Neural Networks 117 (2019) 274–284

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Locally linear SVMs based on boundary anchor points encoding

Baile Xu [a,1], Shaofeng Shen [a,1], Furao Shen [a,*], Jian Zhao [b,*]

[a] State Key Laboratory for Novel Software Technology, Department of Computer Science and Technology, Collaborative Innovation Center of Novel Software Technology and Industrialization, Nanjing University, China
[b] School of Electronic Science and Engineering, Nanjing University, China

ARTICLE INFO

Article history:
Received 17 November 2017
Received in revised form 18 January 2019
Accepted 24 May 2019
Available online 30 May 2019

Keywords:
Locally linear classifier
Kernel SVM
Boundary points
Large-scale data

ABSTRACT

In this paper, we propose a locally linear classifier based on boundary anchor points encoding (LLBAP) to achieve the efficiency of linear SVM and the power of kernel SVM. LLBAP partitions linearly non-separable data into approximately linearly separable parts based on boundary point scanning and local coding. Each part of data is solved by a linear SVM. Experiments on large-scale benchmark datasets demonstrate that the proposed method is more efficient than kernel SVM in both training and testing phases; its efficiency and classification accuracy also outperform other locally linear classifiers on those benchmark datasets.

1. Introduction

The support vector machine (SVM) (Cortes, 1995) is a popular classifier widely used in various machine learning applications. The original formulation of SVM was introduced as a linearly binary classifier (Boser, Guyon, & Vapnik, 2008). By introducing the kernel trick, linear SVM was developed into kernel SVM (Smola, 2008) to process complicated linearly non-separable data. However, the testing costs of kernel SVM are proportional to the number of support vectors, and therefore its application on large-scale datasets is limited. In order to process large-scale linearly non-separable data, it is necessary to develop some classifiers with both the efficiency of linear SVM and the power of kernel SVM.

To address this issue, many locally linear classifiers based on SVMs (Gu & Han, 2013; Kecman & Brooks, 2010; Ladicky & Torr, 2011; Mao, Fu, Wu, & Hu, 2015; Zhang, Berg, Maire, & Malik, 2006) have been proposed in recent years. These methods assume that a sufficiently small part of linearly non-separable data is approximately linearly separable and solve each small part with a linear SVM. The computation costs of locally linear classifiers are proportional to the number of linear SVMs, which is usually much smaller than that of support vectors in kernel SVM. Therefore, locally linear classifiers are more efficient than kernel SVMs.

SVM-KNN (Zhang et al., 2006) employs lazy learning. For each testing data point, SVM-KNN trains a linear SVM with its neighboring data points in training set and then uses it for prediction. A locally linear SVM model similar to SVM-KNN is proposed in Kecman and Brooks (2010), which adopts a weighted Euclidean distance metric to choose the k nearest neighbors. The clustered SVM (CSVM) (Gu & Han, 2013) uses k-means (Macqueen, 1967) to partition the training data into k disjoined parts and trains a linear SVM in each part. The Multiclass Latent Locally Linear SVM (Fornoni, Caputo, & Orabona, 2013) is proposed for multiclass classification based on Latent SVMs.

The Locally Linear SVM (LLSVM) (Ladicky & Torr, 2011) is proposed based on the theory of local coding (Yu, Zhang, & Gong, 2009) and achieved state-of-the-art performances. It first uses k-means to learn a set of anchor points, then approximates the nonlinear classifier by a set of linear classifiers each associated with an anchor point. Local coding can be viewed as a kind of soft partition over the training data in this learning framework. Since local coding gives significant initial information for later classifier training, the performances of LLSVM heavily depend on the quality of local coding, which further depend on the anchor points being used. Based on the same local coding framework, Locally Linear Classifiers with Supervised Anchor Point Learning (LLCSAPL) (Mao et al., 2015) improves the quality of anchor points by jointly training the anchor points and linear classifiers with stochastic gradient descent (SGD).

Both LLSVM and LLCSAPL use k-means to learn the initial coordinates of anchor points, which may lead to unsatisfying results. Specifically speaking, the initial local coordinates may be unsatisfying in two aspects: First, the number of anchor points is decided by some predefined parameters of the unsupervised

* Corresponding authors.
E-mail addresses: blxu@smail.nju.edu.cn (B. Xu), shaofeng_shen@163.com (S. Shen), frshen@nju.edu.cn (F. Shen), jianzhao@nju.edu.cn (J. Zhao).

[1] These authors contributed equally to this work and should be considered co-first authors.

https://doi.org/10.1016/j.neunet.2019.05.023
0893-6080/

learning process, like the number of clusters k in k-means clustering. Setting these parameters is usually important but difficult. Second, labels of training data points that contain useful information for classification are ignored. As a result, many anchor points distribute near the centers of classes, whose contribution to the global decision boundary is trivial. Hence a lot of computational resources are wasted on training local SVMs corresponding to these points. Compared to the anchor point learning method proposed in this paper, the convergence speed of k-means is much slower, especially on large scale datasets. The theoretical bounds of the number of iterations of k-means have been studied in Har-Peled and Sadri (2005) and Arthur and Vassilvitskii (2006).

As Fig. 1 shows, a satisfying anchor point learning method should be able to learn a concise set of anchor points that locate near the ideal decision boundary, namely boundary points. The boundary points are able to partition the underlying nonlinear decision boundary directly, thus the locally linear decision boundary can be constituted in a simple and efficient way.

In this paper, we propose a novel **L**ocally **L**inear SVMs model based on **B**oundary **A**nchor **P**oints encoding (**LLBAP**). LLBAP partitions the training data into overlapping parts by local coding, like LLSVM and LLCSAPL. The main innovations of our work can be summarized as follows:

First, we introduce a new anchor point learning method that automatically decides the number of anchor points and detects the noisy data in the training data. The proposed method is more efficient than k-means clustering because it only needs to iterate several times over the training data.

Second, a boundary point scanning method is proposed to find the anchor points near the centers of desired linear separable parts. High quality data partition is realized by local coding based on boundary points. Boundary point scanning also reduces the number of linear classifiers, and therefore the training and testing speed of LLBAP is further accelerated.

Finally, we propose a novel formulation for locally linear classifiers. Both boundary points and linear SVMs can be optimized efficiently by SGD. LLBAP can fine-tune the positions of boundary points while training linear SVMs, and hence the classification accuracy of LLBAP is improved.

Experiments on large-scale benchmark datasets prove that LLBAP is training and testing efficient compared with kernel SVM. Its efficiency and classification accuracy both outperform other state-of-the-art locally linear classifiers on those datasets.

The remainder of this paper is organized as follows. Section 2 introduces locally linear SVMs based on local coding. Section 3 describes the proposed method. In Section 5, we provide the comparison results between LLBAP and other related methods on real-world datasets.

2. Locally linear SVMs based on local coding

In this section, we briefly describe locally linear SVM models based on local coding, which forms the basis of our work. Given a binary classification dataset $\{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$ with $\boldsymbol{x}_n \in R^D$ and $y_n \in \{-1, 1\}$, for each $(\boldsymbol{x}, y) \in \{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$, a standard linear SVM binary classifier takes the form of:

$$f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b \tag{1}$$

The optimal weight vector $\boldsymbol{w}$ and bias b are obtained by minimizing the following loss function:

$$L(\boldsymbol{w}, b) = \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \frac{1}{N}\sum_{n=1}^{N}\ell(y_n, f(\boldsymbol{x}_n)) \tag{2}$$

where

$$\ell(y, f(\boldsymbol{x})) = \max(0, 1 - yf(\boldsymbol{x})). \tag{3}$$

In nonlinear classification problems, the desired decision boundary can be encoded by varying the weight vector $\boldsymbol{w}$ and bias b of the SVM classifier according to the position of the point $\boldsymbol{x}$ in the feature space as:

$$H(\boldsymbol{x}) = \boldsymbol{w}(\boldsymbol{x})^T\boldsymbol{x} + b(\boldsymbol{x}), \tag{4}$$

where $\boldsymbol{w}(\boldsymbol{x})$ and $b(\boldsymbol{x})$ are functions of the point $\boldsymbol{x}$.

Locally linear SVMs aim to approximate the nonlinear decision boundary by combining locally linear decision boundaries, namely, approximating $\boldsymbol{w}(\boldsymbol{x})$ and $b(\boldsymbol{x})$ with a combination of linear functions. In LLSVM, LLCSAPL and LLBAP, the combination is realized by local coding. The local coding scheme (Yu et al., 2009) is defined over a set of anchor points $V = \{\boldsymbol{v}_j\}_{j=1}^m$. Each data point can be approximated by a linear combination of anchor points, namely

$$\boldsymbol{x} \approx \sum_{\boldsymbol{j=1}}^{m}\gamma_{v_j}(\boldsymbol{x})\boldsymbol{v}_j \tag{5}$$

$$\sum_{\boldsymbol{j=1}}^{m}\gamma_{v_j}(\boldsymbol{x}) = 1. \tag{6}$$

The coefficient $\gamma_{v_j}(\boldsymbol{x})$ represents the degree of relationship of $\boldsymbol{x}$ to boundary point $\boldsymbol{v}_j$. According to Yu et al. (2009), any Lipschitz smooth function $h(\boldsymbol{x})$ defined on a manifold can be approximated by a linear combination of functions $h(\boldsymbol{v})$ over the set of anchor points, namely:

$$h(\boldsymbol{x}) \approx \sum_{j=1}^{m}\gamma_{v_j}(x)h(\boldsymbol{v}_j). \tag{7}$$

Based on the local coding theory, a complex classifier can be approximated by a combination of local classifiers, each of which is related to an anchor point. To deal with linearly non-separable data, a smooth decision boundary with constrained curvature is desired since an unconstrained decision boundary is likely to overfit (Mao et al., 2015). Therefore the functions $\boldsymbol{w}(\boldsymbol{x})$ and $b(\boldsymbol{x})$ can be Lipschitz smooth in the feature space. Apply Eq. (7) to $\boldsymbol{w}(\boldsymbol{x})$ and $\boldsymbol{b}(\boldsymbol{x})$, we obtain:

$$\begin{aligned} H(\boldsymbol{x}) &= \boldsymbol{w}(\boldsymbol{x})^T\boldsymbol{x} + b(\boldsymbol{x}) \\ &= \sum_{j=1}^{m}\gamma_{v_j}(\boldsymbol{x})\boldsymbol{w}(\boldsymbol{v}_j)^T\boldsymbol{x} + \sum_{j=1}^{m}\gamma_{v_j}(\boldsymbol{x})b(\boldsymbol{v}_j) \\ &= \sum_{j=1}^{m}\gamma_{v_j}(\boldsymbol{x})f_{v_j}(\boldsymbol{x}) \end{aligned} \tag{8}$$

$$f_{v_j}(\boldsymbol{x}) = \boldsymbol{w}(\boldsymbol{v}_j)^T\boldsymbol{x} + b(\boldsymbol{v}_j). \tag{9}$$

In locally linear SVMs, $f_{v_j}(\boldsymbol{x})$, $\boldsymbol{w}(\boldsymbol{v}_j)$, $b(\boldsymbol{v}_j)$ can be interpreted as the decision function, the weight vector and bias of a linear SVM regarding the anchor point $\boldsymbol{v}_j$, respectively. According to the training method of locally linear SVMs, $\boldsymbol{w}(\boldsymbol{v}_j)$, and $b(\boldsymbol{v}_j)$ are trained with input data points whose positions in feature space are close to the anchor point $\boldsymbol{v}_j$.

Eqs. (8) and (9) define the locally linear classifier as the weighted sum of m separate linear classifiers, and local coding is used to assign the weights. Existing models adopt a two step training method: first train the positions of anchor points by k-means clustering, and then linear SVMs by SGD. LLBAP adds a intermediate step that selects a set of "boundary points" from anchor points and use these boundary points for local coding and training linear SVMs.

Generally speaking, the local coding method is equivalent to a "soft" partition that divides data into overlapping parts.

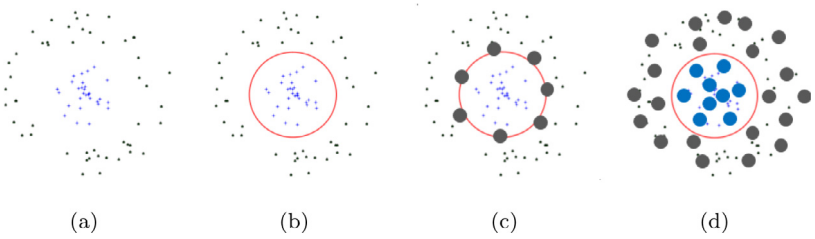

Fig. 1. A binary classification dataset is shown in Fig. 1(a). Fig. 1(b) shows the desired decision boundary that the classification model aims to learn. Fig. 1(c) shows the set of boundary anchor points. Fig. 1(d) shows the anchor points used in LLSVM and LLCSAPL, which are learned by k-means clustering.

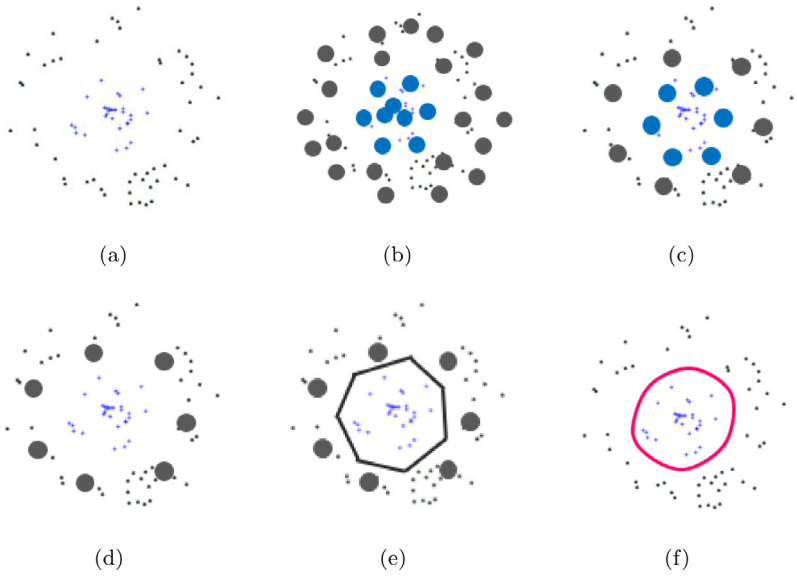

Fig. 2. A brief description of LLBAP's learning procedure based on a nonlinear binary classification task. Fig. 2(a) shows a binary classification dataset. In Fig. 2(b), LLBAP learns two sets of anchor points for the two classes respectively. In Fig. 2(c), LLBAP finds out the boundary points of each set. In Fig. 2(d), only the boundary points from one class are used for local coding. Fig. 2(e)(f) show the piecewise linear decision boundary of each linear SVM and global decision boundary that LLBAP aims to learn, respectively.

In contrast, another kind of partition is "hard" partition that divides data into disjointed parts, such as the clustering based partition method in CSVM. The "hard" partition may result in decision boundaries containing abrupt changes between adjacent subparts, and further result in overfitting. The "soft" partition alleviates this problem and contributes to the construction of smooth decision boundaries.

3. The proposed method

In this section, we present details of the proposed LLBAP. Without loss of generality, we describe the learning process of LLBAP based on a nonlinear binary classification task. As shown in Fig. 2(a), the desired decision boundary of the dataset is a circle. To classify this dataset, LLBAP aims to learn a piecewise linear decision boundary which can fit the desired decision boundary. In detail, as shown in Fig. 2(b)(c)(d), LLBAP first learns two sets of anchor points for the two classes separately, then finds out the boundary points of each class. Since boundary points of a single class are sufficient to form the piecewise linear decision boundary, LLBAP only uses the boundary points of one class for local coding. Next, local coding is used to partition the dataset into overlapping parts. The position of a boundary point and a related linear SVM are trained with respect to each part of data by solving a single optimization problem with SGD. As shown in Fig. 2(e)(f), each boundary point corresponds to a linear decision boundary, and the whole decision boundary of LLBAP is combined from local decision boundaries and smoothed by local coding.

The details of the learning process are described in the following subsections.

3.1. Anchor point learning for each class

Herein, we propose the anchor point learning method of LLBAP. Anchor points are stored as nodes in a graph structure $G = \langle V, E \rangle$, which is built adaptively from an empty graph. We adopt the graph structure to model the topological structure of anchor points. The general learning procedure is shown in Fig. 3. Note that the anchor point learning procedure is applied separately for each class of training data.

This procedure includes three steps: In the initialization step, two anchor points are added to a set of nodes V. In the online anchor point learning step, we add new anchor points and adjust existing anchor points based on a *similarity threshold criterion* (see Definition 1). Noisy anchor points and connections are detected and deleted every once in a while. The connections between anchor points are built according to the *competitive Hebbian rule* (see Definition 2). The connections are helpful for deciding when to add new anchor points and whether an existing anchor point is generated by noisy data, but they are not used after the online learning step. Finally, in the point-tuning step, iterative training is used to slightly adjust the positions of anchor points to get more

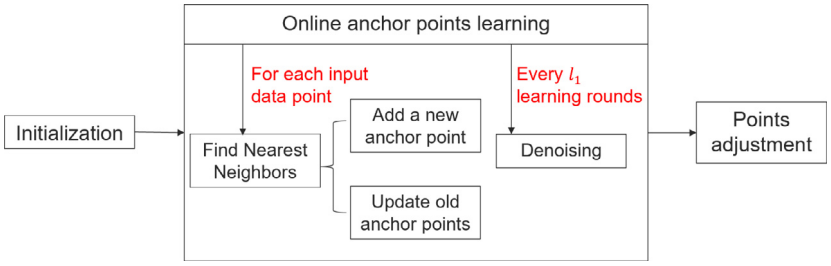

Fig. 3. Anchor point learning procedure.

stable results. The details of anchor point learning are described below.

Definition 1 (*Similarity Threshold*)**.** Denote the set of neighbors of anchor point $\boldsymbol{v}$ as N_v. If N_v is not empty, then the similarity threshold T_v of node $\boldsymbol{v}$ is defined by the largest squared Euclidean distance between $\boldsymbol{v}$ and its neighbors:

$$T_v = \max_{\boldsymbol{i} \in N_v} \|\boldsymbol{i} - \boldsymbol{v}\|^2. \tag{10}$$

Otherwise, T_v is defined by the smallest squared Euclidean distance between $\boldsymbol{v}$ and other anchor points:

$$T_v = \min_{\boldsymbol{i} \in N} \|\boldsymbol{i} - \boldsymbol{v}\|^2. \tag{11}$$

Definition 2 (*Competitive Hebbian Rule*)**.** For each input data point, find its two closest anchor points (measured by squared Euclidean distance) and connect them with an edge.

Initialization. A graph structure is used to store anchor points. Each anchor point $\boldsymbol{v} \in V$ has four attributes: position, winning times M_v, similarity threshold T_v and connections to other points (edges). The set of nodes V is initialized by randomly selecting two data points as new nodes from training data. For each new node $\boldsymbol{v}$, M_v is initialized as 1. T_v can be determined by the similarity threshold criterion.

Each edge $(\boldsymbol{v}_1, \boldsymbol{v}_2)$ has an age attribute $g_{(\boldsymbol{v}_1, \boldsymbol{v}_2)}$. The set of edges E is initialized as an empty set, implying that the initial graph does not have edges in it.

Online competitive learning of input data. After the initial step, data points are learned one-by-one. The procedure of learning a single data point is illustrated in Fig. 4. For each input data point $\boldsymbol{x}$, we first activate its nearest anchor point (measured by Euclidean distance) $\boldsymbol{s}_1$ and second nearest anchor point $\boldsymbol{s}_2$, which are defined by:

$$\boldsymbol{s}_1 = arg \min_{\boldsymbol{v} \in V} \|\boldsymbol{x} - \boldsymbol{v}\|^2 \tag{12}$$

$$\boldsymbol{s}_2 = arg \min_{\boldsymbol{v} \in V - \{\boldsymbol{s}_1\}} \|\boldsymbol{x} - \boldsymbol{v}\|^2. \tag{13}$$

Then, we compute the similarity thresholds $T_{\boldsymbol{s}_1}$ and $T_{\boldsymbol{s}_2}$ of $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ according to Definition 1. Each anchor point represents a local region whose radius is its similarity threshold.

If $\boldsymbol{x}$ lies in both the local regions of its two nearest points, then $\boldsymbol{s}_1$ is updated by:

$$\boldsymbol{s}_1 = \boldsymbol{s}_1 + \frac{1}{M_{\boldsymbol{s}_1}}(\boldsymbol{x} - \boldsymbol{s}_1). \tag{14}$$

This operation is equivalent to pulling $\boldsymbol{s}_1$ towards $\boldsymbol{x}$. $M_{\boldsymbol{s}_1}$ is increased by 1 afterwards. If $\boldsymbol{x}$ lies outside one of the local regions of $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$, a new anchor point $\boldsymbol{v} = \boldsymbol{x}$ is inserted. Here M_v is initialized as 1.

Based on the competitive Hebbian rule, if no edge exists between $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$, we establish an edge $(\boldsymbol{s}_1, \boldsymbol{s}_2)$ between them. The age of the edge $g_{(\boldsymbol{s}_1, \boldsymbol{s}_2)}$ is set as 1, which means that edge is young. If $\boldsymbol{s}_1$ is updated, other edges connected to $\boldsymbol{s}_1$ will become old. Namely, $g_{(\boldsymbol{v}, \boldsymbol{s}_1)}$ is increased by 1 for each edge $(\boldsymbol{v}, \boldsymbol{s}_1) \in E$. If $g_{(\boldsymbol{v}, \boldsymbol{s}_1)}$ exceeds a predefined threshold g_{max}, it is considered as an outdated edge and deleted from E.

Deleting noisy anchor points. A denoising strategy is used to detect and delete anchor points generated by noisy data. Two predefined parameters l_1 and a are used to control the frequency and intensity of denoising. When the number of learned data points reaches a multiple of l_1, the algorithm enters a denoising procedure. We first compute the mean winning times of all anchor points M_{mean}. An anchor point $\boldsymbol{v}$ is marked as noise if the number of its related edges is no more than 1 and M_v is less than the product of a and the mean winning times of all anchor points. Then $\boldsymbol{v}$ and its related edges are removed from the graph.

Point adjustment. After all data points in the training set have gone through the online learning procedure, we adjust the positions of learned anchor points by iterative training. In every iteration, we first find the nearest anchor point $\boldsymbol{v}$ of each data point $\boldsymbol{x}$. Denote the set of examples whose nearest anchor point is $\boldsymbol{v}$ as X_v, then we update $\boldsymbol{v}$ as:

$$\boldsymbol{v} = \frac{1}{|X_v|} \sum_{\boldsymbol{x} \in X_v} \boldsymbol{x}, \tag{15}$$

where $|X_v|$ represents the number of examples in X_v.

A single iteration of point adjustment is equivalent to a single training iteration in k-means clustering. However, instead of running until convergence like k-means, LLBAP only needs a few iterations of point adjustment. The reason is that the quality of anchor points learned in former steps is good enough to represent the training data, and therefore we only need to slightly adjust their positions to get more stable results. More iterative training would also weaken the effect of denoising in the online learning procedure.

Although the result of initialization and online learning can be affected by the input sequence of training data, the result is generally stable if the training data is randomly shuffled. The point adjustment step also improves the robustness of the proposed anchor point learning method.

3.2. Boundary point scanning and local coding

The scanning method is used to find anchor points near the desired decision boundary. The definition of boundary points is given as follows:

Definition 3 (*Boundary Points*)**.** Let $V^p = \{\boldsymbol{v}_j^p\}_{j=1}^{N^p}$ and $V^n = \{\boldsymbol{v}_j^n\}_{j=1}^{N^n}$ denote the sets of anchor points learned from positive and negative examples, respectively. $\boldsymbol{v}_i^p$ is a boundary point of the positive class if there exists $\boldsymbol{v}_j^n \in V^n$ that $\|\boldsymbol{v}_i^p - \boldsymbol{v}_j^n\|^2 \leq \|\boldsymbol{v}_k^p - \boldsymbol{v}_j^n\|^2$ for any $i \neq k$, and vice versa for boundary points of the negative class.

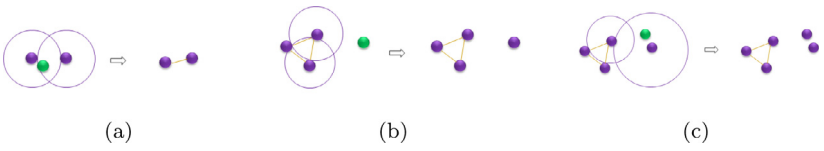

Fig. 4. Adaptive anchor point learning. In the figures, green nodes represent the input data points, other nodes represent the anchor points learned by LLBAP. In Fig. 4(a), the nearest anchor point is pulled to the input data points, and an edge is established between two anchor points. In Fig. 4(b) and (c), new anchor points are inserted adaptively. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

The scanning method is quite straightforward. Specifically, for each anchor point in V^p, find the nearest point $\boldsymbol{v}_j^n$ in V^n, mark $\boldsymbol{v}_j^n$ as a boundary point. In the same way, we can use the anchor points in V^n to find the boundary points in V^p.

In binary classification problems, using the boundary points of a single class is sufficient to form the piecewise linear decision boundary. Hence LLBAP only uses the smaller set of boundary points for efficiency.

Boundary points are used for local coding after they are located. The local coding method in LLBAP is the same as the method described in Section 2. As mentioned in Eqs. (5)–(8), a coefficient function $\gamma_{v_j}(\boldsymbol{x})$ needs to be defined as the measure of relationship between data point $\boldsymbol{x}$ and boundary point $\boldsymbol{v}_j$. In our proposed model, the coefficient function is defined as following:

$$\gamma_{v_j}(\boldsymbol{x}) = \begin{cases} \dfrac{\exp(-\delta\|\boldsymbol{v}_j - \boldsymbol{x}\|^2)}{\sum_{l\in N_k(\boldsymbol{x})}\exp(-\delta\|\boldsymbol{v}_l - \boldsymbol{x}\|^2)} & j \in N_k(\boldsymbol{x}) \\ 0 & otherwise. \end{cases} \tag{16}$$

We use Euclidean distance to measure the similarity between $\boldsymbol{x}$ and $\boldsymbol{v}_j$. $N_k(\boldsymbol{x})$ denotes the set of k-nearest boundary points of $\boldsymbol{x}$. The coding method used here is *localized soft-assignment coding*. In contrast, global coding that expands a data point to all the points ignores the underlying local manifold structure, and therefore leads to unreliable estimation of the membership to distant points. The localized soft-assignment coding scheme alleviates this problem and achieves computational efficiency at the same time due to the sparse representation.

3.3. Locally linear classifiers and training method

We next integrate the above local coding method to our locally linear classifier model. According to Eq. (8), the decision function $H(\boldsymbol{x})$ is determined by the parameters in both the boundary points $(\boldsymbol{v}_j,\ j = 1,\ldots,m)$ and linear SVMs $(\boldsymbol{w}(\boldsymbol{v}_j), b(\boldsymbol{v}_j),\ j = 1,\ldots,m)$. In the former steps, the anchor points are initialized in an unsupervised manner, and hence can be further optimized with supervised training. Therefore we formulate an optimization problem over the parameters in both the boundary points and linear SVMs and solve it by SGD. The loss function for LLBAP is defined as following:

$$Q = \frac{\lambda}{2}\sum_{j=1}^{m}\|\boldsymbol{w}(\boldsymbol{v}_j)\|^2 - \frac{1}{N}\sum_{n=1}^{N}\log\sum_{j=1}^{m}\gamma_{v_j}(\boldsymbol{x}_n)e^{-\ell(y_n, f_{v_j}(\boldsymbol{x}_n))}. \tag{17}$$

This loss function is similar to that of Mixture of Experts (Jacobs, Jordan, Nowlan, & Hinton, 1991). According to Jacobs et al. (1991), if parameters of both points and classifiers are trained by gradient descent in this error function, the model tends to devote a single linear classifier to each training case. That enables the model to adapt to the underlying local manifold structure. Finally, to be in conformity with the loss function, we can obtain the output of each locally linear SVM as:

$$O_{v_j}(\boldsymbol{x}) = e^{-\ell(1, f_{v_j}(\boldsymbol{x}))} - e^{-\ell(-1, f_{v_j}(\boldsymbol{x}))}. \tag{18}$$

Thus, the output of LLBAP is:

$$H(\boldsymbol{x}) = O(\boldsymbol{x}) = \sum_{j=1}^{m}\gamma_{v_j}(\boldsymbol{x})O_{v_j}(\boldsymbol{x}). \tag{19}$$

We minimize the loss function in Eq. (17) with SGD to obtain the optimal parameters. In each iteration of SGD, we randomly pick a labeled data point $(\boldsymbol{x}, y)$ and then update the parameters of boundary points and linear SVMs. Since each data point is approximated by the linear combination of its k-nearest boundary points, only those boundary points and corresponding linear SVMs are optimized in each iteration. If the boundary point $\boldsymbol{v}_j$ belongs to the k-nearest neighbors of $\boldsymbol{x}$, then the change of $\boldsymbol{v}_j$ depends on the partial derivative of the following term L:

$$L = -\log\sum_{j=1}^{m}\gamma_{v_j}(\boldsymbol{x})e^{-\ell(y, f_{v_j}(\boldsymbol{x}))}. \tag{20}$$

The derivative of L with respect to $\boldsymbol{v}_j$ can be expressed as:

$$\frac{\partial L}{\partial \boldsymbol{v}_j} = -\delta(h_j - \gamma_{v_j})\frac{\boldsymbol{x} - \boldsymbol{v}_j}{\|\boldsymbol{x} - \boldsymbol{v}_j\|}, \tag{21}$$

where h_j is defined as:

$$h_j = \frac{\gamma_{v_j}(\boldsymbol{x})e^{-\ell(y, f_{v_j}(\boldsymbol{x}))}}{\sum_{k\in N(\boldsymbol{x})}\gamma_{v_k}(\boldsymbol{x})e^{-\ell(y, f_{v_k}(\boldsymbol{x}))}}. \tag{22}$$

Then, the boundary point $\boldsymbol{v}_j$ is updated as:

$$\boldsymbol{v}_j^{t+1} = \boldsymbol{v}_j^t + \frac{n_0}{\lambda(t+t_0)}(h_j - \gamma_{v_j})\frac{\boldsymbol{x} - \boldsymbol{v}_j^t}{\|\boldsymbol{x} - \boldsymbol{v}_j^t\|^2}, \tag{23}$$

where t denotes the current iteration number.

By computing the partial derivative of L with respect to $\boldsymbol{w}(\boldsymbol{v}_j)$ and $b(\boldsymbol{v}_j)$, we show that the parameters of linear SVMs can be updated as:

$$\boldsymbol{w}(\boldsymbol{v}_j)^{t+1} = \boldsymbol{w}(\boldsymbol{v}_j)^t + \frac{1}{\lambda(t+t_0)}yh_j\boldsymbol{x} - \frac{1}{(t+t_0)}\boldsymbol{w}(\boldsymbol{v}_j)^t \tag{24}$$

$$b(\boldsymbol{v}_j)^{t+1} = b(\boldsymbol{v}_j)^t + \frac{1}{\lambda(t+t_0)}yh_j - \frac{1}{(t+t_0)}b(\boldsymbol{v}_j)^t. \tag{25}$$

In Eqs. (23)–(25), the optimal learning rate is denoted as $\frac{1}{\lambda(t+t_0)}$ (Shalev-Shwartz, Singer, & Srebro, 2007), where λ is the regularization coefficient and t_0 is a positive constant that avoids producing too large steps in the first few iterations. We can harmonize the learning of anchor points and classifiers by changing the value of n_0 in Eq. (23).

3.4. Computational complexity of LLBAP

We first analyze the computational complexity of training LLBAP. Suppose the training dataset contains n data points, where each data point is a d-dimensional feature vector.

In the anchor point learning step, the most time-consuming computation is calculating the pairwise distances between input data points and anchor points. Denoting the maximum number of

Algorithm 1 SGD for LLBAP

Input: $\{\mathbf{v}_j^0\}_{j=1}^m$, $\{\mathbf{w}(\mathbf{v}_j)^0 = \mathbf{0}, b(\mathbf{v}_j)^0 = 0\}_{j=1}^m$, λ, t_0, T, k, δ, $\{\mathbf{x}_n, y_n\}_{n=1}^N$,
Output: $\{\mathbf{v}_j, \mathbf{w}(\mathbf{v}_j), b(\mathbf{v}_j)\}_{j=1}^m$
for t = 1, 2, …T **do**
  Input the labeled example $(\mathbf{x}^t, y^t)$
  Get the k-nearest boundary points $N_k(\mathbf{x}^t)$ of $\mathbf{x}^t$,
  for each point $\mathbf{v}_j$ $(j \in N_k(\mathbf{x}^t)$ **do**
    Compute $\gamma_{\mathbf{v}_j}(\mathbf{x}^t)$, $\frac{\partial L}{\partial \mathbf{v}_j}$, h_j according to Eqs. (16), (21), (22)
    Update $\mathbf{v}_j, \mathbf{w}(\mathbf{v}_j), b(\mathbf{v}_j)$ according to Eqs. (23)–(25)
  end for
end for

anchor points as m, the time complexity of each training iteration is $O(nmd)$, which is equal to the complexity of each iteration of k-means. However, the proposed method only needs one iteration of online learning and a few iterations of point adjustment. Therefore it is more efficient than k-means. It is shown in the experiments of Har-Peled and Sadri (2005) that k-means usually needs tens or hundreds of iterations to converge.

The boundary point scanning step needs to compute pairwise distance between anchor points in different classes, and therefore the time complexity is $O(m^2d)$. Here we denote the number of selected boundary points as m_b.

When training classifiers with SGD, the time complexity of training each linear SVM is $O(nd)$. The number of linear classifiers is equal to the number of boundary points, and therefore its complexity is $O(nm_bd)$ in each iteration. This step is the most time-consuming training step because SGD takes a large number of iterations before convergence.

Compared to LLSVM, the proposed method is more efficient mainly because it takes less iterations when learning anchor points. The computational complexities of the classifier training step in LLSVM and LLBAP depend on the number of linear classifiers, we will provide an experimental analysis of this number in the following section. The computational complexity of LLBAP is also much smaller than the $O(n^2d)$ complexity of kernel SVM.

Now we consider the computational complexity of prediction. When predicting the label of each input data point, the time cost of LLBAP is proportional to the number of linear classifiers and the dimensionality. Therefore the computational complexity of LLBAP for predicting each input data point is $O(m_bd)$, which is the same as LLSVM. Compared to kernel SVM whose prediction time cost is proportional to the number of support vectors, locally linear classifiers are more efficient because the number of linear classifiers is much smaller than that of support vectors. It was proven in Steinwart (2003) that the fraction of support vectors is necessarily greater than the smallest achievable misclassification risk, which means the number of support vectors is closely related to the size of training dataset and the probability that the labels of data are affected by noise. In contrast, the number of linear classifiers in LLBAP is not affected by the size of training data, and therefore the size of training data does not affect the computational complexity of prediction.

4. Analysis experiment

Herein, we experimentally show the decision boundary learning process in LLBAP. As shown in Fig. 5(a), the synthetic dataset is a 2-dimensional dataset whose desired decision boundary is a circle. To learn the piecewise linear decision boundary, LLBAP first learns anchor points for each class. As Fig. 5(b) shows, the distribution of the anchor points is approximate to that of the training data. Then the boundary points are found by using the scanning method as shown in Fig. 5(c). By using the boundary point based local coding, LLBAP partitions the data into overlapping subparts, each of which will be classified by a linear SVM. Finally, both boundary points and linear SVMs are trained by SGD. As shown in Fig. 5(d)(e), each boundary point, which corresponds to a subpart, corresponds to a linear SVM whose learned decision boundary is a line. Those linear decision boundaries are combined into a smooth nonlinear decision boundary by local coding.

Comparing Fig. 5(c) and (d), we find that the boundary points hardly move. This phenomenon indicates that the proposed anchor point learning and boundary point scanning method gives satisfying initial positions for boundary points, which further demonstrates the effectiveness of the boundary points based local coding method. Moreover, the decision boundary of LLBAP shown in Fig. 5(e) is close to that of kernel SVM shown in Fig. 5(f). However, LLBAP only uses 15 boundary points while the kernel SVM uses 137 support vectors. The prediction costs are proportional to the number of boundary points for LLBAP and the number of support vectors for kernel SVM, respectively. Therefore LLBAP is much more efficient than kernel SVM.

Table 1
Details of real-world benchmark datasets.

Datasets	# training	# testing	# feature
USPS	7291	2007	256
MAGIC04	12680	6340	10
MNIST	60000	10000	780
IJCNN	49990	91701	22
SKIN	122528	122529	3
CENSUS	199523	99762	510
KDD99	4898431	311029	127

5. Real-world experiment

5.1. Datasets

We then show the performance of LLBAP in real-world benchmark datasets. The details of these datasets are shown in Table 1. The MAGIC04, CENSUS and KDD99 are available at the UCI repository (Dheeru & Karra Taniskidou, 2017). The SKIN, IJCNN1, USPS and MNIST are obtained from LibSVM website (Chang & Lin, 2011).

Each dataset has a testing set except MAGIC04 and SKIN. In our experiments, we randomly divide MAGIC04 into three parts and select two parts as training set, leaving the remaining as testing set. SKIN is divided into two parts, half for training and half for testing. Features in all datasets are linearly scaled to [0, 1].

5.2. Experimental setting

We compared LLBAP with state-of-the-art methods from the family of locally linear classifiers, including SVM-KNN, CSVM, LLSVM, LLCSAPL. DCSVM (Hsieh, Si, & Dhillon, 2014) is used for comparison as a state-of-the-art method for accelerating kernel SVM training. LLBAP and all the comparing methods are implemented in C++. The experiments are performed on a PC with a 3.20 GHz Intel i5-6500 CPU and 32 GB memory.

5.2.1. Parameters settings

All the parameters in comparing methods are set according to their original papers unless specifically mentioned. The number of clusters in CSVM is set as 8, and the numbers of linear classifiers (anchor points) in LLSVM and LLCSAPL are both fixed as 50 in these experiments.

Parameters in LLBAP include g_{max}, l_1, a (in anchor point learning step) and k, δ, λ (in the classifiers training step). In anchor

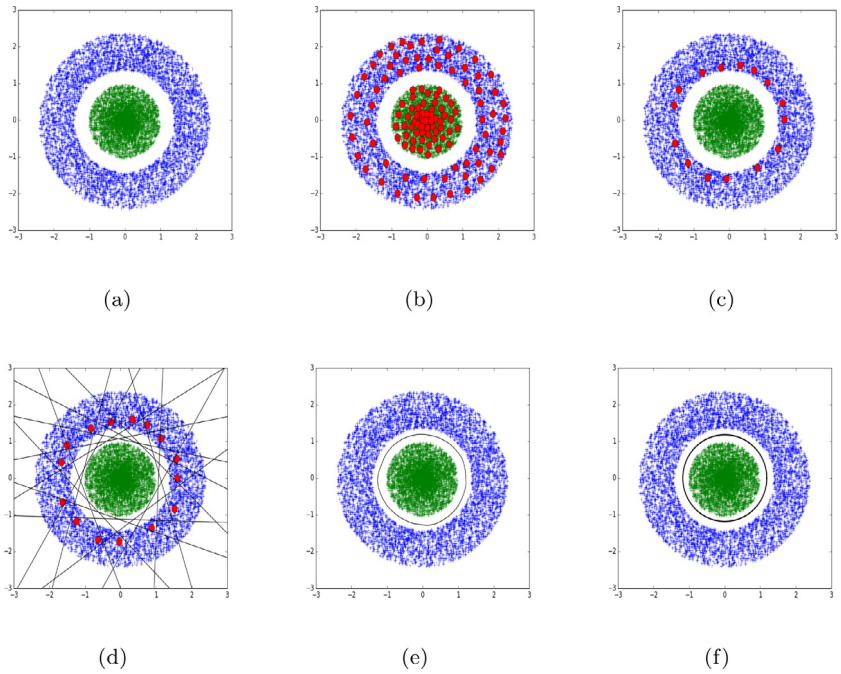

Fig. 5. The learning process of LLBAP. Fig. 5(a) shows the synthetic dataset. Fig. 5(b) shows the anchor points learned by LLBAP. Fig. 5(c) shows the boundary points that LLBAP used for local coding. Fig. 5(d) shows the locally linear SVMs and the boundary points after classifiers training. Fig. 5(e) shows the piecewise linear decision boundary learned by LLBAP. Fig. 5(f) shows the decision boundary of kernel SVM.

point learning step, LLBAP learns a set of boundary points for later training. However, the quality of the boundary points learned in the former step is not very important. The parameter selection for the anchor point learning of LLBAP is easy. Because even if the positions of boundary points are not well learned, they can also be modified later in the classifiers training step. In the following experiments, parameter g_{max}, l_1 and a are set to be constant values (100, 100 and 0.5 respectively). Besides, we have tried some different values of g_{max}, l_1, a and have confirmed that the performance of LLBAP is quite stable with different settings of these parameters.

The other three parameters (k, δ, λ) are for classifiers training. Same as kernel SVM, LLCSAPL and other locally linear classifiers, δ and λ are both tuned by cross-validation searching the grid {1, 2, 5, 10} and {0.00001, 0.000001}. The value of k has an impact on the local coding quality. Herein, k is a constant parameter and set to 8 as in Mao et al. (2015).

5.3. Experimental results

The classification results with training and testing time are shown in Table 2. The number of linear classifiers in LLBAP is automatically decided by boundary point scanning, which is also reported in the table. The bold values represent the best results among the locally linear classifiers (CSVM, SVM-KNN, LLCSAPL, LLSVM and LLBAP).

The underlined values represent the best results among all the methods except kernel SVM and linear SVM. Since SVM-KNN does not have a training step, its training time is marked as "Nan". Besides, "–" means that the training and testing time (includes parameters tuning time) is too long (more than 5 days).

Unsurprisingly, linear SVM performs poorly in all the datasets, although it has low training and testing costs. Kernel SVM achieves the best predictive accuracy, but it is expensive in both training and testing especially in large-scale and high-dimensional datasets. As an accelerating algorithm for kernel SVM training, DCSVM also performs well in terms of predictive accuracy in all datasets. DCSVM introduces a pre-training process that uses divide-and-conquer method to get the initial coefficient of each training data point. Its training speed heavily depends on the performance of its pre-training process, and therefore the training time sometimes even exceeds that of kernel SVM. Moreover, DCSVM is not designed to reduce the number of support vectors. It cannot boost the testing speed of kernel SVM, which is usually more important than training speed in real world applications.

Compared to kernel SVM, locally linear classifiers largely reduce testing time without much loss of accuracy except SVM-KNN. This is because the number of linear SVMs is usually less than that of the support vectors used in kernel SVM. For SVM-KNN, owing to its nature of lazy learning, its testing time even exceeds that of kernel SVM. CSVM is efficient in both training and testing but its prediction performance is relatively bad. Because its performance heavily depends on the result of k-means clustering, it is likely to perform poorly with improper initialization.

LLBAP, LLSVM and LLCSAPL are all developed based on local coding. Therefore we mainly compare LLBAP with LLSVM and LLBAP to show the enhancements of LLBAP. As Table 2 shows, LLBAP outperforms both LLSVM and LLCSAPL in all datasets in terms of predictive accuracy. That is because LLBAP learns better data partition by boundary point scanning and further optimizes the positions of anchor points with supervised training. As a result, in the classifiers training step, LLBAP tends to converge to a more satisfying result. In addition, the experimental results also demonstrate that the training and testing efficiency of LLBAP are better than those of LLSVM and LLBAP. Because LLBAP only uses boundary points for local coding, the total number of linear SVMs in LLBAP is usually less than that of LLSVM and LLCSAPL. We further compare the anchor point learning in LLBAP, LLSVM and LLCSAPL, as well as its effect on predictive accuracy in the following subsection.

5.4. Analysis of anchor point learning

We have experimentally shown the performance of LLBAP on large-scale datasets. Herein, we analyze the learning of anchor

Table 2
Experimental results of LLBAP and other methods.

		USPS	MAGIC04	MNIST	IJCNN	SKIN	CENSUS	KDD99
Kernel SVM	Accuracy (%)	96.81	86.65	98.10	98.79	99.97	95.35	92.51
	Training time (s)	8.55	4.11	1570.58	42.50	431.93	21852.88	5757.50
	Testing time (s)	1.012	0.614	67.298	19.340	25.870	1558.990	154.627
Linear SVM	Accuracy	83.56	79.46	83.91	91.82	92.30	95.09	91.92
	Training time	0.23	0.79	14.45	8.29	0.15	24.51	20.07
	Testing time	0.001	0.001	0.007	0.002	0.001	0.045	0.038
DCSVM	Accuracy	96.81	85.33	98.01	98.51	99.86	95.26	92.53
	Training time	33.58	12.53	1532.88	23.01	119.73	10122	1877.07
	Testing time	2.375	0.625	94.303	7.001	16.609	1887.53	134.707
CSVM	Accuracy	92.58	84.73	93.39	94.70	97.75	95.26	92.22
	Training time	**3.27**	2.89	**97.61**	**12.74**	9.88	120.31	532.67
	Testing time	**0.023**	**0.029**	**0.2035**	**0.511**	0.657	1.469	0.523
SVM-KNN	Accuracy	96.25	85.92	97.79	97.17	99.85	94.63	–
	Training time	Nan	Nan	Nan	Nan	Nan	Nan	–
	Testing time	13.498	8.934	1347.462	680.169	1366.064	30109.366	–
LLCSAPL	Accuracy	95.57	85.76	97.54	98.59	99.50	95.06	91.93
	Training time	31.05	28.13	454.15	75.34	45.37	408.92	2742.48
	Testing time	0.187	0.057	1.62	2.388	0.834	3.727	4.919
LLSVM	Accuracy	95.57	85.34	97.14	98.42	99.15	95.06	91.93
	Training time	18.58	4.62	224.72	30.75	18.51	138.38	1447.54
	Testing time	0.224	0.066	1.572	2.356	0.829	3.727	4.919
LLBAP (Proposed)	Accuracy	**96.32**	**85.92**	**97.81**	**98.69**	**99.87**	**95.32**	**92.45**
	Training time	6.53	**2.88**	103.59	19.29	**8.07**	**62.63**	**243.65**
	Testing time	0.054	0.027	0.472	0.906	**0.270**	**1.254**	**0.423**
	Linear classifiers	23	30	32	45	12	27	3

Table 3
Experimental results of LLBAP using different anchor point learning methods.

		USPS	MAGIC04	MNIST	IJCNN	SKIN	CENSUS	KDD99
Proposed method	Accuracy (%)	96.32	85.92	97.81	98.69	99.87	95.32	92.45
	Training time (s)	1.43	0.30	26.81	2.86	1.83	27.31	149.96
k-means	Accuracy	96.46	85.83	97.70	98.51	99.87	95.08	92.03
	Training time	10.78	4.75	548.53	23.77	31.32	240.11	6751.36
SNC	Accuracy	96.11	85.50	97.96	98.67	99.87	95.06	–
	Training time	194.34	28.15	10867.58	292.15	295.11	22753.95	–

points in LLBAP. LLSVM and LLCSAPL with varying number of anchor points are used for comparison. Specifically, we set the number of anchor points m in LLSVM and LLCSAPL according to the grid {10, 20, 30, …, 200}, and repeat the experiment on benchmark datasets. The predictive accuracies with respect to various number of anchor points are depicted in Fig. 6. In each sub-figure, the x-axis represents the number of anchor points, and the y-axis shows the average value and standard deviation of predictive accuracy on the test data over 5 runs.

As Fig. 6 shows, the predictive accuracies of LLSVM and LLCSAPL are closely related to the number of anchor points. In most cases, the performances of both LLCSAPL and LLSVM improve with the increasing of m while m is small. After m exceeds a threshold, the accuracies will fluctuate in a narrow range. Hence the threshold is a good value for balancing accuracy and efficiency, but it varies in different datasets. For example, it is 30 for LLSVM and 110 for LLCSAPL in SKIN while it is 90 for both LLSVM and LLCSAPL in IJCNN, which makes tuning the parameter m difficult. Although we can set m large enough to ensure predictive accuracy, that will also raise the computation costs of LLSVM and LLCSAPL. An advantage of LLBAP is its ability to adaptively learn this parameter. Moreover, we can see from the results that m in LLBAP is not related to the size of training dataset. For example, KDD99 is a large dataset with millions of data points, but LLBAP only trains 3 linear classifiers on this dataset.

We perform another group of experiments to validate the efficiency of the proposed anchor point learning method. In this group of experiments, we replace the anchor point learning method in LLBAP with two different methods: k-means and a supervised data compression method named Stochastic Neighbor Compression (SNC) (Kusner, Tyree, Weinberger, & Agrawal, 2014). The other training steps, including boundary point scanning and classifier training, are left unchanged. We compare the original LLBAP and the two alternative methods in terms of classification accuracy and the speed of anchor point learning. The numbers of anchor points in k-means and SNC are predefined parameters. We first run the original anchor point learning method, and use the number of learned anchor points as the value of this parameters for k-means and SNC. The results are reported in Table 3. Note that the training times in Table 3 are the times for learning anchor points. SNC cannot be applied on the KDD99 dataset due to an out of memory error.

As shown in Table 3, the classification accuracy of LLBAP is not largely affected by the choice of anchor point learning method, but the training time of the proposed method is much less than those of k-means and SNC. The main reason for this phenomenon is that the proposed method only needs to iterate several times on the data, while k-means and SNC need significantly more iterations to converge.

The performance of LLBAP also benefits from the supervised fine-tuning of boundary points. We validate the effectiveness of the proposed fine-tuning method by comparing LLBAP, LLSVM and LLCSAPL using the same initial anchor points. In this group of experiments, we first train a LLBAP model on the dataset, then use its boundary points to initiate the anchor points in LLSVM and LLCSAPL, and then train linear classifiers accordingly. The comparison of classification accuracies is shown in Table 4.

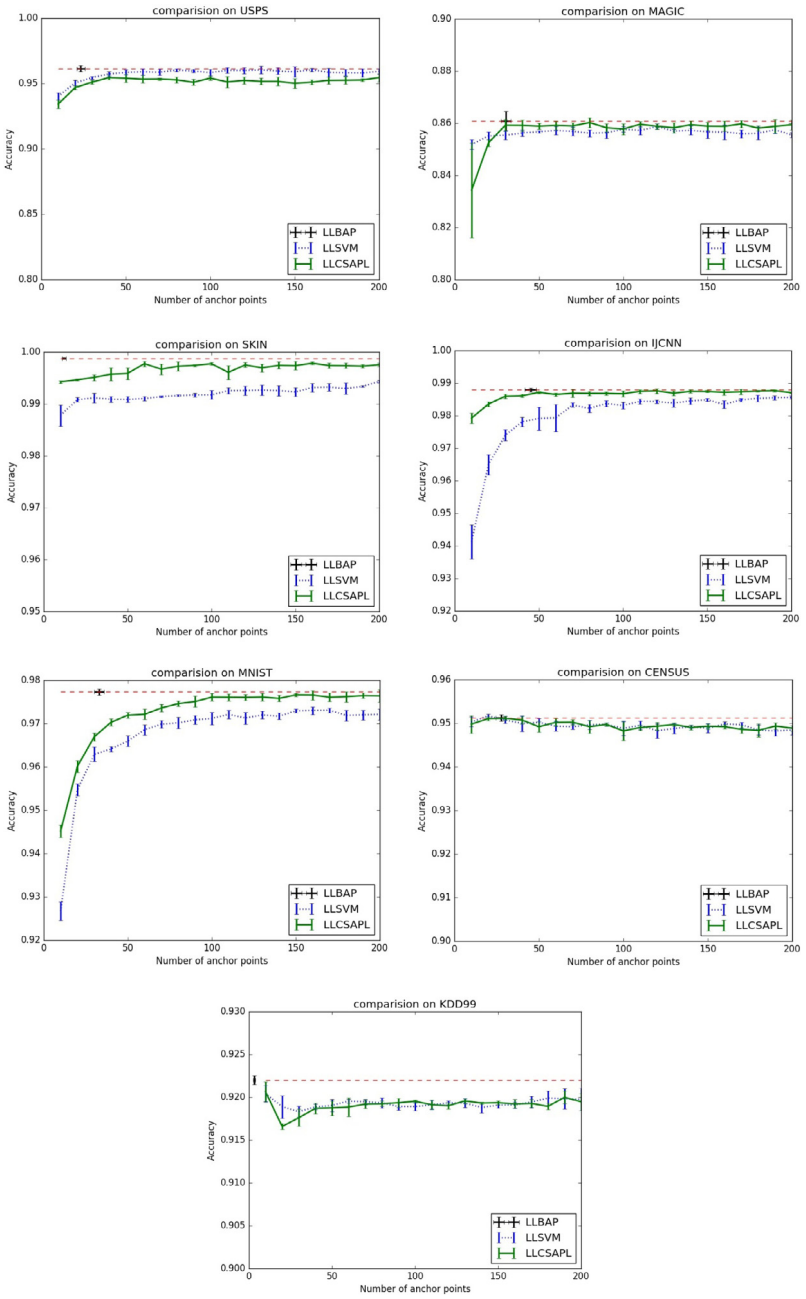

Fig. 6. In all sub-figures, the *x*-axis represents the number of anchor points, and the *y*-axis shows the average value and standard deviation of predictive accuracy on the test data over 5 runs. The sub-figures show the results on USPS, MAGIC04, SKIN, IJCNN, MNIST, CENSUS and KDD99 respectively.

When LLBAP, LLSVM and LLCSAPL use the same initial anchor points, the predictive accuracy of LLBAP still outperforms those of other algorithms on most datasets. The results prove that the proposed joint training method for anchor points and linear classifiers is effective.

Table 4
Accuracies(%) of LLBAP, LLSVM and LLCSAPL using same initial anchor points.

	USPS	MAGIC04	MNIST	IJCNN	SKIN	CENSUS	KDD99
LLBAP	**96.32**	85.92	**97.81**	**98.69**	**99.87**	**95.32**	**92.45**
LLSVM	94.27	85.23	95.29	98.28	98.67	95.23	92.09
LLCSAPL	94.76	**85.96**	96.39	98.49	99.55	95.21	91.98

5.5. Analysis of parameter *k*

Now we investigate how the performances of our method and other local coding based methods change with respect to the value of *k*. Specifically, we vary *k* according to the grid {1, 2, 4, 6, 8, 10, 12, 16, 20}, and repeat the experiments in the same way as the previous subsection. The classification accuracies of those methods with respect to various *k* are depicted in Fig. 7. In each sub-figure, the *x*-axis represents different values of *k*, while the *y*-axis is the averaged classification accuracy on the test data over 10 runs.

We can see that the classification accuracies of LLBAP, LLSVM and LLCSAPL are relatively stable while *k* is moderate (neither too large nor too small). This phenomenon is consistent with the specialty of these methods. The learned decision boundary is not smooth when *k* is small, which may result in overfitting. When *k* is too large, these methods may tend to fit the global shape

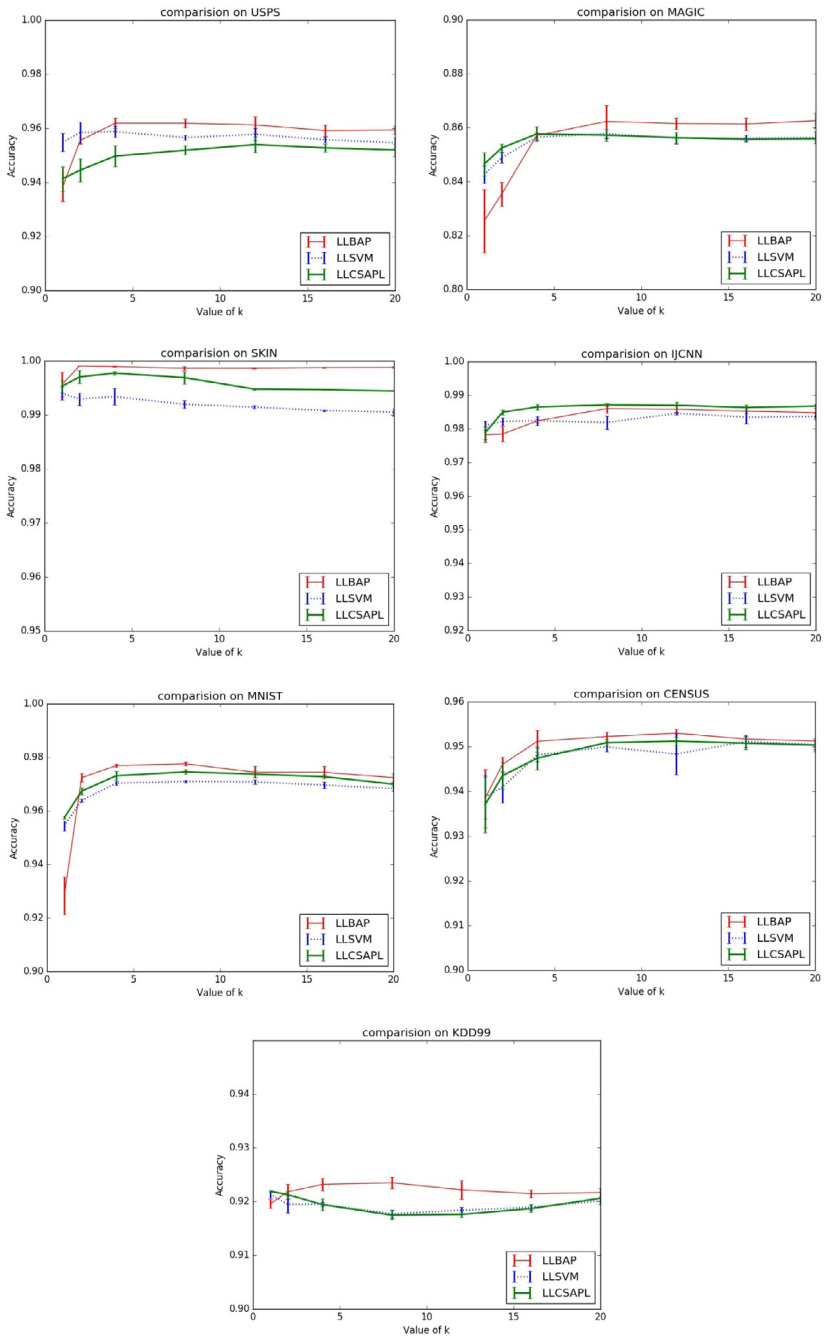

Fig. 7. In all sub-figures, the x-axis represents different values of k, while the y-axis is the averaged classification accuracy on the test data over 10 runs. The sub-figures show the results on USPS, MAGIC04, SKIN, IJCNN, MNIST, CENSUS and KDD99 respectively.

of the decision boundary and ignore the local information. However, this phenomenon does not make k hard to tune. As Fig. 7 shows, when k varies between 6 and 10, their performances are stable.

6. Conclusion

In this paper, we propose a locally linear classifier named LLBAP. Unlike other locally linear classifiers which always partition data based on clustering methods, LLBAP partitions data using local coding with boundary anchor points. Specifically, LLBAP uses a scanning method to adaptively find the boundary points. Boundary point based local coding is used to partition the data into approximately linearly separable parts, each of which is classified by a linear SVM. Furthermore, we formulate an optimization problem over both boundary points and local SVMs, and then solve the problem with a gradient descent method.

We demonstrate the effectiveness of the proposed boundary points based local coding method with a synthetic experiment. Specifically, the initial partition is nearly the same as the final partition. The decision boundary of LLBAP is approximate to that of kernel SVM. By experimentally comparing LLBAP with kernel SVM and other state-of-the-art locally linear classifiers on many benchmark datasets, we prove that LLBAP is both effective and efficient. In our experiments, LLBAP achieves comparable classification performance with kernel SVM while it is significantly more

efficient, and its performance is also superior to those of other state-of-the-art locally linear classifiers.

In the future, we will further enhance the partition method in local linear classifier. Specifically, a supervised boundary point learning method should be designed to ensure that the target of partition learning and that of SVMs learning are totally consistent. In addition, most existing locally linear classifiers are trained in two separate steps (partition learning step and classifier training step) which impose restriction on their efficiency and application scenarios (e.g streaming data). We will try to combine the two learning steps and propose an enhanced method that can be applied to online and incremental learning problems.

Acknowledgments

This work is supported in part by the National Science Foundation of China under Grant Nos. (61876076), Jiangsu NSF, China grant (BK20141319) and the Dengfeng Scholars Program of Nanjing University, China (B1512002).

References

Arthur, D., & Vassilvitskii, S. (2006). How slow is the k-means method? In *Proceedings of the twenty-second annual symposium on computational geometry* SCG '06, (pp. 144–153). New York, NY, USA: ACM, http://dx.doi.org/10.1145/1137856.1137880, URL http://doi.acm.org/10.1145/1137856.1137880.

Boser, B. E., Guyon, I. M., & Vapnik, V. N. (2008). A training algorithm for optimal margin classifier. In *Proceedings of annual ACM workshop on computational learning theory*: *vol. 5*, (pp. 144–152).

Chang, C. C., & Lin, C. J. (2011). *LIBSVM: A library for support vector machines* (pp. 1–27). ACM.

Cortes, C. (1995). Support vector network. *Machine Learning*, *20*(3), 273–297.

Dheeru, D., & Karra Taniskidou, E. (2017). *UCI machine learning repository*. University of California, Irvine, School of Information and Computer Sciences, URL http://archive.ics.uci.edu/ml.

Fornoni, M., Caputo, B., & Orabona, F. (2013). Multiclass latent locally linear support vector machines. In *JMLR W&CP, Asian conference on machine learning*: *vol. 29*, (pp. 229–244).

Gu, Q., & Han, J. (2013). Clustered support vector machines. In *Proceedings of the 16th international conference on artificial intelligence and statistics* (pp. 307–315).

Har-Peled, S., & Sadri, B. (2005). How fast is the k-means method? *Algorithmica*, *41*(3), 877–885.

Hsieh, C. J., Si, S., & Dhillon, I. S. (2014). A divide-and-conquer solver for kernel support vector machines. In *International conference on machine learning* (pp. I–566).

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., & Hinton, G. E. (1991). Adaptive mixtures of local experts. *Neural Computation*, *3*(1), 79–87.

Kecman, V., & Brooks, J. P. (2010). Locally linear support vector machines and other local models. In *International joint conference on neural networks* (pp. 1–6).

Kusner, M. J., Tyree, S., Weinberger, K., & Agrawal, K. (2014). Stochastic neighbor compression. In *International conference on machine learning* (pp. II–622).

Ladicky, L., & Torr, P. (2011). Locally linear support vector machines. In *International conference on machine learning* (pp. 985–992).

Macqueen, J. (1967). Some methods for classification and analysis of multivariate observations. In *Proc. of Berkeley symposium on mathematical statistics and probability* (pp. 281–297).

Mao, X., Fu, Z., Wu, O., & Hu, W. (2015). Optimizing locally linear classifiers with supervised anchor point learning. In *International conference on artificial intelligence* (pp. 3699–3706).

Shalev-Shwartz, S., Singer, Y., & Srebro, N. (2007). Pegasos: Primal estimated sub-gradient solver for SVM. In *Machine learning, proceedings of the twenty-fourth international conference* (pp. 807–814).

Smola, A. J. (2008). Learning with kernels | support vector machines. *Lecture Notes in Computer Science*, *42*(4), 1–28.

Steinwart, I. (2003). Sparseness of support vector machines. *Journal of Machine Learning Research (JMLR)*, *4*(6), 1071–1105.

Yu, K., Zhang, T., & Gong, Y. (2009). Nonlinear learning using local coordinate coding. In *International conference on neural information processing systems* (pp. 2223–2231).

Zhang, H., Berg, A. C., Maire, M., & Malik, J. (2006). SVM-KNN: Discriminative nearest neighbor classification for visual category recognition. In *IEEE computer society conference on computer vision and pattern recognition* (pp. 2126–2136).