An Invariant Large Margin Nearest Neighbour Classifier

M. Pawan Kumar P.H.S. Torr
Oxford Brookes University
{pkmudigonda,philiptorr}@brookes.ac.uk
http://cms.brookes.ac.uk/research/visiongroup

A. Zisserman
University of Oxford
az@robots.ox.ac.uk
http://www.robots.ox.ac.uk/~vgg

Abstract

*The k-nearest neighbour (*kNN*) rule is a simple and effective method for multi-way classification that is much used in Computer Vision. However, its performance depends heavily on the distance metric being employed. The recently proposed large margin nearest neighbour (*LMNN*) classifier [21] learns a distance metric for k*NN *classification and thereby improves its accuracy. Learning involves optimizing a convex problem using semidefinite programming (*SDP*).*

We extend the LMNN *framework to incorporate knowledge about invariance of the data. The main contributions of our work are three fold: (i) Invariances to multivariate polynomial transformations are incorporated without explicitly adding more training data during learning - these can approximate common transformations such as rotations and affinities; (ii) the incorporation of different regularizers on the parameters being learnt; and (iii) for all these variations, we show that the distance metric can still be obtained by solving a convex* SDP *problem. We call the resulting formulation invariant* LMNN *(*ILMNN*) classifier.*

We test our approach to learn a metric for matching (i) feature vectors from the standard Iris dataset; and (ii) faces obtained from TV *video (an episode of 'Buffy the Vampire Slayer'). We compare our method with the state of the art classifiers and demonstrate improvements.*

1. Introduction

Many problems in Computer Vision can be posed in the multi-way classification framework. For example, object recognition [16] (or object category recognition [8]) where a given image needs to be classified as belonging to a particular object (or object category), image retrieval [1] where images in a dataset are classified as satisfying a query or not, and to some extent pose estimation [18] where, given a set of 3-D models in various poses, an image or silhouette is classified as being generated from one of the 3-D models.

The k-nearest neighbour (kNN) rule [6] is one of the oldest and most accurate methods to obtain nonlinear decision boundaries in a multi-way classification problem. Given a training dataset, the kNN classifier labels an unlabelled test example by the majority label among its *k* nearest neighbours. There are two main issues when using the kNN classifier: (i) Which distance metric should be used?; and (ii) How can the nearest neighbours be found efficiently? Many algorithms have been proposed in the literature which address the latter issue of efficiency, e.g. [4, 8, 18]. However, most kNN based methods use Euclidean distance (denoted by kNN-E for short). In other words, they treat all elements of the feature vector equally and fail to capitalize on any statistical regularities in the data.

To overcome this deficiency, some methods have combined other multi-way classifiers with kNN-E [23]. Alternatively, it has been proposed to learn a distance metric from a set of labelled examples. To this end, several methods have been reported which learn a global linear mapping of features to encourage accurate classification by minimizing some cost function [2, 10, 12, 22]. However, these methods rely on gradient descent which is slow and prone to local minima and thus, are limited in their usefulness.

Recently, the problem of learning the linear mapping has been reformulated as a convex semidefinite program (SDP) [9, 21]. This implies that the globally optimal linear mapping can be efficiently estimated from the training data (i.e. in time which is polynomial in the size of the training data). The resulting framework, called the large margin nearest neighbour (LMNN) classifier, effectively learns a Mahalanobis distance metric for kNN classification. However, LMNN does not incorporate invariance of the data to known transformations which has been shown to improve the accuracy of a classifier [11, 19].

A standard way to incorporate invariance when learning a classifier is to augment the training data with transformed versions of each feature vector. However, this increases the amount of training data thereby making the method more computationally expensive. Further, it approximates the infinite points lying on the trajectory (manifold), described by the transformed versions of the feature vector, using a finite number of samples. Due to these disadvantages, previous research has focused on incorporating invariance using the trajectories directly [11, 19].

For kNN classifiers, Simard *et al.* [19] approximate the distance between two trajectories (defined by the transformed versions of two feature vectors) using the distance between their tangent vectors. However, this approximation is inaccurate when the trajectories have high curvatures which limits its application to a restricted set of transformations. For binary classification, Graepel and Herbrich [11] learn a maximum margin separating hyperplane when the training exam-

ples are polynomial trajectories instead of single points. This allows them to integrate invariance to polynomial transformations within the support vector machine (SVM) formulation. However, extending their method to multiple classes is non-trivial. Further, their framework only allows for invariance to univariate polynomial transformations (i.e. transformations over one variable only, such as rotation or translation along one direction).

In this paper, we show how to extend the multi-way LMNN framework to incorporate invariance to multivariate polynomial transformations. Many common transformations such as rotation, scaling and shearing can be approximated as polynomials using Taylor's expansion. Further, multivariate polynomials allow us to handle transformations defined over multiple variables (e.g. the 2-D Euclidean transformation). The resulting framework, which we call the *invariant* large margin nearest neighbour (ILMNN) classifier, can be formulated as a convex SDP. This allows us to learn the distance metric in polynomial time. Further, to address the issue of overfitting, we propose different regularizers for the parameters of the distance metric which can be included within the convex SDP formulation.

Preliminaries: SDP is a strict generalization of commonly used convex optimization frameworks, e.g. linear programming and second-order cone programming. Its ability to model more complex cost functions has lead to new and accurate approaches for several applications [9, 13, 17, 21]. Formally, an SDP is a linear program on the elements of matrices $\mathbf{M}_i$ ($i = 1, \cdots, s$) with the additional constraint that $\mathbf{M}_i$ are positive semidefinite (denoted by $\mathbf{M}_i \succeq 0$), i.e.

$$\begin{array}{ll} \min & \sum_i \mathbf{C}_i \bullet \mathbf{M}_i \\ \text{s.t.} & \sum_i \mathbf{A}_{ij} \bullet \mathbf{M}_i \geq b_j, \mathbf{M}_i \succeq 0, \forall i, j, \end{array} \quad (1)$$

where the operator ($\bullet$) is the Frobenius inner product. The matrices $\mathbf{C}_i$ define the objective function while $\mathbf{A}_{ij}$ specify the constraints. Note that positive scalar variables can be included in SDP by using 1×1 matrices $\mathbf{M}_i$.

Outline: We begin by briefly describing the LMNN formulation presented in [21]. We then propose different regularizers for the parameters of the distance metric. Section 3 outlines the class of transformations which are of interest to us. In section 4, we describe the ILMNN classifier which incorporates invariance to this class of transformations. We test our approach on two problems: (i) recognizing previously unseen data from the Iris dataset; and (ii) recognizing faces in TV video where it is important to address the issue of invariance to multiple transformations in order to handle variations in pose. Results of our method and a comparison with the state of the art classifiers is presented in section 5.

2. The LMNN Classifier

Given a set of feature vectors $\mathbf{x}_i \in \Re^D$ ($i = 1, \cdots, n$) along with their labels y_i and their *target* (correct) neighbours, LMNN learns a Mahalanobis distance metric parameterized by $\mathbf{L}$, i.e.

$$D(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{L}(\mathbf{x}_i - \mathbf{x}_j)\|^2 = (\mathbf{x}_i - \mathbf{x}_j)^\top \mathbf{L}^\top \mathbf{L}(\mathbf{x}_i - \mathbf{x}_j) \quad (2)$$

which encourages accurate kNN classification. Specifically, the parameter $\mathbf{L}$ should minimize the distance between the mappings of a vector and its k target neighbours. Let $\eta_{ij} \in \{0, 1\}$ indicate whether $\mathbf{x}_j$ is a target neighbour of $\mathbf{x}_i$. Thus, $\mathbf{L}$ should be chosen to minimize

$$\psi_1(\mathbf{L}) = \sum_{ij} \eta_{ij} \|\mathbf{L}(\mathbf{x}_i - \mathbf{x}_j)\|^2. \quad (3)$$

Further, the distance of $\mathbf{x}_i$ from a target neighbour $\mathbf{x}_j$ should be less than its distance from an *impostor* (incorrect) neighbour $\mathbf{x}_l$. To this end, Weinberger *et al.* [21] suggest minimizing the sum of the standard hinge loss over triplets of input vectors, i.e.

$$\psi_2(\mathbf{L}) = \sum_{ijl} \nu_{ijl} [1 + \|\mathbf{L}(\mathbf{x}_i - \mathbf{x}_j)\|^2 - \|\mathbf{L}(\mathbf{x}_i - \mathbf{x}_l)\|^2]_+, \quad (4)$$

where $\nu_{ijl} = \eta_{ij}\delta(y_i \neq y_l)$ ($\delta(\cdot)$ is 1 if and only if the argument is true) and the function $[z]_+ = \max\{z, 0\}$. The hinge loss will be zero only when $D(\mathbf{x}_i, \mathbf{x}_l) \geq 1 + D(\mathbf{x}_i, \mathbf{x}_j)$, i.e. when there is a *margin* of width 1 between target and impostor neighbours. The total cost function is given by $\psi(\mathbf{L}) = \psi_1(\mathbf{L}) + \lambda_h \psi_2(\mathbf{L})$ where $\lambda_h \geq 0$ is some constant.

Since $\mathbf{M} = \mathbf{L}^\top \mathbf{L} \succeq 0$, the minimization of the cost function $\psi(\mathbf{L})$ can be formulated as a convex SDP [21], i.e.

$$\begin{array}{ll} \min & \sum_{ij} \eta_{ij} d_{ij} + \lambda_h \sum_{ijl} \nu_{ijl} \xi_{ijl}, \\ \text{s.t.} & (\mathbf{x}_i - \mathbf{x}_j)^\top \mathbf{M} (\mathbf{x}_i - \mathbf{x}_j) = d_{ij}, \\ & (\mathbf{x}_i - \mathbf{x}_l)^\top \mathbf{M} (\mathbf{x}_i - \mathbf{x}_l) - d_{ij} \geq 1 - \xi_{ijl}, \\ & \mathbf{M} \succeq 0, \xi_{ijl} \geq 0, \forall i, j, l. \end{array} \quad (5)$$

The variables d_{ij} and ξ_{ijl} represent the distance between $\mathbf{x}_i$ and $\mathbf{x}_j$ and the hinge loss respectively. Note that the above problem is convex with respect to the variable $\mathbf{M}$ but not $\mathbf{L}$. Hence, the methods described in [2, 10, 12, 22] (which optimize over $\mathbf{L}$) are prone to local minima. Fig. 1 shows an example of the LMNN classifier learnt using 2-D points.

The LMNN classifier sometimes overfits to the training data which results in high generalization error. It is well-known that such problems can be avoided by regularization. To this end, we propose the following three variations of the LMNN classifier.

L^2-LMNN: One of the most commonly used regularizer is to minimize the L^2 (or Frobenius) norm of the parameters. In the case of LMNN, $||\mathbf{L}||^2 = \sum_{a=1}^{D} \mathbf{M}(a, a)$, since $\mathbf{M} = \mathbf{L}^\top \mathbf{L}$. Thus, the minimization of the L^2 norm of $\mathbf{L}$ can be included within the SDP formulation as follows:

$$\min \quad \sum_{ij} \eta_{ij} d_{ij} + \lambda_h \sum_{ijl} \nu_{ijl} \xi_{ijl} + \lambda_r \sum_a \mathbf{M}(a, a), \quad (6)$$

where $\lambda_r \geq 0$ is some constant and the variables d_{ij}, ξ_{ijl} and $\mathbf{M}$ are as defined in the optimization problem (5).

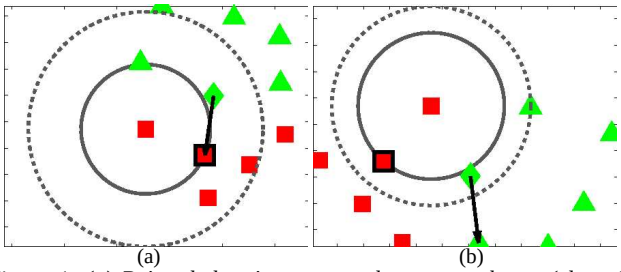

Figure 1. *(a) Points belonging to two classes are shown (class 1: red squares, class 2: green triangles). The solid circle is centred around a point in class 1 with radius equal to the distance from its target neighbour (shown with a black border). The surrounding dashed circle shows the desired margin. A test point (green diamond in class 2) which is misclassifi ed when using* kNN-E *is also shown. The arrow points to the nearest neighbour of the test point in the training dataset. (b) Mapped points using parameter* $\mathbf{L}$ *(obtained by solving the optimization problem (5) with* $k = 1$*). Unlike (a), all points in class 2 lie on or outside the margin. Further, the green diamond is now assigned the correct label.*

D-LMNN: In addition to the L^2 regularization, the form of the parameter $\mathbf{L}$ (and hence $\mathbf{M}$) can also be restricted to a diagonal matrix. In this case the convex optimization problem is a much simpler linear program instead of an SDP since the constraint $\mathbf{M} \succeq 0$ is replaced by $\mathbf{M}(a,a) \geq 0$ and $\mathbf{M}(a,b) = 0$ for all $a \neq b$.

DD-LMNN: We can also use a softer regularization term which favours $\mathbf{M}$ to be diagonally dominant (in contrast to the hard constraint in D-LMNN). This regularization term minimizes the L^1 norm of the off-diagonal elements of $\mathbf{M}$ and can be included within the SDP formulation as follows:

$$\begin{aligned} \min \quad & \textstyle\sum_{ij} \eta_{ij} d_{ij} + \lambda_h \sum_{ijl} \nu_{ijl} \xi_{ijl} + \lambda_r \sum_a M(a,a) + \\ & \lambda_d \textstyle\sum_{a \neq b} t_{ab}, \\ \text{s.t.} \quad & t_{ab} \geq \mathbf{M}(a,b), t_{ab} \geq -\mathbf{M}(a,b), \forall a, b, \end{aligned} \tag{7}$$

where $\lambda_d \geq 0$. Again, the variables d_{ij}, ξ_{ijl} and $\mathbf{M}$ are given in the optimization problem (5).

The globally optimal $\mathbf{M}$ (and hence $\mathbf{L}$) can be estimated in polynomial time by solving the convex problems described above. However, since the worst-case complexity for obtaining the parameter $\mathbf{L}$ is still cubic in the size of the training data, it becomes computationally infeasible to build invariance by augmenting the training data with transformed versions of the feature vectors.

It is desirable, therefore, to incorporate known invariances into the LMNN framework without adding more training data. Further, the resulting problem should retain the property of convexity. The next section describes the class of useful transformations and a result by Lasserre [15] which enable us to achieve this.

3. Polynomial Transformations

Building invariance to a general transformation within the LMNN framework results in a complex and difficult-to-solve problem. In this paper, we consider the case where the transformations can be approximated using multivariate polynomials. As will be seen in the next section, the resulting problem retains the desirable property of convexity.

A multivariate polynomial transformation maps a point $\mathbf{x} \in \Re^D$ to another point $\mathbf{x}' \in \Re^D$ such that the mapping can be represented using polynomials. For example, consider the affine transformation of a point (x_1, x_2). Its mapping $(\theta_1 x_1 + \theta_2 x_2 + \theta_3, \theta_4 x_1 + \theta_5 x_2 + \theta_6)$ can be expressed as a degree 1 multivariate polynomial transformation as follows:

$$\begin{pmatrix} x_1' \\ x_2' \end{pmatrix} = \begin{pmatrix} 0 & x_1 & x_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & x_1 & x_2 & 1 \end{pmatrix} \boldsymbol{\theta}, \tag{8}$$

where $\boldsymbol{\theta} = (1, \theta_1, \theta_2, \cdots, \theta_6)^\top$. In general, a degree r multivariate polynomial transformation $T(\boldsymbol{\theta}, \cdot)$ maps $\mathbf{x}$ to

$$\mathbf{x}' = T(\boldsymbol{\theta}, \mathbf{x}) = \mathbf{X}^\top \boldsymbol{\theta}, \tag{9}$$

where $\boldsymbol{\theta} = (1, \theta_1, \theta_2, \cdots, \theta_m, \theta_1^2, \theta_1\theta_2, \cdots, \theta_m^r)^\top$ and $\mathbf{X} \in \Re^{R \times D}$ is a matrix whose elements are completely determined by $\mathbf{x}$. The above equation provides the trajectory (manifold) which contains the transformed version of the vector $\mathbf{x}$.

An important special case of the above class of transformations is univariate polynomials, where the number of variables $m = 1$. Univariate polynomials contain many useful transformations which are defined over one variable, e.g. the rotation of a point (x_1, x_2) by an angle θ, whose mapping $(x_1 \cos\theta - x_2 \sin\theta, x_1 \sin\theta + x_2 \cos\theta)$ can be approximated to third order using Taylor's expansion as

$$\begin{aligned} \begin{pmatrix} x_1' \\ x_2' \end{pmatrix} &= \begin{pmatrix} x_1(1 - \theta^2/2) - x_2(\theta - \theta^3/6) \\ x_1(\theta - \theta^3/6) + x_2(1 - \theta^2/2) \end{pmatrix} \\ &= \begin{pmatrix} x_1 & -x_2 & -x_1/2 & x_2/6 \\ x_2 & x_1 & -x_2/2 & -x_1/6 \end{pmatrix} \boldsymbol{\theta}, \end{aligned} \tag{10}$$

where $\boldsymbol{\theta} = (1, \theta, \theta^2, \theta^3)^\top$. Nesterov [17] showed that, for every non-negative univariate polynomial $T(\boldsymbol{\theta}, \mathbf{x})$ (i.e. $T(\boldsymbol{\theta}, \mathbf{x}) \geq 0, \forall \theta \in \Re$), there exists a matrix $\mathbf{P} \succeq 0$ such that $T(\boldsymbol{\theta}, \mathbf{x}) \equiv \boldsymbol{\theta}^\top \mathbf{P} \boldsymbol{\theta}$. In other words, non-negative univariate polynomials are *semidefinite-representable* (SD-representable). Note that this was the result used to incorporate invariance to a transformation with one variable in [11].

Unfortunately, the above result by Nesterov does not hold true for all multivariate polynomials. In other words, there exist some multivariate polynomials which are non-negative everywhere on the real axes but are not SD-representable. However, it has been shown that multivariate polynomials which are sum of squares of other polynomials are SD-representable [15].

We can further extend this result by showing that for an SD-representable multivariate polynomial $T(\boldsymbol{\theta}, \mathbf{x})$, the condition $T(\boldsymbol{\theta}, \mathbf{x}) \geq 0, \forall \theta_i \in [-\tau_i, \tau_i]$ is also SD-representable. Note that this is a generalization of corollary 1 in [11] and

can be proved by replacing θ_i with $\tau_i(\theta_i^2-1)/(\theta_i^2+1)$ (detailed proof in technical report [14]). As an illustrative example, consider the case of a univariate polynomial transformation which approximates rotation as shown in equation (10). The non-negativity of this polynomial can be restricted to the range $-\tau \leq \theta \leq \tau$ by constructing the following SD-representable polynomial:

$$\begin{pmatrix} x_1 & -x_2 & -x_1/2 & x_2/6 \\ x_1 & x_2 & -x_1/2 & -x_2/6 \end{pmatrix} \begin{pmatrix} (\theta^2+1)^3 \\ \tau(\theta^2+1)^2(\theta^2-1) \\ \tau^2(\theta^2+1)(\theta^2-1)^2 \\ \tau^3(\theta^2-1)^3 \end{pmatrix} = \begin{pmatrix} c_1 & 0 & c_2 & 0 & c_3 & 0 & c_4 \\ c_5 & 0 & c_6 & 0 & c_7 & 0 & c_8 \end{pmatrix} \boldsymbol{\theta} \quad (11)$$

where $\boldsymbol{\theta} = (1, \theta, \theta^2, \cdots, \theta^6)^\top$, for appropriate values of c_i $(i = 1, \cdots, 8)$.

Next, we show how invariance to multivariate polynomial transformations can be built into the LMNN framework using the SD-representability results.

4. The ILMNN Classifier

In this section we describe the invariant large margin nearest neighbour (ILMNN) classifier. This has the desirable properties of: (i) incorporating invariance of feature vectors to known multivariate polynomial transformations without increasing the size of the training dataset; and (ii) providing the distance metric in polynomial time.

Under a known polynomial transformation $T(\boldsymbol{\theta}, \cdot)$, the goal of the ILMNN classifier is to learn a Mahalanobis distance metric $D(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{L}(\mathbf{x}_i - \mathbf{x}_j)\|^2$ which encourages accurate kNN classification. Recall that the mapping $\mathbf{L}$ is the parameterization of the distance metric. Specifically, the learnt distance metric should satisfy the following criteria:

1. Minimize the maximum distance between the trajectories of $\mathbf{x}_i$ and its target neighbour $\mathbf{x}_j$, i.e. the *target trajectories* $T(\boldsymbol{\theta}_1, \mathbf{x}_i)$ and $T(\boldsymbol{\theta}_2, \mathbf{x}_j)$, $\forall \boldsymbol{\theta}_1, \boldsymbol{\theta}_2$. This would encourage accurate classification in the worst case when the training and test dataset differ significantly due to transformations $T(\boldsymbol{\theta}, \cdot)$.

2. Maximize the minimum distance between the trajectories of $\mathbf{x}_i$ and its impostor neighbour $\mathbf{x}_l$, i.e. the *impostor trajectories* $T(\boldsymbol{\theta}_1, \mathbf{x}_i)$ and $T(\boldsymbol{\theta}_2, \mathbf{x}_l)$, $\forall \boldsymbol{\theta}_1, \boldsymbol{\theta}_2$.

As will be seen, the second criterion results in a convex SDP formulation. However, except for the case of univariate polynomials, the first criterion does not (as explained later). Hence, to satisfy this criterion, we obtain an approximation of the maximum distance between the mapped target trajectories. To this end, we compute the ratio of the maximum distance between the original target trajectories and the corresponding input vectors, i.e.

$$\rho_{ij} = \max_{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2} \frac{\|T(\boldsymbol{\theta}_1, \mathbf{x}_i) - T(\boldsymbol{\theta}_2, \mathbf{x}_j)\|}{\|\mathbf{x}_i - \mathbf{x}_j\|}. \quad (12)$$

Given the training data, the ratio ρ_{ij} can be estimated for each pair of feature vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ using the above equation. We then approximate the maximum distance between the target trajectories under a mapping $\mathbf{L}$ by $\rho_{ij}\|\mathbf{L}(\mathbf{x}_i - \mathbf{x}_j)\|^2$. Note that for some cases (e.g. when all points are collinear and $T(\boldsymbol{\theta}, \cdot)$ defines scaling) ρ_{ij} remains unchanged under any mapping $\mathbf{L}$. In such cases, our approximation is exact.

Under this approximation, the objective function of LMNN defined in equations (3) and (4) can be generalized to satisfy the criteria (1) and (2) as follows:

$$\psi(\mathbf{L}) = \textstyle\sum_{ij} \eta_{ij}\rho_{ij}d_{ij} + \lambda_h \sum_{ijl} \nu_{ijl}[1 + \rho_{ij}d_{ij} - \|\mathbf{L}(\mathbf{X}_i^\top \boldsymbol{\theta}_1 - \mathbf{X}_l^\top \boldsymbol{\theta}_2)\|^2]_+, \quad (13)$$

where $d_{ij} = ||\mathbf{L}(\mathbf{x}_i - \mathbf{x}_j)||^2$. Recall that binary numbers η_{ij} indicate whether $\mathbf{x}_i$ is a target neighbour of $\mathbf{x}_j$ and $\nu_{ijl} = \eta_{ij}\delta(y_i \neq y_l)$. The first term of the above cost function minimizes the maximum distance between the trajectories of target neighbours. The second term defines the hinge loss between the maximum distance of the target trajectories and the minimum distance of the impostor trajectories.

The matrices $\mathbf{X}_i$ are determined completely by the feature vectors provided and hence, the only variable in equation (13) is $\mathbf{L}$. Since $\mathbf{M} = \mathbf{L}^\top\mathbf{L} \succeq 0$, minimizing the cost function $\psi(\mathbf{L})$ defined in equation (13) can be written as

$$\min \quad \textstyle\sum_{ij} \eta_{ij}\rho_{ij}d_{ij} + \lambda_h \sum_{ijl} \nu_{ijl}\xi_{ijl}, \quad (14)$$

$$\text{s.t.} \quad (\mathbf{x}_i - \mathbf{x}_j)^\top \mathbf{M}(\mathbf{x}_i - \mathbf{x}_j) = d_{ij}, \quad (15)$$

$$(\mathbf{X}_i^\top \boldsymbol{\theta}_1 - \mathbf{X}_l^\top \boldsymbol{\theta}_2)^\top \mathbf{M}(\mathbf{X}_i^\top \boldsymbol{\theta}_1 - \mathbf{X}_l^\top \boldsymbol{\theta}_2) - \rho_{ij}d_{ij} \geq 1 - \xi_{ijl}, \quad (16)$$

$$\mathbf{M} \succeq 0, \xi_{ijl} \geq 0, \forall \boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \forall i, j, l, \quad (17)$$

where d_{ij} and ξ_{ijl} are the slack variables which represent the distance between $\mathbf{x}_i$ and $\mathbf{x}_j$ and the hinge loss respectively. Note that equation (16) represents an infinite number of constraints. However, we now show how it can be converted into a single semidefinite constraint.

We observe that the multivariate polynomial in equation (16) is a sum of squares of other polynomials.[1] Thus, the infinite constraints specified by equation (16) (which correspond to criterion (2)) can be replaced by a semidefinite constraint which does not depend on $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$. However, note that when criterion (1) is expressed similarly as a set of infinite constraints we obtain a difference of squared polynomials which is not SD-representable (unless $T(\boldsymbol{\theta}, \cdot)$ is univariate). Hence, criterion (1) is approximated using ρ_{ij} as described above.

In order to obtain the semidefinite constraint, we reformulate equation (16) as $\boldsymbol{\theta}'^\top \mathbf{P}_{ijl} \boldsymbol{\theta}' \geq 0, \forall \boldsymbol{\theta}'$, where $\boldsymbol{\theta}' = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2)^\top$ and

$$\mathbf{P}_{ijl} = \begin{pmatrix} \mathbf{X}_i\mathbf{M}\mathbf{X}_i^\top & -\mathbf{X}_i\mathbf{M}\mathbf{X}_l^\top \\ -\mathbf{X}_l\mathbf{M}\mathbf{X}_i^\top & \mathbf{X}_l\mathbf{M}\mathbf{X}_l^\top \end{pmatrix} - (\rho_{ij}d_{ij} + 1 - \xi_{ijl})\mathbf{E}. \quad (18)$$

Here $\mathbf{E}$ is a square matrix of appropriate dimensions with all elements equal to zero except the $(1,1)^{th}$ element which is

[1] $(\mathbf{X}_i^\top \boldsymbol{\theta}_1 - \mathbf{X}_l^\top \boldsymbol{\theta}_2)^\top \mathbf{M}(\mathbf{X}_i^\top \boldsymbol{\theta}_1 - \mathbf{X}_l^\top \boldsymbol{\theta}_2) = \|\mathbf{L}(\mathbf{X}_i^\top \boldsymbol{\theta}_1 - \mathbf{X}_l^\top \boldsymbol{\theta}_2)\|^2$.

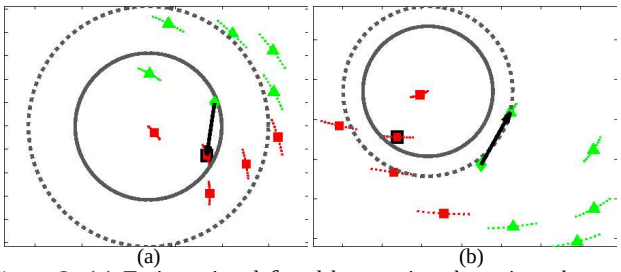

Figure 2. *(a) Trajectories defined by rotating the points shown in Fig. 1 by an angle $-5^o \leq \theta \leq 5^o$. The test point (i.e. the green diamond) is also shown. The solid circle is centred around an endpoint of the trajectory defined by a point in class 1. The radius is the maximum distance between the trajectories of the point and its target neighbour (shown with a black border). The surrounding dashed circle shows the desired margin. (b) Mapped trajectories using parameter* $\mathbf{L}$ *(obtained by solving the optimization problem (19) with $k = 1$). Unlike (a), all impostor trajectories lie on or outside the margin. Further, the test point is now classified correctly.*

1. As equation (16) is SD-representable, it is equivalent to the constraint $\mathbf{P}_{ijl} \succeq 0$. Thus, the above problem can be formulated as the following convex SDP:

$$\begin{aligned} \mathbf{M}^* = \arg\min & \textstyle\sum_{ij} \eta_{ij}\rho_{ij}d_{ij} + \lambda_h \sum_{ijl} \nu_{ijl}\xi_{ijl}, \\ & (\mathbf{x}_i - \mathbf{x}_j)^\top \mathbf{M}(\mathbf{x}_i - \mathbf{x}_j) = d_{ij}, \\ & \mathbf{M} \succeq 0, \mathbf{P}_{ijl} \succeq 0, \xi_{ijl} \geq 0, \forall i, j, l. \end{aligned} \tag{19}$$

The number of variables and constraints present in the above SDP is the same as the number of variables and constraints required to train the LMNN classifier. This is due to the fact that the matrix $\mathbf{P}_{ijl}$ is completely determined by $\mathbf{M}$, d_{ij} and ξ_{ijl}. Note that unlike [11], where approximation using ρ_{ij} is not applicable, the ILMNN framework allows us to handle invariance to multivariate polynomial transformations.

Invariance on Segments: For many applications it may not be desirable to enforce correct classification on the entire trajectory. In such cases, we make use of the SD-representability of non-negative polynomials in the interval $\theta_i \in [-\tau_i, \tau_i]$ (see section 3). The ILMNN framework can then be easily extended to incorporate invariance over a segment of the trajectory. Fig. 2 shows an example of the ILMNN classifier learnt by restricting the invariance to a segment and solving the optimization problem (19).

Regularizers: As in the LMNN framework, the problem of overfitting is avoided by using the regularizers defined in section 2 which can be included within the convex SDP formulation. This results in three variations of the ILMNN classifier: L^2-ILMNN, D-ILMNN and DD-ILMNN. It is worth noting that unlike D-LMNN, the D-ILMNN classifier does not result in a linear program due to the semidefinite constraints $\mathbf{P}_{ijl} \succeq 0$.

5. Experiments

We now present the results of our approach using two datasets: (i) the standard Iris dataset[2] which was used in [21]; and (ii) faces obtained from TV video.

5.1. The Iris Dataset

For the first set of experiments, the goal is to determine whether learning a mapping using trajectories of feature vectors (instead of using only the feature vectors) results in better classification of previously unseen data. Below, we describe the experimental setup and the results obtained in detail.

Dataset: We used the standard Iris dataset which contains 50 feature vectors for each for the 3 classes. Each feature vector is of size 4 and specifies the following attributes: (i) sepal length; (ii) sepal width; (iii) petal length; and (iv) petal width (all measured in centimeters). Using this dataset, we perform 20 experiments. In each experiment, we randomly choose roughly 75% of the dataset for training and the remaining 25% for testing.

Incorporating Invariance: For the ILMNN classifier, we include invariance to slight changes in the scale of the sepal/petal length and width. These transformations (i.e. the scaling) are modelled using a multivariate polynomial (with four variables, one for each dimension) of degree 1. Recall that the ILMNN classifier learns a mapping $\mathbf{L}$ using the trajectories defined by these polynomials. This effectively increases the number of examples used to learn the mapping without changing the size of the training dataset. Using the method described in section 4, we restrict the invariance to segment $[\frac{1}{\sigma_{ij}}, \sigma_{ij}]$ for the i^{th} dimension and the j^{th} class. For each dimension i and class j, the term σ_{ij} is chosen as the average scale variation observed in the training data.

Training the Classifiers: Following the work of Weinberger *et al.* [21], in the absence of prior knowledge, the k target neighbours of each input vector are found using Euclidean distance. All input vectors $\mathbf{x}_l$ such that $y_l \neq y_i$ are considered impostor neighbours of $\mathbf{x}_i$.

The LMNN classifier is trained using the publically available code[3] for solving the optimization problem (5). The ILMNN classifier is trained by solving the optimization problem (19) using the SDPLR software [5]. As noted in [21], the time required to train the classifiers is significantly reduced by using the alternative projection theorem [3] since most slack variables $\xi_{ijl} = 0$. For both the classifiers, no regularization is used in these experiments. We set $k = 1$ for both the classifiers to encourage efficient training. Note, however, that our framework is valid for any value of k.

Results: During testing, the entire training data and the given test vector are mapped using the learnt parameter $\mathbf{L}$. The test vector is then classified using the kNN rule with $k = 1$. We compare the ILMNN classifier with the LMNN and the standard multi-way SVM (M-SVM) classifier. Table 1 shows the average accuracies obtained. By incorporating invariance to scaling, the ILMNN classifier provides the best results. We also trained the LMNN and M-SVM classifiers using a larger dataset. This dataset consists of 10 synthetically

[2] Available at http://www.ics.uci.edu/~mlearn/MLRepository.html

[3] Available at http://seas.upenn.edu/~killanw/Downloads/LMNN.html

scaled versions of each feature vector in the original training dataset. Note that by using the larger dataset, LMNN and M-SVM provide comparable results to ILMNN at the cost of more computation.

Method	Tr. Size	Acc (%)
ILMNN	112	97.37
LMNN	112	92.11
M-SVM	112	93.42
LMNN	1120	97.37
M-SVM	1120	96.71

Table 1. *The second column provides the number of points in the training dataset. The size of the test dataset was fixed to 38 in all experiments. Note that for* LMNN *and* M-SVM *(in rows 4 and 5), additional training data was created by scaling each dimension of the feature vectors present in the original training dataset. The average accuracy obtained is shown in the third column.*

5.2. The Face Dataset

Recognizing faces obtained from TV video is a challenging task due to the presence of large variations in pose. We test the ILMNN classifier using one such dataset and provide a comparison with the state of the art classifiers.

Dataset: We used a publically available dataset[4] consisting of a total of $24,244$ faces (with ground truth) spread over 11 characters present in an episode of 'Buffy the Vampire Slayer'. For each face, thirteen facial features were detected using the method described in [7]. The detected features (along with their numbering) for three faces of 'Buffy' are shown in Fig. 3. Using the location of the these features, faces are mapped onto a common canonical frame. Each face

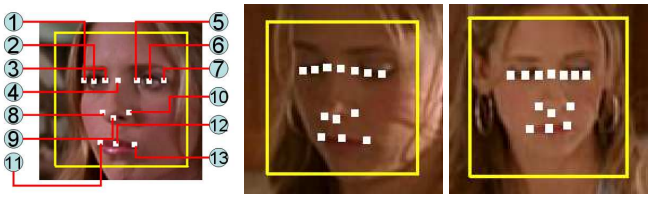

Figure 3. *Examples of 'Buffy'. The white squares indicate the positions of thirteen facial features detected. Note the variation in pose of the faces which is typical of* TV *video.*

is then represented using the local appearance around the detected facial features (similar to [7]). For this work, we use a vector of normalized grayscale values in the circular region with radius 4 around each feature. The descriptor for the face is obtained by simply concatenating these vectors (resulting in a 637 dimensional representation for each face).

We use two different splits of the dataset: (i) In the first experiment, the dataset is randomly split into a training set (30% of the original dataset), a validation set (30%) and a test set (40%). Note that in this setup several test examples lie very close to some training example in the Euclidean distance sense (since they are extracted from nearby frames where appearance does not change much); and (ii) In the second experiment, faces detected in the first 30% of the episode are used as the training set, the next 30% of the episode defines the validation set and the remaining 40% of the episode is the test set. This is a much more demanding setup than the first experiment due to substantial changes in the appearance of faces over an episode.

[4]Available at http://www.robots.ox.ac.uk/~vgg/research/nface/

Figure 4. *Zoomed-in versions of the examples in Fig. 3 showing the right eye. White squares show the location of the detected facial features corresponding to the corners of the eye. The green dots show the correct location of these features.*

For both experiments, the parameters of the classifiers are learnt using the training and validation sets. During the testing stage, the faces in the test set are identified as belonging to one of the 11 characters.

Incorporating Invariance: Due to large variations in the pose of the faces, the facial features are not found reliably. In other words, even after mapping the faces onto a common canonical framework, there remains a residual transformation (e.g. translation and rotation) between the features detected in different images (as shown in Fig. 4). To overcome this problem, we incorporate invariance of the data to multiple transformations within the ILMNN framework. For this work, we consider a 2-D Euclidean transformation. Recall that a Euclidean transformation is specified by three variables: θ (i.e. rotation), t_x (i.e. translation in the x direction), and t_y (i.e. translation in the y direction). However, note that the ILMNN framework allows for more complex transformations (e.g. affine) at the cost of more training time.

The Euclidean transformation of a face is approximated using a multivariate polynomial (with three variables, one each for θ, t_x and t_y) of degree 2 using Taylor's expansion. Images are smoothed by a Gaussian with standard deviation $\sigma = 0.3$ pixels. In order to compute the derivative of the subimage corresponding to a facial feature with respect to a variable (which is required for Taylor's expansion), a small transformation is applied to the subimage (e.g. in the case of θ, it is rotated by a small angle). The derivative is then approximated as the difference between the original subimage and its transformed version. The second derivative is computed similarly. Using the method described in section 4, we restrict the invariance to segments $-5^o \leq \theta \leq 5^o$, $-3 \leq t_x \leq 3$ and $-3 \leq t_y \leq 3$ pixels.

Training the Classifiers: We cluster the training set using k-medoid clustering (where k is chosen as approximately 10% of size of the training set). Both the LMNN and ILMNN classifiers are then trained using the cluster centres. The advantage of this is two fold: (i) since the target neighbours lie far from each other, the classifiers do not learn the trivial parameter $\mathbf{L} = \mathbf{I}$ (i.e. the identity matrix); and (ii) it reduces the computational complexity.

The LMNN classifier is trained using the publically available code (with suitable modifications to incorporate the reg-

ularizers described in section 2). The ILMNN classifier is trained using the SDPLR software [5] together with the alternative projection theorem [3]. We use $k = 1$ for both the classifiers. The values of all regularization constants (λ_h, λ_r and λ_d) are found by cross-validation. Table 2 shows the average training time required for LMNN and ILMNN. Note that the use of the generic SDPLR software (instead of the specialized software used for LMNN) increases the training time for the ILMNN classifier. However, the alternating projection theorem ensures that training is still computationally feasible.

Testing the Classifiers: The incorporation of regularizers encourages a sparse mapping parameter **L** to be learnt. Hence, during testing, the mapped points lie in a lower dimensional space. Since the computational complexity of kNN classification is proportional to the dimensionality of the data, this significantly reduces the testing time (see table 2).

Results: Table 2 shows the accuracies obtained using various classifiers. Below, we describe these results in detail.

Method	Tr.	Te.	Acc-1 (%)	Acc-2 (%)
kNN-E	-	62.2s	83.6	26.7
L^2-LMNN	4h	62.2s	61.2	22.6
D-LMNN	1h	53.2s	85.6	24.3
DD-LMNN	2h	50.5s	84.4	24.5
L^2-ILMNN	24h	62.2s	65.9	24.0
D-ILMNN	8h	**48.2s**	**87.2**	**32.0**
DD-ILMNN	24h	51.9s	86.6	29.8
M-SVM	**300s**	446.6s	62.3	30.0
SVM-KNN	-	2114.2s	75.5	28.1

Table 2. *The second column shows the average training time required for each classifier. Note that, unlike* LMNN *and* ILMNN, M-SVM *was trained using the entire training dataset. The average testing times are shown in column three. The fourth and fifth columns show the accuracy of the classifiers for the two experiments.*

(a) The LMNN Classifier: Fig. 5 shows the precision-recall curves obtained using the LMNN classifier for the two different splits of our dataset. Recall indicates the percentage of test examples whose nearest neighbours lie at a distance less than a threshold τ, while precision corresponds to the percentage of correctly classified examples. For our dataset, the best results are obtained using D-LMNN (which learns a diagonal parameter **L**) and DD-LMNN (which learns a diagonally dominant parameter). The L^2-LMNN classifier overfits to the training data despite a regularization term and hence results in high generalization error. For the first experiment, both D-LMNN and DD-LMNN provide slightly better results than kNN-E classification (i.e. kNN classification with Euclidean distance). For the second experiment, their classification accuracy is slightly worse than kNN-E (possibly because the training data is not representative of the test data).

(b) The ILMNN Classifier: Fig. 6 shows the precision-recall curves obtained using the ILMNN classifier for the two experiments. Similar to the case of LMNN, D-ILMNN provides the best results. The L^2-ILMNN classifier overfits to the training data, while DD-ILMNN learns a nearly diagonal

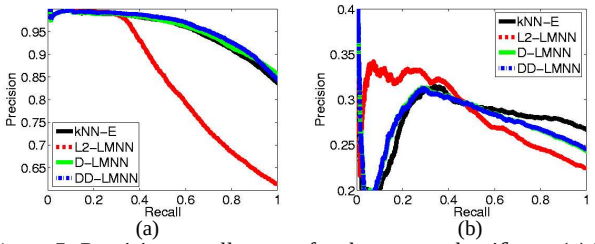

Figure 5. *Precision-recall curves for the* LMNN *classifiers. (a) First experiment. (b) Second experiment. The curves are obtained by varying the threshold τ (see text).*

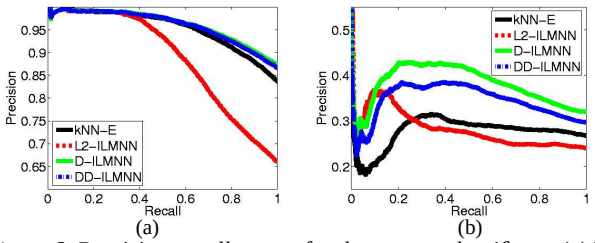

Figure 6. *Precision-recall curves for the* ILMNN *classifiers. (a) First experiment. (b) Second experiment.* ILMNN *incorporates invariance to Euclidean transformation and hence, provides better results.*

parameter and provides comparable results to D-ILMNN. By incorporating invariance to Euclidean transformation, ILMNN outperforms both the kNN-E and the LMNN classifier. Fig. 7 shows examples of true positives for each of the 11 characters obtained using the ILMNN classifier. Most of the errors in recognition occur due to the small size or inaccurate localization of the face.

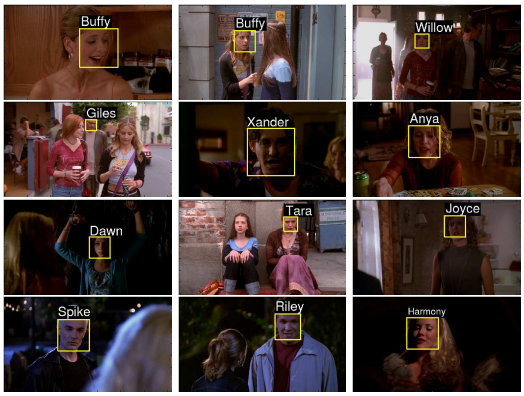

Figure 7. *Examples of true positives for the 11 characters present in the episode obtained using* ILMNN-2.

The D-ILMNN classifier was also trained under further constraints that all dimensions of the feature vector corresponding to the same facial feature share a common weight. Although this is sub-optimal, it indicates which facial features were most useful for the task of classification. For this purpose, we used 10 different training datasets by randomly selecting 60% of the faces in the original dataset. The average weights learnt for each feature vector, together with their standard deviations, are shown in table 3. The two features with the lowest weights are the inner eye corner (feature 3)

	1	2	3	4	5	6	7	8	9	10	11	12	13
Weight	1.19	0.71	0.49	1.32	0.87	1.25	1.04	1.14	1.15	0.67	1.51	1.18	0.48
Std. Dev.	0.14	0.11	0.11	0.12	0.10	0.09	0.14	0.10	0.14	0.13	0.13	0.12	0.18

Table 3. *The average weights and the standard deviation obtained using* D-ILMNN *when all dimensions corresponding to a facial feature share the same weight. The top row lists the feature numbers which correspond to those shown in Fig. 3.*

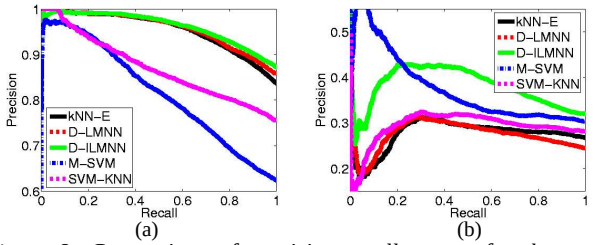

Figure 8. *Comparison of precision-recall curves for the* LMNN, ILMNN, M-SVM *and* SVM-KNN *classifi ers. (a) First experiment. (b) Second experiment. The* ILMNN *classifi er outperforms the state of the art classifi ers on our dataset.*

which is often occluded by the nose, and the mouth corner (feature 13) whose appearance varies due to different facial expressions. The location of the eye centre (feature 2) is computed as the average of positions of the eye corners (features 1 and 3). Hence, its low weight may be related to the problems with feature 3.

(c) Comparison: We compared the performance of ILMNN with two other state of the art classifiers, namely M-SVM and SVM-KNN [23]. The M-SVM classifier was trained using the entire training set instead of just the cluster centres. The value of k for the SVM-KNN classifier is chosen using cross-validation (with a lower bound of $k = 5$ to prevent it from reducing to the kNN-E classifier). Fig. 8 provides a comparison of the precision-recall curves obtained using our method and the above mentioned classifiers. Since the first experiment is more suitable for kNN classification, M-SVM has the worst accuracy. For the second experiment, it provides better results than kNN-E, D-LMNN and SVM-KNN. However, note that by incorporating invariance, D-ILMNN outperforms all the classifiers in both experiments.

6. Discussion

The LMNN and ILMNN formulations presented in this paper are quite generic and can possibly improve distance metrics for other standard features such as SIFT [16]. This needs further investigation. The LMNN framework can be used to learn a mapping in nonlinear space using the kernel trick [9]. For a large and useful class of polynomial transformations, the kernel trick can also be applied within the ILMNN framework. Both the LMNN and the ILMNN formulations can be extended to handle missing or noisy data (similar to the method described in [20]). Further, the D-LMNN classifier can be easily modified to learn a diagonal parameter $\mathbf{L}$ which optimizes χ^2 distance (instead of the Euclidean distance presented in this paper). However, the practical benefits of these extensions have not been evaluated. An interesting direction for future research is to learn many local mappings instead of the global mapping learnt by LMNN and ILMNN.

Acknowledgments: This work was supported by the EPSRC research grant EP/C006631/1(P), the IST Programme of the European Community, under the PASCAL Network of Excellence, IST-2002-506778, and a MURI grant.

References

[1] V. Athitsos, J. Alon, S. Sclaroff, and G. Kollios. BoostMap: A method for effi cient approximate similarity rankings. In *CVPR*, pages 268–275, 2004.
[2] A. Bar-Hillel, T. Hertz, N. Shental, and D. Weinshall. Learning a Mahalanobis metric from equivalence constraints. *JMLR*, 2005.
[3] H. Bauschke and J. Borwein. On projection algorithms for solving convex feasibility problems. *SIAM Review*, 38(3):367–426, 1996.
[4] M. Brown and D. Lowe. Recognizing panoramas. In *ICCV*, pages 1218–1225, 2003.
[5] S. Burer and R. Monteiro. A nonlinear programming algorithm for solving SDP via low-rank factorization. *Mathematical Programming*, 95(2):329–357, 2003.
[6] T. Cover and P. Hart. Nearest neighbour pattern classifi cation. *Trans. on Info. Th.*, IT-13:21–27, 1967.
[7] M. Everingham, J. Sivic, and A. Zisserman. Hello! My name is... Buffy – Automatic naming of characters in TV video. In *BMVC*, 2006.
[8] A. Frome and J. Malik. Object recognition using locality senstive hashing of shape contexts. In *Nearest-neighbour methods in learning and vision*. MIT Press, 2006.
[9] A. Globerson and S. Roweis. Metric learning by collapsing classes. In *NIPS*, 2005.
[10] J. Goldberger, S. Roweis, G. Hinton, and R. Salakhutdinov. Neighbourhood componenets analysis. In *NIPS*, pages 513–520, 2005.
[11] T. Graepel and R. Herbrich. Invariant pattern recognition by semidefinite programming machines. In *NIPS*, pages 33–40, 2004.
[12] A. Holub, Y. Liu, and P. Perona. On constructing facial similarity maps. In *CVPR*, pages 1–8, 2007.
[13] F. Kahl and D. Henrion. Globally optimal estimates for geometric reconstruction problems. In *ICCV*, pages II: 978–985, 2005.
[14] M. P. Kumar, P. H. S. Torr, and A. Zisserman. Invariant large margin nearest neighbour classifi er. Technical report, Oxford Brookes University, 2007.
[15] J. Lasserre. Global optimization with polynomials and the problem of moments. *SIAM Journal of Optimization*, 11:796–817, 2001.
[16] D. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2):91–110, 2004.
[17] Y. Nesterov. Squared functional systems and optimization problems. In *High Performance Optimization*, pages 405–440, 2000.
[18] G. Shakhnarovich, P. Viola, and T. Darrell. Fast pose estimation with parameter sensitive hashing. In *ICCV*, pages 750–759, 2003.
[19] P. Simard, Y. LeCun, and J. Decker. Effi cient pattern recognition using a new transformation distance. In *NIPS*, volume 6, pages 50–58, 1993.
[20] P. Sivaswamy, C. Bhattacharya, and A. Smola. Second order cone programming approaches for handling missing and uncertain data. *JMLR*, 7:1283–1314, 2006.
[21] K. Weinberger, J. Blitzer, and L. Saul. Distance metric learning for large margin nearest neighbor classifi cation. In *NIPS*, 2005.
[22] L. Yang, R. Jin, R. Sukhtankar, and Y. Liu. Effi cient algorithm for local distance metric learning. In *AAAI*, 2006.
[23] H. Zhang, A. Berg, M. Maire, and J. Malik. SVM-KNN: Discriminative nearest neighbor classifi cation for visual category recognition. In *CVPR*, pages 2126–2136, 2006.