Torr, P, Crook, N, Sengupta, S, Sturgess, P and Zhang, Z (2012)

Efficient discriminative learning of parametric nearest neighbor classifiers.
IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 16-21 June 2012, (). pp. 2232 - 2239 .

10.1109/CVPR.2012.6247932

Efficient Discriminative Learning of Parametric Nearest Neighbor Classifiers

Ziming Zhang   Paul Sturgess   Sunando Sengupta   Nigel Crook   Philip H. S. Torr
Oxford Brookes University
Oxford, UK
http://cms.brookes.ac.uk/research/visiongroup

Abstract

Linear SVMs are efficient in both training and testing, however the data in real applications is rarely linearly separable. Non-linear kernel SVMs are too computationally intensive for applications with large-scale data sets. Recently locally linear classifiers have gained popularity due to their efficiency whilst remaining competitive with kernel methods. The vanilla nearest neighbor algorithm is one of the simplest locally linear classifiers, but it lacks robustness due to the noise often present in real-world data.

In this paper, we introduce a novel local classifier, Parametric Nearest Neighbor (P-NN) and its extension Ensemble of P-NN (EP-NN). We parameterize the nearest neighbor algorithm based on the minimum weighted squared Euclidean distances between the data points and the prototypes, where a prototype is represented by a locally linear combination of some data points. Meanwhile, our method attempts to jointly learn both the prototypes and the classifier parameters discriminatively via max-margin. This makes our classifiers suitable to approximate the classification decision boundaries locally based on nonlinear functions. During testing, the computational complexity of both classifiers is linear in the product of the dimension of data and the number of prototypes. Our classification results on MNIST, USPS, LETTER, and Chars74K are comparable and in some cases are better than many other methods such as the state-of-the-art locally linear classifiers.

1. Introduction

Classification continues to be one of the key challenges in computer vision research. Ideally, we would like to learn a classifier f so that $\forall i, (\mathbf{x}_i, y_i), f : \mathbf{x}_i \rightarrow y_i$, where $\mathbf{x}_i \in \mathbb{R}^d$ is a data point and $y_i \in \mathbb{N}$ is its class label.

Among different classifiers, support vector machines (SVMs) are commonly used because of their good generalization. But the data points from distinct neighboring classes are often not well separated when linear kernels are used directly. With the help of the kernel trick, kernel-based SVMs map the original features into a higher dimensional space implicitly, and then classify the data using linear kernels. However, kernel-based SVMs are too computationally intensive for applications with large-scale data sets. To approximate kernels, some explicit feature mapping methods [14, 20] have been proposed for kernel approximation. However, not all the kernel functions can be approximated explicitly using these methods.

Recently, locally linear classifiers [12, 13, 21, 25, 26, 27, 28, 29] have been attracting more and more attention due to their good performance and efficiency. The basic idea behind these approaches is that the optimal classification decision boundaries can be piece-wise approximated locally using linear functions. Notice that all these classifiers above are variants of stacked generalization [24], where the output from the previous layer is the input for the current layer, and all the layers are trained independently, but only the final layer is trained for the classification purpose.

The nearest neighbor algorithm is one of the simplest local classifiers, but its performance is quite sensitive to the noisy data due to its nonparametric property. In this paper, we introduce a novel max-margin based *Parametric Nearest Neighbor* classifiers (P-NN), and its extension Ensemble of P-NN (EP-NN). Our method extends the nonparametric kernel estimation [2], and jointly learns the prototypes, each of which is represented by a locally linear combination of some data points, and the classifier parameters. The classification decision in our classifiers is made based on the minimum weighted squared Euclidean distances between the data points and the prototypes. The computational complexity of our classifiers during testing is linear in the dimension of data and the total number of prototypes, which makes them applicable for large-scale data.

In summary, the main contribution of this paper is that we propose *a max-margin based classifier with good trade-off between accuracy, computational efficiency, and scalability* by parameterizing the nearest neighbor classifier. It turns out that our classifiers can approximate decision boundaries locally as well.

1.1. Related Work

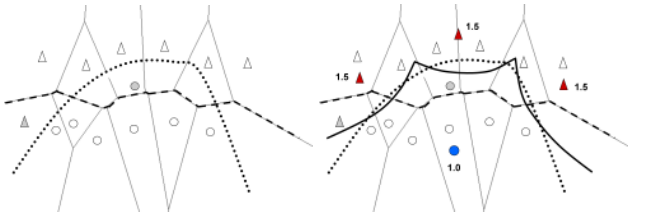
By extending the dimension of data explicitly, the decision boundaries of the kernel-based SVMs in the original feature space can be approximated using linear SVMs. Maji *et al*. [14] proposed a fast piece-wise linearization approach to approximate the intersection kernel with significant savings in terms of computational time and memory. Vedaldi and Zisserman [20] proposed a more general explicit feature mapping method to approximate the homogeneous additive kernels (*e.g*. the intersection kernel and the χ^2 kernel) based on the Fourier sampling theorem.

However, the optimal decision boundary for a classification problem does not necessarily behave as defined by a kernel. In fact, it could be an arbitrary nonlinear function in the feature space. In such cases, it is still possible to approximate an optimal decision boundary locally using linear functions according to Taylor's theorem.

Recently, several locally linear classifiers have been proposed and successfully applied to object recognition. Zhang *et al*. [28] proposed an SVM-KNN classifier, where for each test data point, its K nearest neighbors in the training data are found and used to train multi-class linear SVMs. Similar ideas appeared in [25]. In SVM-KNN, searching for K nearest neighbors for each point can be considered as a coding process, and a few coding methods (*e.g*. LCC [27], improved LCC [26], LLC [21], OCC [29], deep coding network [13]) have been proposed for approximating the nonlinear optimal decision boundaries, which are assumed to be (α, β, p)-Lipschitz smooth functions in order to guarantee theoretical upper bounds of the approximation error. Ladicky and Torr [12] proposed a method for learning locally linear SVMs using encoded data. Notice that among all these approaches, the coding schemes are not designed directly for classification.

Nearest neighbor classifiers [1, 3, 18, 23] have been widely used in computer vision. In [3] Boiman *et al*. proposed a Naive Bayes Nearest Neighbor classifier (NBNN) based on the nonparametric nearest neighbors for image classification, and achieved good results on some benchmark datasets. Behmo *et al*. [1] parameterized NBNN by max-margin methods for both image classification and object detection. Tuytelaars *et al*. [18] built a kernel for SVMs by generating image representations using NBNN. Weinberger and Saul [23] proposed a large margin nearest neighbor classifier (LMNN) which improves the K-nearest neighbor (KNN) classifier by learning the Mahalanobis distance metric from labeled examples. All of these methods suffer from high computational complexity because during testing they need to search for the nearest neighbors for every test point in the entire training data set.

Different from these previous work, our classifiers approximates the optimal decision boundary locally based on nearest neighbors. We summarize the data distribution using prototypes as well as learning the classifier parameters jointly. The functionality of the prototypes in our method is similar to the support vectors in kernel-based SVMs. However, the number of support vectors in kernel-based SVMs could scale linearly with the size of the training data [16], which is highly dependent on the structure of the training data, while the number of the prototypes in our classifiers is pre-defined and fixed during training and testing. In our experiments, we also compare our classifiers with online kernel-based multi-class SVMs with budget learning [22] proposed recently by Wang *et al*., which manages the number of support vectors as well.



(a) Nearest neighbor classifier (b) Parametric nearest neighbor classifier

Figure 1. *Illustration of the differences between (a) the nonparametric nearest neighbor classifier* (i.e. *1-NN) and (b) our parametric nearest neighbor classifier (P-NN), where 2 classes are represented by △ and ◯, the 3 red triangles are the prototypes in class △, the blue circle is the prototype in class ◯, the numbers close to the prototypes are their weights, the dashed curve denotes the decision boundary of 1-NN, the solid hyperbolas in (b) denote the decision boundary of P-NN, and the dotted curve denotes the optimal decision boundary which needs approximation. Clearly, 1-NN makes no attempt to approximate the optimal decision boundary. However, our P-NN learns not only the prototypes in each class but also the classifier parameters* (i.e. *the nonnegative weights of the prototypes and the bias terms for different classes), which approximates the optimal decision boundary locally using hyperbolas based on the weighted squared Euclidean distances. This figure is best viewed in color.*

The rest of the paper is organized as follows. In Section 2 we explain the details of P-NN in terms of formulation, optimization and computational complexity during testing, and then introduce EP-NN in Section 3. In Section 4 we discuss how to implement our classifiers in practice, and show our experimental results and comparison with many other methods in Section 5. Finally we conclude the paper in Section 6.

2. Parametric Nearest Neighbor Classifiers

As illustrated in Fig. 1, in general the decision boundary of a nearest neighbor classifier is locally linear, but it makes no attempt to approximate the optimal decision boundary for classification. On the contrary, our *Parametric Nearest Neighbor* classifier (P-NN) aims to approximate the optimal decision boundary locally by learning both the prototypes for each class and the classifier parameters jointly and discriminatively. In the following sections, we will explain the details of P-NN in terms of formulation, optimization, and computational complexity during testing.

2.1. Formulation

Initially, nearest neighbor classifiers can be considered as nonparametric methods based on Gaussian kernel density estimation. Given a data point $\mathbf{x} \in \mathcal{X} \subset \mathbb{R}^d$ and $|\mathcal{U}_c|$, where $\mathcal{U}_c \subset \mathbb{R}^d$, prototypes for the same class $c \in \mathcal{C}$, where $\mathcal{C}$ denotes the class set, suppose that the window sizes, which are unknown, in the Gaussian kernels for the prototypes in each class are the same, denoted as $h_c \geq 0$, then the probability of the data point $\mathbf{x}$ belonging to a class c can be formulated as follows:

$$p(c|\mathbf{x}) \propto p(\mathbf{x}|c) = \frac{1}{z_c} \sum_{\mathbf{u}_j \in \mathcal{U}_c} \exp\left\{-\frac{\|\mathbf{x}-\mathbf{u}_j\|^2}{2(h_c)^2}\right\} \tag{1}$$

where $\|\cdot\|$ denotes the ℓ_2-norm and $z_c = |\mathcal{U}_c|(2\pi)^{\frac{d}{2}}(h_c)^d$ is a normalization factor of the density function. Following [1], $p(\mathbf{x}|c)$ can be approximated by the largest term in the summation. Then by taking the log-likelihood, Eqn. 1 can be rewritten as:

$$-\log p(c|\mathbf{x}) \approx w_c \left\{\min_{\mathbf{u}_j \in \mathcal{U}_c} \|\mathbf{x}-\mathbf{u}_j\|^2\right\} + b_c \tag{2}$$

where $w_c = \frac{1}{2(h_c)^2} \geq 0$, $b_c = \log z_c + \log p(\mathbf{x}) - \log p(c)$, $p(\mathbf{x})$ and $p(c)$ are the fixed probabilities of data point $\mathbf{x}$ and class c, respectively.

Nonparametric nearest neighbor classifiers assume that given a test data point $\mathbf{x} \in \mathbb{R}^d$, all the w's and b's for all the classes are the same. This leads to the class label prediction rule in nearest neighbor classifiers as follows:

$$c^* = \arg\min_{c \in \mathcal{C}} \|\mathbf{x}-\mathbf{u}_j\|^2, \quad s.t. \quad \mathbf{u}_j \in \mathcal{U}_c. \tag{3}$$

However, as argued in [1], because both the window size h_c and the class prior probability $p(c)$ could vary a lot for different classes, the assumption in the nonparametric nearest neighbor classifiers hardly holds for most cases. On the contrary, our P-NN estimates w's and b's for all the classes as well as learning the prototypes for each class by maximizing the margins.

Given a training data set $(\mathbf{x}_i, y_i)_{i=1,\cdots,|\mathcal{X}|}$ with $|\mathcal{X}|$ data points, where $\forall i, \mathbf{x}_i \in \mathcal{X} \subset \mathbb{R}^d$ is a data point and $y_i \in \mathcal{C} \subset \mathbb{N}$ is its class label, for any class $\forall c \in \mathcal{C}$, P-NN attempts to jointly learn the prototypes $\mathcal{U}_c$ and the class model $(\mathbf{w}_c, \mathbf{b}_c)$, so that the minimum weighted Euclidean distance between each data point $\mathbf{x}_i$ and the prototypes in $\mathcal{U}_{y_i}$ is smaller than the minimum weighted Euclidean distance between $\mathbf{x}_i$ and any prototype in $\mathcal{U}_{\bar{c}} = \bigcup_{c \in \mathcal{C}\setminus\{y_i\}} \mathcal{U}_c$. Therefore, based on the hinge loss, P-NN can be formulated as the following optimization problem:

$$\begin{aligned}
\min_{\mathbf{u},\mathbf{w},\mathbf{b},\boldsymbol{\xi}} \ & \frac{\lambda}{2}\|\mathbf{w}\|^2 + \sum_i \xi_i \\
\text{s.t. } & \forall i, c_i \in \mathcal{C} \setminus \{y_i\}, \\
& \min\nolimits_{c_i} \left\{w_{c_i} \min\nolimits_{\mathbf{u}_k \in \mathcal{U}_{c_i}} \|\mathbf{x}_i - \mathbf{u}_k\|^2 + b_{c_i}\right\} \\
& \quad \geq w_{y_i} \min\nolimits_{\mathbf{u}_j \in \mathcal{U}_{y_i}} \|\mathbf{x}_i - \mathbf{u}_j\|^2 + b_{y_i} + 1 - \xi_i, \\
& \forall i, \ \xi_i \geq 0, \\
& \forall c \in \mathcal{C}, \ w_c \geq 0.
\end{aligned} \tag{4}$$

where $\lambda \geq 0$ is a pre-defined regularization parameter, $\boldsymbol{\xi}$ denotes the set of slack variables, $\mathbf{u}_j \in \mathcal{U}_{y_i}$ (*resp.* $\mathbf{u}_k \in \mathcal{U}_{c_i}$) denotes a prototype in $\mathcal{U}_{y_i}$ (*resp.* $\mathcal{U}_{c_i}$), and w_{y_i} and b_{y_i} (*resp.* w_{c_i} and b_{c_i}) are the class model parameters for class y_i (*resp.* c_i). We denote $(\mathbf{w}, \mathbf{b})$ as the classifier parameters, which are vectors consisting of all w's and b's respectively. Finally, a test data point $\mathbf{x}$ is labeled as:

$$c^* = \arg\min_{c \in \mathcal{C}} \left\{w_c \min_{\mathbf{u}_j \in \mathcal{U}_c} \|\mathbf{x}-\mathbf{u}_j\|^2 + b_c\right\} \tag{5}$$

2.2. Optimization

We adopt an alternating optimization method between learning prototypes and learning classifier parameters to solve the non-convex problem in Eqn. 4.

2.2.1 Learning Prototypes

We update $\mathbf{u}$ and $\boldsymbol{\xi}$ in Eqn. 4 while fixing $\mathbf{w}$ and $\mathbf{b}$ using stochastic gradient descent, similar to the *online-loss-minimization* algorithm in [8]. We say that $\hat{c}_i$ is the *closest class label* to y_i for $\mathbf{x}_i$ if

$$\hat{c}_i = \arg\min_{c_i \in \mathcal{C}\setminus\{y_i\}} \left\{w_{c_i} \min_{\mathbf{u}_k \in \mathcal{U}_{c_i}} \|\mathbf{x}_i - \mathbf{u}_k\|^2 + b_{c_i}\right\}. \tag{6}$$

Letting $g(\mathbf{x}_i, \mathbf{u}; \mathbf{w}, \mathbf{b}) = \xi_i$ be the hinge loss given a data point $\mathbf{x}_i$, and $\hat{c}_i$ be the closest class label to y_i for $\mathbf{x}_i$, then the sub-gradient of g w.r.t. an arbitrary prototype $\mathbf{u}$, denoted as $\frac{\partial g}{\partial \mathbf{u}}$, is: if $\xi_i > 0$, then $\frac{\partial g}{\partial \mathbf{u}_j^*} = 2w_{y_i}\left(\mathbf{u}_j^* - \mathbf{x}_i\right)$ and $\frac{\partial g}{\partial \mathbf{u}_k^*} = 2w_{\hat{c}_i}\left(\mathbf{x}_i - \mathbf{u}_k^*\right)$, where $\mathbf{u}_j^* = \arg\min_{\mathbf{u}_j \in \mathcal{U}_{y_i}} \|\mathbf{x}_i - \mathbf{u}_j\|^2$ and $\mathbf{u}_k^* = \arg\min_{\mathbf{u}_k \in \mathcal{U}_{\hat{c}_i}} \|\mathbf{x}_i - \mathbf{u}_k\|^2$; otherwise, $\frac{\partial g}{\partial \mathbf{u}} = \mathbf{0}$. Then we can use the following equation to update $\mathbf{u}$ given a data point $\mathbf{x}_i$:

$$\forall \mathbf{u} \in \left\{\mathcal{U} = \bigcup_{c \in \mathcal{C}} \mathcal{U}_c\right\}, \quad \mathbf{u}^{(t+1)} = \mathbf{u}^{(t)} - \eta_t \frac{\partial g}{\partial \mathbf{u}^{(t)}} \tag{7}$$

where η_t and $\frac{\partial g}{\partial \mathbf{u}^{(t)}}$ denote the learning rate parameter and the sub-gradient for $\mathbf{u}$ at iteration $t \in \mathbb{N}$, respectively.

Alg. 1 and Alg. 2 show our learning algorithms, where we use some training points as the initial prototypes, because at the beginning we want to guarantee that the data points and the prototypes are definitely in the same class, or not. Other clustering algorithms such as K-Means could be used as well to initialize the prototypes.

```
Algorithm 1: Initialization of the prototypes for each class:
$\mathcal{U} = InitializePrototypes(\mathcal{X}, \mathcal{Y}, \{|\mathcal{U}_c|\})$
Input  : training data $(\mathcal{X}, \mathcal{Y})$, number of prototypes per class
         $\{|\mathcal{U}_c|\}_{c\in\mathcal{C}}$
Output: prototypes $\mathcal{U} = \bigcup_{c\in\mathcal{C}} \mathcal{U}_c$
foreach $c \in \mathcal{C}$ do
    $\mathcal{U}_c \leftarrow \emptyset$;
    repeat
        Randomly select data $(\mathbf{x}, y) \in (\mathcal{X}, \mathcal{Y})$ so that $\mathbf{x} \notin \mathcal{U}_c$
        and $y = c$;
        $\mathcal{U}_c \leftarrow \mathcal{U}_c \bigcup \{\mathbf{x}\}$;
    until $|\mathcal{U}_c|$ data points has been added;
end
return $\mathcal{U} = \bigcup_{c\in\mathcal{C}} \mathcal{U}_c$;
```

```
Algorithm 2: Stochastic gradient descent for learning prototypes:
$\mathcal{U} = LearnPrototypes(\mathcal{X}, \mathcal{Y}, \{\eta_i\}, \mathcal{U}, \mathbf{w}, \mathbf{b})$
Input  : training data $(\mathcal{X}, \mathcal{Y})$, learning rate $\{\eta_i\}$, prototypes $\mathcal{U}$,
         classifier parameters $(\mathbf{w}, \mathbf{b})$
Output: prototypes $\mathcal{U} = \bigcup_{c\in\mathcal{C}} \mathcal{U}_c$
foreach $(\mathbf{x}_i, y_i) \in (\mathcal{X}, \mathcal{Y})$ do
    if $\min_{c_i\in\mathcal{C}\setminus\{y_i\}} \left\{ w_{c_i} \min_{\mathbf{u}_k\in\mathcal{U}_{c_i}} \|\mathbf{x}_i - \mathbf{u}_k\|^2 + b_{c_i} \right\} <$
    $w_{y_i} \min_{\mathbf{u}_j\in\mathcal{U}_{y_i}} \|\mathbf{x}_i - \mathbf{u}_j\|^2 + b_{y_i} + 1$ then
        $\mathbf{u}_j^* = \arg\min_{\mathbf{u}_j\in\mathcal{U}_{y_i}} \|\mathbf{x}_i - \mathbf{u}_j\|^2$;
        $\mathbf{u}_k^* = \arg\min_{\mathbf{u}_k\in\mathcal{U}_{\hat{c}_i}} \|\mathbf{x}_i - \mathbf{u}_k\|^2$;
        $\mathbf{u}_j^* \leftarrow \mathbf{u}_j^* + \eta_i w_{y_i}(\mathbf{x}_i - \mathbf{u}_j^*)$;
        $\mathbf{u}_k^* \leftarrow \mathbf{u}_k^* - \eta_i w_{\hat{c}_i}(\mathbf{x}_i - \mathbf{u}_k^*)$;
    end
end
return $\mathcal{U} = \bigcup_{c\in\mathcal{C}} \mathcal{U}_c$;
```

2.2.2 Learning Classifier Parameters

We update $\mathbf{w}$, $\mathbf{b}$ and $\boldsymbol{\xi}$ in Eqn. 4 while fixing $\mathbf{u}$. Then given data $(\mathbf{x}_i, y_i)$, letting $\mathbf{v}_i$ be a $|\mathcal{C}|$-dimensional vector consisting of 0's, where $|\mathcal{C}|$ is the number of classes, and $\hat{c}_i$ be the closest class label to y_i for $\mathbf{x}_i$, we set $\mathbf{v}_i(\hat{c}_i) = \min_{\mathbf{u}_k\in\mathcal{U}_{\hat{c}_i}} \|\mathbf{x}_i - \mathbf{u}_k\|^2$ and $\mathbf{v}_i(y_i) = -\min_{\mathbf{u}_j\in\mathcal{U}_{y_i}} \|\mathbf{x}_i - \mathbf{u}_j\|^2$, where $\mathbf{v}_i(\cdot)$ denotes the value at a particular bin of vector $\mathbf{v}_i$. Therefore, Eqn. 4 can be rewritten as follows:

$$\begin{aligned} \min_{\mathbf{w},\mathbf{b},\boldsymbol{\xi}} \quad & \frac{\lambda}{2}\|\mathbf{w}\|^2 + \sum_i \xi_i \qquad (8) \\ \text{s.t.} \quad & \forall i,\ \mathbf{w}^T\mathbf{v}_i + b_{\hat{c}_i} - b_{y_i} \geq 1 - \xi_i, \\ & \forall i,\ \xi_i \geq 0, \\ & \forall c \in \mathcal{C},\ w_c \geq 0. \end{aligned}$$

where $(\cdot)^T$ denotes the matrix transpose operator. Notice that both $\hat{c}_i$ and $\mathbf{v}_i$ are dependent on the classifier parameters $(\mathbf{w}, \mathbf{b})$. So if the classifier parameters are updated, $\hat{c}_i$ and $\mathbf{v}_i$ should be updated as well. Thus, we present an iterative optimization algorithm to solve Eqn. 8 as shown in Alg. 3, where $\boldsymbol{\Omega}$ denotes a set of triplets.

Finally, based on Alg. 1-3, we can jointly learn the prototypes and the classifier parameters by maximizing the margin in an alternating manner, as presented in Alg. 4, where FLT_MAX denotes the max value that we can set to w's so that Eqn. 4 can be optimized from its biggest value.

```
Algorithm 3: Iterative optimization for solving Eqn. 8:
$(\mathbf{w}, \mathbf{b}) = LearnClassifiers(\mathcal{X}, \mathcal{Y}, \mathcal{U}, \mathbf{w}, \mathbf{b})$
Input  : training data $(\mathcal{X}, \mathcal{Y})$, prototypes $\mathcal{U}$, classifier parameters
         $(\mathbf{w}, \mathbf{b})$
Output: classifier parameters $(\mathbf{w}, \mathbf{b})$
$\boldsymbol{\Omega} \leftarrow \emptyset$;
repeat
    foreach $(\mathbf{x}_i, y_i) \in (\mathcal{X}, \mathcal{Y})$ do
        Calculate $\hat{c}_i$ using Eqn. 6 and $\mathbf{v}_i \in \mathbb{R}^{|\mathcal{C}|}$;
        $\boldsymbol{\Omega} \leftarrow \boldsymbol{\Omega} \bigcup \{(\mathbf{v}_i, y_i, \hat{c}_i)\}$;
    end
    Update $\mathbf{w}, \mathbf{b}$ based on $\boldsymbol{\Omega}$ using Eqn. 8;
until Classifier parameters converged;
return $\mathbf{w}, \mathbf{b}$;
```

```
Algorithm 4: Alternating optimization for solving Eqn. 4
Input  : training data $(\mathcal{X}, \mathcal{Y})$, learning rate $\{\eta_i\}$, number of
         prototypes per class $\{|\mathcal{U}_c|\}_{c\in\mathcal{C}}$
Output: prototypes $\mathcal{U}$, classifier parameters $(\mathbf{w}, \mathbf{b})$
foreach $c \in \mathcal{C}$ do
    $w_c \leftarrow$ FLT_MAX, $b_c \leftarrow 0$;
end
$\mathcal{U} = InitializePrototypes(\mathcal{X}, \mathcal{Y}, \{|\mathcal{U}_c|\})$;
repeat
    $\mathcal{U} = LearnPrototypes(\mathcal{X}, \mathcal{Y}, \{\eta_i\}, \mathcal{U}, \mathbf{w}, \mathbf{b})$;
    $(\mathbf{w}, \mathbf{b}) = LearnClassifiers(\mathcal{X}, \mathcal{Y}, \mathcal{U}, \mathbf{w}, \mathbf{b})$;
until Converged;
return $\mathcal{U}, \mathbf{w}, \mathbf{b}$;
```

2.3. Computational Complexity During Testing

Assuming that the computational complexities of the unit operations $+$, $-$, $*$, $\leq$, and $\geq$ are the same, denoted as $O(1)$, then in general the computational complexities of the $\min$ operator is $O(n)$[1], where n is the dimension of data. By counting how many unit operations involved for classifying a test data point, we will know the computational complexity of P-NN.

Suppose we have a test data point $\mathbf{x} \in \mathbb{R}^d$, $|\mathcal{U}_c|$ prototypes per class, with $|\mathcal{C}|$ classes. Since the distance between $\mathbf{x}$ and a prototype $\mathbf{u}$ is $\|\mathbf{x} - \mathbf{u}\|^2 = \|\mathbf{x}\|^2 - 2\mathbf{x}^T\mathbf{u} + \|\mathbf{u}\|^2$, where $\|\mathbf{x}\|^2$, $\|\mathbf{u}\|^2$ and $2\mathbf{u}$ can be pre-calculated, the computational complexity of calculating distances is $(2d + 2) \cdot O(1)$. Then the computational complexity of P-NN during testing can be described as:

[1] In practice, the complexity of the $\min$ operator depends on the data structure. At most, it is $O(n)$.

$$\left\{\sum_{c\in\mathcal{C}}|\mathcal{U}_c|(2d+2)+\sum_{c\in\mathcal{C}}|\mathcal{U}_c|+2|\mathcal{C}|+|\mathcal{C}|\right\}\cdot O(1) \tag{9}$$
$$\leq \left\{\sum_{c\in\mathcal{C}}|\mathcal{U}_c|(2d+6)\right\}\cdot O(1)=O(dN)$$

where $N = \sum_{c\in\mathcal{C}}|\mathcal{U}_c|$ is the total number of prototypes. Thus, the computational complexity of P-NN during testing is linear in the product of the dimension of data and the total number of prototypes, which is the same as some locally linear methods such as LLC [21] and LL-SVM [12].

3. Ensemble of P-NN Classifiers

P-NN assumes that the window sizes in the Gaussian kernel density estimation are the same for all the prototypes in the same class, while varying for different classes. However, this assumption is quite strong, because even for the prototypes in the same class, the window sizes may vary individually.

In order to relax this assumption, we take advantage of the random initialization of the prototypes in P-NN due to the non-convexity of Eqn. 4, similar to random forest [6]. In this way, we further introduce an *Ensemble of P-NN* classifier (EP-NN) to boost the classification accuracy. We call the set of learned prototypes in each P-NN a *base learner*. Rather than learning one base learner with many prototypes for each class, which risks overfitting the training data, EP-NN jointly learns multiple base learners with reasonable amount of prototypes per class.

Given a training data set $(\mathbf{x}_i, y_i)_{i=1,\cdots,|\mathcal{X}|}$, EP-NN is formulated as below to jointly learn $|\mathcal{L}|$ base learners and the classifier parameters, where $l \in \mathcal{L}$ denotes the l^{th} base learner in $\mathcal{L}$:

$$\min_{\mathbf{u},\mathbf{w},\mathbf{b},\boldsymbol{\xi}} \ \frac{\lambda}{2}\sum_{c\in\mathcal{C}}\|\mathbf{w}_c\|^2+\sum_i \xi_i \tag{10}$$
$$\begin{aligned}\text{s.t. }& \forall i, c_i \in \mathcal{C}\setminus\{y_i\},\\ & \min\nolimits_{c_i}\left\{\sum\nolimits_{l\in\mathcal{L}} w^l_{c_i}\min\nolimits_{\mathbf{u}_k\in\mathcal{U}^l_{c_i}}\|\mathbf{x}_i-\mathbf{u}_k\|^2+b_{c_i}\right\}\\ & \geq \sum\nolimits_{l\in\mathcal{L}} w^l_{y_i}\min\nolimits_{\mathbf{u}_j\in\mathcal{U}^l_{y_i}}\|\mathbf{x}_i-\mathbf{u}_j\|^2+b_{y_i}+1-\xi_i,\\ & \forall i,\ \xi_i\geq 0,\\ & \forall c\in\mathcal{C}, \forall l\in\mathcal{L},\ w^l_c\geq 0.\end{aligned}$$

We can easily modify Alg. 1-4 to solve Eqn. 10 by considering all the base learners together for each update. In the same way, we can easily extend EP-NN by taking multi-source information into account. Notice EP-NN shares the same computational complexity during testing as P-NN.

4. Implementation

In order to compare other locally linear methods easily, especially the coding based locally linear methods, as well as making a fast implementation of our classifiers, in practice we followed the stacked generalization framework and implemented our classifiers approximately in a 2-stage way: first encoding data and then training a multi-class linear SVM. Empirically the classification accuracy of this implementation is very close to that based on Alg. 4, with much faster training speed and less care of parameter tuning.

(I) Encoding data. We learn each base learner independently so that this process can be parallelized. After the first update of the prototypes in Alg. 4, we stop updating prototypes, because empirically we find that these prototypes are good enough for classification.

To encode data, we map each data point into a distance based sparse vector. Given a data point $\mathbf{x} \in \mathcal{X}$ and $|\mathcal{L}|$ base learners, letting $\forall l \in \mathcal{L}, \mathbf{v}^l_i \in \mathbb{R}^{|\mathcal{C}|}$ be a vector, where $|\mathcal{C}|$ is the number of classes, we set the c^{th} bin in $\mathbf{v}^l_i$ as $\mathbf{v}^l_i(c) = \min_{\mathbf{u}_j\in\mathcal{U}^l_c}\|\mathbf{x}_i-\mathbf{u}_j\|^2$, where $c = \arg\min_{c\in\mathcal{C}}\left\{\min_{\mathbf{u}_j\in\mathcal{U}^l_c}\|\mathbf{x}_i-\mathbf{u}_j\|^2\right\}$, and 0's to other bins. Further, we denote $\mathbf{v}_i$ as our encoded feature vector by concatenating all $\mathbf{v}^l_i$'s and normalizing it using ℓ_1-norm. Notice that our distance based feature vectors are $|\mathcal{C}| \times |\mathcal{L}|$ dimensional, but in each vector only $|\mathcal{L}|$ bins are non-zeros.

(II) Training a standard multi-class linear SVM. By taking the encoded data as the input, we can train the following standard multi-class linear SVM [9] for classification:

$$\min_{\mathbf{w},\mathbf{b},\boldsymbol{\xi}} \ \frac{\lambda}{2}\sum_{c}\|\mathbf{w}_c\|^2+\sum_{i,c_i}\xi_{i,c_i} \tag{11}$$
$$\begin{aligned}\text{s.t. }& \forall i, \forall c_i\in\mathcal{C}\setminus\{y_i\},\\ & \left[\mathbf{w}^T_{c_i}\mathbf{v}_i+b_{c_i}\right]-\left[\mathbf{w}^T_{y_i}\mathbf{v}_i+b_{y_i}\right]\geq 1-\xi_{i,c_i},\\ & \xi_{i,c_i}\geq 0.\end{aligned}$$

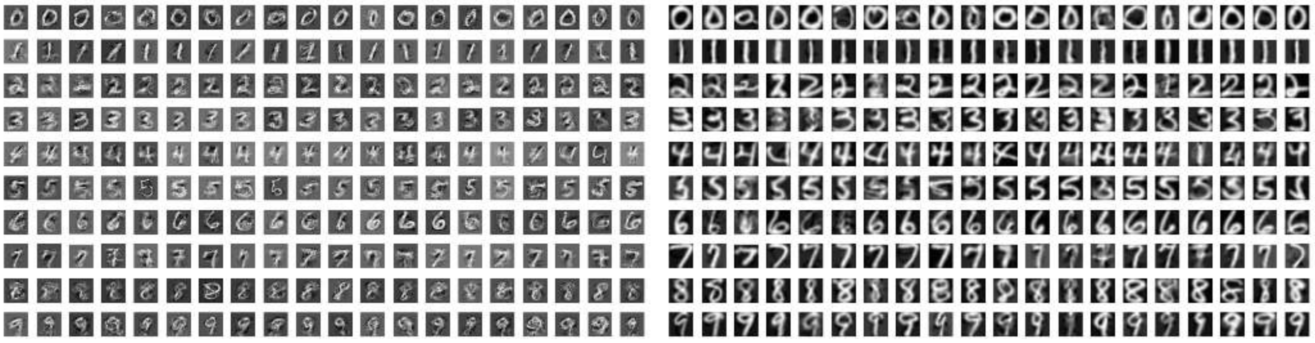
Here we relax Eqn. 8 by (1) removing the nonnegative constraints on $\mathbf{w}$, and (2) allowing that the weights of the prototypes in the same class can be changed for different classification cases, rather than fixed values.

5. Experiments

In our experiments, we test P-NN and EP-NN[2] on four datasets: MNIST, USPS, LETTER, and Chars74K [10].

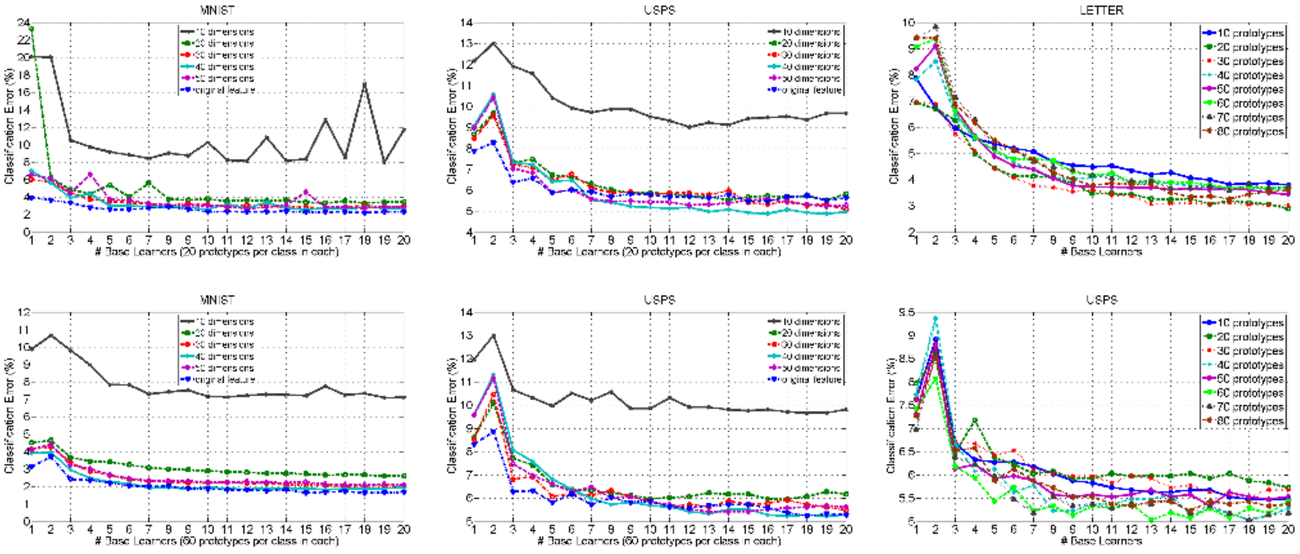
MNIST contains 40000 training and 10000 test gray-scale images with resolution 28×28 pixels, which are vectorized directly into 784 dimensional vectors. The label of each image is one of the 10 digits from 0 to 9. USPS contains 7291 training and 2007 test gray-scale images with resolution 16×16 pixels, directly stored as 256 dimensional vectors, and the label of each image still corresponds to one of the 10 digits from 0 to 9. LETTER contains 16000 training and 4000 testing images, each of which is represented

[2] Our implement can be downloaded from https://sites.google.com/a/brookes.ac.uk/zimingzhang/code



(a) MNIST (b) USPS

Figure 2. *Some examples of the jointly learned prototypes by our classifiers on (a) MNIST and (b) USPS, 20 prototypes per class.*



(a) (b) (c)

Figure 3. *Performance of EP-NN: classification error* vs. *the number of base learners on (a) MNIST and (b) USPS with different dimensions of data using 20 (top) and 60 (bottom) prototypes per class; (c) LETTER (top) and USPS (bottom) with different numbers of the prototypes per class in each base learner from 10 to 80, step by 10, using original features. When the number of base learners is equal to 1, EP-NN turns into P-NN. Clearly, EP-NN boosts the performance of P-NN significantly. This figure is best viewed in color.*

as a 16 dimensional vector, and the label of each image corresponds to one of the 26 letters from A to Z. The features we used are the raw features. Chars74K comprises 62 classes (0-9, A-Z, a-z), 7705 characters obtained from natural images, 3410 hand drawn characters using a tablet PC, and 62992 synthesized characters from computer fonts, with over 74K images in total. We first resized each image into a gray image with 8 × 8 pixels, then randomly split it into two independent sets, 7400 images as test data and the rest as training data, and finally vectorize each image directly into a 64 dimensional vector as our feature.

To learn the prototypes in each base learner, we randomly select at most 10^5 data points from the training set, where each data point is allowed to be selected repeatedly, and fix the learning rate to 0.1. LIBLINEAR [11] is employed as our multi-class SVM solver.

We first visualize some of the learned prototypes for MNIST and USPS in Fig. 2, respectively. Each prototype is represented as a linear combination of different training data points and plays a role of a weak classifier. We can roughly see the digit represented by each prototype, which demonstrates the good discriminability of the learned prototypes.

Then we test the robustness of our classifiers w.r.t. dimensions of features, numbers of prototypes per class in each base learner, and numbers of base learners. To build low-dimensional features, we directly apply singular value decomposition (SVD) to the original data in MNIST and USPS and take the top-K values in the coefficient vector of each data point. Notice that when the number of base learners is equal to 1, EP-NN actually turns into P-NN. Fig. 3 summarizes the comparison results among the three factors:

Table 1. *Classification error rate comparison (%) between our methods and others on MNIST, USPS, LETTER, and Chars74K. All kernel methods use the RBF kernel. Most results are cited from [29]. With much lower-dimensional and sparser inputs for training linear SVMs, the results of both P-NN and EP-NN are comparable to the best among these different methods.*

Methods		MNIST	USPS	LETTER	Chars74K[§]
Ours (40 prototypes per class)	**Parametric Nearest Neighbors (P-NN)**	3.13	7.87	6.95	29.46
	Ensemble of P-NN (EP-NN) (20 base learners)	1.65	4.88	2.90	19.53
Nearest Neighbors	Nearest Neighbor (1-NN)	3.09	5.08	4.35	18.69
	K Nearest Neighbors (KNN)	2.92	4.88	4.35	18.69
	LMNN [23]	1.70	***0.91***	3.60	20.18
Linear SVMs	Linear SVM (10 passes) [4]	12.00	9.57	41.77	72.68
	LIBLINEAR [11]	8.18	8.32	30.60	54.61
Kernel SVMs	LIBSVM [7]	***1.36***	4.58	2.12	16.86
	MCSVM [9]	1.44	4.24	2.42	-
	$\text{SVM}_{\text{struct}}$ [17]	1.40	4.38	***2.40***	-
	LA-RANK (1 pass) [5]	1.41	4.25	2.80	-
Locally linear classifiers	Linear SVM + LCC (4096 anchor points) [27]	1.90	-	-	-
	Linear SVM + improved LCC (4096 anchor points) [26]	1.64	-	-	-
	Linear SVM + LLC (4096 anchor points) [21]	2.28	4.38	4.12	20.88
	Linear SVM + DCN ($L_1 = 64, L_2 = 512$) [13]	1.51	-	-	-
	LL-SVM (100 anchor points, 10 passes) [12]	1.85	5.78	5.32	-
	LIB-LLSVM + OCC [29]	1.61	3.94	6.85	18.72
	ALH [25][†]	2.15	4.19	2.95	***16.26***
Others	BP^M+MRG [22]	-	6.10	10.50	-
	Linear SVM + EFM (Intersection kernel) [19, 20][‡]	9.11	8.12	8.22	29.08

[†] The results shown here are the best, respectively, by running ALH software downloaded from http://www.people.vcu.edu/~vkecman/.

[‡] The input dimensions for training linear SVMs shown in the brackets are the ones which return the best results, respectively, by running VLFEAT [19].

[§] The results on this dataset are generated by the public codes of those methods.

(I) P-NN: From Fig. 3(a) and (b), P-NN seems a little sensitive to very low-dimensional data (*e.g.* 10 or 20). However, when the feature dimension is higher, P-NN behaves stably within 2% difference, and performs best using the original features. From Fig. 3(c), we can see clearly that more prototypes per class does not guarantee a better performance using P-NN, as we expected, but its performance is still reasonably stable within 3% difference.

(II) EP-NN: From the figures in Fig. 3, we can see that EP-NN really boosts the classification accuracy of P-NN significantly. With only 2 base learners, EP-NN performs worse than P-NN, because sometimes each prototype may disagree with each other, leading to weak discrimination between classes. However as we increase the number of base learners, the majority will tend to agree giving better discrimination, as demonstrated by our empirical results. Also, the same phenomenon has been observed in [15]. Similar to P-NN, based on the same number of base learners, reasonably higher dimensional data leads to better results but more prototypes have no guarantee on better results.

Next, we list our comparison with many other methods in Table 1, where both P-NN and EP-NN work comparably and in some cases better than the others. It is worth mentioning that the inputs for training multi-class linear SVMs in our methods are usually much lower-dimensional and sparser than the others due to the nearest neighbor search. For instance, on MNIST EP-NN achieves *1.65%* classification error rate using 150 dimensional input vectors for SVMs, each of which contains only *15* non-zero elements, one non-zero element per base learner, while OCC achieves 1.61% using $784 \times 90 = 70560$ dimensional inputs, and DCN achieves 1.51% using $64 \times 512 = 32768$ dimensional inputs.

Finally, we show our computational time during testing in Table 2, with comparison to LL-SVM, LIBSVM and the nearest neighbor classifier (*i.e.* 1-NN) based on kd-tree. All the algorithms were run on a single thread of a 2.67 GHz CPU. Table 2 verifies our discussion in Section 2.3 that the computational complexity of our methods is linear in the product of the dimension of the data and the total number of the prototypes.

6. Conclusion

In this paper, we propose a novel parametric local classifier called Parametric Nearest Neighbor (P-NN), and its extension Ensemble of P-NN (EP-NN). Our classifiers extend

Table 2. *Computational time comparison of different classifiers, in seconds, on MNIST, USPS, LETTER, and Chars74K.*

Methods (/s)	MNIST (784-dim)	USPS (256-dim)	LETTER (16-dim)	Chars74K (64-dim)
EP-NN (totally 100 prototypes)	$\boldsymbol{3.36\times10^{-4}}$	$\boldsymbol{1.74\times10^{-4}}$	1.24×10^{-4}	1.24×10^{-4}
1-NN (kd-tree)	6.90×10^{-2}	3.20×10^{-3}	$\boldsymbol{4.81\times10^{-5}}$	1.24×10^{-4}
LL-SVMs (100 anchor points)	4.70×10^{-4}	-	-	-
LIBSVM (RBF kernel)	4.60×10^{-2}	-	-	-

the analysis of the Gaussian kernel density estimation, and attempt to learn the prototypes for nearest neighbor search and the classifier parameters jointly and discriminatively. The decision boundary of our classifiers consists of a set of nonlinear functions, since we use the minimum weighted squared Euclidean distance between the data and the prototypes as the classification criterion. The computational complexity of our classifiers during testing is linear in the product of the dimension of the data and the pre-defined total number of the prototypes, which makes our classifiers suitable for large-scale data. We implement P-NN and EP-NN by following the stacked generalization framework, where each data point is mapped into a very sparse vector based on the minimum distance across the classes in each base learner, and as the inputs a multi-class linear SVM is trained for classification. The experimental results on MNIST, USPS, LETTER, and Chars74K datasets demonstrate that our classifiers are robust to the dimension of data, the number of the prototypes per class in each base learner, and the number of base learners as well, and achieve comparable or even better classification accuracy than many other methods. Overall, our classifiers can achieve good trade-off between accuracy, computational efficiency, and scalability.

References

[1] R. Behmo, P. Marcombes, A. Dalalyan, and V. Prinet. Towards optimal naive bayes nearest neighbor. In *ECCV'10*, pages 171–184, Berlin, Heidelberg, 2010. Springer-Verlag. 2, 3

[2] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. 1

[3] O. Boiman, E. Shechtman, and M. Irani. In defense of nearest-neighbor based image classification. In *CVPR'08*, pages 1–8, 2008. 2

[4] A. Bordes, L. Bottou, and P. Gallinari. SGD-QN: Careful quasi-Newton stochastic gradient descent. *JMLR*, 10:1737–1754, 2009. 7

[5] A. Bordes, L. Bottou, P. Gallinari, and J. Weston. Solving multiclass support vector machines with larank. In *ICML '07*, pages 89–96, New York, NY, USA, 2007. ACM. 7

[6] L. Breiman. Random forests. *Machine Learning*, 45:5–32, October 2001. 5

[7] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`. 7

[8] K. Crammer, R. Gilad-Bachrach, A. Navot, and N. Tishby. Margin analysis of the lvq algorithm. In *NIPS'02*, pages 462–469, 2002. 3

[9] K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *JMLR*, 2:265–292, March 2002. 5, 7

[10] T. E. de Campos, B. R. Babu, and M. Varma. Character recognition in natural images. In *Proceedings of the International Conference on Computer Vision Theory and Applications, Lisbon, Portugal*, February 2009. 5

[11] 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, 7

[12] L. Ladicky and P. H. Torr. Locally linear support vector machines. In *ICML'11*, 2011. 1, 2, 5, 7

[13] Y. Lin, T. Zhang, S. Zhu, and K. Yu. Deep coding networks. In *NIPS '10*, Cambridge, MA, 2010. MIT Press. 1, 2, 7

[14] S. Maji, A. Berg, and J. Malik. Classification using intersection kernel support vector machines is efficient. In *CVPR'08*, pages 1–8, 2008. 1, 2

[15] R. E. Schapire, Y. Freund, P. Bartlett, and W. S. Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26(5):1651–1686, 1998. 7

[16] I. Steinwart. Sparseness of support vector machines—some asymptotically sharp bounds. In S. Thrun, L. Saul, and B. Schölkopf, editors, *NIPS'04*. MIT Press, Cambridge, MA, 2004. 2

[17] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *JMLR*, 6:1453–1484, December 2005. 7

[18] T. Tuytelaars, M. Fritz, K. Saenko, and T. Darrell. The NBNN kernel. In *ICCV'11*, 2011. 2

[19] A. Vedaldi and B. Fulkerson. VLFeat: An open and portable library of computer vision algorithms. `http://www.vlfeat.org/`, 2008. 7

[20] A. Vedaldi and A. Zisserman. Efficient additive kernels via explicit feature maps. *PAMI*, 2011. 1, 2, 7

[21] J. Wang, J. Yang, K. Yu, F. Lv, T. Huang, and Y. Gong. Locality-constrained linear coding for image classification. In *CVPR'10*, pages 3360–3367, 2010. 1, 2, 5, 7

[22] Z. Wang, K. Crammer, and S. Vucetic. Multi-class pegasos on a budget. In J. Frnkranz and T. Joachims, editors, *ICML*, pages 1143–1150. Omnipress, 2010. 2, 7

[23] K. Weinberger and L. Saul. Distance metric learning for large margin nearest neighbor classification. *JMLR*, 10:207–244, 2009. 2, 7

[24] D. H. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992. 1

[25] T. Yang and V. Kecman. Adaptive local hyperplane classification. *Neurocomputing*, 2008. 1, 2, 7

[26] K. Yu and T. Zhang. Improved local coordinate coding using local tangents. In *ICML'10*, 2010. 1, 2, 7

[27] K. Yu, T. Zhang, and Y. Gong. Nonlinear learning using local coordinate coding. In *NIPS'09*, 2009. 1, 2, 7

[28] H. Zhang, A. Berg, M. Maire, and J. Malik. Svm-knn: Discriminative nearest neighbor classification for visual category recognition. In *CVPR'06*, pages II: 2126–2136, 2006. 1, 2

[29] Z. Zhang, L. Ladicky, P. H. Torr, and A. Saffari. Learning anchor planes for classification. In *NIPS'11*, 2011. 1, 2, 7