and R.A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software (TOMS)*, 3(3):209–226, 1977.

[FL01] Gary William Flake and Steve Lawrence. Efficient SVM regression training with SMO, 2001.

[GR06] A. Globerson and S. Roweis. Metric learning by collapsing classes. *Advances in Neural Information Processing Systems*, 18:451, 2006.

[GRHS05] J. Goldberger, S. Roweis, G. Hinton, and R. Salakhutdinov. Neighbourhood components analysis. *Advances in Neural Information Processing Systems*, 17:513–520, 2005.

[GW92] R.C. Gonzalez and R.E. Woods. Digital imaging processing, 1992.

[HCYZ05] X. He, D. Cai, S. Yan, and H.J. Zhang. Neighborhood preserving embedding. In *Tenth IEEE International Conference on Computer Vision, 2005. ICCV 2005*, volume 2, 2005.

[HLLM06] SCH Hoi, W. Liu, MR Lyu, and W.Y. Ma. Learning distance metrics with contextual constraints for image retrieval. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, 2006.

[HN03] X. He and P. Niyogi. Locality preserving projections. *Advances in neural information processing systems*, 16:153–160, 2003.

[HT96] T. Hastie and R. Tibshirani. Discriminant adaptive nearest neighbor classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(6):607–616, 1996.

[KT03] J.T. Kwok and I.W. Tsang. Learning with idealized kernels. In *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*, volume 20, page 400, 2003.

[LMGY04] T. Liu, A.W. Moore, A. Gray, and K. Yang. An investigation of practical approximate nearest neighbor algorithms. *Advances in neural information processing systems*, 2004.

[NST+94] W.J. Nash, T.L. Sellers, S.R. Talbot, A.J. Cawthorn, and W.B. Ford. The population biology of abalone (haliotis species) in Tasmania. I. Blacklip Abalone (h. rubra) from the north coast and islands of Bass Strait. *Sea Fisheries Division, Technical Report*, 48, 1994.

[RS00] S.T. Roweis and L.K. Saul. Nonlinear dimensionality reduction by locally linear embedding, 2000.

[SHWP02] N. Shental, T. Hertz, D. Weinshall, and M. Pavel. Adjustment learning and relevant component analysis. *LECTURE NOTES IN COMPUTER SCIENCE*, pages 776–792, 2002.

[SJ04] M. Schultz and T. Joachims. Learning a distance metric from relative comparisons. In *Advances in Neural Information Processing Systems 16: Proceedings of the 2003 Conference*, page 41. The MIT Press, 2004.

[TC00] R. Todeschini and V. Consonni. *Handbook of molecular descriptors*. DE, 2000.

[TSL00] J.B. Tenenbaum, V. Silva, and J.C. Langford. A global geometric framework for nonlinear dimensionality reduction, 2000.

[WBS06] K. Weinberger, J. Blitzer, and L. Saul. Distance metric learning for large margin nearest neighbor classification. *Advances in neural information processing systems*, 18:1473, 2006.

[XNJR03] E.P. Xing, A.Y. Ng, M.I. Jordan, and S. Russell. Distance metric learning with application to clustering with side-information. *Advances in neural information processing systems*, pages 521–528, 2003.

[Yan07] L. Yang. An overview of distance metric learning, 2007.

[YJ06] L. Yang and R. Jin. Distance metric learning: A comprehensive survey. *Michigan State Universiy*, 2006.

[YJS07] L. Yang, R. Jin, and R. Sukthankar. Bayesian active distance metric learning. In *Proc. Uncertainty in Artificial Intelligence*, 2007.

[YJSL06] L. Yang, R. Jin, R. Sukthankar, and Y. Liu. An efficient algorithm for local distance metric learning. In *PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*, volume 21, page 543. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.