Optimizing 1-Nearest Prototype Classifiers

Paul Wohlhart, Martin Köstinger, Michael Donoser, Peter M. Roth, Horst Bischof
Institute for Computer Vision and Graphics, Graz University of Technology, Austria
{wohlhart,koestinger,donoser,pmroth,bischof}@icg.tugraz.at

Abstract

The development of complex, powerful classifiers and their constant improvement have contributed much to the progress in many fields of computer vision. However, the trend towards large scale datasets revived the interest in simpler classifiers to reduce runtime. Simple nearest neighbor classifiers have several beneficial properties, such as low complexity and inherent multi-class handling, however, they have a runtime linear in the size of the database. Recent related work represents data samples by assigning them to a set of prototypes that partition the input feature space and afterwards applies linear classifiers on top of this representation to approximate decision boundaries locally linear. In this paper, we go a step beyond these approaches and purely focus on 1-nearest prototype classification, where we propose a novel algorithm for deriving optimal prototypes in a discriminative manner from the training samples. Our method is implicitly multi-class capable, parameter free, avoids noise overfitting and, since during testing only comparisons to the derived prototypes are required, highly efficient. Experiments demonstrate that we are able to outperform related locally linear methods, while even getting close to the results of more complex classifiers.

1. Introduction

Over the last decade we have seen an impressive progress in important fields of computer vision like image classification and object detection, which is mainly a result of developments in supervised machine learning and the steadily growing amount of available training data. Large-scale datasets for effectively training classifiers have already proven to be quite beneficial in several contexts. For example, the seminal work on using random forest classifiers on single depth images for estimating human poses (Microsoft Kinect™) [15] heavily exploits a huge set of synthetically generated human pose depth images during training. Also, the recent advances on the PASCAL visual object recognition challenge can largely be attributed to the steady increase in training database sizes [19].

Since powerful classifiers such as non-linear kernel Support Vector Machines (SVMs) have shown excellent performance in diverse computer vision applications it seems to be a natural choice to directly apply them to the available large-scale datasets. Unfortunately, such non-linear kernel classifiers are in general computationally too expensive for applications in large-scale settings. For this reason, explicit feature mappings have been proposed that approximate specific kernels in a linear manner. For example in [11] an approximation of the popular intersection kernel was described. This approach was then generalized to the set of homogeneous, additive kernels (including the intersection and chi-squared kernel) in [16]. Nevertheless, not all kernels can be approximated in such a manner. Furthermore, optimal classification decision boundaries do not necessarily exhibit the structure as defined by a kernel.

On the contrary, linear classifiers are fast enough to be applicable in large-scale scenarios, especially due to recent developments in efficiently solving the corresponding optimization problem. For example, in [14] a stochastic gradient descent approach for solving the optimization problem of linear SVMs was introduced, where the total runtime does not directly depend on the size of the training set. The approach also extends to non-linear kernels by solely working on the primal, nevertheless, in these cases runtime scales linearly with training set size.

Unfortunately, a global, linear model is seldom powerful enough to separate the data. For this reason, locally linear classifiers have been attracting a lot of attention recently. The core idea of these methods is to define the optimal decision boundary by locally linear approximations that are defined on prototypes. These prototypes (or anchor points) reside in the same space as the input data points and the classifier is then trained and evaluated in the context of a local neighborhood around them. A recent example for such a classifier is the Locally Linear SVM (LL-SVM) proposed in [9], where the non-linear decision boundary is approximated with many linear SVMs. Data points are soft-assigned to anchor points identified by k-means and SVM parameters are jointly learned in a common optimization problem. This can be solved by stochastic gradient descent,

with the same convergence guarantees as for standard, linear SVMs. In a related work a multi-class classifier denoted as Parametric Nearest Neighbor (PNN) and its ensemble extension (EPNN) was proposed [18]. The core idea is to embed data points into a new feature space, again defined by soft assignments to prototypes, and to learn a linear max-margin classifier on this new representation. An iterative method is introduced, that alternately optimizes the prototypes and then the classifier parameters. Interestingly, reasonable performance is only obtained when the optimization is not run until convergence and an ensemble of such base classifiers is formed.

The necessity for large-scale classification has in general led to a comeback of non-parametric classifiers like the nearest neighbor algorithm. Nearest neighbor classifiers have several interesting properties: (a) they allow implicit multi-class decisions and naturally handle even a huge number of classes, (b) they, in general, avoid overfitting, which is important to obtain reasonable classification accuracy and (c) they do not require any training effort. For example, in [1] it was shown that for Bag-of-words based image classification, simple adaptations in the visual word quantization and in the image-to-image distance calculation are sufficient to achieve state-of-the-art results using a simple nearest neighbor method as subsequent classifier.

Nearest neighbor classification especially shows promising performance in combination with metric learning (*e.g.*, [17]), where the assignment to the nearest neighbors is based on a Mahalanobis distance metric instead of the common Euclidean distance. In general, such methods aim at improving the nearest neighbor classification, *e.g.*, by ensuring that all k-nearest neighbors belong to the same class, whereas examples from differing classes should be separated by a large margin. In such a way, these methods are a counterpart to max-margin classifiers like SVMs, where nearest neighbor assignments replace the weighted sum of (kernelized) distances to support vectors.

An interesting way to further reduce the complexity of nearest neighbor classifiers is to constrain the analysis to selected prototypes, instead of considering all training data samples during testing. Obviously, performance is heavily depending on a reasonable selection of the prototypes. Finding optimal prototypes from labeled data has a long history, where methods are in general referred to as Learning Vector Quantization (LVQ). This field was initiated by the seminal work of [8], where a heuristic was used to move prototypes close to training samples of the same class and away from other classes. The method was later adapted in [13], where the problem was formulated as an optimization problem maximizing the likelihood ratio between the probability of correct classification and the total probability in a Gaussian Mixture Model (each prototype is represented as a Gaussian). In [3] a supervised vector quantization based on max-margin principles was introduced and it is demonstrated that the original LVQ [8] represents a specific instance of the proposed framework. Remarkably, the authors demonstrate that prototype based classification is even able to outperform nearest neighbor classification using all training data due to improved generalization properties caused by a reduction of over-fitting to noise.

In this paper, we address large-scale classification by a prototype based nearest neighbor approach. We aim to go beyond the most related methods of [9, 18], where local codings based on prototypes are used in a subsequently applied classifier, by completely focusing on the most simple variant: a 1-nearest neighbor (NN) algorithm. Instead of considering all N training samples in the 1-NN algorithm as done in [8, 13, 3], we aim at discriminatively deriving a small, optimal set of P prototypes with $P \ll N$ and base our classification decision completely on the single label of the nearest neighbor within this prototype set.

To reach this goal, we present a differentiable probabilistic relaxation of 1-NN classification, based on soft assignments of samples to prototypes, that can be optimized using gradient descent. It has one parameter controlling the softness of the assignment. Letting this parameter go to infinity leads to pure 1-NN classification. Thus, in contrast to related methods, we can provide discriminative prototypes directly optimized for subsequent pure 1-NN classification, by gradually decreasing the softness of the assignment while optimizing the prototypes.

Our method exhibits the same advantages as standard prototype based nearest neighbor classification: (a) It is parameter free, (b) it is implicitly able to handle large numbers of classes, (c) it avoids over-fitting to noise and generalizes well to previously unseen samples as also demonstrated in the experiments, and (d) it has very low computational costs during testing, since only distances to the P prototypes have to be evaluated (with possible further improvements using fast nearest neighbor assignment methods like KD-trees). In thorough experiments, we demonstrate that our discriminatively learned prototypes consistently outperform the k-means baseline, even with very low numbers of prototypes. Furthermore, we show that already a very small number of prototypes is sufficient to obtain reasonable classification accuracy on several datasets. Our method naturally allows to integrate learned Mahalanobis metrics like [17]. However, we show that by learning the prototypes in the proposed discriminative manner, learning Mahalanobis metrics becomes less important.

The outline of the paper is as follows. We first show in Section 2, how to relax the 1-nearest neighbor classification to a soft-max variant, which then allows to find the prototypes using gradient descent by minimizing the empirical risk of misclassification over the entire training dataset. We derive formulations for the exponential and the hinge

loss, and discuss possible influences on the results. Section 3 gives illustrative results on toy examples, as well as a thorough comparison to state-of-the-art on three standard classification datasets.

2. Method

Given a training set $\mathcal{T}$ with labeled samples (x_i, y_i), where $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$, the goal is to define a classifier $f(x) = y$ that correctly classifies test samples from the same distribution. As discussed before, to reduce the model complexity and the test time, our proposed classifier is a prototype based 1-nearest neighbor algorithm. We aim at deriving discriminative, labeled prototypes $(p_j, \theta_j) \in \mathcal{P},\ p_j \in \mathbb{R}^d,\ \theta_j \in \{-1, 1\}$, that are optimized for simple 1-nearest neighbor classification:

$$f(x) = \theta_k, \text{ where } k = \arg\min_j ||x - p_j|| \ . \tag{1}$$

To get an optimal classifier, the prototypes p_j must be arranged such as to minimize the empirical risk of misclassification. Thus, we formulate the optimization of the prototypes as a gradient descent on a loss function over the training examples. Since the original formulation of 1-NN shown in Eq. (1) is not differentiable, we relax the hard assignment of samples to prototypes to a soft-max formulation over sample similarities (*i.e.*, soft-min over distances). The following sections describe the setup of the relaxed classifier.

2.1. Classifier

Given the set of optimized prototypes, we define the classifier as the weighted sum of the prototypes' labels:

$$f(x) = \sum_{j=1}^{|\mathcal{P}|} \theta_j w_j(x) \ . \tag{2}$$

The weights are defined by a soft-max over the similarities between input sample x and prototypes p_j:

$$w_j(x) = \frac{s(x, p_j)^\gamma}{\sum_{k=1}^{|\mathcal{P}|} s(x, p_k)^\gamma} \ , \tag{3}$$

where $s(x_1, x_2)$ is the similarity of samples x_1 and x_2, and γ controls the degree of softness in the assignment, which will be discussed in Sec. 2.2. The weights for all prototypes for one training sample sum up to 1, *i.e.*, $\forall_x : \sum_{j=1}^{|\mathcal{P}|} w_j(x) = 1$, resulting in a probabilistic output for the overall classifier.

The similarity of two samples is defined as the negative exponential of the distance between the samples:

$$s(x_1, x_2) \quad = \quad \exp(-d_\star(x_1, x_2)) \ , \tag{4}$$

where $d_\star$ is a distance and typically chosen to be the squared Euclidean distance:

$$d_{\text{euc}}(x_1, x_2) \quad = \quad ||x_1 - x_2||^2 \ . \tag{5}$$

Inspired by the recent developments in metric learning, we also consider learned Mahalanobis distance metrics M to define the distances between samples:

$$d_{\text{metric}}(x_1, x_2) \quad = \quad (x_1 - x_2)^T M (x_1 - x_2) \ . \tag{6}$$

In practice, we do not use d_{metric} directly, but decompose M into $M = L^T L$ and project the input features x onto L by $\hat{x}_i = L x_i$. In this way, we can again use the Euclidean distance to measure distances in the transformed space.

2.2. Choosing γ

Notice, if γ is set to infinity (or a very high value in practice) when calculating the weights in Eq. (3), every sample gets assigned a weight of 1 only for its closest prototype. Thus, in the soft-max classifier shown in Eq. (2) the sample is directly assigned the label of its closest prototype, exactly as in the original 1-NN formulation. With lower values of γ more weight is given to the next closest prototypes and more distant interactions are established, until for $\gamma = 0$ all locality vanishes and the result of a classification is just the prior distribution of the training labels. This indicates that the choice of γ essentially influences the results.

We would like to emphasize that the soft-max classifier defined in the previous section is only used as a surrogate for our finally applied classifier. The discriminatively derived prototypes are then supposed to be used in 1-NN classification, as in Eq. (1). If we were sacrificing the benefit of the simple 1-NN classification and looking for an optimal soft-max classifier as defined in Eq. (2), we would also need to optimize for γ.

However, for our goal of defining an effective 1-NN classifier, we consider an iterative approach, where we start with a low value for γ. In this way, we allow more than one prototype to have influence on a sample, and thus vice versa, let each sample have influence on more than one prototype. For the current γ-value, we derive optimal prototypes using a gradient descent approach as outlined in the next section. Then, in each iteration, we gradually increase γ and again optimize the prototypes, until virtually all samples are only influenced by a single prototype, arriving at pure 1-NN classification. In such a way, we can avoid local minima resulting from overfitting.

In practice, we initialize γ such that the difference between the highest and the second highest prototype weight is less than 0.5 for at least 80% of the training samples. We then increase γ until for every training sample there is significant weight for only a single prototype (*i.e.*, the sum over

all other weights drops below a very low threshold), but calculation of the weights is still numerically stable. As intermediate steps we interpolate 10 values between the lowest and the highest value on a logarithmic scale.

2.3. Empirical loss over training data

In order to optimize the prototypes with gradient descent, we need to design a derivable loss function, measuring the quality of the output of the current probabilistic classifier setup. Given the definition of the classifier, the loss over all training samples is defined by

$$\mathcal{L}(\mathcal{T}, \Delta_\star) = \sum_{i=1}^{|\mathcal{T}|} \Delta_\star(f(x_i), y_i) \ , \tag{7}$$

where $\Delta_\star$ is a function measuring the loss of the classification outcome. In particular, we apply two types of loss functions, namely the exponential loss

$$\Delta_{\text{exp}}(f(x_i), y_i) = \exp\left(-y_i f(x_i)\right) \tag{8}$$

and the hinge loss

$$\Delta_{\text{hinge}}(f(x_i), y_i) = \max(0, \delta - y_i f(x_i)) \ , \tag{9}$$

which is more robust to outliers, where with δ the margin can be adjusted. Due to the probabilistic formulation of our classifier, an output of more than 0.5 for the correct class means that no other class can have a larger value and δ can be adjusted to preserve a margin but not to overfit. A hinge loss formulation was also used in [18].

2.4. Derivatives w.r.t. prototypes

In order to perform the gradient descent, we need the derivatives w.r.t. the prototype vectors. Since we have different choices at some parts of the formulation, we show the partial derivative for each component individually in Figure 1.

2.5. Multiclass

For notational convenience we have only discussed binary classification up to this point. In the case of multiclass classification tasks with C classes we adopt a lifted formulation and extend training and prototype labels to vectors $\mathbf{y}_i, \theta_j \in \{-1, 1\}^C$ (*i.e.*, $\mathbf{y}_i = (y_{i,1}, \ldots, y_{i,C})$), where $y_{i,c_i} = 1$ for a sample x_i of class c_i and all other entries are -1. The overall classifier then takes the $\max$ over the predictions for each class

$$F(x) = \arg\max_c f_c(x) \ , \tag{19}$$

where $f_c(x) = \sum_{j=1}^{|\mathcal{P}|} \theta_{j,c} w_j(x)$.

Total Loss

$$\frac{\partial \mathcal{L}(\mathcal{T}, \Delta_\star)}{\partial p_l} = \sum_{i=1}^{|\mathcal{T}|} \frac{\partial \Delta_\star(f(x_i), y_i)}{\partial p_l} \tag{10}$$

Loss Functions

$$\frac{\partial \Delta_{\text{exp}}(f(x_i), y_i)}{\partial p_l} = -y_i \exp(-y_i f(x_i)) \frac{\partial f(x_i)}{\partial p_l} \tag{11}$$

$$\frac{\partial \Delta_{\text{hinge}}(f(x_i), y_i)}{\partial p_l} = \begin{cases} -y_i \frac{\partial f(x_i)}{\partial p_l} & \text{if } y_i f(x_i) < \delta \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

Classifier

$$\frac{\partial f(x_i)}{\partial p_l} = \sum_{j=1}^{|\mathcal{P}|} \theta_j \frac{\partial w_j(x_i)}{\partial p_l} \tag{13}$$

Weights

$$\frac{\partial w_j(x_i)}{\partial p_l} = \frac{\gamma w_j(x_i)}{s(x_i, p_j)} \frac{\partial s(x_i, p_j)}{\partial p_l} - \tag{14}$$

$$- \frac{\gamma w_j(x_i)}{s(x_i, p_l)} w_l(x_i) \frac{\partial s(x_i, p_l)}{\partial p_l} \tag{15}$$

Similarities

$$\frac{\partial s(x_i, p_j)}{\partial p_l} = -s(x_i, p_j) \frac{\partial d_\star(x_i, p_j)}{\partial p_l} \tag{16}$$

Distances

$$\frac{\partial d_{\text{euc}}(x_i, p_j)}{\partial p_l} = \begin{cases} -2(x_i - p_l) & \text{if } l = j \\ 0 & \text{else} \end{cases} \tag{17}$$

Using the Euclidean distance $d_\star = d_{\text{euc}}$, the derivation of the classifier w.r.t. the prototypes thus reduces to

$$\frac{\partial f(x_i)}{\partial p_l} = 2\gamma w_l(x_i)(x_i - p_l)\left(\theta_l - f(x_i)\right) \ . \tag{18}$$

Figure 1. Partial derivatives for each individual part of the classification loss w.r.t prototype locations, as needed for the optimization with gradient descent.

The multi-class loss is then defined as the sum over the losses for each individual class:

$$\mathcal{L}(\mathcal{T}, \Delta_\star) = \sum_{c=1}^{C} \mathcal{L}_c(\mathcal{T}, \Delta_\star) \ , \tag{20}$$

where $\mathcal{L}_c(\mathcal{T}, \Delta_\star) = \sum_{i=1}^{|\mathcal{T}|} \Delta_\star(f_c(x_i), y_{i,c})$.

We also considered a formulation that optimizes the margin between the classification of the target class and the out-

put for the most confident non-target class for each sample, instead of the output for all classes:

$$\mathcal{L}(\mathcal{T}, \Delta_\star) = \sum_{i=1}^{|\mathcal{T}|} \Delta_\star(f_{c_i}(x_i), y_{i,c_i}) + \max_{c_j \neq c_i} \Delta_\star(f_{c_j}(x_i), y_{i,c_j}) \ . \tag{21}$$

Such a formulation was chosen in [18]. However, although this formulation yields meaningful results on toy examples, it consistently produced considerably worse result than the formulation in Eq. (20) on real world data, which is why we excluded it from further considerations.

3. Experiments

We first outline characteristics of our proposed method for selecting optimal prototypes for 1-NN classification on two toy examples in Section 3.1. Then, Section 3.2 provides a thorough quantitative comparison of our algorithm to related work on real world computer vision datasets.

In all experiments, we initialize our method using prototypes defined by applying k-means clustering for each class individually. We also experimented with taking a random subset of the training data as initial prototypes. This usually leads to similar results, but is more susceptible to local minima, especially if modes of the distribution do not get a nearby, initial prototype. In contrast, k-means initialization ensures a certain coverage of the training sample distribution. Starting from the k-means initialization, we apply our iterative prototype updating as described in the previous section using the non-linear conjugate gradient descent method (`ncg`) from the Poblano Toolbox [4]. Our final classifier is then defined by using the obtained prototypes in a 1-NN assignment strategy, as defined in Eq. (1).

3.1. Illustrative toy examples

We illustrate how prototypes evolve during our optimization for two 2D toy examples, namely a dataset consisting of three classes with multimodal distributions and a dataset of 10 classes with regular structure. Figures 2 and 3 show prototypes and the consequently emerging decision surfaces for 1-nearest neighbor classification using either the exponential or the hinge loss during our iterative prototype update method. Results are shown starting from k-means clustering and updating until convergence (Fig. 2(a)-2(e) and 3(a)-3(e)). Additionally, we illustrate classification performance on train and test data during optimization (Fig. 2(f)-(g) and 3(f)-(g)), where the x-axis represents the value of γ used to optimize the prototypes in the current iteration of the algorithm, the y-axis represents values of γ used in the soft-max over similarities, and the height of the mesh reflects the corresponding classification accuracy.

Note that the decision surfaces become much smoother during prototype evolution and that overfitting, prevalent for the initial k-means clustering, is drastically reduced. The hinge loss smoothes the decision boundary more than the exponential loss, where for higher values of γ the decision boundaries tend to fit more around single samples that are scattered into areas that are more densely populated by samples from differing classes. Note also the very different configuration of the finally obtained prototypes for the two different loss functions. Generally, with the hinge loss the prototypes tend to be grouped together to more generic locations, whereas with the exponential loss prototypes distribute more to also capture instances in low density areas. Whether this is beneficial depends on the distribution and especially separability of the data. The difference is also reflected in the performance plots, which show steadily increased training accuracy for increasing γ values in both cases, but a slight breakdown in the test accuracy for the exponential loss. However, the performance is still better compared to the k-means initialization.

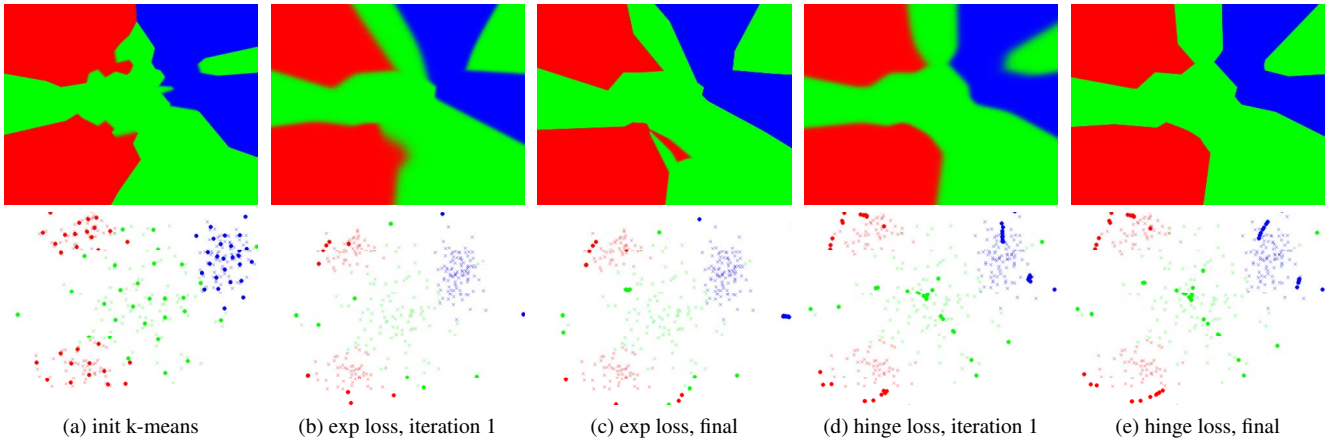
3.2. Image classification

We evaluate our method on three widely used classification datasets, each providing sufficient training samples per class, such that a reduction to a small set of discriminative prototypes really pays off.

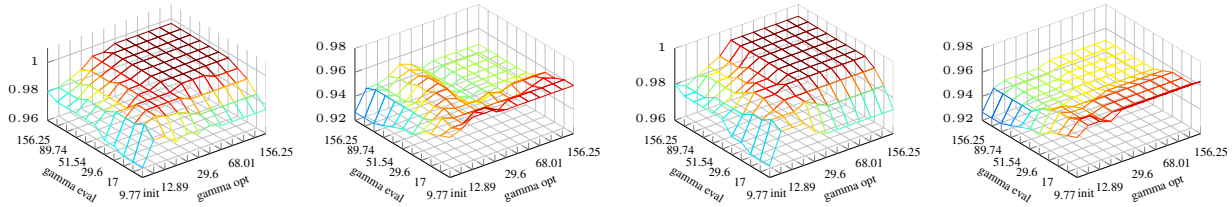
The first database is USPS [7], which contains 7291 training and 2007 test gray-scale images of the digits '0' to '9', stored with a resolution of 16×16 pixels. Additionally, we test on the LETTER database [6], which contains 16000 training and 4000 testing images of the letters 'A' to 'Z'. Each of them is represented by a 16 dimensional feature vector. Finally, we also evaluate on MNIST [10]. It contains 60000 training and 10000 test gray-scale images of the digits '0' to '9' with a resolution of 28×28 pixels. For MNIST we took the publicly available data from [12] and LETTER and USPS are taken from [9].

We list classification errors on these three datasets for all variants of our method in comparison to related work in Table 1. Our proposed Nearest Prototype Classifier (NPC) with exponential loss is denoted by *NPC-e* and with hinge loss *NPC-h*. We show results for different numbers k of prototypes per class. For example, a value of $k = 15$ means a reduction to 0.25% of the number of available training samples for MNIST, 2.06% for USPS and 2.4% for LETTER. Table 1 further shows results if using a learned global Mahanalobis metric obtained by applying LMNN [17] instead of standard Euclidean distances. Additionally, the table outlines results for the most related state-of-the-art approaches [9, 18], as well as linear and RBF-kernel SVMs.

Discussion As can be seen in Table 1 our proposed Nearest Prototype Classifier (NPC) achieves highly competitive error rates in comparison to related state-of-the-art methods although solely relying on simple prototype based 1-nearest

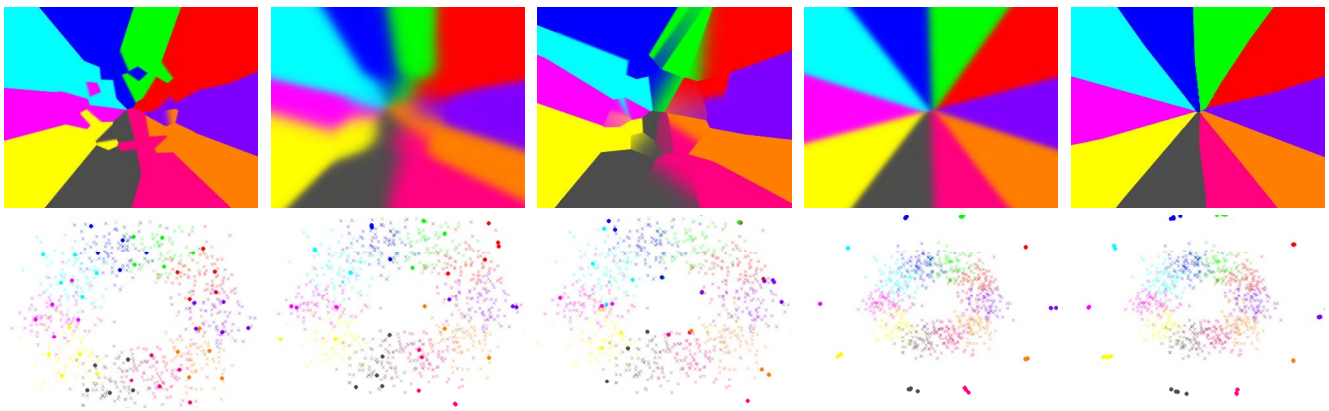


(a) init k-means (b) exp loss, iteration 1 (c) exp loss, final (d) hinge loss, iteration 1 (e) hinge loss, final

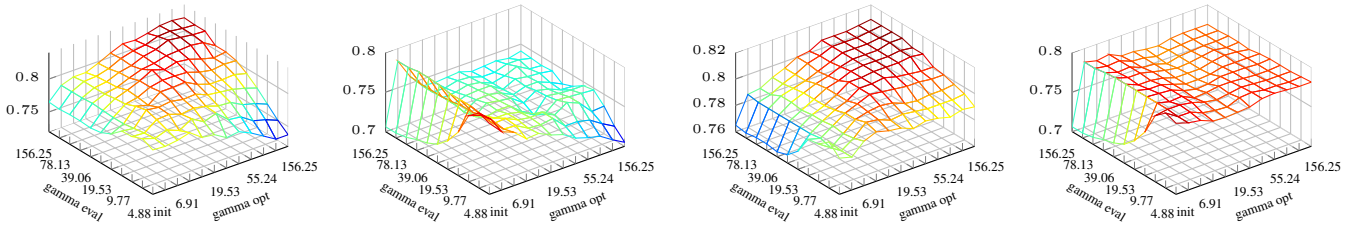


(f) train (left) and test (right) accuracy, exp loss (g) train (left) and test (right) accuracy, hinge loss

Figure 2. Evolution of prototypes during learning on a 2D toy example with 3 classes. Crosses denote samples, circles are prototypes.



(a) init, k-means (b) exp loss, iteration 1 (c) exp loss, final (d) hinge loss, iteration 1 (e) hinge loss, final



(f) train (left) and test (right) accuracy, exp loss (g) train (left) and test (right) accuracy, hinge loss

Figure 3. Same as in Figure 2 for a 10 class toy example. Each of the classes if defined by a single Gaussian. However the different Gaussians overlap significantly, such that samples of individual classes get scattered into the area of the neighboring ones.

k	dist	method	USPS	Letter	MNIST
all	euc	1-nn	5.08	4.35	2.43
	lmnn	1-nn	5.03	2.92	2.38
15	euc	k-means	7.67	13.75	4.98
		NPC-h	6.23	4.77	2.24
		NPC-e	5.38	3.13	2.11
	lmnn	k-means	6.43	7.40	5.08
		NPC-h	5.73	4.33	2.55
		NPC-e	5.43	3.20	2.22
30	euc	k-means	6.88	8.23	4.15
		NPC-h	5.78	4.33	2.18
		NPC-e	4.98	3.43	1.72
	lmnn	k-means	5.98	5.25	4.17
		NPC-h	6.03	3.25	2.32
		NPC-e	5.33	3.00	1.84
50	euc	k-means	6.38	6.55	3.50
		NPC-h	5.73	4.70	1.89
		NPC-e	**4.58**	3.35	**1.57**
	lmnn	k-means	5.68	4.35	3.48
		NPC-h	5.33	3.95	2.09
		NPC-e	5.28	2.70	1.76
100	euc	k-means	5.23	5.83	3.11
		NPC-h	5.68	4.37	1.87
		NPC-e	4.98	2.85	1.69
	lmnn	k-means	5.43	3.67	2.99
		NPC-h	5.78	2.93	1.80
		NPC-e	5.03	**2.55**	1.61
40	PNN [18]		7.87	6.59	3.13
800	EPNN [18]		4.88	2.90	1.65
LL-SVM [9]			5.78	5.32	1.85
LIBSVM-RBF [2]			4.58	2.12	1.36
Linear SVM [5]			8.32	23.63	8.18

Table 1. Comparison of all variants of our method and related work on USPS, LETTER and MNIST. All numbers are percentages of the classification error. The scores of the best setting per database for our algorithm are marked bold.

neighbor classification. In general, using our learned prototypes consistently outperforms the baseline, where prototypes are defined by k-means clustering results. In several cases (*e.g.* USPS, NPC-e, k=50) our nearest prototype classifier even outperforms 1-NN classification using all training data samples, although only a few discriminative prototypes are considered (*e.g.*, for k = 50 only 7% of the number of training samples of USPS). This clearly demonstrates the ability of our method to identify powerful prototypes in a discriminative manner, while reducing overfitting to noise and improving generalization properties.

As opposed to the simple 2D toy examples, the exponential loss generally performs a bit better than the hinge loss. Apparently, for this kind of data and the relatively low number of prototypes, it is more important to better fit to all the data and overfitting is not so much of an issue.

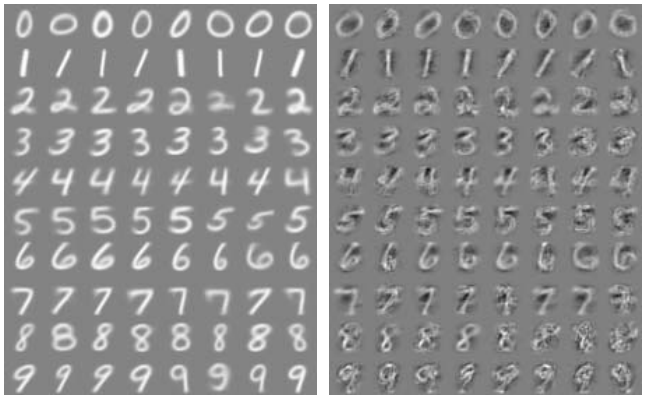
The effect of using the learned distance metric instead of Euclidean distances on the classification error varies over the different settings. When using all training samples or the initial k-means prototypes for the 1-NN classification, the learned metric performs consistently better. Nevertheless, if using our discriminatively learned prototypes, the plain Euclidean distance shows better performance, especially when the number of learned prototypes is low.

Compared to state-of-the-art methods our Nearest Neighbor Classifier (NPC) delivers highly competitive results. Already with only 15 prototypes, it consistently outperforms PNN [18], which uses 40 prototypes per class. If using 50 prototypes, we also improve over the ensemble version EPNN [18], which uses 800 prototypes (40 per class $\times$ 20 weak classifiers). This might be devoted to the more discriminative nature of our prototypes. In comparison to LL-SVM [9], our method also performs better starting from $k = 15$ on USPS and LETTER, and $k = 30$ for MNIST. LL-SVM relies on only 100 anchor-points in total (not per class), but afterwards has to re-evaluate a weighted sum over multiclass SVMs at runtime. Additionally, as shown in [9], increasing the number of anchor points does not further improve the results of LL-SVM.

Using our best setup, NPC even gets close to classification error rates of significantly more complex models (in terms of test time operations) such as non-linear kernel SVMs. For example, if using 50 prototypes per class, Euclidean distances and the exponential loss we reach the same performance as a multiclass SVM with RBF kernel on the USPS dataset. On the other two datasets, results are in close range (using LMNN for the LETTER database).

3.3. Visualization of learned prototypes

Since the input to our classifier is a vector representation of raw pixel values, it is also possible to visualize the learned prototypes in comparison to standard k-means results. Figure 4 visualizes obtained prototypes for the MNIST dataset, where the discriminatively identified prototypes are sorted per class in descending importance, analyzing the sum of weights assigned by the training samples. As can be seen, prototypes obtained by k-means clustering purely capture generative aspects of different modes of the training data, while our learning approach adds discriminative information. For example, around the strokes of the individual symbols, as well as in the center of the digit "0", there must not be other scribbles. Our prototypes reflect this by featuring negative values in those areas, with the purpose of increasing the distance to samples of other classes. This effect is also more pronounced than in [18], most likely because we let the optimization procedure converge to a minimum instead of stopping after a single iteration.



(a) initial prototypes (b) learned discriminative prototypes

Figure 4. Samples of some of the prototypes for MNIST (a) initialized by k-means and (b) learned with our algorithm. Intensity levels are scaled such that 0 is grey, lighter pixels are positive and darker negative.

4. Conclusion

Prototype based 1-nearest-neighbor (NN) classification is one of the simplest and fastest supervised machine learning methods. In this paper we showed that by a probabilistic relaxation of the 1-NN classification into a soft-max formulation, we are able to derive optimal prototypes by gradient descent in a discriminative manner. Since during testing we only have to evaluate distances to the few prototypes obtained, our proposed method is highly efficient, where a further speedup would be possible using approximated nearest neighbor methods like KD-trees. We further demonstrated in the experiments that based on our properly learned set of discriminative prototypes, we can achieve state-of-the-art results on challenging datasets, even getting close to the performance of significantly more complex classifiers like non-linear kernel SVMs.

Additionally, we have noticed that, when starting with a setting considering more long range interactions, prototypes tend to group together at generic locations. This motivates further research on how to start with a rather high number of prototypes and select the best ones afterwards. Additionally, it would be interesting to build an anytime algorithm, by ranking prototypes by importance, such that during testing the distance calculation could begin at the most important ones and stop after a time budget is used up, always giving the best possible classification up to that point.

Acknowledgement. This work was supported by the Austrian Science Foundation (FWF) project Advanced Learning for Tracking and Detection in Medical Workflow Analysis (I535-N23), the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 601139 (CultAR) and by the Austrian Research Promotion Agency (FFG) project FACTS (832045).

References

[1] O. Boiman, E. Shechtman, and M. Irani. In defense of Nearest-Neighbor based image classification. In *Proc. CVPR*, 2008.

[2] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Trans. on Intelligent Systems and Technology*, 2:27:1–27:27, 2011.

[3] K. Crammer, R. Gilad-bachrach, A. Navot, and N. Tishby. Margin analysis of the LVQ algorithm. In *Advances NIPS*, 2002.

[4] D. M. Dunlavy, T. G. Kolda, and E. Acar. Poblano v1.0: A matlab toolbox for gradient-based optimization. Technical Report SAND2010-1422, Sandia National Laboratories, Albuquerque, NM and Livermore, CA, Mar. 2010.

[5] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *JMLR*, 9:1871–1874, 2008.

[6] P. W. Frey and D. J. Slate. Letter recognition using holland-style adaptive classifiers. *Machine Learning*, 6(2):161–182, 1991.

[7] J. Hull. A database for handwritten text recognition research. *IEEE Trans. on PAMI*, 16(5):550–554, May 1994.

[8] T. Kohonen. *Self-organization and associative memory*. Springer-Verlag, New York, 1989.

[9] L. Ladicky and P. H. S. Torr. Locally linear support vector machines. In *Proc. ICML*, 2011.

[10] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.

[11] S. Maji, A. C. Berg, and J. Malik. Classification using intersection kernel support vector machines is efficient. In *Proc. CVPR*, 2008.

[12] D. Ramanan and S. Baker. Local distance functions: A taxonomy, new algorithms, and an evaluation. *IEEE Trans. on PAMI*, 33:794–806, 2011.

[13] S. Seo and K. Obermayer. Soft learning vector quantization. *Neural Computation*, 2002.

[14] S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal estimated sub-gradient solver for svm. In *Proc. ICML*, 2007.

[15] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from single depth images. In *Proc. CVPR*, 2011.

[16] A. Vedaldi and A. Zisserman. Efficient additive kernels via explicit feature maps. *IEEE Trans. on PAMI*, 34(3), 2011.

[17] K. Weinberger and L. Saul. Distance metric learning for large margin nearest neighbor classification. *JMLR*, 10:207–244, 2009.

[18] Z. Zhang, P. Sturgess, S. Sengupta, N. Crook, and P. H. S. Torr. Efficient discriminative learning of parametric nearest neighbor classifiers. In *Proc. CVPR*, 2012.

[19] X. Zhu, C. Vondrick, D. Ramanan, and C. C. Fowlkes. Do we need more training data or better models for object detection? In *Proc. BMVC*, 2012.