On Unsupervised Training of Multi-Class RLS Classifiers

Tapio Pahikkala[1],*Member, ACM, IEEE* Antti Airola[1], Fabian Gieseke[2], and Oliver Kramer[3]

[1] *University of Turku & Turku Centre for Computer Science, Turku, Finland*
[2] *University of Copenhagen, Department of Computer Science, Copenhagen, Denmark*
[3] *Carl von Ossietzky Universität Oldenburg, Computer Science Department, Oldenburg, Germany*

E-mail: tapio.pahikkala@utu.fi, antti.airola@utu.fi, fabian.gieseke@diku.dk, oliver.kramer@uni-oldenburg.de

Abstract

In this work we present the first efficient algorithm for unsupervised training of multiclass regularized least-squares classifiers. The approach is closely related to the unsupervised extension of the support vector machine classifier known as maximum margin clustering, which has received recently considerable attention, though mostly considering the binary classification case. We present a combinatorial search scheme that combines steepest descent strategies with powerful meta-heuristics for avoiding bad local optima. The regularized least-squares based formulation of the problem allows us to use matrix algebraic optimization enabling constant time checks for the intermediate candidate solutions during the search. Our experimental evaluation indicates the potential of the novel method and demonstrates its superior clustering performance over a variety of competing methods on real-world data sets. Both time complexity analysis and experimental comparisons show that the method can scale well to practical sized problems.

Keywords Unsupervised Learning, Multi-Class Regularized Least-Squares Classification, Maximum Margin Clustering, Combinatorial Optimization

1 Introduction

Unsupervised learning belongs to the most important tasks at the beginning of each data mining process: In an early phase, no labeled data at all are given, and the task consists in extracting reasonable information based on the patterns only. Various unsupervised learning tasks like clustering or dimensionality reduction have been proposed in the literature over the years [1]. In this work we concentrate on clustering, which plays a central role in a variety of real-world applications in computer vision, information retrieval, marketing, context modeling [2] and many other fields [3]. Roughly speaking, clustering techniques aim at grouping objects into *clusters*, so that objects with similar characteristics belong to the same cluster, and those with different properties to different ones.

In recent years, the supervised learning method known as the *support vector machine* (SVM) [4, 5] as well as other regularized learning schemes have been extended to unsupervised learning settings, in most cases under the name of *maximum margin clustering* (MMC) [6]. These extensions aim at finding a partition of the unlabeled patterns into classes, so that a subsequent application of the underlying supervised model yields the overall best result. In general, these unsupervised extensions induce combinatorial or non-convex

optimization tasks that are difficult to address. However, since the obtained models have been proven to outperform standard clustering techniques in many experimental analyses, they have received considerable attention over the last years.

A preliminary version of this paper was published in the Proceedings of The 12th IEEE International Conference on Data Mining (ICDM 2012) [7]. The theory behind the algorithm framework as well as the discussion and motivation behind the proposed meta-heuristic have been considerably extended in the current article, and the performance evaluation now includes a running time comparison with the baseline methods.

1.1 Related Work

Xu *et al.* [6] were among the first ones who formalized the extension of support vector machines to unsupervised learning scenarios. Their optimization approach is based on reformulating the original combinatorial task as semidefinite programming problem [8], which can then be addressed via standard solvers. An extension of this framework based on semidefinite programming as well is provided by Valizadegan and Jin [9]. In contrast to the work of Xu *et al.* [6], however, they show how to reduce the number of involved optimization variables, which yields a more efficient optimization framework. A recent local search approach for the linear case is given by Zhao *et al.* [10]. Their optimization framework is based on a combination of recently proposed cutting plane schemes and concave-convex procedures. Similar ideas have also been presented by Li *et al.* [11].

An alternative approach for the binary case is proposed by Zhang *et al.* [12]. Basically, their simple but surprisingly effective approach is based on iteratively applying a support vector machine model to improve an "initial guess" that is obtained via an auxiliary clustering framework. One of the key ingredients of their framework is the replacement of the original hinge loss by the ε-insensitive or the square loss. As pointed out by Zhang *et al.* [12], the resulting models *"can more easily get out of a poor solution"*. These ideas are extended by Gieseke *et al.* [13], who propose matrix-based update strategies that can be used to significantly speed up stochastic search frameworks. In line with the approach of Zhang *et al.* [12], they resort to the square loss instead of the hinge loss.

1.2 Contribution

While there exists a significant body of research on extending supervised regularized classifiers to clustering under the framework of maximum margin clustering, almost all of the work has concentrated on the binary case. The exception is the cutting plane multi-class method of Zhao *et al.* [14]. The method can be efficiently trained, but is, as demonstrated in our experiments, prone to getting stuck at bad local minima leading to inferior clustering performance. While Xu *et al.* [15] also formalize a method for the multi-class setting, their method has a runtime complexity of $\mathcal{O}(n^7)$ for n patterns, which becomes impractical for real-world problems.

In this work we extend the concept of supervised *one-vs-all multi-class regularized least-squares classification* [16] to unsupervised learning settings. As reported by Zhang *et al.* [12] and Gieseke *et al.* [13], the square loss depicts a very reasonable choice in the context of such clustering settings and offers desirable computational shortcuts for corresponding optimization strategies. The contribution provided in this work is twofold:

1. Firstly, we show how to enhance simple steepest descent strategies by means of a powerful meta-heuristic that effectively avoids bad local optima. While being a seemingly simple modification, we

demonstrate that this minor adaptation provides major improvements to the clustering accuracy compared to straightforward stochastic search and steepest descent implementations.

2. Secondly, in line with our previous work [13], we provide computational shortcuts for assessing the quality of the intermediate clustering candidate solutions. As we show, these shortcuts render function calls possible to be conducted in $\mathcal{O}(1)$ time, which paves the way for an exhaustive search in the large combinatorial search space.

The meta-heuristic (which we call a *shaking* strategy) is an important algorithmic ingredient for the unsupervised one-vs-all extension which we address. We experimentally analyze our approach on various data sets; the results demonstrate that our approach is capable of yielding better clustering accuracies than conventional techniques in most cases.

So far, no publicly available implementation can be found in the literature that takes care of the interesting multi-class maximum margin principle, and we consider the approach presented in this work to be a valuable candidate for such difficult multi-class clustering settings. Our implementation is made available as part of the RLScore software library at `http://users.utu.fi/aatapa/RLScore/`.

2 Mathematical Background

In this section we provide the mathematical notations and the mathematical background related to the general concept of regularized kernel methods [4, 5], which encompasses the regularized least-squares classification framework and support vector machines as a special case. We start from the standard supervised setting and proceed to re-formalize the central concepts for the unsupervised learning setting. To simplify the notation, we denote the set of all $n \times m$ matrices with real coefficients by $\mathbb{R}^{n \times m}$. Given a particular matrix $\boldsymbol{M} \in \mathbb{R}^{n \times m}$, we denote its element in the i-th row and j-th column by $\boldsymbol{M}_{i,j}$, and we use y_i to denote the i-th coordinate of a vector $\boldsymbol{y} \in \mathbb{R}^n$.

2.1 Binary Classification Scenarios

We start by depicting the binary cases, for both supervised and unsupervised learning settings. The multi-class scenarios that are central for the work at hand are described afterwards.

2.1.1 Supervised Regularized Kernel Methods

Regularized least-squares and support vector machines can be seen as a special case of so-called *regularized kernel methods* [4, 5]. We briefly define these settings and then show how to extend the corresponding supervised models to unsupervised (multi-class) learning settings.

Let X be an arbitrary set and let $k : X \times X \to \mathbb{R}$ be a *kernel function* that can be seen as a similarity measure for the elements in this space. For a given labeled training set

$$\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\} \subset X \times Y$$

with $Y = \{-1, +1\}$ the regularized risk minimization problem is defined as

$$\operatorname*{argmin}_{f \in \mathcal{H}_k} \left\{ \sum_{i=1}^{n} l\big(y_i, f(\boldsymbol{x}_i)\big) + \lambda \|f\|^2_{\mathcal{H}_k} \right\}, \quad (1)$$

where f is the prediction function (also called *model*) that maps a given data pattern to a real-valued prediction and $\|\cdot\|_{\mathcal{H}_k}$ is a norm in a *reproducing kernel Hilbert space* $\mathcal{H}_k$ induced by the kernel function k. The disagreement between the predictions and the true labels is measured via a loss function $l : Y \times \mathbb{R} \to [0, \infty)$ that gives rise to the *empirical risk*, which, in turn, measures how well the prediction function fits to all training patterns. The *regularization*

parameter $\lambda \in \mathbb{R}_+$ determines the trade-off between the first term of the task (1) and the complexity of the prediction function f [4, 5].

Two prominent representatives of this family of regularization methods are support vector machines and the concept of *regularized least-squares classification* [17]. The first one stems from the use of the *hinge loss* $l\big(y, f(\boldsymbol{x})\big) = \max(0, 1 - yf(\boldsymbol{x}))$, whereas the latter one is based on the *square loss* $l\big(y, f(\boldsymbol{x})\big) = \big(y - f(\boldsymbol{x})\big)^2$. By the representer theorem [18], any solution $f^* \in \mathcal{H}_k$ of the task (1) has the form

$$f^*(\cdot) = \sum_{i=1}^{n} a_i k(\boldsymbol{x}_i, \cdot) \tag{2}$$

with coefficients $\boldsymbol{a} = (a_1, \ldots, a_n)^{\mathrm{T}} \in \mathbb{R}^n$. Hence, by plugging in the square loss into the objective and by using $\|f^*\|^2_{\mathcal{H}_k} = \boldsymbol{a}^{\mathrm{T}}\boldsymbol{K}\boldsymbol{a}$ [5] with kernel matrix $\boldsymbol{K} \in \mathbb{R}^{n\times n}$ consisting of entries $\boldsymbol{K}_{i,j} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$, we can rewrite the task at hand as

$$\operatorname*{argmin}_{\boldsymbol{a}\in\mathbb{R}^n} J(\boldsymbol{a}) \tag{3}$$

with

$$J(\boldsymbol{a}) = (\boldsymbol{y} - \boldsymbol{K}\boldsymbol{a})^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{K}\boldsymbol{a}) + \lambda \boldsymbol{a}^{\mathrm{T}}\boldsymbol{K}\boldsymbol{a}. \tag{4}$$

The objective $J(\boldsymbol{a})$ of the above optimization task is convex and differentiable with respect to $\boldsymbol{a} \in \mathbb{R}^n$. Thus a global minimizer can analytically be obtained by enforcing $\frac{\partial}{\partial \boldsymbol{a}} J(\boldsymbol{a}) = 0$, that is, a minimizer of the objective is given by

$$\boldsymbol{a}^* = \boldsymbol{G}\boldsymbol{y} \tag{5}$$

with

$$\boldsymbol{G} = (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1},$$

where $\boldsymbol{I} \in \mathbb{R}^{n\times n}$ is the identity matrix [17]. Although the resulting models are, in general, not sparse (as it is often the case for standard support vector machines), the above closed-form solution is a desirable property [5, 17]. As we will see below, this is especially the case in the context of the considered clustering scenarios.

By substituting the minimizer (5) into (2), it becomes rewritten as the following linear combination of the training labels

$$\begin{aligned} f(\cdot) &= \sum_{i=1}^{n} (\boldsymbol{G}\boldsymbol{y})_i k(\boldsymbol{x}_i, \cdot) \\ &= \sum_{i,j=1}^{n} \boldsymbol{G}_{i,j} y_j k(\boldsymbol{x}_i, \cdot) \\ &= \sum_{j=1}^{n} y_j b_j(\cdot)\ , \end{aligned} \tag{6}$$

where the n basis functions $b_j(\cdot)$ are defined as

$$b_j(\cdot) = \sum_{i=1}^{n} \boldsymbol{G}_{i,j} k(\boldsymbol{x}_i, \cdot)\ .$$

Note that the basis functions are completely independent on the labels as they only depend of the input data points and of the regularization parameter. This observation is central with regard to this work, since several of the computational shortcuts we take advantage of in the algorithms arise from this property.

In the literature, the representation (6) is sometimes called the dual form of (2) and the basis functions are called the equivalent kernels, because (6) resembles the kernel smoothing technique. For more in depth discussion about these issues, we refer to Girosi *et al.* [19] and the references therein.

Accordingly, we obtain the minimum of the objective function (3) as the following closed form expression:

$$\begin{aligned} &\min_{\boldsymbol{a}\in\mathbb{R}^n} J(\boldsymbol{a}) \\ &= (\boldsymbol{y} - \boldsymbol{K}\boldsymbol{G}\boldsymbol{y})^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{K}\boldsymbol{G}\boldsymbol{y}) + \lambda \boldsymbol{y}^{\mathrm{T}}\boldsymbol{G}\boldsymbol{K}\boldsymbol{G}\boldsymbol{y}. \end{aligned}$$

Inspired by this, we define a new objective function in which the above closed form expression is written as a function of the label vector $\boldsymbol{y}$:

$$\begin{aligned} F(\boldsymbol{y}) =& (\boldsymbol{y} - \boldsymbol{K}\boldsymbol{G}\boldsymbol{y})^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{K}\boldsymbol{G}\boldsymbol{y}) \\ &+ \lambda \boldsymbol{y}^{\mathrm{T}}\boldsymbol{G}\boldsymbol{K}\boldsymbol{G}\boldsymbol{y}\ . \end{aligned} \tag{7}$$

This provides a powerful and easy to use approach for testing different labelings for the given data. Below, we will use this function as a basic tool when considering the least-squares-based clustering algorithms.

2.1.2 Unsupervised Least-Squares Extension

As pointed out by Zhang *et al.* [12], the square loss depicts an ideal candidate for the maximum margin principle from a practical point of view. Further, the above closed-form solution can also be used to greatly speed up the computations induced by a variety of search strategies [13]. The direct extension of the supervised regularized kernel methods (1) to the unsupervised case for a given unlabeled training set $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \subset X$ has the form [6]:

$$\operatorname*{argmin}_{\boldsymbol{y}\in\{-1,+1\}^n,\, f\in\mathcal{H}_k} \left\{ \sum_{i=1}^{n} l\big(y_i, f(\boldsymbol{x}_i)\big) + \lambda \|f\|^2_{\mathcal{H}_k} \right\}$$

Hence, the difficult part is the additional integer optimization variable $\boldsymbol{y} \in \{-1, +1\}^n$ that encodes the partition of the given unlabeled patterns. To avoid trivial solutions, some form of a *balancing constraint* is usually added for such clustering settings, which is of the form

$$\left| \frac{1}{n} \sum_{i=1}^{n} y_i \right| < b_c$$

with a user-defined parameter $b_c \in [0, 1]$. By again considering the square loss, we can take advantage of the tools established in the previous section and can rewrite the above optimization problem in terms of the label vector only:

$$\operatorname*{argmin}_{\boldsymbol{y}\in\{-1,+1\}^n} F(\boldsymbol{y}). \tag{8}$$

with $F(\boldsymbol{y})$ being the closed form given in (7).

2.2 Multi-Class Classification Scenarios

We can now address the multi-class learning settings that are the basis of the optimization schemes derived in this work. Like above, we start by outlining the supervised models followed by their unsupervised extensions.

2.2.1 Supervised Multi-Class Extension

In the literature, several ways to extend the concept of support vector machines and their variants to multi-class settings have been proposed. As reported by Rifkin and Klautau [16] the so-called one-versus-all multi-class classification setting depicts a valuable candidate for such learning scenarios, and we follow this line of research for the unsupervised case.

In such multi-class supervised settings, we are given a training set

$$\{(\boldsymbol{x}_1, c_1), \ldots, (\boldsymbol{x}_n, c_n)\} \subset X \times \mathcal{C}$$

with $\mathcal{C} = \{1, \ldots, |\mathcal{C}|\}$ as set of all possible class labels. In a nutshell, one aims at deriving models $f_1, \ldots, f_{|\mathcal{C}|}$ such that a new pattern $\boldsymbol{x} \in X$ is assigned to the class whose associated model is the most confident. Formally, given the $|\mathcal{C}|$ models, the class label for a data point $\boldsymbol{x}$ is obtained as:

$$f(\boldsymbol{x}) = \operatorname*{argmax}_{c\in\mathcal{C}} f_c(\boldsymbol{x}) \;, \tag{9}$$

that is, the model with the largest real-valued prediction determines the class label.

It is easy to see that the above multi-class classification rule can also be written as

$$f(\boldsymbol{x}) = \operatorname*{argmin}_{c\in\mathcal{C}} L\left(c, f_1(\boldsymbol{x}), \ldots, f_{|\mathcal{C}|}(\boldsymbol{x})\right) \;, \tag{10}$$

where L is the following type of a one-versus-all multi-class loss function:

$$L\left(c, f_1(\boldsymbol{x}), \ldots, f_{|\mathcal{C}|}(\boldsymbol{x})\right) = \sum_{h=1}^{|\mathcal{C}|} l(p_h(c), f_h(\boldsymbol{x})),$$

and

$$p_h(c) = \begin{cases} 1 & \text{if } h = c \\ -1 & \text{otherwise} \end{cases} \;. \tag{11}$$

Here, the one-versus-all multi-class loss is decomposed to a sum of ordinary type of binary loss functions.

Let $\boldsymbol{c} \in \mathcal{C}^n$ be the vector containing the class labels of the training examples. Further, in order to support the forthcoming considerations below, we overload the notation p_h by defining it also for vectors of class labels, which leads to the following mapping from the class label vector to $\{-1, +1\}$ -valued vectors, one per each class $h \in \mathcal{C}$:

$$p_h(\boldsymbol{c}) = \begin{pmatrix} -1 \\ \vdots \\ -1 \end{pmatrix} + 2 \sum_{j=1}^{n} \delta_{c_j h} \boldsymbol{e}^j, \qquad (12)$$

where δ is the *Kronecker delta* (i.e., we have $\delta_{c_j h} = 1$ if $c_j = h$ and $\delta_{c_j h} = 0$ otherwise), and $\boldsymbol{e}^j$ is the j-th standard basis vector of $\mathbb{R}^n$. Hence, the i-th component of vector $p_h(\boldsymbol{c})$ equals $+1$ in case the i-th training pattern belongs to the class h, and equals -1 otherwise (see Figure 1 for an example).

A variety of different objectives (and loss functions) have been proposed in the literature [16]. In the following, we consider extensions of the supervised models (1) to the multi-class case having the form

$$\operatorname*{argmin}_{f_1,\ldots,f_{|\mathcal{C}|} \in \mathcal{H}_k} M\left(f_1, \ldots, f_{|\mathcal{C}|}, \boldsymbol{c}\right),$$

where

$$M\left(f_1, \ldots, f_{|\mathcal{C}|}, \boldsymbol{c}\right) = \sum_{h=1}^{|\mathcal{C}|} \sum_{i=1}^{n} \left(l\left(p_h(c_i), f_h(\boldsymbol{x}_i)\right) + \lambda \|f_h\|_{\mathcal{H}_k}^2 \right)$$

for some vector of class labels $\boldsymbol{c}$. Hence, the goal of the learning process is the search of binary-valued prediction functions $f_1, \ldots, f_{|\mathcal{C}|}$ that minimize the above risk. Given that the one-versus-all loss is decomposed into a sum of binary ones, the process can be considered as training $|\mathcal{C}|$ binary models *independently*. As pointed out above, such frameworks have been shown to work as well as other sophisticated multi-class schemes for such supervised settings, see Rifkin and Klautau [16].

Finally, let us consider the one-versus-all classification with the squared loss. Using analogous arguments as in the binary classification case, the minimizer of the above objective function has a closed form, which can be written simply as

$$\min_{f_1,\ldots,f_{|\mathcal{C}|} \in \mathcal{H}_k} M\left(f_1, \ldots, f_{|\mathcal{C}|}, \boldsymbol{c}\right) = Q(\boldsymbol{c})$$

where

$$Q(\boldsymbol{c}) = \sum_{h=1}^{|\mathcal{C}|} F(p_h(\boldsymbol{c})), \qquad (13)$$

and F is defined via (7).

2.2.2 Unsupervised Least-Squares Extension

Exactly as for the binary case, one can extend the multi-class framework depicted above to unsupervised settings by considering the class membership vector $\boldsymbol{c} \in \mathcal{C}^n$ as additional optimization variable. For the square loss, this leads to the following optimization problem:

$$\operatorname*{argmin}_{\substack{\boldsymbol{c} \in \mathcal{C}^n \\ f_1,\ldots,f_{|\mathcal{C}|} \in \mathcal{H}_k}} M(f_1, \ldots, f_{|\mathcal{C}|}, \boldsymbol{c}) = \operatorname*{argmin}_{\boldsymbol{c} \in \mathcal{C}^n} Q(\boldsymbol{c}).$$

Hence, one is again given a mixed-integer programming problem; the key problem is to find an appropriate assignment for the integer variable $\boldsymbol{c}$ such that the induced $|\mathcal{C}|$ supervised binary classification tasks yield the overall best results. Note that, while the objectives seem to be independent from each other, they interact via the vector $\boldsymbol{c}$, since changing the class membership of a *single* training instance leads to the modification of *two* of the induced classification models $f_1, \ldots, f_{|\mathcal{C}|}$.

Note that additional constraints can (and should) be used to enforce appropriate ratios of the cluster sizes. In the next section, we propose an efficient algorithm for finding accurate clustering solutions that aim at minimizing the above objective subject to such cluster

$$\boldsymbol{c} = \begin{pmatrix} 1 \\ 2 \\ 1 \\ 3 \\ 2 \\ 4 \\ 2 \end{pmatrix} \quad p_1(\boldsymbol{c}) = \begin{pmatrix} +1 \\ -1 \\ +1 \\ -1 \\ -1 \\ -1 \\ -1 \end{pmatrix} \; p_2(\boldsymbol{c}) = \begin{pmatrix} -1 \\ +1 \\ -1 \\ -1 \\ +1 \\ -1 \\ +1 \end{pmatrix} \; p_3(\boldsymbol{c}) = \begin{pmatrix} -1 \\ -1 \\ -1 \\ +1 \\ -1 \\ -1 \\ -1 \end{pmatrix} \; p_4(\boldsymbol{c}) = \begin{pmatrix} -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ +1 \\ -1 \end{pmatrix}$$

Fig. 1. A vector of class labels (left) and the corresponding mappings (right).

constraints, along with computational shortcuts for assessing the quality of intermediate candidate solutions.

3 Algorithmic Framework

In this section we describe the basic ideas behind the proposed algorithm without getting into the computational details. Due to the discrete nature of the clustering problem, we employ direct optimization methods for searching appropriate labels for the data points. In the literature, this type of methods are often referred to as hill climbing algorithms (see e.g. Russel and Norvig [20]).

There are different variants of hill climbing, such as stochastic and steepest hill climbing. In addition, there are so-called meta-algorithms that are built on top of the hill climbing algorithms, such as climbing with random restarts, etc. Here, we focus mainly on the idea of the steepest descent hill climbing, in which all closest neighbors of the current solution are compared, and the current solution is replaced with the neighbor having the lowest value of the objective function. In our case, the set of closest neighbors consists of label vectors that differ from the current solution *only by one entry*. In addition, we propose a meta-algorithm we call *shaking* that uses the idea of steepest descent so that it is less likely to get stuck to local minima with inferior clustering performance than the basic steepest descent search.

For convenience, we use $S(\boldsymbol{c}, j, d)$, where $\boldsymbol{c} \in \mathcal{C}^n$, $j \in \{1, \ldots, n\}$, and $d \in \mathcal{C}$, to denote the value of the objective function Q defined in equation (13) for a cluster label vector, whose entries are equal to those of $\boldsymbol{c}$ except that the label of the jth data point has been switched from c_j to d. This allows us to denote the search directions in the space of cluster label vectors so that each direction corresponds to switching the cluster label of a single data point. Armed with the above notation, and a vector of initial cluster assignments $\boldsymbol{c} \in \mathcal{C}^n$ for n data points, we next consider the search algorithms for solving the clustering problems.

3.1 Basic Descent Strategies

One of the most straightforward approaches is the so-called *stochastic hill climbing*, in which the algorithm simply goes trough the data points one at a time and switches its current class label to another one if it decreases the objective value, and stops when a local optimum is found. This type of algorithms were proposed for binary clustering in our previous work [13]. In our experiments, we use a similar algorithm modified for multi-class cluster-

ing as a baseline method. The modification can take advantage of the computational short-cuts presented in the appendix, and it is therefore computationally as efficient as the other methods proposed in this paper.

Algorithm 1 STOCHASTIC DESCENT

```
Initialize c ∈ C^n randomly
loop
    b ← True
    for j = 1, ..., n do
        d ← argmin_{d∈C} S(c, j, d)
        if c_j ≠ d then
            c_j ← d
            b ← False
        end if
    end for
    if b then ▷ Stop if local optimum found
        break
    end if
end loop
```

Algorithm 2 STEEPEST DESCENT

```
Initialize c ∈ C^n randomly
loop
    j, d ← argmin_{j∈{1,...,n},d∈C} S(c, j, d)
    if c_j = d then
        break
    else
        c_j ← d
    end if
end loop
```

Another framework is the basic *steepest descent search* (see Algorithm 2) for the multiclass clustering problem. The idea is that during each iteration the algorithm finds a pair (j, d), where j is the index of the data point, for which switching the cluster label would decrease the value of the objective function the most and d is the corresponding new cluster label. The algorithm stops when a local minimum is found, that is, switching the cluster assignment of a single data point will not decrease the objective value.

3.2 Avoiding Local Minima via Shaking

Both the stochastic and steepest descent methods *can easily get stuck in local minima* corresponding to inferior clusterings of the data; the existence and severity of this problem is confirmed in our experiments. For this reason, we propose to improve the steepest descent algorithm with a simple, but surprisingly effective trick (see Algorithm 3).

Instead of traversing the search space exactly towards the steepest descent direction, the algorithm iterates through the clusters and each cluster at a time claims a number of points from the other clusters. The point the cluster d claims next is determined by the steepest descent direction. That is, the point for which switching the cluster label to d would decrease the objective value the most (or increase the least), is assigned to the cluster d.

The number of points claimed by the clusters is determined by the following formula:

$$\alpha = \frac{n}{2^i|\mathcal{C}|} + \frac{n}{|\mathcal{C}|} - |\{h \mid c_h = d\}|. \qquad (14)$$

Firstly, it depends on the round i of the outer loop, which is encoded in the first term of (14) via the exponential factor in the nominator. During the first iterations, all clusters claim a large number of points, but the value of the first term decreases exponentially with respect to the iteration index of the loop.

To prevent the smaller clusters from disappearing completely, the number of points claimed by a cluster also depends on the number of points already assigned to the cluster in question, that is, the small clusters claim more points than the large ones. This behavior is ensured by the sum of the second and third terms of (14), which is positive for smaller than average sized clusters and negative for the larger ones. Note that this definition of α is

just an ad hoc heuristic and better ones may be designed, for example, if we have a prior knowledge about the cluster sizes. Nevertheless, as we show in the experiments, even this simple approach performs considerably better than the stochastic hill climbing and the basic steepest descent, since it avoids many of the inferior local minima into which the basic approaches get stuck.

Algorithm 3 STEEPEST DESCENT WITH SHAKING

```
Initialize c ∈ C^n randomly
for i = 0, ..., s do
  for d ∈ C do
    α ← n/(2^i |C|) + n/|C| − |{h | c_h = d}|
    for j = 1, ..., α do
      j ← argmin_{j∈{1,...,n}, d≠c_j} S(c, j, d)
      c_j ← d
    end for
  end for
end for
```

Intuitively, the idea of this approach can be considered as "shaking" the solution so that the local minima can be avoided. In the beginning, the solution is shaken profusely, but the intensity quickly decreases with the rounds of the outer loop. Because of the exponential decrease, the number s of shakings can be set to a small constant. In our experiments, we always use the value $s = 20$. This turned out to be a comparatively robust parameter choice in preliminary experiments.

The behavior of the proposed method is demonstrated in Figure 2 with a toy example consisting of three Gaussian clusters. First, the patterns are labeled randomly. Then, each cluster claims points from other clusters at a time and the number of points claimed decreases during each iteration.

The shaking heuristic is a powerful optimization approach due to its dynamic mechanism that controls the magnitude of changes according to a cooling scheme depending on the iteration number. In this regard, the shaking heuristic is similar to dynamic mutation rate control employing an external cooling scheme like simulated annealing (SA) [21] does. SA adapts the probability to accept a solution deterioration according to an exponential cooling scheme. A change of solution $\boldsymbol{c}$ that increases the fitness value by ΔS is accepted with a decreasing probability

$$p(\Delta S, T) = e^{-\Delta S/T}$$

depending on the fitness change ΔS and temperature T that must be increased in the course of the optimization run (e.g., depending on the iteration number). Our shaking heuristic differs from SA, as each solution change is accepted, but the magnitude of change is controlled dynamically. The dynamic scheme that is employed depends on the iteration number in an exponential kind of way. Linear decrease of the probably turned out to be less successful. It is possible to introduce a further parameter that controls the speed of the decrease of α. Extensions with this regards will be subject to future work.

A further characteristic of the shaking heuristic is that not a random part of the solution is optimized, but the search concentrates on particular solution parts, i.e., one cluster in each iteration employing greedily the steepest descent approach. A greedy cluster assignment is a recommendable optimization strategy, in particular in case of large solution spaces. Another important part of the shaking heuristic are the second and the third summands of equation (14). They give small clusters the chance to develop, discouraging the the optimization process from losing found clusters. Equipped with this knowledge, the algorithm has an advantage over pure direct search.

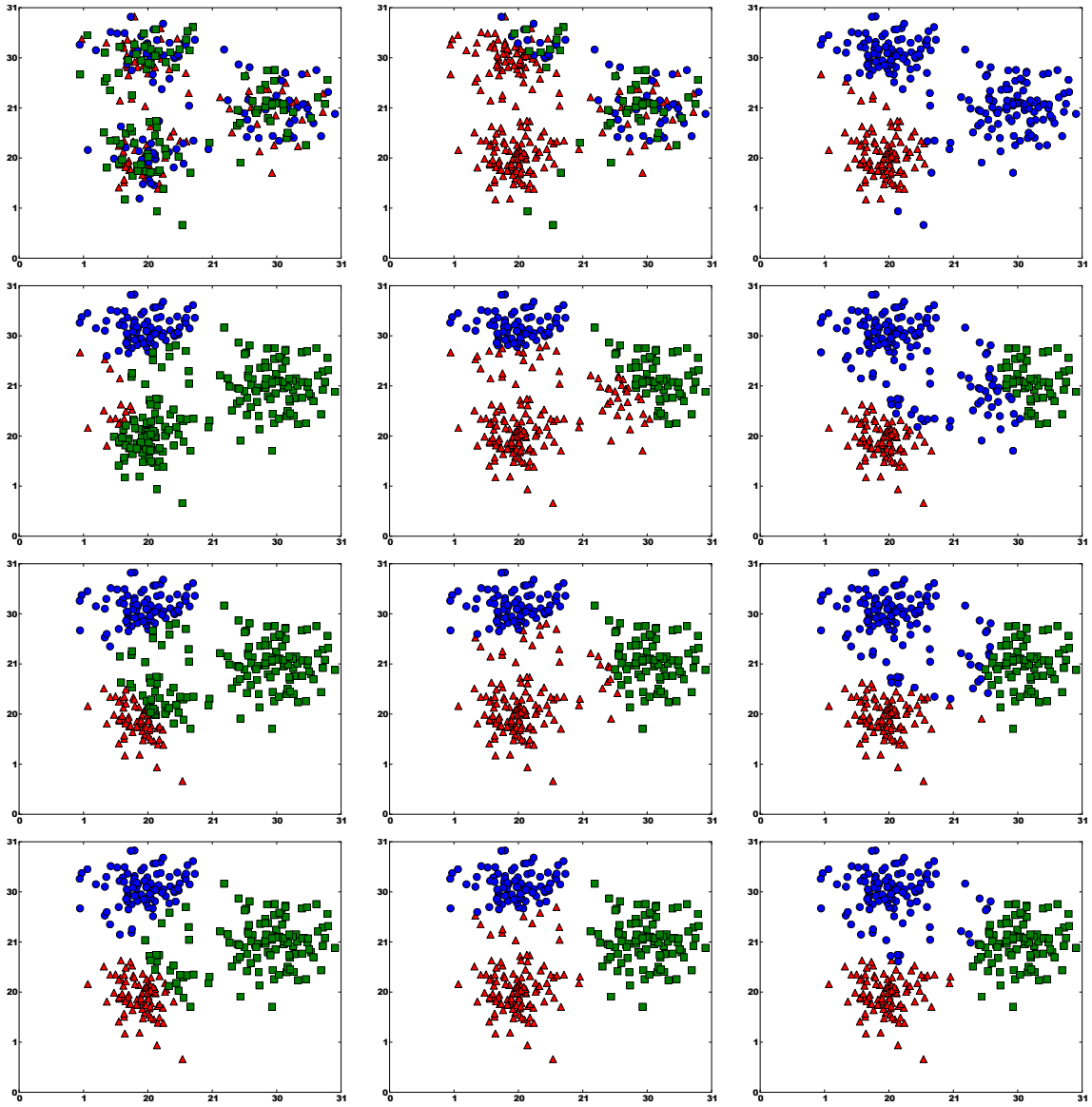

Fig. 2. Demonstration of the shaking meta-algorithm on three Gaussian clusters. The first image contains the initial random assignments of the cluster labels. The next three images correspond to the first round of the outermost loop during which each class claims a large number of data points at a time. During the following rounds, the classes keep claiming data points but the number of points claimed decreases exponentially with the index of the outermost loop.

3.3 Efficient Optimization via Matrix-Based Updates

As observed above, all considered search algorithms can be formulated in terms of the operation $S(\boldsymbol{c}, j, d)$ that computes the objective value for a certain neighbor of the label vector $\boldsymbol{c}$. In particular, we note that the shaking heuristic requires roughly $\mathcal{O}(s|\mathcal{C}|n^2)$ calls of $S(\boldsymbol{c}, j, d)$.

A naive approach for computing it would require retraining the classifiers from scratch each time the operation is used, which would clearly be computationally infeasible except for very small data sets. For example, the computational complexity of training support vector machine classifiers with non-linear kernels is $\mathcal{O}(n^3)$ in the worst case, and hence the overall complexity of running the shaking heuristic together with those would be of order $\mathcal{O}(s|\mathcal{C}|n^5)$.

The next theorem characterizes one of the main contributions of this paper. This offers massive runtime savings compared to the naive implementation mentioned above.

Theorem 1. *The computational complexity of Algorithm 3 is*

$$\mathcal{O}((s|\mathcal{C}| + r)n^2),$$

where r is the rank of the kernel matrix.

Proof. For the sake of exposition, we defer the lengthy proof to the appendix. The general idea is that given a certain amount of time spent in the preprocessing phase, one can test each possible flip of a coordinate in $\mathcal{O}(1)$ time, which is far less compared to a naive implementation that would take $\mathcal{O}(n^2)$ time per flip. Algorithm 3 performs $\mathcal{O}(s|\mathcal{C}|n^2)$ calls for the function $S(\boldsymbol{c}, j, d)$, each with $\mathcal{O}(1)$ cost, and the initialization of the caches requires $\mathcal{O}(n^2 r)$ time. □

As we show in the experimental evaluation, both the shaking framework as well as the computational shortcuts are important algorithmic ingredients to address the challenging combinatorial task induced by the unsupervised multi-class extension of the maximum margin principle considered in this work.

4 Experiments

In the experiments we evaluate the accuracy of the proposed unsupervised multi-class regularized least-squares algorithm (UMC-RLS) on several real-world and synthetic data sets with various baseline methods.

4.1 Methods

As baseline methods we employ the cutting plane multi-class MMC method (CPMMC) [14], K-means clustering (more specifically, k-means++ [22]), Gaussian mixture models fitted using expectation maximization with full covariance structure (GMM) [23], and spectral clustering using a 10-nearest neighbors-graph as similarity graph (SC) [24]. Additionally, to demonstrate the importance of the shaking heuristic, we also provide results for simpler variants of the proposed method, where the combinatorial search over the cluster assignments is based either on pure stochastic search (Stoc-RLS) or on the direction of steepest descent without the shaking heuristic (Steep-RLS). As discussed above, the stochastic variant extends our previous implementation for the binary case [13] to the multi-class setting.

UMC-RLS, Steep-RLS, Stoc-RLS, and CPMMC are based on `Python` and the `Numpy` and `Scipy` libraries. Further, the CPMMC implementation uses the `CVXOPT` optimization library for solving the quadratic programs arising during training the method. The method was originally formulated only for the linear case, but as suggested by Zhao *et al.* [14], the method can be straightforwardly kernelized by running it on a feature representation generated by eigen decomposing the kernel matrix (see, e.g., [25] for a detailed discussion). The

Table 1. Comparison of the clustering methods. Mean ARI and the one standard deviation based on ten repetitions are provided.

data set	size	classes	SC	GMM	K-means	CPMMC	Stoc-RLS	Steep-RLS	UMC-RLS
Coil	1440	20	0.74 ± 0.01	0.55 ± 0.01	0.58 ± 0.02	-	0.31 ± 0.10	0.43 ± 0.07	$\mathbf{0.84 \pm 0.06}$
Coil 1-4	288	4	0.70 ± 0.00	0.50 ± 0.00	0.49 ± 0.00	0.47 ± 0.08	0.29 ± 0.11	$0.\mathbf{0}.09$	$\mathbf{1.00 \pm 0.00}$
Coil 5-8	288	4	0.74 ± 0.06	0.61 ± 0.03	0.60 ± 0.04	0.62 ± 0.05	0.35 ± 0.06	0.37 ± 0.10	$\mathbf{0.97 \pm 0.09}$
Iris	150	3	0.43 ± 0.00	$\mathbf{0.96 \pm 0.00}$	0.70 ± 0.09	0.69 ± 0.08	0.45 ± 0.25	0.56 ± 0.13	$\mathbf{0.96 \pm 0.00}$
Letter	500	4	0.38 ± 0.00	0.45 ± 0.00	0.43 ± 0.00	0.39 ± 0.10	0.19 ± 0.12	0.20 ± 0.12	$\mathbf{0.46 \pm 0.09}$
Moons	500	2	0.99 ± 0.00	0.78 ± 0.00	0.65 ± 0.00	0.69 ± 0.45	0.21 ± 0.35	0.07 ± 0.05	$\mathbf{1.00 \pm 0.00}$
USPS 1-4	500	4	0.66 ± 0.00	0.48 ± 0.00	0.61 ± 0.01	0.59 ± 0.19	0.38 ± 0.12	0.38 ± 0.08	$\mathbf{0.85 \pm 0.02}$
USPS 5-8	500	4	$\mathbf{0.93 \pm 0.00}$	0.64 ± 0.00	0.67 ± 0.00	0.57 ± 0.11	0.26 ± 0.12	0.34 ± 0.14	0.91 ± 0.02

implementation, which is based on the primal formulation of the optimization problem, does not scale to large problem sizes, in effect restricting our comparison to problems with few hundred training examples, and few clusters at most. Zhao *et al.* [14] note that one can derive a dual version of the method with much better scalability, but neither technical details nor a corresponding implementation are provided. The K-means, GMM and SC implementations are from the `scikit-learn` library available at `http://scikit-learn.org`.

4.2 Data Sets

We perform experiments on eight tasks that represent a wide variety of application domains (see Table 1). The Iris, Letter, and USPS data sets are standard benchmarks from the UCI repository. From Letter, we choose the first 4 classes in the data. We split the USPS into two tasks, USPS 1-4 and USPS 5-8, which both contain four classes from the original data. We use the full COIL image recognition data set [26], as well as two subsets, COIL 1-4 and COIL 5-8. Moons is a well-known artificial benchmark data set with a non-linear structure. The Letter and USPS data sets are sub-sampled so that they have at most 500 examples, in order to allow for running the experiments with CPMMC. For the full COIL data set we do not present results for CPMMC, as the implementation does not scale to the considered number of patterns and clusters.

4.3 Clustering Performance

We estimate the clustering performance of the compared methods using the *adjusted rand index* (ARI) [27]. After parameter selection, each method is run 10 times, with mean ARI of the repetitions being used to represent the final performance. All kernel methods employ a Gaussian kernel in our experiments.

Similarly to the setups of [14, 28], we choose the regularization parameter λ and kernel width σ for the RLS-based methods an CPMMC using grid search. In preliminary experiments we noticed that the methods tended to favor small regularization parameter values, therefore λ is chosen from grid $\{2^{-10}, 2^{-9} \dots 2^{-1}\}$. Kernel width σ is chosen from the grid $\{0.1\sigma_0, 0.2\sigma_0, \dots, \sigma_0\}$, where σ_0 is the maximum distance between two data points in the data set. For each tested parameter, the performance is computed as the mean over 10 repetitions of clustering with different random initializations. Following [14,28], we set the er-

ror tolerance parameters α and ϵ for CPMMC both to 0.01. The parameter l of CPMMC was set to 10, based on preliminary experiments.

Table 2. Comparison of the clustering methods. The maximum ARI out of 10 repetitions is provided.

data set	SC	GMM	K-means	CPMMC	Stoc-RLS	Steep-RLS	UMC-RLS
Coil	0.75	0.57	0.62	-	0.43	0.58	**1.00**
Coil 1-4	0.70	0.50	0.49	0.56	0.50	0.43	**1.00**
Coil 5-8	0.79	0.65	0.65	0.67	0.42	0.51	**1.00**
Iris	0.43	**0.96**	0.62	0.76	0.85	0.92	**0.96**
Letter	0.38	0.45	0.43	0.49	0.37	0.36	**0.57**
Moons	0.99	0.78	0.65	0.99	0.95	0.14	**1.0**
USPS 1-4	0.66	0.48	0.62	**0.92**	0.51	0.52	0.88
USPS 5-8	**0.93**	0.65	0.68	0.68	0.51	0.57	**0.93**

Table 1 presents the mean ARI with standard deviations for the considered methods and data, with the results for best performing methods highlighted in each row. On seven of the eight considered data sets, UMC-RLS either outperforms all the other methods, or shares the place of best performing method with one other baseline method. On the USPS 5-8 data, UMC-RLS is the second best performing method. SC and GMM are the most competitive baselines. SC outperforms the other methods on USPS 5-8 data, and is the only baseline method also able to solve the Moons problem. GMM performs as well as UMC-RLS on Iris, and almost as well on Letter. K-means and CPMMC are not competitive with UMC-RLS, but can still on some of the data sets outperform either SC or GMM. The mean clustering performances of Stoc-RLS and Steep-RLS are very poor. On most runs CPMMC, Stoc-RLS and Steep-RLS seem to get stuck in bad local minima. Thus, the shaking heuristic implemented by UMC-RLS proves to be crucial in order achieving stable and good performance.

Next, we compare the methods based on the maximum ARI achieved out of the final ten runs. The experiment allows us to estimate whether the performance of some of the considered methods could be significantly improved by using random restarts based metaheuristics. The results are presented in Table 2. In this setting, CPMMC becomes competitive with the SC and GMM methods, even outperforming all the methods on USPS1 data sets. Stoc-RLS and Steep-RLS results are also greatly improved compared to mean results. Still, even if we compare the maximum ARI out of 10 repetitions for the other methods (Table 2) to the mean ARI out of 10 repetitions for UMC-RLS (Table 1) UMC-RLS still appears to be the overall best performing method.

To conclude, the experimental results suggest that the proposed UMC-RLS method often achieves high clustering performance on real-world problems, and seems to represent currently the state-of-the art among multi-class MMC type of methods. Further, the results highlight the importance of the shaking heuristic, as otherwise the combinatorial search will typically not lead to a good clustering solution.

4.4 Runtime

Finally, we explore also experimentally the scalability of the proposed method. All the runtime experiments were performed on a computer with Intel i7-3770 3.40GHz processor, 16 GB of memory and 64-bit Ubuntu 13.04 operating system. The CPMMC is not considered in the comparison, since its implementation is highly suboptimal in terms of running time.

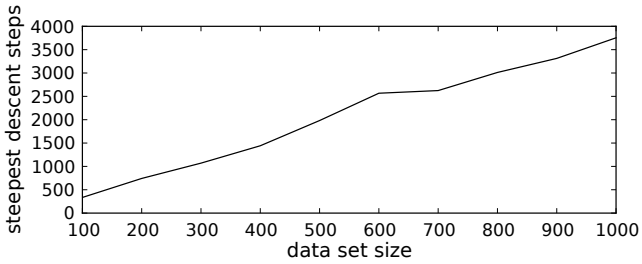

Fig. 3. Number of steepest descent steps required by UMC-RLS as a function of the data set size.

In Figure 3 we have plotted the number of

steepest descent steps executed while running UMC-RLS for varying sized random subsets of the COIL data set with 20 clusters. From the plot it can be seen that the number of steps required grows linearly in the size of the data set, as can be expected from the complexity considerations. Thus the experiment further verifies that the steepest descent search can be executed efficiently for the proposed method.

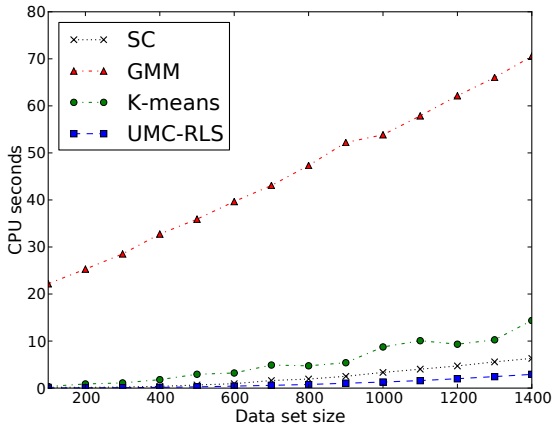

Fig. 4. CPU runtimes for the COIL data set.

In Figure 4 we have the CPU running times for the different methods on varying sized random subsets of the COIL data set. It can be observed that when dealing with such a moderately sized data set the UMC-RLS implementation actually outperforms the baseline methods in terms of computational speed. To evaluate the behaviour of the methods on larger data sets we further performed a scaling experiment on the MNIST handwritten character recognition data set. The dimensionality of the data is 784, and it contains 10 clusters representing different digits. As can be seen in Figure 5, once the sample size becomes several thousands, the higher computational complexity of UMC-RLS begins to dominate the running time of the method, the K-means and SC have better scalability. Still, even with 8000 data points UMC-RLS clustering can be run in a bit over five minutes on a modern desktop computer.

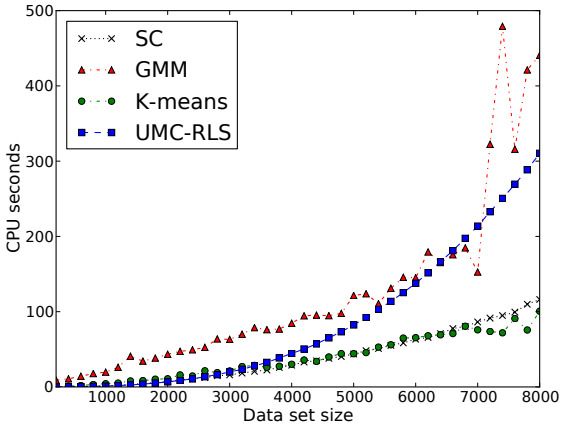

Fig. 5. CPU runtimes on MNIST data set.

The computational bottleneck for UMC-RLS remains the computation of the eigen decomposition of the kernel matrix needed during initialization. Here, kernel matrix approximation techniques could be of great benefit, in case one needs to scale the method to very large data sets.

5 Discussion and Future work

As shown by the experimental results, the proposed shaking heuristic provides considerably better results than the simple greedy approach relying on the steepest descent directions only. Still, the heuristic was the first non-trivial one we tested, and hence it is of very ad-hoc nature. We expect that far better heuristics can be produced if we can encode prior knowledge about the classification problem into it, as is often possible in practical problems. Heuristics could also be designed for other variations of unsupervised classification, such as learning with partial class memberships, for example [29].

The main computational bottleneck of the proposed algorithm is computing the eigen decomposition of the kernel matrix, whose time complexity is cubic with respect to data set

size in the worst case. For large data sets, a standard practise in kernel-based learning is to employ sparse kernel matrix approximation techniques, such as the well know Nyström method [30], which will decrease the complexity of performing the decomposition to linear time, usually without considerably harming the classification performance. Another bottleneck is due to the linear cost of a single steepest descent step, which causes the overall time complexity to become quadratic if the amount of steps also grows linearly with respect to data set size, as is usually the case with the shaking heuristic. To remedy this, we intend to develop such variations of the method and the heuristic that, instead of doing global steepest descent steps with linear cost, would employ the steps on small local subsets of the data at the time so that the step costs would scale with the subset sizes rather than the overall data set size. This would also reduce the memory size of the cache matrices required by the method (see Appendix), since the caches would have to be constructed for the subsets only and reconstructed when the subset would be changed. If the size of the subsets can be fixed to a small constant, the overall computational and memory complexities of the method will be linear in the overall data set size. This requires considerably more sophisticated heuristics that also take care of changing the local subsets when needed.

The performance of kernel-based learning methods depends considerably on the hyper-parameters values, such as those of the regularization and kernel parameters. However, tuning the values properly is very challenging in unsupervised learning. There exists several methods for measuring the cluster validity that do not depend on the class-labels of the data points but work in completely unsupervised fashion [31, 32]. In the future, we also intend to investigate the potential of these methods for setting the hyper-parameter values. The problem is not as severe in tasks like image segmentation, where the method is used in an interactive fashion so that the user can tune the parameter values by hand in order to get results that are satisfactory enough.

Yet another research direction not yet covered in this paper is the semi-supervised extension of the proposed method. Given a small set of data points with known class labels and a large set with unknown labels, the proposed ideas can be easily modified so that the known class labels are kept fixed, while the labels of the other points are switched with the meta-heuristic just like in the fully unsupervised case. The problem, of course, becomes considerably easier, since the classes can in most practical cases be assumed to be concentrated around the points with known labels and the method does not have to be initialized with a completely random labeling or with a simpler clustering algorithm. Moreover, instead of the shaking proposed here for the completely unsupervised learning, new meta-heuristics can be specially tailored for the semi-supervised case, for example, such that keep claiming unlabeled points to the classes one by one from the vicinity of the labeled points until all points have been labeled. With the help of the efficient computational short-cuts proposed in this paper for searching the steepest label switching directions, it becomes possible to develop and test several different types of new heuristics also for the semi-supervised problems.

6 Conclusion

In this work we proposed a multi-class extension of the binary maximum margin principle. Our framework is based on the least-squares variant of the original problem formulation, which has been experimentally proven to be a valuable candidate for such clustering settings, see, e.g., Zhang *et al.* [12] or Gieseke *et al.* [13]. So far, only little work has been done

related to the interesting extension of these schemes to the multi-class case though. This is the focus of the work at hand dealing with the unsupervised extension of the corresponding one-vs-all multi-class setting.

The key contributions provided in this work are (1) a carefully designed steepest descent strategy, and (2) its extremely efficient implementation. The former contribution is based on a new shaking strategy that effectively avoids getting trapped in bad local optima during early stages of the optimization process. The latter contribution is based on a series of non-trivial matrix-based update steps that take care of the intermediate optimization tasks induced by the global shaking steepest descent framework. The experimental evaluation takes into account a variety of real-world data sets, and the clustering accuracy of our approach is compared to the ones of state-of-the-art methods. The results demonstrate the superior performance of the proposed framework and, hence, indicates that the unsupervised regularized least-squares approach is a promising clustering variant, given that one addresses the induced combinatorial optimization task appropriately.

Appendix

In Section 3 we gave an intuitive description of the proposed search algorithm, and pretended a claim about its overall computational complexity. Here, we show in detail how the claimed complexity can be achieved. The consideration can be divided into the following three fundamental parts:

A. *Initialization of cache memories:* Before starting the actual search, certain cache memories have to be initialized that the subsequent parts will take advantage of.

B. *Computation of the steepest descent directions:* The proposed search algorithm computes these directions before deciding, for which data point the cluster label should be switched next.

C. *Update of the caches:* whenever a cluster label of a data point is switched, the cache memories have to be updated in order to maintain the ability to compute the steepest descent directions efficiently.

We go through these phases one by one before summarizing them in the proof of Theorem 1.

6.1 Initialization of Cache Memories

First, we reformulate the objective function of the regularized least-squares framework. Let $\boldsymbol{K} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{\mathrm{T}}$ be the eigen decomposition of the kernel matrix, let $\widetilde{\boldsymbol{\Lambda}} = (\boldsymbol{\Lambda} + \lambda\boldsymbol{I})^{-1}$, and let $F(\boldsymbol{y})$ be defined as in (7). We can prove the following:

Lemma 1.

$$F(\boldsymbol{y}) = 1 - \boldsymbol{y}^{\mathrm{T}}\boldsymbol{V}\boldsymbol{\Lambda}\widetilde{\boldsymbol{\Lambda}}\boldsymbol{V}^{\mathrm{T}}\boldsymbol{y}$$

Proof. Using standard linear algebra techniques, we obtain the following decomposition

$$\begin{aligned}
&F(\boldsymbol{y}) \\
&= (\boldsymbol{y} - \boldsymbol{K}\boldsymbol{G}\boldsymbol{y})^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{K}\boldsymbol{G}\boldsymbol{y}) + \lambda\boldsymbol{y}^{\mathrm{T}}\boldsymbol{G}\boldsymbol{K}\boldsymbol{G}\boldsymbol{y} \\
&= \boldsymbol{y}^{\mathrm{T}}\boldsymbol{V}\left(\boldsymbol{I} - 2\boldsymbol{\Lambda}\widetilde{\boldsymbol{\Lambda}} + \boldsymbol{\Lambda}^2\widetilde{\boldsymbol{\Lambda}}^2 + \lambda\boldsymbol{\Lambda}\widetilde{\boldsymbol{\Lambda}}^2\right)\boldsymbol{V}^{\mathrm{T}}\boldsymbol{y} \\
&= \boldsymbol{y}^{\mathrm{T}}\boldsymbol{V}\left(\boldsymbol{I} + (-2\boldsymbol{I} + \widetilde{\boldsymbol{\Lambda}}\boldsymbol{\Lambda} + \lambda\widetilde{\boldsymbol{\Lambda}})\boldsymbol{\Lambda}\widetilde{\boldsymbol{\Lambda}}\right)\boldsymbol{V}^{\mathrm{T}}\boldsymbol{y} \\
&= \boldsymbol{y}^{\mathrm{T}}\boldsymbol{V}\left(\boldsymbol{I} + (-2\boldsymbol{I} + \widetilde{\boldsymbol{\Lambda}}(\boldsymbol{\Lambda} + \lambda\boldsymbol{I}))\boldsymbol{\Lambda}\widetilde{\boldsymbol{\Lambda}}\right)\boldsymbol{V}^{\mathrm{T}}\boldsymbol{y} \\
&= \boldsymbol{y}^{\mathrm{T}}\boldsymbol{V}\left(\boldsymbol{I} + (-2\boldsymbol{I} + \boldsymbol{I})\boldsymbol{\Lambda}\widetilde{\boldsymbol{\Lambda}}\right)\boldsymbol{V}^{\mathrm{T}}\boldsymbol{y} \\
&= \boldsymbol{y}^{\mathrm{T}}\boldsymbol{V}\left(\boldsymbol{I} - \boldsymbol{\Lambda}\widetilde{\boldsymbol{\Lambda}}\right)\boldsymbol{V}^{\mathrm{T}}\boldsymbol{y} \\
&= 1 - \boldsymbol{y}^{\mathrm{T}}\boldsymbol{V}\boldsymbol{\Lambda}\widetilde{\boldsymbol{\Lambda}}\boldsymbol{V}^{\mathrm{T}}\boldsymbol{y}. \qquad (15)
\end{aligned}$$

□

Given the kernel matrix containing all pairwise kernel evaluations between the training data points, the computation of the *compact decomposition*, in which only the eigen vectors corresponding to the nonzero eigenvalues are computed, requires $\mathcal{O}(r^2 n)$ time, where r is the

rank of the kernel matrix. In the compact decomposition, the matrix $\boldsymbol{V} \in \mathbb{R}^{n \times r}$ contains the r eigenvectors and $\boldsymbol{\Lambda} \in \mathbb{R}^{r \times r}$ is a diagonal matrix containing the r nonzero eigenvalues. It is also worth pointing out that the eigen decomposition of the kernel matrix is often used to turn the kernel-based clustering setting into a linear one (as is done in the multi-class MMC experiments by, e.g., Zhao *et al.* [14]); it therefore forms a common computational bottleneck for the kernel based competitors considered in our experimental evaluation.

Assumption 1. *Assume that we are given the compact eigen decomposition of the kernel matrix* $\boldsymbol{K} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{\mathrm{T}}$*, where* $\boldsymbol{V} \in \mathbb{R}^{n \times r}$*,* $\boldsymbol{\Lambda} \in \mathbb{R}^{r \times r}$*, and* r *is the rank of the kernel matrix, as well as an initial vector of class labels* $\boldsymbol{c} \in \mathcal{C}^n$*. In the initialization phase, we prepare the following cache memories which are updated whenever the vector of class labels is changed:*

- *The* $n \times n$*-matrix*

$$\boldsymbol{R} = \boldsymbol{V}\boldsymbol{\Lambda}\widetilde{\boldsymbol{\Lambda}}\boldsymbol{V}^{\mathrm{T}}, \tag{16}$$

- *the vectors* $\boldsymbol{R}p_c(\boldsymbol{c}), \forall c \in \mathcal{C}$*,*
- *as well as the values* $Q(\boldsymbol{c})$ *and* $F(p_h(\boldsymbol{c})), \forall h \in \mathcal{C}$*.*

The computational complexity of the initialization phase is dominated by the computation of $\boldsymbol{R}$, which can be done in $\mathcal{O}(rn^2)$ time given the compact decomposition of the kernel matrix of rank r.

6.2 Computation of the Steepest Descent Directions

The next lemma concerns the efficient computation of $S(\boldsymbol{c}, j, d)$, given that certain intermediate results have already been computed and cached. Its proof also encompasses implementation details of the search algorithms that take advantage of the computational shortcuts.

Lemma 2. *Assume that we have cache memories given in Assumption 1 available. Then, the value of* $S(\boldsymbol{c}, j, d)$ *can be computed in a constant time.*

Proof. Let $\boldsymbol{y} \in \{-1, 1\}^n$ and let us denote $\hat{\boldsymbol{y}} = \boldsymbol{y} - 2y_j\boldsymbol{e}^j$, that is, $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$ are two ± 1-valued vectors differing from each other only by the jth entry. Moreover, we denote $\boldsymbol{t} = \boldsymbol{R}\boldsymbol{y}$. Then, continuing from (15), if we already know $F(\boldsymbol{y})$, $\boldsymbol{R}$, and $\boldsymbol{t}$, the value of F for $\hat{\boldsymbol{y}}$ can be computed from

$$\begin{aligned} F(\hat{\boldsymbol{y}}) &= n - \hat{\boldsymbol{y}}^{\mathrm{T}}\boldsymbol{R}\hat{\boldsymbol{y}} \\ &= n - \left(\boldsymbol{y} - 2y_j\boldsymbol{e}^j\right)^{\mathrm{T}}\boldsymbol{R}\left(\boldsymbol{y} - 2y_j\boldsymbol{e}^j\right) \\ &= n - \boldsymbol{y}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{y} + 4y_j\boldsymbol{y}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{e}^j - 4y_j^2{\boldsymbol{e}^j}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{e}^j \\ &= F(\boldsymbol{y}) + 4y_j{\boldsymbol{e}^j}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{y} - 4{\boldsymbol{e}^j}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{e}^j \\ &= F(\boldsymbol{y}) + 4y_jt_j - 4\boldsymbol{R}_{j,j}. \end{aligned}$$

Moreover, we observe that we can define a simple formula for calculating the difference in the objective values if a single entry of the ± 1-valued vector is flipped as follows:

$$D(\boldsymbol{y}, j) = F(\hat{\boldsymbol{y}}) - F(\boldsymbol{y}) = 4y_jt_j - 4\boldsymbol{R}_{j,j}. \tag{17}$$

Putting together (12), (13), and (17), $S(\boldsymbol{c}, j, d)$ can be formally written as

$$S(\boldsymbol{c}, j, d) = Q(\boldsymbol{c}) + D(p_{c_j}(\boldsymbol{c}), j) + D(p_d(\boldsymbol{c}), j).$$

The formula contains the objective value adjustments of both the old cluster c_j, and the new one d of the aggregate objective function (13) that allows the cluster assignments of a single data point to have only one positive entry. Thus, given the assumptions about cached intermediate results, we can calculate the objective value change caused by switching a single entry of the cluster label vector $\boldsymbol{c}$ in $\mathcal{O}(1)$ time. □

6.3 Updates of the Caches

After the steepest descent direction is found, and the cluster label of the corresponding patterns is switched, part of the cache memories given in Assumption 1 have to be updated

accordingly in order to maintain the ability to perform fast searches. As shown in the following lemma, the update operation does now slow down the computation of the steepest descent directions.

Lemma 3. *The cache memories given in Assumption 1 can be updated in $\mathcal{O}(n)$ time after the cluster label of a single pattern is switched.*

Proof. Given that $\boldsymbol{R}p_{c_j}(\boldsymbol{c})$ is stored in memory, the vector $\boldsymbol{R}(p_{c_j}(\boldsymbol{c}) - 2\boldsymbol{e}^j)$ can be obtained from

$$\boldsymbol{R}(p_{c_j}(\boldsymbol{c}) - 2\boldsymbol{e}^j) = \boldsymbol{R}p_{c_j}(\boldsymbol{c}) - 2(\boldsymbol{R}_j)^{\mathrm{T}}$$

in $\mathcal{O}(n)$ time. The vector $\boldsymbol{R}(p_d(\boldsymbol{c}) + 2\boldsymbol{e}^j)$ can be computed analogously. The values $Q(\boldsymbol{c})$, $F(p_{c_j}(\boldsymbol{c}))$, and $F(p_d(\boldsymbol{c}))$ are obtained in a constant time as implied by the proof of Theorem 2. The matrix $\boldsymbol{R}$ does not depend on $\boldsymbol{c}$ and, thus, does not have to be updated. □

6.4 Runtime Proof

Putting everything together, we arrive to the proof of Theorem 1.

Proof. [Proof of Theorem 1] As shown in Lemma 2, the evaluation of $S(\boldsymbol{c}, j, d)$ can be performed in constant time by taking advantage of the cache memories defined in Assumption 1. Since there are $|\mathcal{C}|$ clusters and n data points, finding the steepest descent direction requires $\mathcal{O}(|\mathcal{C}|n)$ time. Lemma 3 in turn shows that the cache memories can be updated according to the steepest descent direction in $\mathcal{O}(n)$, but this is dominated by the time required for finding the next steepest descent direction. Altogether, Algorithm 3 performs $\mathcal{O}(s|\mathcal{C}|n^2)$ calls for the function $S(\boldsymbol{c}, j, d)$, each with a constant time complexity, and the initialization of the caches requires $\mathcal{O}(n^2 r)$ time. The proof follows. □

Acknowledgments

We would like to thank the anonymous reviewers for their comments. Tapio Pahikkala gratefully acknowledges support from the Academy of Finland (grant 134020) and Fabian Gieseke from the German Academic Exchange Service (DAAD).

References

[1] Hastie T, Tibshirani R, Friedman J. *The Elements of Statistical Learning*. New York, NY, USA: Springer, 2nd ed. 2009.

[2] Bao T, Cao H, Chen E, Tian J, Xiong H. An unsupervised approach to modeling personalized contexts of mobile users. *Knowledge and Information Systems*. 2012;31(2):345–370.

[3] Jain A, Dubes R. *Algorithms for clustering data*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc. 1988.

[4] Schölkopf B, Smola A. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA, USA: MIT Press. 2001.

[5] Steinwart I, Christmann A. *Support Vector Machines*. New York, NY, USA: Springer-Verlag. 2008.

[6] Xu L, Neufeld J, Larson B, Schuurmans D. Maximum margin clustering. In: *Advances in Neural Information Processing Systems 17*, edited by Saul L, Weiss Y, Bottou L. MIT Press. 2005; pp. 1537–1544.

[7] Pahikkala T, Airola A, Gieseke F, Kramer O. Unsupervised Multi-Class Regularized Least-Squares Classification. In: *The 12th IEEE International Conference on Data Mining (ICDM 2012)*, edited by Zaki M, Siebes A, Yu J, Goethals B, Webb G, Wu X. Brussels, Belgium: IEEE Computer Society. 2012; pp. 585–594.

[8] Boyd S, Vandenberghe L. *Convex Optimization*. New York, NY, USA: Cambridge University Press. 2004.

[9] Valizadegan H, Jin R. Generalized Maximum Margin Clustering and Unsupervised Kernel Learning. In: *Advances in Neural Information Processing Systems 19*, edited by Schölkopf B, Platt J, Hoffman T. MIT Press. 2007; pp. 1417–1424.

[10] Zhao B, Wang F, Zhang C. Efficient Maximum Margin Clustering via Cutting Plane Algorithm. In: *Proceedings of the SIAM International Conference on Data Mining*. Atlanta, GA, USA: Society for Industrial and Applied Mathematics. 2008; pp. 751–762.

[11] Li Y, Tsang I, Kwok J, Zhou Z. Tighter and Convex Maximum Margin Clustering. In: *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics*, edited by van Dyk D, Welling M, vol. 5 of *JMLR: Workshop and Conference Proceedings*. Clearwater Beach, FL, USA: JMLR. 2009; pp. 344–351.

[12] Zhang K, Tsang I, Kwok J. Maximum margin clustering made practical. In: *Proceedings of the 24th International Conference on Machine Learning*, edited by Ghahramani Z, vol. 227 of *ACM International Conference Proceeding Series*. Corvallis, Oregon, USA: ACM. 2007; pp. 1119–1126.

[13] Gieseke F, Pahikkala T, Kramer O. Fast Evolutionary Maximum Margin Clustering. In: *Proceedings of the 26th International Conference on Machine Learning*, edited by Bottou L, Littman M, vol. 382 of *ACM International Conference Proceeding Series*. Montreal, QC, Canada: ACM. 2009; pp. 361–368.

[14] Zhao B, Wang F, Zhang C. Efficient Multiclass Maximum Margin Clustering. In: *Proceedings of the 25th international conference on Machine learning*, edited by Cohen W, McCallum A, Roweis S, vol. 307 of *ACM International Conference Proceeding Series*. Helsinki, Finland: ACM. 2008; pp. 1248–1255.

[15] Xu L, Schuurmans D. Unsupervised and Semi-Supervised Multi-Class Support Vector Machines. In: *Proceedings of the 20th national conference on Artificial intelligence*, edited by Veloso M, Kambhampati S. Pittsburgh, PA, USA: AAAI Press. 2005; pp. 904–910.

[16] Rifkin R, Klautau A. In Defense of One-Vs-All Classification. *Journal of Machine Learning Research*. 2004;5:101–141.

[17] Rifkin R, Yeo G, Poggio T. Regularized Least-Squares Classification. In: *Advances in Learning Theory: Methods, Models and Applications*, edited by Suykens J, Horvath G, Basu S, Micchelli C, Vandewalle J, vol. 190 of *NATO Science Series III: Computer and System Sciences*, chap. 7, pp. 131–154. Amsterdam, The Netherlands: IOS Press. 2003;.

[18] Kimeldorf G, Wahba G. Some results on Tchebycheffian spline functions. *Journal of Mathematical Analysis and Applications*. 1971;33(1):82–95.

[19] Girosi F, Jones M, Poggio T. Regularization Theory and Neural Networks Architectures. *Neural Computation*. 1995; 7(2):219–269.

[20] Russell S, Norvig P. *Artificial Intelligence: A Modern Approach*. Upper Saddle River, NJ, USA: Prentice Hall Press, 3rd ed. 2009.

[21] Kirkpatrick S, Gelatt C, Vecchi M. Optimization by Simulated Annealing. *Science*. 1983;220(4598):671–680.

[22] Arthur D, Vassilvitskii S. k-means++: the advantages of careful seeding. In: *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, edited by Bansal N, Pruhs K, Stein C. New Orleans, LA, USA: Society for Industrial and Applied Mathematics. 2007; pp. 1027–1035.

[23] Dempster A, Laird N, Rubin D. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*. 1977;39(1):1–38.

[24] Shi J, Malik J. Normalized Cuts and Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2000;22(8):888–905.

[25] Schölkopf B, Mika S, Burges C, Knirsch P, Müller KR, Rätsch G, Smola A. Input space versus feature space in kernel-based methods. *IEEE Transactions On Neural Networks*. 1999;10(5):1000–1017.

[26] Nene S, Nayar S, Murase H. Columbia Object Image Library (COIL-100). *Tech. Rep. CUCS-006-96*, Department of Computer Science, Columbia University, New York, NY, USA. 1996.

[27] Hubert L, Arabie P. Comparing Partitions. *Journal of Classification*. 1985; 2(1):193–218.

[28] Wang F, Zhao B, Zhang C. Linear Time Maximum Margin Clustering. *IEEE Transactions on Neural Networks*. 2010; 21(2):319–332.

[29] Waegeman W, Verwaeren J, Slabbinck B, De Baets B. Supervised learning algorithms for multi-class classification problems with partial class memberships. *Fuzzy Sets and Systems*. 2011;184(1):106–125.

[30] Williams C, Seeger M. Using the Nyström Method to Speed Up Kernel Machines. In: *Advances in Neural Information Processing Systems 13*, edited by Leen T, Dietterich T, Tresp V. MIT Press. 2001; pp. 682–688.

[31] Halkidi M, Batistakis Y, Vazirgiannis M. On Clustering Validation Techniques. *Journal of Intelligent Information Systems*. 2001;17(2-3):107–145.

[32] Zhao Q. Cluster validity in clustering methods. Ph.D. thesis, University of Eastern Finland. 2012.

Tapio Pahikkala currently acts as a Professor of Intelligent Systems in University of Turku, Finland. He received his PhD in computer science from University of Turku in 2008 and held an Academy of Finland postdoctoral position during 2010-2012. His research focuses on machine learning, pattern recognition, algorithmics, and computational intelligence. He has authored more than eighty peer reviewed scientific publications and served in program committees of numerous scientific conferences. He is a member of both ACM and IEEE, and has served as a member in several committees of IEEE societies.

Antti Airola received his M.Sc. degree in software engineering and the D.Sc. (Tech) degree in information and communication technology from the University of Turku, Finland in 2006 and 2011 respectively. He is currently a university teacher at the University of Turku. His research interests include both basic and applied research in machine learning, with focus on regularized kernel methods.

Fabian Gieseke received his Diploma degrees in mathematics and computer science from the University of Münster, Germany, and his PhD in computer science from the Carl von Ossietzky University Oldenburg, Germany. He is currently working as a postdoctoral researcher at the University of Copenhagen, Denmark. His research interests include support vector machines and their extensions to semi- and unsupervised learning settings as well as large-scale applications in astronomy.

Oliver Kramer is Juniorprofessor for Computational Intelligence at the University of Oldenburg in Germany. His main research interests are data mining, evolutionary optimization, and the application of computational intelligence techniques to renewable energy systems. He received a PhD in computer science from the University of Paderborn, Germany, in 2008. After a postdoc stay at the TU Dortmund, Germany, from 2007 to 2009, and the International Computer Science Institute in Berkeley (USA) in 2010, he became Juniorprofessor at the Bauhaus University Weimar, Germany. Since August 2011 he is affiliated to the Department of Computing Science at the University of Oldenburg.