NON-LINEAR METRIC LEARNING

Jeppe Johan Waarkjær Olsen
Christian Petersen

Technical University of Denmark

ABSTRACT

Following some of the interest in metric learning, the article will discuss the theory of metric learning, and investigate the Non-linear-Large-Margin-Nearest-Neighbour(NLMNN)[1]. By introducing a new optimization schema, based on simplex projection[2][3], the model is able to achieve better results when testing on both synthetic- and real data, compared with regular classification algorithms, such as K-Nearest-Neighbours and Large-Margin-Nearest-Neighbour(LMNN)[4].

***Index Terms*—** Metric Learning, χ^2-distance, K-Nearest-Neighbours

1. INTRODUCTION

Measuring similarities between examples is often of interest in many applications. One can even create regression and classification models based on similarity metrics like the K-Nearest-Neighbour (KNN) algorithm, that just makes predictions based on the euclidean distance to its neighbours. While simple and popular, KNN often falls short due to it only using the euclidean distance, which has lead researcher to come up with more adaptive ways of learning similarities. This paper will focus on the Large-Margin-Nearest-Neighbour (LMNN) model [4], which introduces a learn-able transformation, and the non-linear χ^2-LMNN [1], bases on the χ^2-distance.

2. METRIC LEARNING

A metric is defined as a similarity measure between two elements. In the KNN algorithm, the metric of choise is the euclidean distance, defined as the l_2-norm of the difference:

$$\mathcal{D}(\mathbf{x_i}, \mathbf{x_j}) = \|\mathbf{x_i} - \mathbf{x_j}\|_2 \tag{1}$$

where $\mathbf{x_i} \in \mathbb{R}^d$ is a vector of features for the ith datapoint. In the LMNN algorithm by Weinberger et al.[4] a learn-able linear transformation $\mathbf{L} \in \mathbb{R}^{d \times d}$ was introduced, to give a more flexible metric

$$\mathcal{D}_L(\mathbf{x_i}, \mathbf{x_j}) = \|\mathbf{L}(\mathbf{x_i} - \mathbf{x_j})\|_2 \tag{2}$$

In practice one uses the square of the distance

$$\mathcal{D}_L(\mathbf{x_i}, \mathbf{x_j})^2 = \|\mathbf{L}(\mathbf{x_i} - \mathbf{x_j})\|_2^2 = (\mathbf{x_i} - \mathbf{x_j})^T \mathbf{L}^T \mathbf{L} (\mathbf{x_i} - \mathbf{x_j}) \tag{3}$$

If we let $\mathbf{M} = \mathbf{L}^T\mathbf{L}$ we see that this is just a mahalanobis distance.

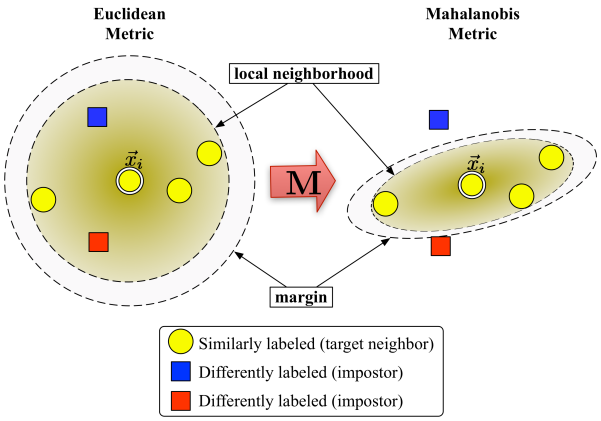

Fig. 1. Visualization of the transformation learned in LMNN (taken from [4])

In order to train the transformation, a few definitions are needed. Given a set of data points $\{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n}\} \in \mathbb{R}^d$, with labels $\{y_1, y_2, \ldots, y_n\} \in \{1, 2, \ldots, C\}$, the target neighbours of $\mathbf{x_i}$ are the set of k nearest neighbours, with the same label as y_i, prior to any transformation of the input. We will let $j \rightsquigarrow i$ denote if x_j is a target neighbour of x_j. Similar for each target neighbour x_j of data point x_i ,we define the imposters x_k as the set of points with different label, $y_i \neq y_k$, that are closer than the target neighbour plus a margin l, after the linear transformation, meaning that

$$\mathcal{D}_L(x_i, x_k)^2 \leq \mathcal{D}_L(x_i, x_j)^2 + \ell \tag{4}$$

This is visualized on Fig.1, where we see that we want learn the transform such that the imposters no longer are within the local neighbourhood. Using this, we can define the objective function we want our transformation L to minimize:

$$\min_{\mathbf{L}} \mathcal{L} = \sum_{i,j:j\rightsquigarrow i} \mathcal{D}_{\mathbf{L}}\left(\mathbf{x}_i, \mathbf{x}_j\right)^2 \quad + \\ \mu \sum_{k:y_k \neq y_i} \left[\ell + \mathcal{D}_{\mathbf{L}}\left(\mathbf{x}_i, \mathbf{x}_j\right)^2 - \mathcal{D}_{\mathbf{L}}\left(\mathbf{x}_i, \mathbf{x}_k\right)^2\right]_+ \tag{5}$$

where $[\cdot]_+ = max(\cdot, 0)$ is the hinge loss. It is argued in [4], that the margin parameter ℓ can be set to 1, since it should result in the same transformation up to a scaling factor. The loss function consists of two terms. The first term minimizes the loss by pulling the target neighbours closer, and the second term penalizes the imposters.

2.1. Non-Linear Metric Learning

The non-linear χ^2-LMNN (or simply NLMNN) method is the adaption of the regular LMNN algorithm. Where LMNN uses the mahalanobis distance measure, NLMNN uses the χ^2-distance defined as:

$$\chi^2\left(\mathbf{x}_i, \mathbf{x}_j\right) = \frac{1}{2} \sum_{f=1}^{d} \frac{\left([\mathbf{x}_i]_f - [\mathbf{x}_j]_f\right)^2}{[\mathbf{x}_i]_f + [\mathbf{x}_j]_f} \tag{6}$$

The χ^2-distance is only defined on histogram data, meaning that each datapoint $\mathbf{x}$ lies within the probability simplex $\mathcal{S}^d = \{\mathbf{x} \in \mathbb{R}^d | x_i \geq 0 \quad \forall i \in \{1, .., d\}, \sum_i x_i = 1\}$, which just corresponds to the features being non-negative and summing to one. This might seem like a strict constraint, but many problems work on histogram data, like bag of words representations for Natural Language Processing(NLP), and visual codebooks for computer vision, which can be normalized to lie in the simplex.

Similar to LMNN, NLMNN uses a learnable parameter L, which gives rise to a non-linear transformation.

$$\chi^2_{\mathbf{L}}\left(\mathbf{x}_i, \mathbf{x}_j\right) = \chi^2\left(\mathbf{L}\mathbf{x}_i, \mathbf{L}\mathbf{x}_j\right) \tag{7}$$

In order to optimize the transformation matrix L, we reuse the same loss function as for the linear case, just replacing the distance metrix:

$$\min_{\mathbf{L}\in\mathcal{P}} \mathcal{L} = \sum_{i,j:j\sim i} \chi^2_{\mathbf{L}}\left(\mathbf{x}_i, \mathbf{x}_j\right) \\ +\mu \sum_{k:y_i \neq y_k} \left[\ell + \chi^2_{\mathbf{L}}\left(\mathbf{x}_i, \mathbf{x}_j\right) - \chi^2_{\mathbf{L}}\left(\mathbf{x}_i, \mathbf{x}_k\right)\right]_+ \tag{8}$$

Following this, ℓ now becomes a hyper-parameter as it no longer can be fixed as 1, since it will the value will have an impact on the result.

2.2. Dimension Reduction

It is possible to perform dimension reduction using both LMNN and NLMNN, by noting that the transformation matrix doesn't have to be square, meaning that $\mathbf{L} \in \mathbb{R}^{d,r}, r \leq d$.

2.3. Optimization

In the case of LMNN, the loss function was optimized using gradient decent. The target neighbours are fixed at the beginning of the algorithm, while the imposters are updated after each gradient step. A similar approach can be used for NLMNN, but since an arbitrary transformation L, might bring the data out of the simplex, we need to constrain L in to be a simplex-preserving transformation which we define as $\mathcal{P} = \{\mathbf{L} \in \mathbb{R}^{d\times d} : \forall \mathbf{x} \in \mathcal{S}, \mathbf{L}\mathbf{x} \in \mathcal{S}^d\}$. We note that this is true if and only if $\mathbf{L} \geq 0$ and each column is normalized $\sum_i L_{ij} = 1, \forall j$. We will now describe two methods of ensuring that $\mathbf{L} \in \mathcal{P}$

2.3.1. Softmax transformation

It's possible to frame the optimization problem, as an unconstrained problem, by introducing a change of variable, and using a function that enforces the properties of the simplex-preserving transform. It can be seen, that if we let $\mathbf{L} = f(\mathbf{A})$, where f is the column-wise softmax function:

$$[f(\mathbf{A})]_{ij} = \frac{e^A_{ij}}{\sum_k e^{A_{kj}}}, \tag{9}$$

the resulting $\mathbf{L}$, will both have non-negative values, and be column-wise normalized. This mean, that we now are optimizing over $\mathbf{A}$ instead of $\mathcal{L}$. Using the chain rule, the gradient can be found by

$$\frac{\partial \mathcal{L}}{\partial A_{pq}} = \sum_{i,j:j\sim i} \frac{\partial \chi^2_{\mathbf{L}}(\mathbf{x}_i, \mathbf{x}_j)}{\partial A_{pq}} \\ +\mu \sum_{k:y_i \neq y_k} \left[\frac{\partial \chi^2_{\mathbf{L}}(\mathbf{x}_i, \mathbf{x}_j)}{\partial A_{pq}} - \frac{\partial \chi^2_{\mathbf{L}}(\mathbf{x}_i, \mathbf{x}_k)}{\partial A_{pq}}\right]_+ \tag{10}$$

If we first define

$$t_{ijp} = \frac{[\mathbf{L}\mathbf{x}_i]_p - [\mathbf{L}\mathbf{x}_j]_p}{[\mathbf{L}\mathbf{x}_i]_p + [\mathbf{L}\mathbf{x}_j]_p}, \tag{11}$$

we can calculate the partial derivative as

$$\frac{\partial \chi^2_{\mathbf{L}}(\mathbf{x}_i, \mathbf{x}_j)}{\partial A_{pq}} = L_{pq}\Bigg(\left(t_{ijp}(x_{iq} - x_{jq}) - \frac{t^2_{ijp}(x_{iq} + x_{jq})}{2}\right) \\ - \sum_{l=1}^{r} L_{lq}\left(t_{ijl}(x_{iq} - x_{jq}) - \frac{t^2_{ijl}(x_{iq} + x_{jq})}{2}\right)\Bigg) \tag{12}$$

2.3.2. Simplex projection

Instead of transforming the optimization problem into an unconstrained one, by using the softmax function, another way of ensuring the simplex-preserving properties of $\mathcal{L}$, is to take

the gradient step of $\mathcal{L}$ with respect to $\mathbf{L}$, and then afterwards project $\mathbf{L}$ back into $\mathcal{P}$. [2] The gradient with respect to $\mathbf{L}$, can again be calculated as

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial L_{pq}} &= \sum_{i,j:j\sim i} \frac{\partial \chi^2_{\mathbf{L}}(\mathbf{x}_i, \mathbf{x}_j)}{\partial L_{pq}} \\ +\mu \sum_{k:y_i \neq y_k} &\left[\frac{\partial \chi^2_{\mathbf{L}}(\mathbf{x}_i, \mathbf{x}_j)}{\partial L_{pq}} - \frac{\partial \chi^2_{\mathbf{L}}(\mathbf{x}_i, \mathbf{x}_k)}{\partial L_{pq}} \right]_+\end{aligned} \tag{13}$$

where the partial derivative can be shown to be

$$\frac{\partial \chi^2_{\mathbf{L}}(\mathbf{x}_i, \mathbf{x}_j)}{\partial L_{pq}} = t_{ijp}(x_{iq} - x_{jp}) - \frac{1}{2} t^2_{ijp}(x_{iq} + x_{jq}) \tag{14}$$

The projection back into the simplex, can be done by solving the following optimization for each column in $\mathbf{L}$

$$\begin{aligned} \min_{\widehat{\mathbf{L}}_i} \quad & \tfrac{1}{2}\|\mathbf{L}_i - \widehat{\mathbf{L}}_i\|^2 \\ \text{s.t.} \quad & \widehat{\mathbf{L}}_i^T \mathbf{1} = 1 \\ & \widehat{\mathbf{L}}_i \geq 0 \end{aligned} \tag{15}$$

where $\widehat{\mathbf{L}}_i$ is the i'th column of the projection. This can be solved efficiently in $\mathcal{O}(r \log(r))$ time [3]

3. IMPLEMENTATION

Both optimization methods for NLMNN were implemented in python [1]. To speed things up the `numba`-python package [5] was used to compile, and parallelize the code. The two dominating computations in the algorithm, is calculating the gradient, and finding the imposters. We introduced parallelism by calculating the rows of the gradient in parallel, and finding the pairwise distances between all datapoints in parallel. The code was executed on a 48 core CPU on the DTU HPC cluster.A dynamic learning rate similar to that of [4] was introduced, where each successful gradient step would increase the learning rate by 1.01, while if a gradient step that would not decrease the loss was detected, the learning rate would be halved, until a step that decreased the loss could be taken, or until 20 learning rate reductions were made. The algorithms were set to stop when convergence was detected, as defined by the loss function decreasing less than 0.01 in a gradient step. Both methods were initialized with $\mathbf{L} = softmax(10\mathbf{I} + 0.01 \cdot \mathbf{1}\mathbf{1}^T)$

4. DATA

To evaluate the performance, we ran the NLMNN model on two datasets. First we tested on the wine dataset[6], which consists of three classes and 13 attributes. This dataset is relative small, and will serve as a sanity check. Secondly we used the webcam dataset [7] from which we choose the same 295 images of 10 different classes (back pack, bike, calculator, headphones, keyboard, laptop computer,monitor, mouse, mug, projector) as [2] and [1]. From these images 800 features has been extracted. Since the dimensionalty is high for this data, we will use this to test the dimension reduction or our models.

[1] available at `https://github.com/jeppe742/nonLinearMetricLearning`

5. RESULTS

Both optimization methods were initially tried on both datasets, but similar to [2], we found that the projection method performed significantly better than using the softmax transformation. The softmax method produced significantly smaller gradients, and had problems converging. From Fig.2 we see the model trained on the webcam dataset with r=10 dimension reduction. Here we see that for three different learning rates, the projection method in general converges faster, but also to a minima that is smaller than the softmax method.

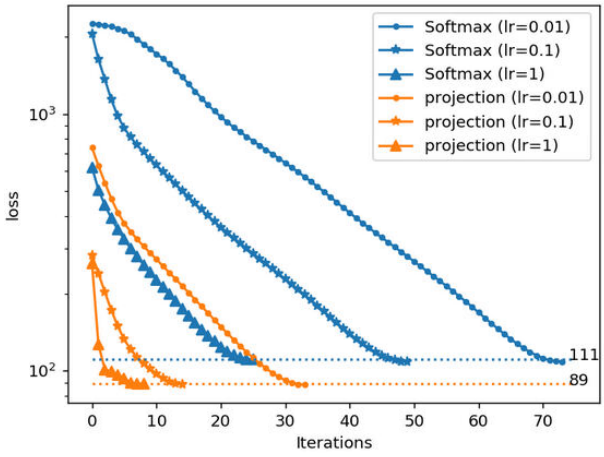

Fig. 2. Training curve on webcam dataset with r=10. Note the difference in learning rate.

5.1. Wine dataset

For obvious reasons, the projection method was used for the remaning results. The model were first evaluated on the wine dataset, where the parameters μ, l were chosen by gridsearch, on crossvalidation with 5 folds. Baselines were calculated in the form of KNN with euclidian distance, LMNN and KNN using the χ^2-distance. From Table.1 we see that the NLMNN model is able to out perform both of the linear metric models. We do however note, that simply KNN with the χ^2-distance is very good, and our learned transformation deviates only slightly from the identity (see Appendix A)

	Accuracy
Euclidean	0.726 ± 0.071
LMNN	0.888 ± 0.064
χ^2	0.923 ± 0.053
NLMNN	$\mathbf{0.927 \pm 0.050}$

Table 1. Accuracy results on the wine. Errors are given as error on the mean calculated from cross validation

	r=10	r=20	r=800
Euclidean	0.784 ± 0.013	0.787 ± 0.008	0.767 ± 0.015
LMNN	0.786 ± 0.022	0.824 ± 0.015	0.842 ± 0.012
χ^2	0.410 ± 0.018	0.554 ± 0.020	0.928 ± 0.012
NLMNN	$\mathbf{0.837 \pm 0.008}$	$\mathbf{0.881 \pm 0.012}$	$\mathbf{0.936 \pm 0.009}$

Table 2. Accuracy results on the webcam dataset with different dimension reductions. Errors are given as error on the mean calculated from cross validation

5.2. Webcam dataset

Next we evaluated the dimensional reduction capabilities of the models, on the webcam dataset, with r=10 and r=20. The same baselines were used, where dimensionality reduction was performed by using PCA before feeding the data to the euclidean KNN and LMNN. The optimal μ, ℓ for NLMNN were chosen by grid-searching μ using cross validation, then picking the best value and searching over ℓ using that. From Fig.3 we see that both μ and ℓ can have an effect on the performance of the model.

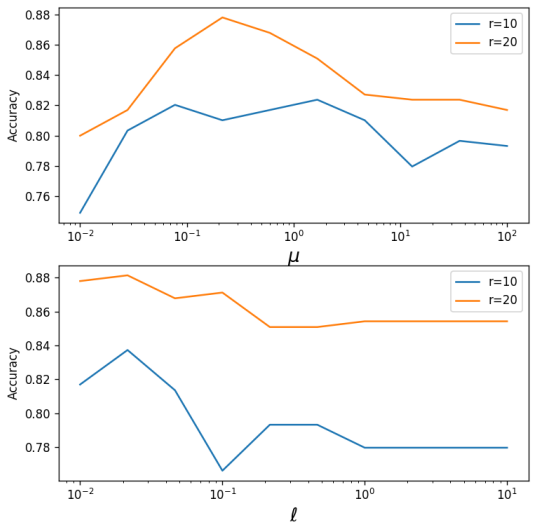

Fig. 3. Mean accuracy on webcam dataset, for different hyperparameter values.

Choosing the best parameters, we get the results in Table 4. Here we see that our model is able to outperform the baselines, at two different levels of dimension reduction.

6. DISCUSSION

As mentioned, the softmax method performed very poorly during training, which caused us to not be able to produce any meaningful results with this method. The main concern with the method, is that it produces extremely small gradient steps. From (12) and (14) we see that the the gradient used in the projection method, is a term in the gradient of the softmax method. We know that $L_{ij} \in [0, 1], \forall i, j$, which means that we take the gradient from (14), subtract a term greater than 0 and finally multiplying it with a number less than 1. Not surprisingly, this will result in a smaller gradient. One might argue, that the projection method allow for large gradients since, we don't constrain the gradient, but do the projection later. Our experiments does however show, than even when projecting back into the simplex after the larger gradient back, the resulting update to the loss function is significantly larger than the softmax method. The performance of the NLMNN algorithm depends on both μ and ℓ, but ℓ also has significant influence on the runtime of the optimization. Calculating the gradient requires $\mathcal{O}(d \cdot r \cdot n \cdot I)$ operations, where I is the average number of imposters per target neighbour. Since ℓ controls the margin, setting it too high can result in each target neighbour having all n data points as imposters, in which case the gradient can become very computational heavy. Another consequence of this scenario, is that our implementation stores the imposters in a matrix which in this case would require $\mathcal{O}(n^2 \cdot k)$ space, which for larger datasets can consume a very large amount of RAM. With the fast convergence of the projection method, the number of imposters are however very quickly reduced to a manageable amount. In our experiments the very first gradient step on the webcam dataset reduced the number of imposters from more than 200k to several hundreds.

On the wine dataset, our improved results were mostly due to the χ^2-distance being a better metric for the problem, but on the webcam dataset, our model achieves very good results, even though the χ^2-distance by it self is worse than the euclidean. This confirms that the performance of our model is due to it being able to learn a good transformation.

7. CONCLUSION

We have shown that the original NLMNN model by [1] using softmax transformation performed very poorly, but by changing to the projection method suggested in [2], the model is able to outperform the linear LMNN on both a small dataset, and when using the dimension reduction capabilities of both models, on an image dataset.

8. REFERENCES

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

[2] Wei Yang, Luhui Xu, Xiaopan Chen, Fengbin Zheng, and Yang Liu, "Chi-squared distance metric learning for histogram data," *Mathematical Problems in Engineering*, vol. 2015, 2015.

[3] Weiran Wang and Miguel A Carreira-Perpinán, "Projection onto the probability simplex: An efficient algorithm with a simple proof, and an application," *arXiv preprint arXiv:1309.1541*, 2013.

[4] Kilian Q Weinberger, John Blitzer, and Lawrence K Saul, "Distance metric learning for large margin nearest neighbor classification," in *Advances in neural information processing systems*, 2006, pp. 1473–1480.

[5] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert, "Numba: A llvm-based python jit compiler," in *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, New York, NY, USA, 2015, LLVM '15, pp. 7:1–7:6, ACM.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[7] A. Holub G. Griffin and P. Perona, "Caltech-256 object category dataset," *Caltech Technical Report*, 2007.

Appendix

A. LEARNED TRANSFORMATION MATRIX FOR WINE DATASET

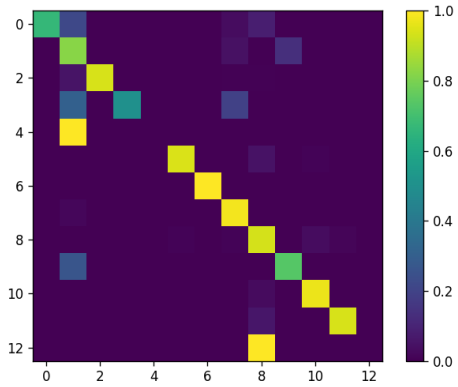

Fig. 4.

B. LEARNED TRANSFORMATION MATRIX FOR WEBCAM DATASET (R=10)

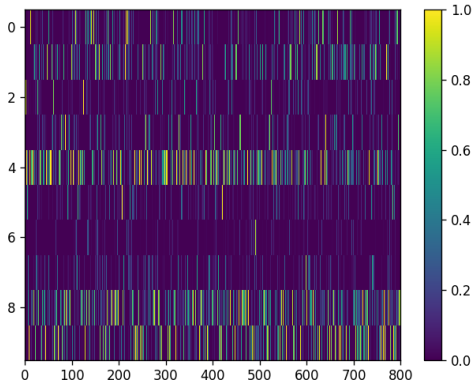

Fig. 5.

C. LEARNED TRANSFORMATION MATRIX FOR WEBCAM DATASET (R=20)

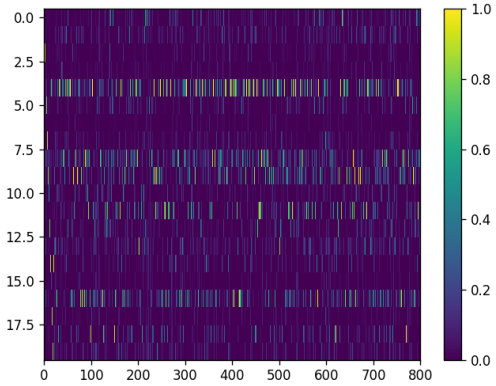

Fig. 6.

	Accuracy
Euclidean	0.767 ± 0.015
LMNN	0.203 ± 0.012
χ^2	0.928 ± 0.012
NLMNN	$\mathbf{0.936 \pm 0.009}$

Table 3.

	r=10	r=20	r=800
Euclidean	0.784 ± 0.013	0.787 ± 0.008	0.767 ± 0.015
Euclidean (Kedem)	0.673 ± 0.016	$0.727 \pm .008$	0.562 ± 0.08
LMNN	0.786 ± 0.022	0.824 ± 0.015	0.203 (0.842) ± 0.012
χ^2	0.410 ± 0.018	0.554 ± 0.020	0.928 ± 0.012
NLMNN	$\mathbf{0.837 \pm 0.008}$	$\mathbf{0.881 \pm 0.012}$	$\mathbf{0.936 \pm 0.009}$

Table 4. Accuracy results on the webcam dataset with different dimension reductions. Errors are given as error on the mean calculated from cross validation