Metric alignment: linear approximation of arbitrary similarity measures.

Carsten van Weelden
cvanweelden@gmail.com
0518824

Supervisor:
Jan van Gemert
J.C.vanGemert@uva.nl

University of Amsterdam
The Netherlands

August 19, 2014

Abstract

In this thesis we investigated the idea of learning a metric from absolute distance constraints instead of binary similarity/dissimilarity or relative constraints. The metric learning problem is cast as an unconstrained regression problem minimizing squared prediction error between the learned metric and a given target metric. This allows the alignment of the metric in feature space to a target metric in output space. Our results show that our metric correctly minimizes the squared prediction error, but does not necessarily maximize alignment to the target metric.

Contents

Chapter 1

Introduction

In machine learning tasks, objects of interest are represented using a set of features. Features correspond to attributes of the object that can be encoded in some way. For example the features representing images may simply be the pixel values of an image, but may also be more complex features computed from the basic pixel values. Each object is thus represented by a feature vector containing the feature values. We will refer to the space of all possible feature vectors as the *feature space*.

Now suppose we would like to measure similarities between objects. For example, we have some images of a Ferrari car and we want to find images of similar cars. Assuming that similar objects result in similar feature values, we can express similarity between objects as a distance function in feature space. For our $\mathbb{R}^d$ feature space we can therefore measure similarity by Euclidean distance between the points in the feature space given by the feature vectors. Thus, starting from the feature representations of our input images we would expect to find images of similar cars nearer in feature space than images of dissimilar cars.

However, Euclidean distance is not necessarily the best measure for similarity. In most cases features are predetermined since they measure basic attributes of the object, but the type of similarity we want to measure is specific to the task. With Euclidean distance each feature has an equal impact on the distance, even though some features are much more relevant to the type of similarity we want to measure than others. For example in our car images example, we would want to focus on similarities in color if we are simply looking for other red cars, but we would want to look at more complex similarities in shape if we are looking for other racing cars.

Several machine learning methods use a distance function at the core of their algorithm. For example, the k-nearest neighbors (kNN) algorithm classifies objects by searching the training data for feature vectors that are closest to the input vector. The objects corresponding to these nearest neighbors then vote on the class to assign to the input object. It is clear that for this to be effective, distances in feature space should be small for objects of the same class and large for objects of differing class. Similarly, the k-means clustering algorithm clusters samples based on their distances in feature space. This will cluster similar objects together if they are close together in feature space. These methods require us to carefully choose a feature representation and distance

function in order to be effective.

Instead of hand-crafting a distance function, we can automatically learn a suitable distance function in a given feature space. This approach is called *metric learning*. Metric learning methods define a parametric distance function and then learn the best parameters for this function. Learning the parameters requires a set of training samples in the given feature space and a set of constraints on the distances between training samples. Parameters of the distance function are learned that best fit these distance constraints. The constraints thus represent the similarity measure specific to our task. For example, if we are still looking for images of similar racing cars, we would provide samples of racing cars and then constrain the distances between these to be small, while constraining the distances to other types of cars to be large.

Metric learning has been researched most in the context of multi-class classification problems and ranking problems. Because of this background, the constraints generally take the form of similarity/dissimilarity constraints or relative distance constraints. Similarity and dissimilarity constraints are generated from class labels: objects of the same class are constrained to have a small pairwise distance while objects of different classes are constrained to have a larger distance. In contrast, relative distance constraints are used for tasks where it might be difficult to make absolute similarity judgments, but where we can identify one object to be more similar to a given object than some other object. Relative distance constraints constrain one pairwise distance to be smaller than another pairwise distance.

However, there is a more direct approach to the metric learning problem: instead of generating similarity/dissimilarity or relative distance constraints, a metric can be trained on absolute distance constraints that directly specify what the distance between a given pair of points in feature space should be. Absolute distance constraints make more fine-grained similarity judgments than the binary similarity/dissimilarity constraints while supplying stronger supervision than relative distance constraints. Despite these advantages, absolute distance constraints have received little attention in the metric learning literature. Therefore, this thesis investigates the use of absolute distance constraints for metric learning. In doing so it answers the following research questions:

- How can we apply metric learning methods to problems with absolute distance constraints?
- How does metric learning with absolute distance constraints compare to methods using similarity/dissimilarity or relative distance constraints?

There is a specific class of machine learning problems that requires absolute distance constraints, namely *structured prediction* problems. Structured prediction problems are classification problems in which the goal is not to predict a single label, but to predict a complex output structure. For example, predicting a label sequence. Unlike a single class label, which is always either correct or incorrect, structured outputs can be partially correct. Therefore, structured prediction problems define a real-valued loss function which measures the divergence between two output structures. When we apply a prediction algorithm our goal is to predict an output that has the lowest loss relative to a given ground-truth output structure. Modifying the kNN classification algorithm to work as structured prediction algorithm is straightforward: instead of a simple

voting scheme, the outputs corresponding to the k nearest neighbors in feature space are combined in some task-specific way. Since our goal is to predict output structures that have the lowest loss relative to the ground truth, this method is most effective when small distances in feature space correspond to low loss. By generating absolute distance constraints which constrain distances between points in feature space to be proportional to the loss between the corresponding outputs, we can learn a distance function that ensures nearest neighbors in feature space correspond to low-loss solutions. We refer to this method as *metric alignment* since satisfying these constraints aligns the feature space to the structured-output space in which the loss function is measured.

This thesis describes the metric alignment method in detail and compares it against existing methods for metric learning. We apply our method to images taken from a semantic segmentation problem in order to predict semantic overlap between image patches, and to images from an attribute-based classification problem in order to predict an attribute vector from the image features. We compare our method against existing methods where we threshold the real-valued loss in order to generate similarity/dissimilarity constraints. In summary, the main contributions of this thesis are as follows:

- We describe a method for metric learning with absolute distance constraints in the context of structured prediction problems.
- We introduce two learning problems based on existing datasets that can be used to evaluate metric learning methods with absolute distance constraints.
- We evaluate our method against existing methods on these learning problems.

Chapter 2

Metric Learning

Metric learning methods learn a distance function specific to a task through supervised learning. Supervised learning requires a training set containing data points and some sort of side information about those points which informs the learning process. In practice, this side information comes in the form of class labels for each point in classification problems or the correct structured output in structured prediction problems. This side information is often referred to as the ground-truth.

Training points are assumed to be d-dimensional real-valued feature vectors in a Euclidean feature space $\mathcal{X} = \mathbb{R}^d$. Let $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n]$ be the matrix of all the training points, with each column being a single feature vector $\mathbf{x}_i \in \mathcal{X}$. Each element of a training vector corresponds to a single feature value. Distances between points in the feature space can be measured by the Euclidean distance $\|\mathbf{x}_i - \mathbf{x}_j\| = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^T(\mathbf{x}_i - \mathbf{x}_j)}$, although in practice the squared distance is often used since dropping the square root makes the distance easier to compute.

2.1 Linear distance functions

The goal of metric learning is to learn a distance function in the feature space. A distance function, or metric, is a function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}^+$ having the following properties:

$$d(\mathbf{x}_i, \mathbf{x}_j) \geq 0 \qquad \text{non-negativity,} \tag{2.1a}$$

$$d(\mathbf{x}_i, \mathbf{x}_j) = 0 \leftrightarrow i = j \qquad \text{identity of indiscernibles,} \tag{2.1b}$$

$$d(\mathbf{x}_i, \mathbf{x}_j) = d(\mathbf{x}_j, \mathbf{x}_i) \qquad \text{symmetry,} \tag{2.1c}$$

$$d(\mathbf{x}_i, \mathbf{x}_k) \leq d(\mathbf{x}_i, \mathbf{x}_j) + d(\mathbf{x}_j, \mathbf{x}_k) \qquad \text{triangle inequality.} \tag{2.1d}$$

In addition, a distance function is called a pseudo-metric if it conforms to all these conditions except (2.1b) which is replaced by $d(\mathbf{x}_i, \mathbf{x}_i) = 0$, meaning that points still have zero distance to themselves but might also have zero distance to other points. Metric learning is often concerned with learning a pseudo-metric, e.g. in classification problems we would not necessarily want to make a distinction between objects of the same class, hence the distance between them is not constrained to be larger than zero.

Most metric learning methods learn a linear distance function. A linear distance function is limited to rotating and scaling the dimensions along which the

distance is measured. Because of this it can also be computed by first applying a linear transformation to the input space and then measuring the Euclidean distance in the transformed space. This has the benefit that algorithms that rely on a Euclidean distance, such as nearest-neighbor look up with spatial indexing techniques, can be used off-the-shelf.

The distance function is often parameterized as a metric with a $d \times d$ parameter matrix $\mathbf{A}$:

$$d_{\mathbf{A}}(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{A} (\mathbf{x}_i - \mathbf{x}_j)}. \tag{2.2}$$

If $\mathbf{A}$ is positive definite then $d_{\mathbf{A}}$ is a metric and if $\mathbf{A}$ is positive semi-definite then $d_{\mathbf{A}}$ is a pseudo-metric. If $\mathbf{A}$ equals the identity matrix then $d_{\mathbf{A}}$ is simply the Euclidean distance.

In the metric learning literature this form of distance function is often called a Mahalanobis distance, although this term was originally introduced as a metric to measure distance between random vectors from the same distribution. [Mahalanobis, 1936] The original Mahalanobis distance is defined as:

$$d_{\text{MAHALANOBIS}}(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^T \Sigma^{-1} (\mathbf{x}_i - \mathbf{x}_j)}, \tag{2.3}$$

where Σ is the covariance of the distribution, or an estimate thereof. This metric is especially helpful for calculating the distance of a random vector to the mean of the distribution, since the metric ensures that for a Gaussian distribution the iso-surface corresponds to a constant probability of being generated by the distribution. Computing Mahalanobis distance between vectors is equivalent to calculating the Euclidean distance between whitened vectors. This can be easily seen by realizing that whitening data causes the covariance to be equal to the identity matrix, thus $\Sigma^{-1} = \mathbf{I}^{-1} = \mathbf{I}$, making the calculation for the Mahalanobis distance to be equal to Euclidean distance.

Because a Mahalanobis metric is a linear metric, it can be interpreted as the Euclidean distance in a linear transformation of the feature space. This is done by substituting $\mathbf{A} = \mathbf{G}^T \mathbf{G}$ and transforming points in the feature space by $\mathbf{G}$ in the following way:

$$\begin{aligned}
d_{\mathbf{A}}(\mathbf{x}_i, \mathbf{x}_j) &= \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{A} (\mathbf{x}_i - \mathbf{x}_j)} \\
&= \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{G}^T \mathbf{G} (\mathbf{x}_i - \mathbf{x}_j)} \\
&= \sqrt{(\mathbf{G}(\mathbf{x}_i - \mathbf{x}_j))^T \mathbf{G} (\mathbf{x}_i - \mathbf{x}_j)} \\
&= \sqrt{(\mathbf{G}\mathbf{x}_i - \mathbf{G}\mathbf{x}_j)^T (\mathbf{G}\mathbf{x}_i - \mathbf{G}\mathbf{x}_j)} \\
&= \|\mathbf{G}\mathbf{x}_i - \mathbf{G}\mathbf{x}_j\|
\end{aligned} \tag{2.4}$$

Here $\mathbf{G}$ is a $r \times d$ matrix with r being the rank of $\mathbf{A}$. Therefore, if the distance matrix $\mathbf{A}$ is not full-rank, transforming the points in feature space corresponds to projecting onto a lower-dimensional space $\mathbb{R}^r$. This can be useful for high-dimensional problems since it has the same effects as dimension reduction. Reducing the dimensionality of the problem gives a more compact feature representation and distance calculations are less expensive. Furthermore, high-dimensional spaces tend to be sparse as volume increases exponentially with the number of dimensions, often referred to as the curse of dimensionality. Hence, several metric learning methods try to learn low-rank metrics.

2.2 Constraints

Metric learning methods require supervision in the form of constraints on the learned distance. When objects can easily be judged to be either similar or dissimilar, these can be given as a set of similarity/dissimilarity constraints. This is given as a set $\mathcal{S}$ of pairs (i, j) that are similar and a set $\mathcal{D}$ of pairs that are dissimilar. These constraints have the following form:

$$d_{\mathbf{A}}(\mathbf{x}_i, \mathbf{x}_j) \leq u \qquad (i, j) \in \mathcal{S}, \tag{2.5a}$$

$$d_{\mathbf{A}}(\mathbf{x}_i, \mathbf{x}_j) \geq \ell \qquad (i, j) \in \mathcal{D}. \tag{2.5b}$$

They constrain the distances between similar pairs to be lower than some upper bound u and distances between dissimilar pairs to be larger than some lower bound ℓ. Suitable bounds have to be chosen for each problem. This is generally done by computing the 5th and 95th percentiles of the sampled distribution of distances between random pairs and equating these to u and ℓ respectively.

Another popular choice for distance constraints is relative constraints. Relative constraints constrain one distance to be larger or smaller than another distance. Relative constraints are defined as a set $\mathcal{R}$ of triplets of points (i, j, k) and have the following form:

$$d_{\mathbf{A}}(\mathbf{x}_i, \mathbf{x}_j) < d_{\mathbf{A}}(\mathbf{x}_i, \mathbf{x}_k) \qquad (i, j, k) \in \mathcal{R}. \tag{2.6}$$

Thus for each triplet $(i, j, k) \in \mathcal{R}$ the distance between points $\mathbf{x}_i$ and $\mathbf{x}_j$ should be smaller than the distance between points $\mathbf{x}_i$ and $\mathbf{x}_k$.

Distance constraints are generated from the ground truth in the training set. For small problems with a small training set we can take a comprehensive approach and enumerate all possible pairs or triplets. However, for most real-world problems the training set has to be large and enumerating all pairs would be intractable. Therefore we have to resort to taking a sample of pairs from the training set. There are two main approaches to this sampling: global sampling and local sampling. Global sampling is the most general and corresponds to simply picking random pairs or triplets from the training set. This gives the most representative sample of distances in the feature space. However, many methods that use distance function in feature space are only concerned with pairs of points that have small pairwise distances. For example, kNN only looks at the nearest neighbors of a point and any points that have a larger distance are ignored. In this case we can also restrict our samples to only consist of pairs or triplets that are contained in a local neighborhood: local sampling. One way to do this, which is specifically useful for kNN applications is to sample a random point and then sample one or two random points amongst the nearest neighbors of this point in order to form a pair or triplet. Since the nearest neighbors depend on the distance function that is used they may change during the learning process. Therefore, we might want to iterate this sampling during the learning process, which we will refer to as iterative local sampling.

2.3 Optimization and regularization

While most linear metric learning methods parametrize the distance function in the same way, as a Mahalanobis metric defined by (2.2), they differ in the way that they optimize the parameter matrix $\mathbf{A}$.

In [Kulis, 2012] the author present a general formulation of the metric learning problem that encompasses most metric learning methods. The problem consists of minimizing a loss function that includes a cost term and a regularization term and is given by:

$$\mathcal{L}(\mathbf{A}) = r(\mathbf{A}) + \lambda \sum_{i=1}^{m} c_i \left(\mathbf{X}^T \mathbf{A} \mathbf{X}\right). \tag{2.7}$$

Here $r(\mathbf{A})$ is a regularization term which is a function of the parameter matrix $\mathbf{A}$, $c_1, ..., c_m$ are loss functions which encode the given constraints and λ is a trade-off parameter between the regularization term and the costs. The way in which these cost functions are defined, the regularization function that is chosen and the way in which the resulting loss is optimized are the factors that distinguish the different linear metric learning methods.

In (2.7) the cost functions are given as functions of $\mathbf{X}^T\mathbf{A}\mathbf{X}$ in order to be as general as possible. The term $\mathbf{X}^T\mathbf{A}\mathbf{X}$ denotes the pairwise distance matrix under the learned metric. This is simply there to show that the cost functions are evaluated on $d_{\mathbf{A}}$, the learned distance function. In practice the cost functions depend on the given constraints and generally only include the distances between the given pair or triplets of training points. For example, a popular choice is to encode similarity constraints as a hinge loss:

$$c(\mathbf{X}^T\mathbf{A}\mathbf{X}) = \max(0, d_{\mathbf{A}}(x_i, x_j) - u) \qquad \forall (i,j) \in \mathcal{S}, \tag{2.8}$$

$$c(\mathbf{X}^T\mathbf{A}\mathbf{X}) = \max(0, \ell - d_{\mathbf{A}}(x_i, x_j)) \qquad \forall (i,j) \in \mathcal{D}. \tag{2.9}$$

And the equivalent formulation for a hinge loss function on relative distance constraints would be:

$$c(\mathbf{X}^T\mathbf{A}\mathbf{X}) = \max(0, d_{\mathbf{A}}(x_i, x_j) - d_{\mathbf{A}}(x_i, x_k)) \qquad \forall (i,j,k) \in \mathcal{R}. \tag{2.10}$$

Optionally, a regularization term is added to avoid over-fitting. Common examples are the Frobenius norm $\|\mathbf{A}\|_F$, which is the matrix equivalent to standard L_2 regularization and the trace norm $\mathrm{tr}(\mathbf{A})$ which is similar to L_1 regularization in that it prefers low-rank solutions.

The cost functions and regularization terms are generally chosen such that there is a tractable way to minimize the resulting loss function. This optimization can be cast as a minimization over the matrix $\mathbf{A}$ or by substituting $\mathbf{A} = \mathbf{G}^T\mathbf{G}$ it can be done as a minimization over $\mathbf{G}$. In the first case this results in a constrained optimization problem since $\mathbf{A}$ has to be positive semi-definite (PSD). E.g. by reprojecting onto the cone of PSD matrices during the optimization by setting negative eigenvalues to 0. In the second case the problem is unconstrained since $\mathbf{G}^T\mathbf{G}$ will always yield a PSD matrix, but will be in a quadratic form instead of a linear form.

Chapter 3

Related work

3.1 Linear metric learning

There is a large body of work in the metric learning literature, therefore this thesis does not aim to provide a complete overview. Instead, this section highlights the differences between the main metric learning approaches and the work presented in this thesis. Readers looking for a recent survey are directed to [Kulis, 2012], which discusses metric learning approaches with reference to a unified model for regularized transformation learning, a model which we discussed in Section 2.3. Readers looking for a more complete discussion of specific methods are directed to [Bellet et al., 2013] and [Yang and Jin, 2006].

3.1.1 Similarity/dissimilarity constraints

Most metric learning methods have been formulated in the context of a classification problem. This context determines the type of background knowledge that is available for training the metric. Hence, most popular metric learning methods use similarity/dissimilarity constraints, with pairs of inputs being constrained to be similar if they share the same classification label as output, and dissimilar if their output label differs. [Davis et al., 2007, Guillaumin et al., 2009b, Kostinger et al., 2012] Because these methods expect input pairs to be assigned to be either similar or dissimilar, they cannot be applied to a problem where similarity between pairs lies on a continuous scale. In comparison, the method presented in this thesis is specifically designed to do just that, by fitting the learned metric to a target metric which can take any value. The method presented in this thesis can thus be applied to both binary similarity as well as continuous similarity.

3.1.2 Relative distance constraints

Metric learning methods that use relative distance constraints in the context of ranking problems, such as [Schultz and Joachims, 2003], are more similar to the work in this thesis because relative constraints can be used to indirectly encode a real-valued target metric. However, since these constraints do not directly encode the target distance they provide less supervision than the absolute distance constraints that are defined in this thesis. Also, because of the large number

of training triplets that would be needed to fully specify a target metric, these methods are less applicable to large datasets than the method presented in this thesis which needs only a sampling of pairs.

Relative distance constraints are also used in the large-margin metric learning approach, as exemplified by [Weinberger and Saul, 2009, Frome et al., 2007], which has become popular because of good results on classification problems and its similarity to the well-known support vector machine (SVM) approach. Note however that because of the large-margin formulation these methods still depend on a binary similarity judgements for pairs of training points and are thus not applicable to problems with a real-valued target metric.

3.1.3 Metric learning in structured prediction

Structured prediction problems are characterized by a real-valued loss function. The metric learning method described in this thesis can be applied to structured prediction problems by identifying the target metric with this loss function. In [McFee and Lanckriet, 2010] metric learning is also applied to a structured prediction problem. The main differences with the approach described in this thesis is that McFee & Lanckriet apply their method specifically to ranking problems and use a structural SVM approach to learning. Since they specify their ranking problem as finding the best interleaving of relevant and irrelevant points, this method also requires binary similarity judgements. They do make use of the structured loss function in training the metric, but only to find maximally violated constraints for the cutting-plane algorithm that they use. Therefore, this method is not applicable to fitting a metric to a target metric such as is done by the method described in this thesis.

In [Guillaumin et al., 2009a] metric learning is applied to the problem of image annotation, which can also be viewed as a structured problem. Like the work in this thesis, they use a nearest-neighbor approach to solving the problem. But instead of learning a distance function in feature space, they learn a weighted combination of base distance functions picked by hand. They optimize their weights by maximizing the log-likelihood for the training output, while this thesis takes the approach of directly optimizing the distance function instead of optimizing the resulting predictions.

Attribute-based classification is one of the structured prediction problems investigated in this thesis. In [Akata et al., 2013] label embedding is used as an approach to attribute-based classification. This can be viewed as learning a distance function between input features and attribute vectors. In comparison, this thesis presents a method that learns a distance function between pairs of input vectors. Thus, while this thesis takes a pure metric learning approach, [Akata et al., 2013] takes a label embedding, or scoring function, approach.

Note that, although [Bellet et al., 2013] investigates the application of metric learning to structured data, they focus on a structured input space, while the work in this thesis focuses on a structured output space. The methods discussed in [Bellet et al., 2013] rely on binary similarity/dissimilarity constraints for training.

3.1.4 Metric learning as regression

The idea of approaching metric learning as a regression problem, as is done in this thesis, is not new. In [Lowe, 1995] a weighted Euclidean metric is learned through conjugate gradient descent on a squared prediction error measure. Although this approach is similar in solution to the work in this thesis, the target is very different. Like most metric learning methods, this approach relies on the classification problem consisting of a set of discrete labels.

The method in [Weinberger and Tesauro, 2007] learns a metric that minimizes regression error for a kernel regression algorithm. Since they formulate their error as the quadratic prediction error and minimize this using stochastic gradient descent (SGD), they have a very similar derivation of the learning problem as the work presented in this thesis. However, Weinberger $ Tesauro only apply their method directly to the kernel regression problem. Thus in their method the target metric is directly the function to be learned, while the method in this thesis uses the target metric to learn a better representation of the input features.

The regression problem for metric learning from a squared prediction error is further investigated in [Meyer et al., 2011]. The work in this thesis uses exactly the same formulation for the SGD problem as Meyer et al. and they further show fixed-rank and scale invariant variants, which are applicable as solving methods for the work in this thesis. However, in [Meyer et al., 2011] this method is only applied to a classical classification metric learning problem with similarity/dissimilarity constraints. The work in this thesis goes further than that and describes how this method can be applied to solve metric learning problems for structured prediction problems with absolute distance constraints.

3.2 MDS and related techniques

Like the method in this thesis, multidimensional scaling (MDS) and related techniques also use target distances as a training signal. This set of techniques includes MDS [Venna and Kaski, 2006, Chen and Buja, 2009, Chen and Buja, 2013], kernel PCA [Schölkopf et al., 1997], isomap [Tenenbaum et al., 2000], and others. What they have in common is that they take as input a distance matrix, which is comparable with a target metric, since a target metric could also be specified as a distance matrix. However, all these methods learn an embedding of the training points in a lower-dimensional space. Thus instead of solving for the parameters of the distance function, they solve for the point coordinates. Because of this, these methods are very applicable to dimension reduction and visualization, but not to metric learning.

[Chen and Buja, 2013] discusses a family of optimization functions, called stress functions, for MDS methods. The squared error function that this thesis uses corresponds to the most naive stress function and [Chen and Buja, 2013] show that by changing the stress function the characteristics of the solution can be tailored to a specific problem. Thus, finding a method to minimize these stress functions with respect to the metric parameters instead of to the point coordinates might change the characteristics of the method presented in this thesis and make it more broadly applicable.

3.3 Representation learning

The method presented in this thesis learns a feature transformation which makes aligns distances in feature space to a target metric, thus resulting in a feature representation which makes learning problems that are characterized by that target metric easier to solve. According to a recent survey, the problem of representation learning is "learning representations of the data that make it easier to extract useful information when building classiers or other predictors". [Bengio et al., 2013] These descriptions seem very similar. However, the field of representation learning deals with unsupervised methods that learn a completely new representation that can be considered helpful in a general way, while this thesis presents a method that adapts a given representation in a supervised, task-specific way.

Chapter 4

Metric alignment

Our metric alignment method is based on the intuition that, if the ground truth in our training data has a structure on which we can define a real-valued loss function, we can use this loss function to inform our metric learning. Specifically, we learn a metric that minimizes the difference between the distance in feature space and the loss between the corresponding ground-truth output structures. The learned metric thus makes distances in feature space predictive of loss in ground-truth space.[1] This predictive power is especially useful when applying nearest neighbor methods to structured prediction problems.

4.1 Target metric

The metric alignment method takes as input a *target metric* and a set of *absolute distance constraints*. The target metric is a function $\hat{d}$ which assigns a distance to pairs (i, j) of input points. For our applications we define the target metric to be equal to the loss ℓ for prediction output $\mathbf{y}_j$ when the correct output is $\mathbf{y}_i$:

$$\hat{d}(i, j) = \ell(\mathbf{y}_i, \mathbf{y}_j). \tag{4.1}$$

Although we use the structured loss as target metric, the notion of a target metric is more general than that. In fact, the target metric can be any non-negative value as long as it is known for a number of training pairs much larger than the number of dimensions of the feature space. It could even be background knowledge provided by hand. However, since it will be approximated by a pseudo-metric function, it needs to be non-negative and the approximation will be better for target metrics that themselves are of a pseudo-metric nature.

4.2 Absolute distance constraints

Absolute distance constraints are the equivalent of similarity/dissimilarity constraints that use a target metric instead of upper or lower bounds. Absolute distance constraints are given as a set $\mathcal{A}$ of pairs $(i, j) \in \mathcal{A}$, each associated with

[1] Note that the output structure need not necessarily induce any specific type of space, all we need is a positive, real-valued loss function defined on a large enough sample of pairs from the training set.

a ground-truth distance given by the target metric. They are thus defined as follows:

$$d_{\mathbf{A}}(\mathbf{x}_i, \mathbf{x}_j) = \hat{d}(i,j) \qquad (i,j) \in \mathcal{A}. \tag{4.2}$$

Here $d_{\mathbf{A}}(\mathbf{x}_i, \mathbf{x}_j)$ is the learned metric evaluated on the pair of points in feature space $(\mathbf{x}_i, \mathbf{x}_j)$ that corresponds to the pair $(i,j) \in \mathcal{A}$. And $\hat{d}(i,j)$ is the value of the target metric corresponding to that pair. The set of pairs $\mathcal{A}$ can be sampled in the same way as the set of pairs $\mathcal{S}$ in the case of similarity constraints. Note however that if the target metric is non-symmetric, the pairs should be treated as ordered pairs and preferably both (i,j) and (j,i) should be included in the set.

4.3 Loss function

In order to learn a metric from absolute distance constraints we minimize the quadratic prediction error on the target distances given by the constraint. Thus our loss function corresponds to the mean squared error (MSE) over the training set:

$$\mathrm{MSE}_{\mathcal{A}}(\mathbf{A}) = \frac{1}{|\mathcal{A}|} \sum_{(i,j)\in\mathcal{A}} \left(\hat{d}_{ij} - d_{\mathbf{A}}(\mathbf{x}_i, \mathbf{x}_j)\right)^2. \tag{4.3}$$

Minimizing the MSE over the training pairs corresponds to minimizing the empirical risk under a quadratic loss function.

4.4 Convex optimization using stochastic gradient descent

In order to fit the transformation matrix to the training set we use stochastic gradient descent (SGD). Gradient descent is an iterative first order optimization algorithm that finds a local minimum of an error function. On each iteration it changes the parameters by a small step in the direction opposite to the gradient of the error function with respect to the parameters. Since the negative gradient is the direction of steepest descent, each step leads to a reduction in error. When the error function is defined as a sum of individual errors on examples in a training set, SGD is a stochastic version of gradient descent where at each step the gradient is computed on the error of a single example. This does not require a summation over all examples in the training set for each update, which is beneficial when the training set is large.

In order to apply SGD to our learning problem we first define our metric using the decomposition $\mathbf{A} = \mathbf{G}^T\mathbf{G}$ to remove the PSD constraint on $\mathbf{A}$ and turn the problem into a unconstrained optimization problem. Further, we use squared distances to remove the square root to further simplify the gradient derivation. Our distance function is thus defined as:

$$\begin{aligned} d_{\mathbf{A}}(\mathbf{x}_i, \mathbf{x}_j)^2 &= (\mathbf{x}_i - \mathbf{x}_j)^T\mathbf{A}(\mathbf{x}_i - \mathbf{x}_j) && (4.4)\\ &= (\mathbf{x}_i - \mathbf{x}_j)\mathbf{G}^T\mathbf{G}(\mathbf{x}_i - \mathbf{x}_j) && (4.5)\\ &= \|\mathbf{G}(\mathbf{x}_i - \mathbf{x}_j)\|^2 && (4.6) \end{aligned}$$

We then define an error function which has the same minimum as (4.3) but is easier to differentiate and optimize because of its form:

$$E_{\mathcal{A}}(\mathbf{G}) = \sum_{(i,j)\in\mathcal{A}} \frac{1}{4}\left(\hat{d}(i,j)^2 - \|\mathbf{G}(\mathbf{x}_i - \mathbf{x}_j)\|^2\right)^2. \tag{4.7}$$

Here $\mathbf{x}_i - \mathbf{x}_j$ is the pairwise difference between two vectors in the input space $\mathcal{X}$ and $\hat{d}(i,j)^2$ is the squared target distance for this pair of input vectors. And $\|\mathbf{G}(\mathbf{x}_i - \mathbf{x}_j)\|^2$ is our learned squared metric, which is efficient to compute as the inner product of the transformed difference vector with itself. The constant vector $\frac{1}{4}$ is there only for convenience in differentiation.

Interpreting the optimization objective as $E_{\mathcal{A}}(\mathbf{G}) = \sum_{(i,j)\in\mathcal{A}} E_{ij}(\mathbf{G})$ the corresponding gradient of the error E_{ij} with respect to $\mathbf{G}$ on a single pairwise distance constraint $(i,j) \in \mathcal{A}$ is given by:

$$\nabla E_{ij}(\mathbf{G}) = \frac{\delta}{\delta \mathbf{G}} \frac{1}{4}\left(\hat{d}(i,j)^2 - \|\mathbf{G}(\mathbf{x}_i - \mathbf{x}_j)\|^2\right)^2 \tag{4.8}$$

$$= \frac{1}{2}\left(\hat{d}(i,j)^2 - \|\mathbf{G}(\mathbf{x}_i - \mathbf{x}_j)\|^2\right) \frac{\delta}{\delta \mathbf{G}}\left(\hat{d}(i,j)^2 - \|\mathbf{G}(\mathbf{x}_i - \mathbf{x}_j)\|^2\right) \tag{4.9}$$

$$= -\left(\hat{d}(i,j)^2 - \|\mathbf{G}(\mathbf{x}_i - \mathbf{x}_j)\|^2\right) \mathbf{G}(\mathbf{x}_i - \mathbf{x}_j)(\mathbf{x}_i - \mathbf{x}_j)^T. \tag{4.10}$$

We initialize our transformation as the identity transformation $\mathbf{G} = \mathbf{I}$ and then update it at each iteration as:

$$\mathbf{G} \leftarrow \mathbf{G} - \eta_t \nabla E_{ij}(\mathbf{G}). \tag{4.11}$$

Here t is the iteration number, $\nabla E_{ij}(\mathbf{G})$ is the gradient computed over the distance constraint for points $(i,j) \in \mathcal{A}$ and η_t is the learning rate: a gain parameter which controls the step size. The learning rate should decrease and the schedule with which it decreases is important for the convergence speed of the optimization [Xu, 2011]. We calculate η_t at each time step t as follows:

$$\eta_t = \frac{\eta_0}{(1 + \eta_0 t)}, \tag{4.12}$$

where η_0 is a hyper parameter that we set by 2-fold cross validation on the training set.

Chapter 5

Experiments and results

We evaluate the metric learning method described in this thesis in two ways. Firstly, we evaluate how well the learned metric approximates the target metric. In other words, how well does the method align feature space to the structured output space? This can be measured by the mean squared error (MSE) between the learned distances in feature space and the loss in output space. Secondly, we evaluate how much the learned metric improves performance over the default Euclidean metric on the overall prediction task. We do this by measuring the task-specific performance of a kNN classifier using the learned metric and see if this improves over the Euclidean metric. We compare both these results against a baseline metric learning algorithm that uses binary similarity constraints, namely the ITML algorithm. [Davis et al., 2007]

We run these evaluations on two datasets taken from structured prediction problems. The first dataset comes from an attribute-based classification problem, while the second comes from a semantic segmentation problem. Both these datasets have a real-valued loss defined on the structured output space. This real-valued loss allows us to test the idea of aligning the feature space to the output space through metric alignment by using the loss function as the target metric.

5.1 Datasets

5.1.1 Attribute-based classification dataset

Attribute-based classification is an approach to object recognition which can be applied to problems where there is no training data for the test categories. This problem is known as zero-shot or zero-data learning. In attribute-based classification, classes are described by high-level "semantic" attributes. These attributes are properties of an object for which a human can decide whether this property applies to the object. In this way, previously unseen classes can be described by its attributes. By learning the connection between visual features and attributes a classifier can correctly classify members of these unseen classes.

For our evaluation we use the animals with attributes (AwA) dataset[1] described in [Lampert et al., 2009,Lampert et al., 2014]. This dataset contains im-

[1] Available at `http://attributes.kyb.tuebingen.mpg.de/`.

ages of 50 different species of mammals. A set of 85 semantic attributes describes each class and the class-attributes table is available in both a continuous-valued matrix as well as a binary one obtained by thresholding at the mean value.

Image features

Several pre-computed features are distributed along with the AwA set in order to promote repeatability of experiments done using the dataset. We make use of the SURF histogram features. *Speeded up robust features* (SURF), originally presented in [Bay et al., 2006], are scale and rotation invariant image features that are similar to the well-known SIFT feature. [Lowe, 2004] Scale-invariant interest points are detected by finding locally maximal response to an approximate Hessian filter on the lightness values of an image. This gives comparable performance to the Harris-Laplace detector used in SIFT, but because the approximation to the Hessian can be computed from integral images it is faster to compute. A dominant orientation is estimated for each point using Haar-wavelet responses which are also cheap to compute using integral images. The descriptor itself user Haar-wavelets as well: it sums responses to a vertical and horizontal wavelet for a local neighborhood in a 4×4 grid.

The SURF-histogram feature used to describe each image in the AwA set is visual bag of word (BoW) approach to classification based on these SURF features. A visual codebook is created by clustering the SURF features computed from the dataset using the k-means clustering algorithm. Each cluster centroid in descriptor space is then taken as a visual word. The feature vector for a specific image is then formed by computing the SURF descriptors for the image, assigning each descriptor to the closest cluster centroid and creating a histogram of these assignments. The pre-computed features use a codebook of 2000 visual words.

We pre-process these features by first L1 normalizing them. Then we standardize them by subtracting the mean and dividing by the standard deviation. This ensure the data has zero-mean and unit-variance. We further reduce the dimensionality to 128 dimensions using the principal component analysis algorithm in order to make the computations more tractable.

Target metric and constraints

In order to generate the constraints on which we train the metric learning algorithm we sample $100n$ random pairs of points without replacement, where n is the number of feature vectors in the training set. These pairs form the set $\mathcal{A}$ on which our absolute distance constraints are defined. The absolute distance constraints are formed by choosing a target metric and calculating its value for each of the sampled pairs in $\mathcal{A}$. For the AwA set we choose Hamming distance between the attribute vectors corresponding to the classes of the images in the pair. The Hamming distance is calculated as:

$$d_{\mathrm{H}}(i,j) = \|\mathbf{a}_i - \mathbf{a}_j\|_{\mathrm{H}}, \tag{5.1}$$

$$= \sum_{k=1}^{n} \left[a_{i,k} = a_{j,k}\right], \tag{5.2}$$

where $(\mathbf{x}_i, \mathbf{x}_j)$ is one of the sampled pairs in $\mathcal{A}$ and $\mathbf{a}_i$ and $\mathbf{a}_j$ are the corresponding attribute vectors consisting of n binary attribute values $a_{i,k}$. Further

[·] denote Iverson brackets which take the value 1 if the enclosed equation evaluates as true. The absolute distance constraints are expressed as:

$$d_{\mathbf{A}}(\mathbf{x}_i, \mathbf{x}_j) = d_{\mathrm{H}}(i, j) \qquad (i, j) \in \mathcal{A}, \tag{5.3}$$

where $d_{\mathbf{A}}(\mathbf{x}_i, \mathbf{x}_j)$ is the metric distance being learned by our metric alignment algorithm.

In order to train our baseline metric learning algorithm we also need a set of binary similarity constraints. Since each image in the AwA set is assigned to single class, these are easy to generate by constraining the distance between pairs of the same class to be small and pairs of differing classes to be large. We thus divide our sampled pairs $(i, j) \in \mathcal{A}$ into two disjoint sets $\mathcal{S}$ and $\mathcal{D}$ for similar and dissimilar classes respectively. The constraints are defined as:

$$d_{\mathbf{A}}(\mathbf{x}_i, \mathbf{x}_j) \leq u \qquad (i, j) \in \mathcal{S}, \tag{5.4}$$

$$d_{\mathbf{A}}(\mathbf{x}_i, \mathbf{x}_j) \geq \ell \qquad (i, j) \in \mathcal{D}, \tag{5.5}$$

where we set u and l to be the 5^{th} and 95^{th} percentile of the distribution of distances on a small subset of 5000 pairs sampled from the training set.

Task description

We use the AwA set in the context of two learning problems: a standard multiclass classification problem and a zero-shot learning problem as described in [Lampert et al., 2014]. For the multiclass task we randomly divide the images 50/50 into a training and a test set while keeping the distribution among classes the same. We then learn our metric on the training set using the constraint described above. Next we classify the test images using a k-nearest neighbor classifier. The parameter k is tuned by 2-fold cross validation on the training set using the Euclidean metric, and was set to $k = 50$.

In zero-shot learning, the classes in the dataset are split, such that the set of classes in the train set is disjoint from the set of classes in the test set. Therefore, this task can not be solved using a regular k-NN classifier. Lampert et al. describe two different approaches to attribute based classification for zero-shot learning: *direct attribute prediction* and *indirect attribute prediction*. For simplicity we will constrain ourselves to the direct attribute prediction method. In this method a probabilistic classifier is learned that predicts the probability $p(a|\mathbf{x})$: the probability for an attribute value a given the image feature $\mathbf{x}$. We calculate this probability by finding the k-nearest neighbors of the feature vector and calculating the proportion that take the same attribute value:

$$p(a|\mathbf{x}) = \frac{1}{k} \sum_{i=1}^{k} \left[a^i = a \right]. \tag{5.6}$$

Here a^i is the value for the attribute of the i^th nearest neighbor and $[\cdot]$ denotes the Iverson brackets. Using these probabilities the test class z is found through MAP prediction assuming independent attributes:

$$h(\mathbf{x}) = \underset{z \in \mathcal{Z}}{\operatorname{argmax}} \prod_{m=1}^{M} \frac{p(a_m^z|\mathbf{x})}{p(a_m^z)}, \tag{5.7}$$

where $\mathcal{Z}$ is the set of unseen test classes, a_m^z is the m^th element of the attribute vector for class z and $\mathbf{x}$ is the input feature. We calculate $p(a_m^z|\mathbf{x})$ using the k nearest neighbors and we calculate the prior $p(a_m^z)$ as the proportion of training classes having attribute value a_m^z.

5.1.2 Semantic segmentation dataset

Semantic segmentation is a computer vision problem in which the objective is to segment input images into contiguous regions by assigning labels to each pixel. The labels are drawn from a predefined set of object and background class labels. Annotations for the training set are often provided by hand and usually not all pixels receive a label in these annotations.

For our evaluation we use the Microsoft research Cambridge (MSRCv1) dataset.[2] The MSRCv1 set consists of 240 images, segmented into 9 classes. The images are loosely segmented, meaning that the annotations are not pixel-perfect. The segmentations cover most of each image with some of the pixels not having a label. The 9 classes contain both foreground object classes, e.g. car, cow, face, and background classes, e.g. grass, sky. There is limited variation in appearance and scene within each class as the pictures have been taken in the same manner, often around the same location.

Image features

For the MSRCv1 set we use CSIFT color descriptor. [Abdel-Hakim and Farag, 2006] The CSIFT descriptor is calculated the same as regular SIFT descriptors, but instead of calculating it on the gray-scale image, a SIFT descriptor is calculated on the three channels of a color invariance model presented in [Geusebroek et al., 2001]. We calculate the CSIFT descriptors using the color descriptor software from [van de Sande et al., 2011]. From each training image we extract feature vectors $\mathbf{x}$ on a regular hexagonal grid $(\mathbf{u}, \mathbf{v})$, with $\mathbf{x}$ computed from the image area centered on image coordinates (u_i, v_i). Each of these feature vectors $\mathbf{x}_i$ is paired to a label patch $\mathbf{y}$ taken from the training annotation for the image and centered at (u_i, v_i). We again normalize and standardize these features and reduce the dimensionality to 128 dimensions using principal component analysis.

Target metric and constraints

For the MSRCv1 set, the target metric we use for the constraints is the segmentation overlap. This is given as the proportion of overlapping pixel labels between the segmentation patches corresponding to the image patches in the pair (i, j) and is calculated as:

$$d_{\mathrm{P}}(i,j) = 1 - \mathrm{Acc}(\mathbf{y}_i, \mathbf{y}_j) \tag{5.8}$$

$$= 1 - \frac{\sum_{k=1}^{n} [y_{i,k} = y_{j,k}]}{\sum_{k=1}^{n} [y_{i,k} \in \mathcal{L} \wedge y_{j,k} \in \mathcal{L}]}, \tag{5.9}$$

where $\mathbf{y}_i$ is the segmentation patch corresponding to the image patch represented by the feature vector $\mathbf{x}_i$. This patch consists of n elements, here indexed

[2]The MSRC datasets are available at `http://research.microsoft.com/en-us/projects/objectclassrecognition/`.

by k, and $\mathcal{L}$ denotes the set of possible labels. This measure counts the number of pixels that have the same label as a proportion of the number of pixels that have been labeled in both patches, and subtracts this from one to turn it into a distance function.

Because the majority of patches does not overlap at all, we do not sample the pairs globally as we do with the AwA set. Instead, for each point in the training set we find the 100 nearest neighbors and use these neighbor pairs as the set $\mathcal{A}$ for our constraints. These constraints are thus given as:

$$d_{\mathbf{A}}(\mathbf{x}_i, \mathbf{x}_j) = d_{\mathrm{P}}(i, j) \qquad (i, j) \in \mathcal{A}, \tag{5.10}$$

Since the label patches are not assigned to a single class, we cannot generate binary similarity constraints based on class assignment. Instead, we threshold the target metric d_{P}. Because d_{P} ranges from 0 to 1 we set the threshold at 0.5:

$$d_{\mathbf{A}}(\mathbf{x}_i, \mathbf{x}_j) \leq u \qquad \text{if} \quad d_{\mathrm{P}}(i, j) < 0.5, \tag{5.11}$$

$$d_{\mathbf{A}}(\mathbf{x}_i, \mathbf{x}_j) \geq \ell \qquad \text{if} \quad d_{\mathrm{P}}(i, j) \geq 0.5. \tag{5.12}$$

Again we set u,l to be the 5^{th} and 95^{th} percentile of the distances distribution over a sample of pairs from the train set.

Task description

Segmentation benchmarks consist of images $\mathbf{I}$ where each pixel $\mathbf{I}_{(u,v)}$ is usually given as a triple (r, g, b) denoting the color of a single pixel and segmentations $\mathbf{S}$ of which each element $\mathbf{S}_{(u,v)} \in \mathcal{Y}$ assigns a label to pixel $\mathbf{I}_{(u,v)}$ of the image. Here $\mathcal{Y} = \{1, 2, \ldots, n\}$ is a predetermined label set where each label corresponds to one of n visual classes.

We randomly divide the images of each data set 50/50 between a training set and test set. In order to keep the class distribution for each subset similar we split the MSRCv1 set per image sequence, since each sequence of photographs was taken in a single context and thus shows a similar distribution of classes.

For our dataset we extract square patches from the images and the corresponding segmentations. The task is then to create the correct label patch $\mathbf{y}$ given a test feature vector $\mathbf{x}$ extracted from the image patch. We do this by finding the k-nearest neighbors of the feature vector among the training set and combining the corresponding k label patches by majority voting for each pixel separately. The parameter k is tuned by 2-fold cross validation on the training set using the Euclidean metric, and was set to $k = 10$.

5.2 Metric alignment experiment

The first experiment is designed to test how well the metric learning method presented in this thesis can approximate the given target metric, with the target metric being the loss function for the learning problem at hand. If the learned distance in feature space is closely aligned to the loss, this means that the space is well-clustered: small distances in feature space correspond to small loss between the corresponding output structures. This is beneficial for prediction algorithms making predictions in this feature space, an effect which we will evaluate in the next experiment.

Table 5.1: Mean square error (MSE) between learned metric distance and target metric distance on test sets.

	Dataset	
Metric	AwA	MSRCv1
Euclidean	303.86	279.49
ITML	302.55	279.46
Metric Alignment	114.00	279.00

We quantify the alignment between the learned metric and the target metric in the form of a squared prediction error, namely the mean squared error between the distances calculated according to the learned metric and the target metric. We evaluate this on a sample of pairs from the test set, sampled in the same way as the training pairs, either globally or locally. Our MSE evaluation measure on the test set is thus calculated as:

$$\mathrm{MSE}_{\mathcal{T}}(d_{\mathbf{A}}, \hat{d}) = \frac{1}{|\mathcal{T}|} \sum_{(i,j)\in\mathcal{T}} \Big(d_{\mathbf{A}}(\mathbf{x}_i, \mathbf{x}_j) - \hat{d}(i,j) \Big)^2 . \tag{5.13}$$

Here $d_{\mathbf{A}}$ is the learned metric and $\hat{d}$ is the target metric, either d_{H} for the AwA set or d_{P} for the MSRCv1 set. The set $\mathcal{T}$ of pairs (i, j) is the sample of pairs from the test set.

We compare the metric learned by our algorithm against two baselines. Firstly, the Euclidean metric, given by $\mathbf{A} = \mathbf{I}$. Secondly, against a metric learned by information-theoretic metric learning (ITML). [Davis et al., 2007] While the metric alignment method is trained using the absolute distance constraints, ITML is trained using the binary similarity constraints.

5.2.1 Results

For our alignment test we train metric alignment on the absolute distance constraints for both the AwA and the MSRCv1 datasets and compare this against the Euclidean metric and against a metric learned by the ITML algorithm on binary similarity/dissimilarity constraints. For each dataset we sample $100n$ pairs from the test set, where n is the number of points in the test set. We evaluate the mean squared error (MSE) between the learned metric distance between the feature vectors and the target metric distance.

Table 5.1 presents our results. It shows that for the AwA dataset, metric alignment reduces the MSE compared to both the Euclidean metric and the ITML algorithm. However, for the MSRCv1 dataset this effect is negligible. Furthermore, the alignment between the learned metric and the target metric still leaves a considerable error term, showing that the alignment is not complete.

These results show that the metric alignment method is at least able to move the metric on the feature space towards the target metric in the output space and that this alignment is closer in terms of MSE on at least one dataset than that between the original feature space and the target metric and closer than a metric learned by the ITML algorithm on binary similarity constraints and the target metric.

5.3 Structured prediction experiment

Our second experiment tests the effect of metric alignment on the ability of a k-nearest neighbors (kNN) classifier to predict the correct output. kNN is a non-parametric classification method, which means that its decision function is not characterized by a parameterized model. Instead, the class for an input point is decided to be the majority class amongst the most similar points in the training set, the nearest neighbors. Although it requires storing of the whole training set, which may become problematic for large sets, the kNN classification method is simple, requires little parameter tuning and is inherently applicable to multi-class problems. The kNN classifier depends on nearest neighbors having low loss relative to the ground truth output, thus learning a metric with the loss as target metric should improve prediction performance.

We implement our kNN algorithm using fast library for approximate nearest neighbors (FLANN). [Muja and Lowe, 2009] The training features are stored in a set of randomized kd-trees. This allows for fast and parallel retrieval of nearest neighbors. We set the FLANN parameters to create 4 trees and check a maximum of 1024 leafs during retrieval.

We evaluate this experiment on the prediction accuracy. We calculate the normalized multiclass accuracy, which is the mean of the true positive rates of each class:

$$Acc = \frac{1}{|\mathcal{Z}|} \sum_{z \in \mathcal{Z}} \frac{\mathrm{TP}_z}{\mathrm{TP}_z + \mathrm{FN}_z}. \tag{5.14}$$

Here $\mathcal{Z}$ is the set of all test classes: all classes for the multiclass and segmentation tasks, but only the hidden classes for the zero-shot task. Further, TP_z and FN_z denote the true positives respectively false negatives for class $z \in \mathcal{Z}$. Again, we evaluate these measures against the same baselines as in the metric alignment experiment: Euclidean metric and ITML.

5.3.1 Results

For our classification accuracy test we use a kNN algorithm to solve the AwA multiclass classification and zero-shot learning tasks and the MSRCv1 patch segmentation tasks. The neighbors are sought using the original Euclidean feature space, using a metric learned by the ITML algorithm on binary similarity/dissimilarity constraints, and using our metric alignment algorithm on the absolute distance constraints generated using the target metric.

For testing we sample $100n$ pairs from each test set, where n is the number of points in the test set. We evaluate the mean class accuracy of the predicted classes or segmentations. The results are displayed in Table 5.2.

These results clearly show that while the metric learned using the ITML algorithm has no impact on the prediction accuracy, the metric learned using the metric alignment algorithm has a negative effect on the prediction accuracy for the AwA dataset. This is the same set on which the MSE measuring the alignment improved the most.

Table 5.2: Mean class accuracy on the AwA multiclass and zero-shot tasks and MSRCv1 patch segmentation task.

	Dataset		
	AwA		MSRCv1
Metric	multiclass	zero-shot	
Euclidean	0.129	0.252	0.368
ITML	0.128	0.251	0.368
Metric Alignment	0.114	0.210	0.368

Table 5.3: Pearson correlation (Pearson r) between learned metric distances and target metric distances on the test sets.

	Dataset	
Metric	AwA	MSRCv1
Euclidean	0.124	0.166
ITML	0.126	0.166
Metric Alignment	0.100	0.165

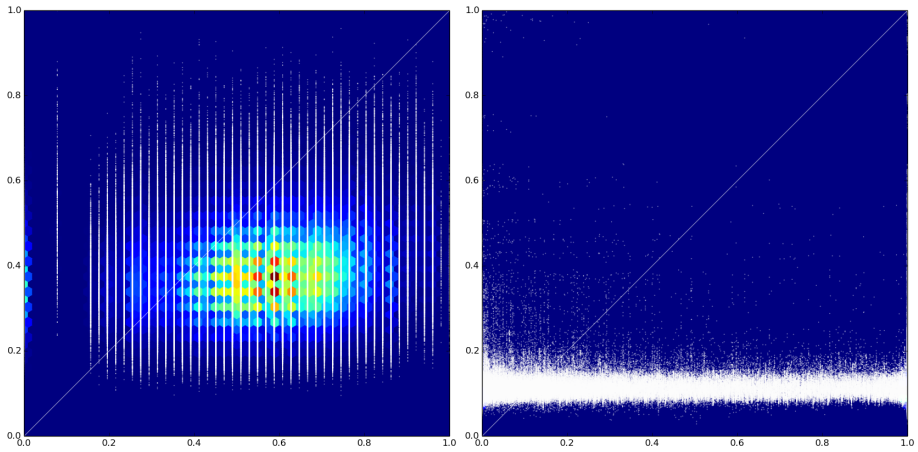
5.4 Analysis

In order to resolve the apparent contradiction between the alignment results as measured by MSE on the fit of the learned metric to the target metric and by the prediction accuracy as measured by the mean class accuracy, we take a closer look at what exactly it is that our metric learning algorithm is learning in the alignment experiment.

Firstly, we look at the Pearson correlation statistic between the learned metric distances and the target metric distances for the sample pairs taken from the test sets. This statistic measures the linear correlation between the two sets of measurements. In other words, this measures the proportionality between the learned metric distance and the target distance. This is exactly the alignment we are trying to achieve through metric learning. It ranges between 1 and -1, with 1 being exact linear correlation, -1 being inverse linear correlation and 0 being no linear correlation. The results are shown in Table 5.3.

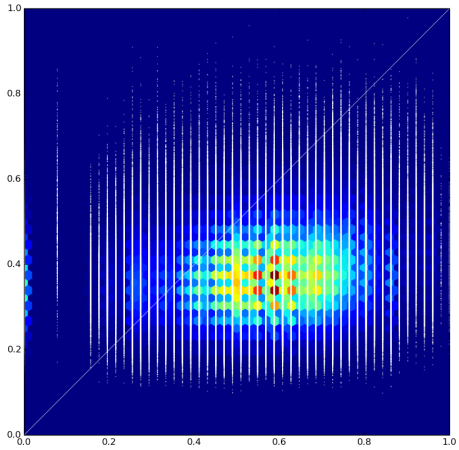
The Pearson correlation results show that our metric alignment algorithm markedly *reduces* the correlation between the learned metric and the target metric. Thus the MSE decreases, since that is the measure that we minimize in our algorithm, but the alignment decreases as well. This is also shown in Figure 5.1. Here we plot the learned metric distances along against the target distances. The learned distances are plotted along the y-axis, scaled to be between 0 and 1, and the target distances are plotted along the x-axis, also scaled.Testing pairs are shown as a white scatter plot, with the density visualized as a color jet with blue being low density and dark red being highest density. Also shown is the diagonal. The more points lying along the diagonal, the better the alignment. The plot for metric alignment seems to show that the distances get squashed in a narrower range of learned distances. This hurts the alignment.

One possible reason why this might happen has to do with our choice for a squared error measure. Since it is squared, this is very vulnerable to outliers. Because of our high-dimensional features there can easily be many pairs with a large distance between them in feature space. This is shown in the scatter plot by the outliers on the top half of the plot and the tendency of the metric learning algorithm to squash these values into a more narrow range.

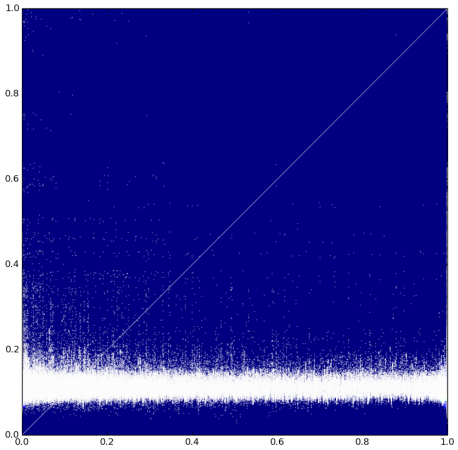


(a) Euclidean AwA

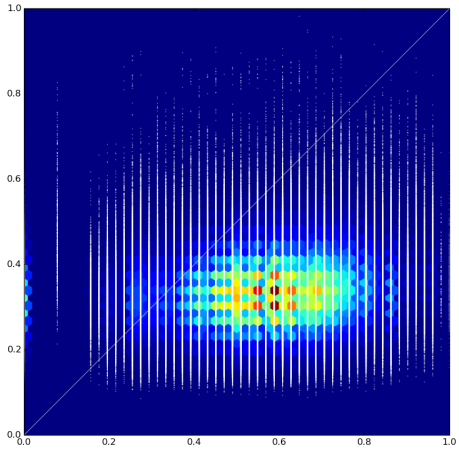
(b) Euclidean MSRCv1



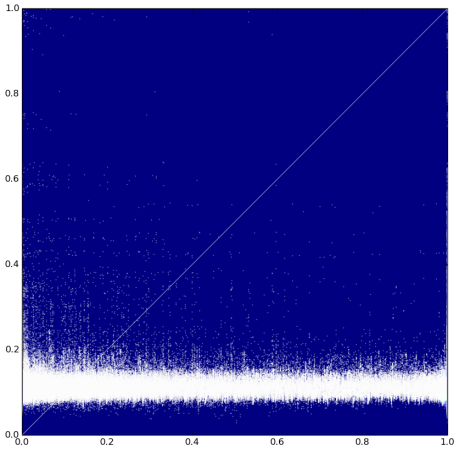
(c) ITML AwA



(d) ITML MSRCv1



(e) Metric Alignment AwA



(f) Metric Alignment MSRCv1

Figure 5.1: Plots showing the alignment between learned and target distances. The learned distances are plotted along the y-axis, scaled to be between 0 and 1, and the target distances are plotted along the x-axis, also scaled. Testing pairs are shown as a white scatter plot, with the density visualized as a color jet with blue being low density and dark red being highest density. Also shown is the diagonal.

Chapter 6

Conclusions

In this thesis we investigated the idea of learning a metric from absolute distance constraints instead of the binary similarity/dissimilarity or relative constraints that are in common use. Some problems, especially structured prediction problems, do not allow for the simplistic approach of dividing data points based on class membership. The complex nature of the structured output space calls for a continuous loss function instead of the 0/1-loss used in simple classification problems. In order to adapt metric learning to such problems we introduced the notion of absolute distance constraints which take the continuous loss function as target metric.

We show that a metric can be learned from the absolute distance constraints by casting this problem as a regression problem on the squared prediction error in regard to the target metric. This can be solved by decomposition of the Mahalanobis matrix such that the metric can be optimized through a stochastic gradient descent algorithm. Our first experiment shows that our method learns a metric that minimizes this the mean of the squared prediction error.

However, the squared measure that we minimize seems vulnerable to outliers. Although our results show that alignment to the target metric is correlated to performance on the prediction task, it also shows that minimizing the mean squared error might not optimize the alignment. This in turn hurts the prediction performance.

Although the learning tasks presented in this thesis proved too challenging for the proposed algorithm, it can be applied to metric learning problems characterized by a continuous loss function, such as structured prediction problems, which can not be handled with metric learning methods relying on binary similarity/dissimilarity constraints or large margin relative constraints. The learning tasks that we introduce form a challenging testing ground for further development in this direction, with the problem lying in both the learning of a metric which has a positive effect on the prediction accuracy, as in the sheer size of the constraint set that is generated which makes efficient optimization difficult.

Bibliography

[Abdel-Hakim and Farag, 2006] Abdel-Hakim, A. E. and Farag, A. A. (2006). CSIFT: A SIFT descriptor with color invariant characteristics. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 1978–1983. IEEE.

[Akata et al., 2013] Akata, Z., Perronnin, F., Harchaoui, Z., and Schmid, C. (2013). Label-embedding for attribute-based classification. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 819–826. IEEE.

[Bay et al., 2006] Bay, H., Tuytelaars, T., and Van Gool, L. (2006). Surf: Speeded up robust features. In *Proceedings of the 9th European Conference on Computer Vision*, pages 404–417. Springer.

[Bellet et al., 2013] Bellet, A., Habrard, A., and Sebban, M. (2013). A survey on metric learning for feature vectors and structured data. Technical report, arXiv:1306.6709 [cs.LG].

[Bengio et al., 2013] Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828.

[Chen and Buja, 2009] Chen, L. and Buja, A. (2009). Local multidimensional scaling for nonlinear dimension reduction, graph drawing, and proximity analysis. *Journal of the American Statistical Association*, 104(485):209–219.

[Chen and Buja, 2013] Chen, L. and Buja, A. (2013). Stress functions for nonlinear dimension reduction, proximity analysis, and graph drawing. *The Journal of Machine Learning Research*, 14(1):1145–1173.

[Davis et al., 2007] Davis, J. V., Kulis, B., Jain, P., Sra, S., and Dhillon, I. S. (2007). Information-theoretic metric learning. In *Proceedings of the 24th International Conference on Machine Learning*, pages 209–216. ACM.

[Frome et al., 2007] Frome, A., Singer, Y., Sha, F., and Malik, J. (2007). Learning globally-consistent local distance functions for shape-based image retrieval and classification. In *IEEE 11th International Conference on Computer Vision*, pages 1–8. IEEE.

[Geusebroek et al., 2001] Geusebroek, J.-M., Van den Boomgaard, R., Smeulders, A. W. M., and Geerts, H. (2001). Color invariance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(12):1338–1350.

[Guillaumin et al., 2009a] Guillaumin, M., Mensink, T., Verbeek, J., and Schmid, C. (2009a). Tagprop: Discriminative metric learning in nearest neighbor models for image auto-annotation. In *IEEE 12th International Conference on Computer Vision*, pages 309–316. IEEE.

[Guillaumin et al., 2009b] Guillaumin, M., Verbeek, J., and Schmid, C. (2009b). Is that you? Metric learning approaches for face identification. In *IEEE 12th International Conference on Computer Vision*, pages 498–505. IEEE.

[Kostinger et al., 2012] Kostinger, M., Hirzer, M., Wohlhart, P., Roth, P. M., and Bischof, H. (2012). Large scale metric learning from equivalence constraints. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2288–2295. IEEE.

[Kulis, 2012] Kulis, B. (2012). Metric learning: A survey. *Foundations and Trends in Machine Learning*, 5(4):287–364.

[Lampert et al., 2009] Lampert, C. H., Nickisch, H., and Harmeling, S. (2009). Learning to detect unseen object classes by between-class attribute transfer. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 951–958. IEEE.

[Lampert et al., 2014] Lampert, C. H., Nickisch, H., and Harmeling, S. (2014). Attribute-based classification for zero-shot visual object categorization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(3):453–465.

[Lowe, 1995] Lowe, D. G. (1995). Similarity metric learning for a variable-kernel classifier. *Neural Computation*, 7(1):72–85.

[Lowe, 2004] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110.

[Mahalanobis, 1936] Mahalanobis, P. C. (1936). On the generalized distance in statistics. *Proceedings of the National Institute of Science, India*, 2(1):49–55.

[McFee and Lanckriet, 2010] McFee, B. and Lanckriet, G. R. (2010). Metric learning to rank. In *Proceedings of the 27th International Conference on Machine Learning*, pages 775–782. IMLS.

[Meyer et al., 2011] Meyer, G., Bonnabel, S., and Sepulchre, R. (2011). Regression on fixed-rank positive semidefinite matrices: A riemannian approach. *The Journal of Machine Learning Research*, 12:593–625.

[Muja and Lowe, 2009] Muja, M. and Lowe, D. G. (2009). Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Application*, pages 331–340. INSTICC.

[Schölkopf et al., 1997] Schölkopf, B., Smola, A., and Müller, K.-R. (1997). Kernel principal component analysis. In *7th International Conference on Artificial Neural Networks*, pages 583–588. ENNS.

[Schultz and Joachims, 2003] Schultz, M. and Joachims, T. (2003). Learning a distance metric from relative comparisons. In *Advances in Neural Information Processing Systems 16*, pages 41–48. NIPS.

[Tenenbaum et al., 2000] Tenenbaum, J. B., De Silva, V., and Langford, J. C. (2000). A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323.

[van de Sande et al., 2011] van de Sande, K. E. A., Gevers, T., and Snoek, C. G. M. (2011). Empowering visual categorization with the gpu. *IEEE Transactions on Multimedia*, 13(1):60–70.

[Venna and Kaski, 2006] Venna, J. and Kaski, S. (2006). Local multidimensional scaling. *Neural Networks*, 19(6):889–899.

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

[Weinberger and Tesauro, 2007] Weinberger, K. Q. and Tesauro, G. (2007). Metric learning for kernel regression. In *International Conference on Artificial Intelligence and Statistics*, pages 612–619. AISTATS.

[Xu, 2011] Xu, W. (2011). Towards optimal one pass large scale learning with averaged stochastic gradient descent. ePrint arXiv:1107.2490 [cs.LG].

[Yang and Jin, 2006] Yang, L. and Jin, R. (2006). Distance metric learning: A comprehensive survey. Technical report, Michigan State Universiy. `http://www.cs.cmu.edu/~liuy/frame_survey_v2.pdf`.