Metric Learning for Kernel Regression

Kilian Q. Weinberger
Computer and Information Sciences
U. of Pennsylvania, Philadelphia, PA 19104
kilianw@seas.upenn.edu

Gerald Tesauro
IBM T.J. Watson Research Center
19 Skyline Drive, Hawthorne, NY 10532
gtesauro@us.ibm.com

Abstract

Kernel regression is a well-established method for nonlinear regression in which the target value for a test point is estimated using a weighted average of the surrounding training samples. The weights are typically obtained by applying a distance-based kernel function to each of the samples, which presumes the existence of a well-defined distance metric. In this paper, we construct a novel algorithm for supervised metric learning, which learns a distance function by directly minimizing the leave-one-out regression error. We show that our algorithm makes kernel regression comparable with the state of the art on several benchmark datasets, and we provide efficient implementation details enabling application to datasets with $\sim$O(10k) instances. Further, we show that our algorithm can be viewed as a supervised variation of PCA and can be used for dimensionality reduction and high dimensional data visualization.

1 Introduction

One of the oldest and most commonly used algorithms for regression is kernel regression [1]. In kernel regression, the target value of a test input is computed as a weighted average of the function values observed at training inputs. The weight of each training input is computed using a *kernel function*, which typically decays rapidly with distance between itself and the test input. This is so that the estimated test point value has strongest dependence on nearby training points. A common approach in kernel regression combines a Euclidean distance metric with Gaussian kernels, which decay exponentially with squared distance rescaled by a kernel width factor.

We note two important drawbacks of standard methods for kernel regression. First, they require an *a priori* well-defined distance metric on the input space, which may preclude their usage in datasets where such metrics are not meaningful. For example, the well-known Boston housing dataset contains 13 input features representing completely disparate quantities such as pollution levels, crime rates, pupil-teacher ratios, etc.. Secondly, a pre-defined distance metric may not be particularly relevant to the regression task at hand. For example, if certain input features are completely irrelevant to the regression task, they ideally should not contribute at all to the distance metric, whereas a Euclidean distance would ascribe to them as much weight as the most significant features.

Inspired by recent innovations on metric learning for classification [3] and feature selection [8], we propose a novel method to learn a task-specific (pseudo-)metric over the input space in which small distances between two vectors imply similar target values. This metric gives rise to an appropriate kernel function with parameters set entirely from the data. In addition to performing regression, we will also show that our algorithm, which we refer to as *Metric Learning for Kernel*

Regression (MLKR), can be viewed as an algorithm for dimensionality reduction. This opens the possibility to use MLKR as a pre-processing tool for a whole variety of independent machine learning algorithms and data visualization. Data visualization can be particularly helpful to gain deeper understanding of the underlying structure and behavior of the target function with respect to the input.

This paper is organized as follows: Section 2 establishes notation and briefly gives background on the specifics of kernel regression. In Section 3, we introduce our algorithm for supervised distance metric learning. Section 4 illustrates how our algorithm can be interpreted as a variation of Principal Component Analysis [6], and how it can be used for task-specific dimensionality reduction. In Section 5 we evaluate the capabilities of MLKR for data visualization on a synthetically generated data set, and compare its regression accuracy on several standard data sets with various common regression algorithms. Section 6 compares our algorithm to related published work, and we conclude in Section 7.

2 Kernel regression problem setting

The standard regression task is to estimate an unknown function $f : \mathcal{R}^D \rightarrow \mathcal{R}$ based solely on a training set of (possibly noisy) evaluations $(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n)$. Here $y_i = f(\vec{x}_i) + \epsilon$, where ϵ represents some small noise. Specifically, for some distribution of test points $\{\vec{x}_t\}$, our task is to find an estimator $\hat{f} : \mathcal{R}^D \rightarrow \mathcal{R}$ of f that minimizes some loss function (e.g., squared error) over the distribution of test points.

In kernel regression, the value of $\hat{y}_t \approx f(\vec{x}_t)$ is approximated by a weighted average of the training samples:

$$\hat{y}_i = \frac{\sum_{j \neq i} y_j k_{ij}}{\sum_{j \neq i} k_{ij}} \tag{1}$$

where $k_{ij} = k(\vec{x}_i, \vec{x}_t) \geq 0$ is referred to as the *kernel function*. A wide variety of kernel functions have been studied in prior literature, e.g., Gaussian, triangular, spherical, wave, etc.. To make the estimate $\hat{y}_t$ truly local, the kernel function $k(\vec{x}_i, \vec{x}_t)$ should decay rapidly with increasing (squared) distance $d(\vec{x}_i, \vec{x}_t)$. The optimal rate of decay depends on the noisiness and smoothness of the function f, the density of the training samples, and the scale of the input features. Hence the choice of kernel function may require great care, especially in high dimensional spaces. Several publications elaborate on how to chose the right kernel function for a given data set [11]. Additionally, kernel functions may have several tunable parameters pertaining to the rate of decay; these also must be set properly to obtain good performance on the regression task.

3 Metric learning algorithm

We now present our algorithm *Metric Learning for Kernel Regression* (MLKR) for learning an appropriate distance function for kernel regression. Our approach applies to any distanced-based kernel function $k(\vec{x}_i, \vec{x}_j) = k_D(D_\theta(\vec{x}_i, \vec{x}_j))$ with differentiable dependence on parameters θ specifying the distance function D_θ. Specifically, MLKR consists of setting initial values of θ, and then adjusting the values using a gradient descent procedure:

$$\Delta\theta = -\epsilon \frac{\partial \mathcal{L}}{\partial \theta} \tag{2}$$

where ϵ is an adaptive step-size, and the loss function $\mathcal{L}$ is the cumulative leave-one-out quadratic regression error of the training samples:

$$\mathcal{L} = \sum_i (y_i - \hat{y}_i)^2 \tag{3}$$

with $\hat{y}_i$ defined as in (1). Of course, other methods for minimizing (3) such as conjugate gradient, stochastic gradient or BFGS may also be used and might lead to faster convergence results.

While MLKR may apply to many types of kernel functions and distance metrics, we hereafter focus the exposition on a particular instance of the Gaussian kernel and Mahalanobis metric, as these are used in our empirical work. The Gaussian kernel is generally defined as follows:

$$k_{ij} = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{d(\vec{x}_i, \vec{x}_j)}{\sigma^2}}, \tag{4}$$

where $d(\vec{x}_i, \vec{x}_j)$ is the squared distance between the vectors $\vec{x}_i$ and $\vec{x}_j$. As the constant factor before the exponent in eq. (4) cancels out in eq.

(1), we will drop it for simplification. Also, we will absorb σ^2 in $d()$ and can fix $\sigma = 1$.

A Mahalanobis metric is a generalization of the Euclidean metric, in which the squared distance between two vectors $\vec{x}_i$ and $\vec{x}_j$ is defined as

$$d(\vec{x}_i, \vec{x}_j) = (\vec{x}_i - \vec{x}_j)^\top \mathbf{M} (\vec{x}_i - \vec{x}_j), \qquad (5)$$

where $\mathbf{M}$ can be any symmetric positive semidefinite[1] real matrix. (Setting $\mathbf{M}$ to the identity matrix recovers the standard Euclidean metric.) Figure 1 illustrates the difference between a Mahalanobis metric and the Euclidean metric on a synthetic regression example.

Unfortunately, learning the matrix $\mathbf{M}$ directly requires enforcing a positive semi-definite constraint during optimization. This is non-linear and expensive to satisfy. One way to eliminate this expensive constraint is to decompose $\mathbf{M}$ as

$$\mathbf{M} = \mathbf{A}^\top \mathbf{A}, \qquad (6)$$

where the i^{th} row of $\mathbf{A}$ is the vector $\sqrt{\lambda_i}\vec{v}_i^\top$, where $\vec{v}_i$ is the i^{th} eigenvector and λ_i is the corresponding eigenvalue of $\mathbf{M}$. Substituting (6) into (5) enables us to express the Mahalanobis distance as the Euclidean distance after the mapping $\vec{x} \rightarrow \mathbf{A}\vec{x}$:

$$d(\vec{x}_i, \vec{x}_j) = \|\mathbf{A}(\vec{x}_i - \vec{x}_j)\|^2. \qquad (7)$$

Equation (6) allows us to rewrite (3) in terms of the unconstrained matrix $\mathbf{A}$, instead of $\mathbf{M}$. Let $\hat{y}_i$ be the target estimate of vector $f(\vec{x}_i)$. The gradient of (3) with respect to $\mathbf{A}$ can be stated as

$$\frac{\partial \mathcal{L}}{\partial \mathbf{A}} = 4\mathbf{A} \sum_i (\hat{y}_i - y_i) \sum_j (\hat{y}_j - y_j) k_{ij} \vec{x}_{ij} \vec{x}_{ij}^\top, \quad (8)$$

where $\vec{x}_{ij} = (\vec{x}_i - \vec{x}_j)$. We note that minimizing (3) gives us a kernel function in a entirely data-driven, non-parametric way. There are no parameters to tune, apart from the initialization of $\mathbf{A}$ and the gradient stepsize. Also, the algorithm learns its own scaling (in the form of $\mathbf{A}$) and is therefore invariant to the scaling of the input vectors. In our experiments, we usually initialized with the identity matrix. If local minima are a concern, one can use several runs with different random initializations and choose the outcome with minimum training error $\mathcal{L}$.

[1] A matrix $\mathbf{M}$ is called positive semi-definite if all its eigenvalues are non-negative. This is the case if and only if for any real vector $\vec{v}$ the following holds: $\vec{v}^\top \mathbf{M} \vec{v} \geq 0$.

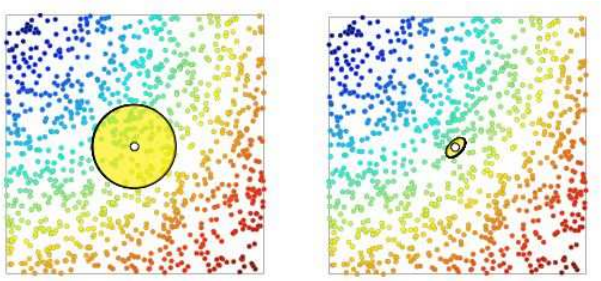

Figure 1: An illustration of kernel regression under varying distance metrics. The color of the points represents the function value. The circle shows the radius that encapsulates 95% of the weights. The function value is estimated at a test point at the center. *Left:* With the use of the Euclidean Distance metric, the kernel is spherical and ignores the present structure that points along the diagonal share similar function values. *Right:* After training, under the Mahalanobis metric, the kernel function follows the direction of the target function. The radius has also shrunk and has therefore adapted to the relatively densely sampled and noiseless training samples.

Efficient optimization

A naïve computation of the gradient (8) requires us to compute the outer product between all pair of vectors $\vec{x}_i, \vec{x}_j$. This would result in a complexity of order $O(D^2n^2)$. Luckily, we can approximate the gradient by making use of the fast decay of k_{ij}. In our implementation we only considered the $c = 1000$ nearest neighbors of each vector $\vec{x}_i$ and also disregarded weights of very small values ($k_{ij} < e^{-34}$). Both cutoff values turned out to be overly conservative and seemed to have no noticeable influence on the quality of the outcome. To avoid an expensive nearest neighbor search in each iteration, we cached the nearest neighbors of each point in a simple heap-tree data structure, which we updated every 15 gradient steps. The new complexity ($O(D^2nc + n^2)$) is still quadratic in n, but only for the rare nearest neighbor search when the heap tree is corrected. These two changes al-

lowed us to apply our algorithm on data sets of size $n = 30,000$ and higher. For even bigger data sets, the quadratic nearest neighbor search could be sped up with sophisticated tree data structures, such as cover trees [2] or kd-trees [7].

4 Dimensionality reduction

Many machine learning algorithms require some sort of dimensionality reduction as pre-processing of the input data. Recently, there have been several new methods for linear dimensionality reduction [5]. In this section, we illustrate that MLKR can also be understood as a supervised version of principal component analysis [6], that - in contrast to PCA - is independent of the scaling of the input features and generates locally meaningful distances with respect to the regression value.

Principal Component Analysis

One of the most commonly used algorithms for dimensionality reduction is Principal Component Analysis (PCA) [6]. PCA projects the input vectors onto the low dimensional sub-space that minimizes the sum-squared reconstruction-error of the input vectors. It has been shown that the principal components are the leading eigenvectors of the covariance matrix (of the centered inputs $\vec{x}_i$ - up to a constant factor $\frac{1}{n}$)

$$\mathbf{C} = \sum_i \vec{x}_i \vec{x}_i^\top. \tag{9}$$

One major drawback of PCA is that the outcome heavily depends on the scaling of the input features. For example, if the individual features of $\vec{x}_i$ contain time measures, and we change the unit of one feature from milliseconds to hours, the resulting embedding found with PCA would be drastically different. A second disadvantage of PCA is that the projection is entirely unsupervised and ignores any side information that might be known about the input data.

We can show that learning the matrix $\mathbf{M}$ from section 3 is equivalent to learning a specially constructed "covariance" matrix for PCA. In comparison to $\mathbf{C}$, $\mathbf{M}$ is independent of the scale of the individual input dimensions. For now let us assume that the covariance matrix of the input is the identity matrix (if not, the input can always be whitened through multiplication by the matrix $\mathbf{C}^{-\frac{1}{2}}$.)[2] Let $\mathbf{M} = \mathbf{A}^\top\mathbf{A}$ be the solution of minimizing (3). The covariance matrix (9) of the input after the mapping $\vec{x}_i \to \mathbf{A}\vec{x}_i$ is the matrix $\mathbf{C}^{\mathbf{A}} = \mathbf{A}\mathbf{A}^\top$ (because of the prior whitening). Let $\vec{v}$ be the leading eigenvector of $\mathbf{C}^{\mathbf{A}}$. One way to combine MLKR with dimensionality reduction would be to first map $\vec{x}_i \to \mathbf{A}\vec{x}_i$ and then perform PCA after the mapping. More explicitly, we obtain the first dimension of our embedding by

$$\vec{x}_i \to \vec{v}^\top \mathbf{A}\vec{x}_i. \tag{10}$$

Let $\vec{w}$ be the leading eigenvector of $\mathbf{M}$. It is easy to verify that $\vec{w} = \mathbf{A}^\top\vec{v}$. Hence, the mapping in eq. (10) is equivalent to $\vec{x}_i \to \vec{w}^\top\vec{x}_i$.

It follows that if we project onto the leading eigenvectors of $\mathbf{M}$, the mapping $\vec{x}_i \to \mathbf{A}\vec{x}_i$ is in fact PCA in the projected space. This gives us an effective method for supervised dimensionality reduction. In other words, by learning the matrix $\mathbf{M}$, we learn a covariance matrix on which we can perform PCA.

Additionally, if the desired dimensionality is known before the optimization, the dimensionality reduction can be directly incorporated into the minimization of (3). $\mathbf{M}$ can be constraint to be of low rank by choosing $\mathbf{A}$ to be rectangular (ie $\mathbf{A} \in \mathcal{R}^{r\times D}$) which naturally leads to a mapping $\mathbf{A} : \mathcal{R}^D \to \mathcal{R}^r$, where $r < D$.

5 Results

We performed several tests to evaluate the capabilities of MLKR. First we demonstrate the abilities of MLKR for dimensionality reduction and feature selection on an artificially generated data set. Further, we conducted extensive tests on standard regression data sets and compared MLKR with a variety of different state of the art algorithms. We minimized the objective (3) with the simple gradient descent algorithm (2). In cases where the normalizer in (4) leads to division by zero (due to inadequate machine preci-

[2]Please note, that whitening the input does not change the best solution of MLKR. If $\mathbf{A}$ is the best solution for the original vectors then $\mathbf{A}\mathbf{C}^{\frac{1}{2}}$ leads to the same results for the whitened vectors.

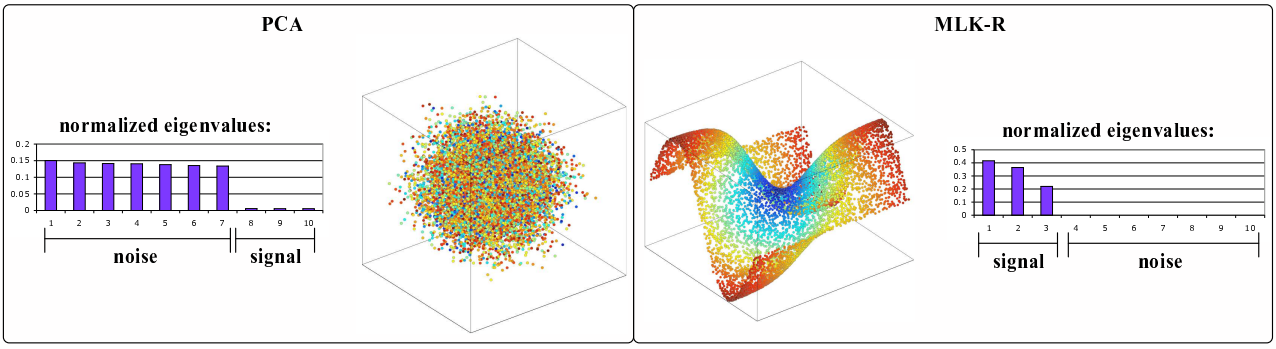

Figure 2: The result of a 3d projection of a manifold data set under PCA and MLKR. *Left:* The projection onto the leading PCA principal components projects directly onto the noise and ignores the signal. The seven large eigenvalues would suggest to keep the noise and ignore the signal. *Right:* The MLKR projection finds the three relevant features and projects out the noise perfectly. The three eigenvalues suggest to keep the signal and cut off the noise.

sion), we assigned the target value of the closest training point.

Supervised dimensionality reduction

We tested the abilities of MLKR for dimensionality reduction on a synthetic data set. We generated $n = 8000$ input vectors of dimensionality $D = 10$. The input data was constructed as follows: First, we sampled data points from a three dimensional "twin-peak" manifold with $\vec{x}_i = (r_i, s_i, \sin(r_i)\tanh(s_i))^\top$ for some randomly sampled points (r_i, s_i). The target value was computed as $y_i = \|\vec{x}_i\|_2$. The dataset was then embedded into a ten dimensional space by adding seven dimensions of high uniform noise (the noise level was set to 1000% of the original input magnitude.) Finally, we multiplied the input vectors by a randomly generated rotation matrix which resulted in all ten input dimensions containing a comparable mixture of noise and signal.

Figure 2 shows the low dimensional projections of these input vectors found with PCA and MLKR. As the noise has much higher magnitude than the input signal, PCA returns principal components that project entirely onto the high variance noise and therefore ignore the relevant signal. The eigenvalue spectrum of the covariance matrix reveal that the (low-variance) signal is represented by the bottom eigenvectors, which are generally dropped.

MLKR on the other hand learns a transformation of the space that suppresses the noise entirely and restores the signal. The final result is a perfect recovery of the clean input signal. The three non-zero eigenvalues indicated that MLKR has recovered the three dimensions that are relevant to the target function.

In addition to the synthetically generated data, we also applied MLKR to images from the FG-NET face data set[3]. The FG-NET data base consists of $n = 984$ frontal face images of 82 persons. The images are accompanied by the age of each person, at the time when the image was taken, as additional side information. We preprocessed the images by aligning the eyes (through rotation and translation), cropping the edges and rescaling the images to 50×50 pixels. We then applied MLKR to the set of 100 dimensional eigenfaces (after removing some distinct outliers).

Figure 3 displays the two dimensional projection of images obtained with MLKR and PCA. For better visualization, we superimposed several representative images on top of their low dimensional coordinates. The MLKR representation shows a clear correlation between position and age (the arrows highlight the direction of increasing age), whereas the PCA representation seems mostly influenced by angle and illumination.

[3]More information on the FG-NET face image data set is available at http://fgnet.cycollege-pms.net/

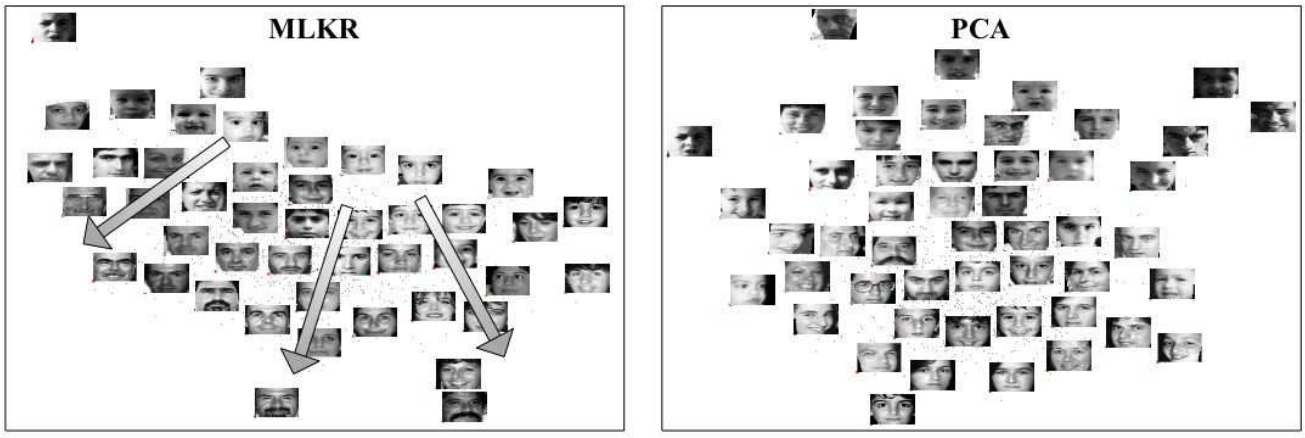

Figure 3: The two dimensional projections of the FG-NET face data set set obtained with PCA and MLKR. A number of representative images are superimposed on top of their low dimensional coordinates. MLKR was trained with the persons age as target and aligns the images accordingly (the arrows indicate age increasing directions) whereas the PCA embedding correlates images mostly according to viewing angle and illumination.

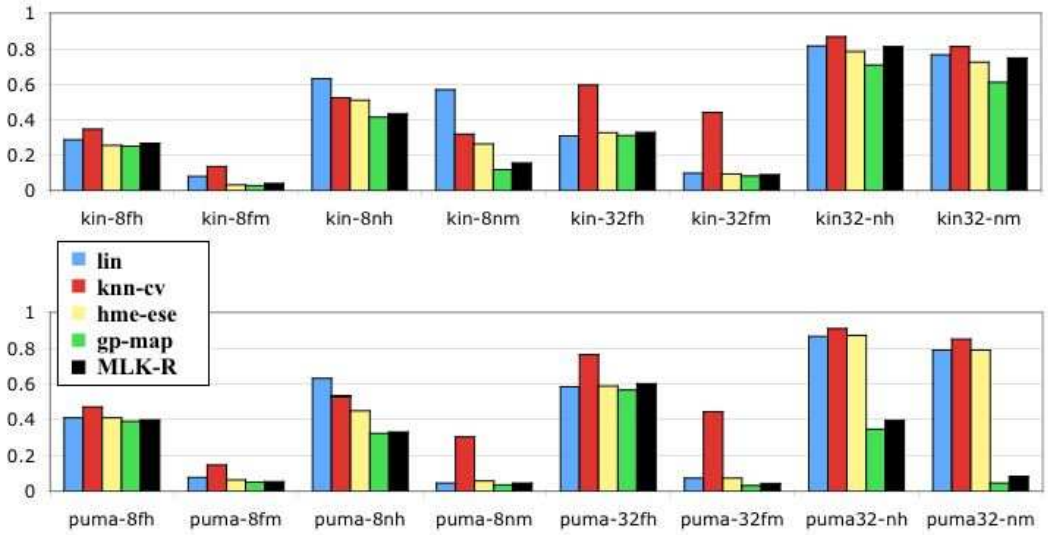

Figure 4: The regression error on various robot arm regression data sets, obtained from the *Delve* package. The applied algorithms are linear regression (lin-1), k-nearest neighbor regression with cross validation for parameter setting (knn-cv-1), hierarchical mixture of experts trained with early stopping (hme-ese) [12], Gaussian processes for regression [10] (gp-map-1) and MLKR.

Regression

In addition to the data sets from the previous section, we also evaluated our algorithm on several standard regression problems from the *Data for Evaluating Learning in Valid Experiments (Delve)* collection[4]. We used sixteen different data sets generated by two synthetic robot arms[5]. Half of the sixteen data sets had 32 dimensions and the other half were of dimension 8.

The *Delve* evaluation package randomly split each data set into four disjoint training sets of size

[4]The Delve package is available at: http://www.cs.toronto.edu/~delve/

[5]For more details on the specific data please see http://www.cs.toronto.edu/~delve/data/pumadyn/desc.html and http://www.cs.toronto.edu/~delve/data/kin/desc.html

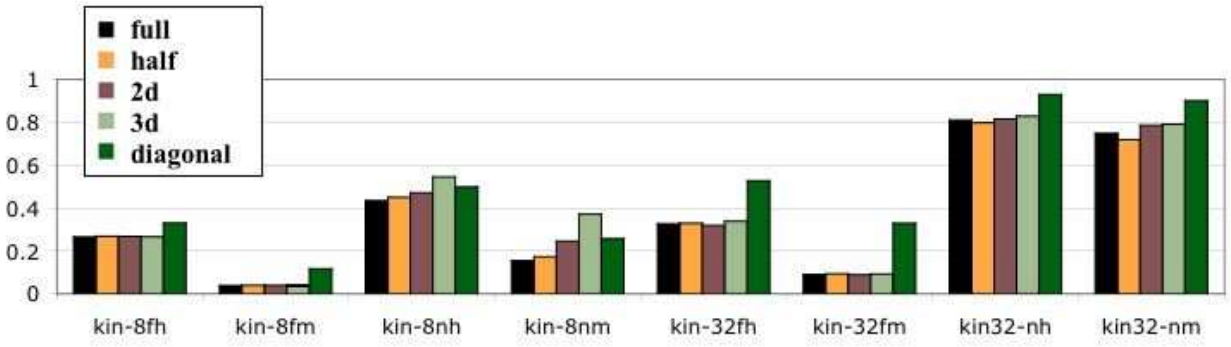

Figure 5: The results of MLKR under varying matrix restrictions. We compared the results of MLKR with a full matrix, a $2 \times d$, $3 \times d$, $\frac{d}{2} \times d$ and a diagonal matrix. The results of the low-rank constraint variations are surprisingly close to the full rank version of MLKR.

$n = 1024$ and one unique test set with 1024 instances. The squared error is computed from the results of the four individual runs. We would like to point out that MLKR could be trained on much bigger data sets (we successfully ran it on proprietary data of size $n > 30000$). We chose the *Delve* evaluation package as an objective method to compare with previously published work. All runs finished within a matter of several minutes. To avoid overfitting, we trained all versions of MLKR with a validation data set for early stopping (of size 10% of the training data).

Figure 4 shows the sum-squared-test error of six different algorithms. We compared MLKR with four standard regression algorithms: linear regression, k-nearest neighbor regression with cross validation for parameter setting, hierarchical mixture of experts trained with early stopping [12] and Gaussian processes for regression [10]. All results are relative to a baseline algorithm which is based entirely on the mean and median of the training target values[6].

It can be observed that MLKR outperforms most alternative algorithms consistently and is almost as precise as Gaussian Process Regression, an algorithm generally considered state of the art.

In addition to the full matrix version, we also evaluated four constrained variants of MLKR. First we constrained MLKR to learn a rectangular matrix that reduces the dimensionality of the input from $d \in \{8, 32\}$ to $2, 3, \frac{d}{2}$. We also constrained the matrix $\mathbf{A}$ to be diagonal, which simply rescales the input features. Figure 5 shows that restricting the dimensionality to $3d$ or even $2d$ has surprisingly little impact on the quality of the regression.

[6]For more information see http://www.cs.toronto.edu/~delve/methods/base-1/home.html

6 Related Work

Over the last several years there have been many different algorithms on metric learning and kernel regression. As far as we know, our work is the first to combine these two fields.

In 2005 Goldberger et al. introduced a novel algorithm for metric learning, which they refer to as neighbourhood component analysis (NCA) [3]. (Our work was inspired by this publication.) Similarly to MLKR, the authors learn a Mahalanobis metric over the input space which appears in the objective function in terms of a soft-max distribution. The main differences between NCA and MLKR are that MLKR minimizes a regression error loss function, whereas NCA is designed for classification, and that NCA's probabilistic formulation assumes a specific soft-max probability model.

In 2006, Takeda et al. introduced an algorithm for kernel regression [4]. Similarly to our work, Takeda et al. can be interpreted as learning a Mahalanobis matrix for a Gaussian regression ker-

nel. However, instead of function estimation, the authors apply their algorithm on image denoising. In contrast to MLKR, which learns the Mahalanobis matrix via loss function minimization, Takeda et al. estimate the covariance matrix from statistics of the local pixel space.

Navot et al. published a novel and effective method for nearest neighbor feature selection [8] in 2006. Their method learns weights for each input feature as an indication of its significance. In comparison, MLKR learns a linear transformation for dimensionality reduction and regression.

Finally, Keller et al. (2006) [9] applied neighborhood component regression to function approximation for reinforcement learning. Our paper focusses more on the adaptation of NCA for kernel regression.

7 Conclusion

We introduced a novel metric learning algorithm for kernel regression, which we refer to as *Metric Learning for Kernel Regression*. We demonstrated that the algorithm can give rise to state-of-the-art regression results on several standard regression benchmark data sets. We also illustrated that MLKR can be used for dimensionality reduction and can be viewed as a supervised alternative to PCA.

As future work, we are also interested in exploring the possibilities of combining the learned metric with various other machine learning algorithms. We believe that MLKR can be combined with recent work in non-linear dimensionality reduction, e.g., Maximum Variance Unfolding (MVU) [13]. One could view MLKR as a first step that learns a metric with local meaningful distances and MVU as a second step that finds a non-linear mapping into a low dimensional representation that preserves these local distances. We hope that this two step approach will allow function approximation in very high dimensional spaces with (relatively) few training samples.

Acknowledgements

The authors would like to thank Amir Navot, Sam Roweis, Lawrence Saul and Lyle Ungar for helpful discussions. Kilian Weinberger would also like to thank Irina Rish, Rajarshi Das and Alina Beygelzimer for stimulating discussion during his time spent at IBM.

References

[1] J. K. Benedetti. On the nonparametric estimation of regression functions. *Journal of the Royal Statistical Society*, 39:248–253, 1977.

[2] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *Proc. of the 23rd Intl. Conf. on Machine Learning*, Pittsburgh, USA, 2006.

[3] J. Goldberger, S. Roweis, G. Hinton, and R. Salakhutdinov. Neighbourhood components analysis. In L. K. Saul et al., editors, *Advances in Neural Information Processing Systems 17*, pages 513–520. MIT Press, 2005.

[4] P. Milanfar H. Takeda, S. Farsiu. Robust kernel regression for restoration and reconstruction of images from sparse, noisy data. In *Intl. Conf. on Image Processing*. Atlanta, GA, 2006.

[5] X. He and P. Niyogi. Locality preserving projections. In S. Thrun et al., editors, *Advances in Neural Information Processing Systems 16*. MIT Press, 2004.

[6] I. T. Jolliffe. *Principal Component Analysis*. Springer-Verlag, New York, 1986.

[7] A. Moore. A tutorial on kd-trees. Extract from PhD Thesis, 1991. Available from http://www.cs.cmu.edu/*sim*awm/papers.html.

[8] A. Navot, L. Shpigelman, N. Tishby, and E. Vaadia. Nearest neighbor based feature selection for regression and its application to neural activity. In Y. Weiss et al., editors, *Advances in Neural Information Processing Systems 18*, pages 995–1002. MIT Press, 2006.

[9] D. Precup P. Keller, S. Mannor. Automatic basis function construction for approximate dynamic programming and reinforcement learning. In *Proceedings of the Twenty-third International Conference on Machine Learning (ICML 2006)*, Pittsburgh, U.S.A., 2006.

[10] C. E. Rasmussen and C. K. I Williams. *Gaussian Processes for Machine Learning*. Springer-Verlag, Cambridge, MA, 2006.

[11] H.G. Müller T. Gasser. *Kernel Estimation of Regression Functions.*, pages 23–67. Springer-Verlag, Heidelberg, 1979.

[12] D. S. Touretzky et al., editors. *Constructive Algorithms for Hierarchical Mixtures of Experts*. MIT Press, 1996.

[13] K. Q. Weinberger and L. K. Saul. Unsupervised learning of image manifolds by semidefinite programming. *Intl. Journal of Computer Vision*, 70(1):77–90, 2006.