# Unsupervised Multi-Class Regularized Least-Squares Classification

Tapio Pahikkala, Antti Airola
University of Turku
Turku Centre for Computer Science
Turku, Finland
Email: {tapio.pahikkala, antti.airola}@utu.fi

Fabian Gieseke, Oliver Kramer
Carl von Ossietzky Universität Oldenburg
Computer Science Department
Oldenburg, Germany
Email: {f.gieseke, oliver.kramer}@uni-oldenburg.de

***Abstract*—Regularized least-squares classification is one of the most promising alternatives to standard support vector machines, with the desirable property of closed-form solutions that can be obtained analytically, and efficiently. While the supervised, and mostly binary case has received tremendous attention in recent years, unsupervised multi-class settings have not yet been considered. In this work we present an efficient implementation for the unsupervised extension of the multi-class regularized least-squares classification framework, which is, to the best of the authors' knowledge, the first one in the literature addressing this task. The resulting kernel-based framework efficiently combines steepest descent strategies with powerful meta-heuristics for avoiding local minima. The computational efficiency of the overall approach is ensured through the application of matrix algebra shortcuts that render efficient updates of the intermediate candidate solutions possible. 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.**

***Index Terms*—Unsupervised Learning, Multi-Class Regularized Least-Squares Classification, Maximum Margin Clustering**

## I. 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, and many other fields [2]. 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) [3], [4], and related regularized learning schemes have been extended to unsupervised learning settings, in most cases under the name *maximum margin clustering* (MMC) [5]. 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 during the recent years.

Xu *et al.* [5] were among the first ones who formalized the extension of SVMs to unsupervised learning scenarios. Their optimization approach is based on reformulating the original combinatorial task as semidefinite programming problem [6], which can then be addressed via standard solvers. An extension of this framework is provided by Valizadegan and Jin [7]. Their approach is based on semidefinite programming. In contrast to Xu *et al.* [5], they show how to reduce the number of involved optimization variables, thus showing how to reduce the computational runtime. A recent local search approach for the linear case is given by Zhao *et al.* [8]. 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.* [9].

An alternative approach for the binary case is suggested by Zhang *et al.* [10]. Basically, their simple but surprisingly effective approach is based on iteratively applying a support vector machine model to improve kind of an "initial guess" that is obtained via an auxiliary clustering framework. One of the key ingredients of their framework, however, is the replacement of the original hinge loss by the $\varepsilon$-insensitive or the square loss. As pointed out by Zhang *et al.* [10], the resulting models *"can more easily get out of a poor solution"*. These ideas are extended by Gieseke *et al.* [11], 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.* [10], they resort to the square loss in this context.

**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.* [12]. While Xu *et al.* [13] 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* [14] to unsupervised learning settings. As reported by Zhang *et al.* [10] and Gieseke *et al.* [11], the square loss depicts a very reasonable choice in the context of such clustering settings and offers desirable computational shortcuts for optimization strategies that address the resulting combinatorial tasks. The particular 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 local minima with a suboptimal clustering performance. While being a seemingly simple modification, we demonstrate that this minor yet crucial adaptation provides major improvements to the clustering accuracy compared to straightforward stochastic search and steepest descent implementations.
2) Secondly, in line with the work of Gieseke *et al.* [11], 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.[1]

## II. Mathematical Background

In this section we provide the mathematical notations and the mathematical background related to the general concept of regularized kernel methods [3], [4], 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 $\{1,\ldots,n\}$ of natural numbers by $[n]$. Further, the set of all $n \times m$ matrices with real coefficients are denoted by $\mathbb{R}^{n\times m}$. Given a particular matrix $\mathbf{M} \in \mathbb{R}^{n\times m}$, we denote its element in the $i$-th row and $j$-th column by $\mathbf{M}_{i,j}$. For two index sets $R = \{i_1,\ldots,i_r\} \subseteq [n]$ and $S = \{k_1,\ldots,k_s\} \subseteq [m]$, we use $\mathbf{M}_{R,S}$ to denote the sub-matrix that only contains the rows and the columns of $\mathbf{M}$ that are indexed by $R$ and $S$. Finally, we use the shorthand $\mathbf{M}_{R,[m]} = \mathbf{M}_R$, and use $y_i$ to denote the $i$-th coordinate of a vector $\mathbf{y} \in \mathbb{R}^n$.

[1]It is worth pointing out that, 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 will be made available as part of the RLScore software library at http://staff.cs.utu.fi/~aatapa/software/RLScore/

### A. 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.

*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* [3], [4]. 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 $T = \{(\mathbf{x}_1, y_1),\ldots,(\mathbf{x}_n, y_n)\} \subset X \times Y$ with $Y = \{-1,+1\}$, the regularized risk minimization problem is defined as

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

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 $f$.

Two prominent representatives of this family of regularization methods are support vector machines and the concept of *regularized least-squares classification* [15]. The first one stems from the use of the *hinge loss* $L\big(y, f(\mathbf{x})\big) = \max(0, 1 - yf(\mathbf{x}))$, whereas the latter one is based on the *square loss* $L\big(y, f(\mathbf{x})\big) = \big(y - f(\mathbf{x})\big)^2$. By the representer theorem [16], any solution $f^* \in \mathcal{H}_k$ of the task (1) has the form

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

with appropriate coefficients $\mathbf{a} = (a_1,\ldots,a_n)^{\mathrm{T}} \in \mathbb{R}^n$. Hence, by plugging in the square loss into the objective and by using $||f^*||^2_{\mathcal{H}_k} = \mathbf{a}^{\mathrm{T}}\mathbf{K}\mathbf{a}$ [4] with kernel matrix $\mathbf{K} \in \mathbb{R}^{n\times n}$ consisting of entries $\mathbf{K}_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)$, we can rewrite the task at hand as

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

with

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

The objective $J(\mathbf{a})$ of the above optimization task is convex and differentiable with respect to $\mathbf{a} \in \mathbb{R}^n$. Thus a global minimizer can analytically be obtained by enforcing $\frac{\partial}{\partial \mathbf{a}} J(\mathbf{a}) = 0$. One can therefore obtain an optimal solution via

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

with

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

and where $\mathbf{I} \in \mathbb{R}^{n \times n}$ is the identity matrix [15]. 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 [4], [15]. As we will see below, this is especially the case in the context of the considered clustering scenarios.

*2) Unsupervised Least-Squares Extension:* As pointed out by Zhang *et al.* [10], 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 [11]. The direct extension of the supervised regularized kernel methods (1) to the unsupervised case for a given unlabeled training set $T = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset X$ has the form [5]:

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

Hence, the difficult part is the additional integer optimization variable $\mathbf{y} \in \{-1, +1\}^n$ 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

$$\left| \frac{1}{n} \sum_{i=1}^{n} \max(0, y_i) - b_c \right| < \varepsilon$$

with user-defined parameters $b_c \in [0, 1]$ and $\varepsilon \in \mathbb{R}^+$. By again considering the square loss and by substituting (5) back into (4), we obtain

$$\underset{\mathbf{y} \in \{-1,+1\}^n}{\operatorname{argmin}} \; F(\mathbf{y}) \tag{6}$$

with

$$F(\mathbf{y}) = (\mathbf{y} - \mathbf{KGy})^{\mathrm{T}}(\mathbf{y} - \mathbf{KGy}) + \lambda \mathbf{y}^{\mathrm{T}}\mathbf{GKGy} \tag{7}$$

as resulting optimization task for the case of the square loss.

## *B. Multi-Class Classification Scenarios*

We are now ready to 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.

*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 [14] the so-called one-versus-all multi-class classification settings 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 $T = \{(\mathbf{x}_1, c_1), \ldots, (\mathbf{x}_n, c_n)\} \subset X \times \mathcal{C}$ 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 $\mathbf{x} \in X$ is assigned to the class whose associated model is the most confident. A variety of different objectives (and loss functions) have been proposed in the literature [14]. In the following, we consider extensions of the supervised models (1) to the multi-class case having the form

$$\underset{f_1, \ldots, f_{|\mathcal{C}|} \in \mathcal{H}_k}{\operatorname{argmin}} \sum_{h=1}^{|\mathcal{C}|} \left( \sum_{i=1}^{n} L\big(c_i, f_h(\mathbf{x}_i)\big) + \lambda ||f_h||^2_{\mathcal{H}_k} \right),$$

where the loss function $L$ can be defined via a binary encoding of the class memberships: Let $\mathbf{c} \in \mathcal{C}^n$ be the vector containing the class labels of the training examples. Further, let

$$p_h(\mathbf{c}) = \begin{pmatrix} -1 \\ \vdots \\ -1 \end{pmatrix} + 2 \sum_{j=1}^{n} \delta_{c_j h} \mathbf{e}^j \in \{-1, +1\}^n \tag{8}$$

be a binary vector defined for each class $h \in \mathcal{C}$, where $\delta$ 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 $\mathbf{e}^j$ is the $j$-th standard basis vector of $\mathbb{R}^n$. Hence, the $i$-th component of vector $p_h(\mathbf{c})$ equals $+1$ in case the $i$-th training pattern belongs to the class $h$, and equals $-1$ otherwise. Based on these definitions, one can formulate the loss in the multi-class setting for the square loss as

$$\underset{f_1, \ldots, f_{|\mathcal{C}|} \in \mathcal{H}_k}{\operatorname{argmin}} \sum_{h=1}^{|\mathcal{C}|} \left( \sum_{i=1}^{n} \big(p_h(\mathbf{c})_i - f_h(\mathbf{x}_i)\big)^2 + \lambda ||f_h||^2_{\mathcal{H}_k} \right).$$

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. It can therefore be seen 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 [14].

*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 $\mathbf{c} \in \mathcal{C}^n$ as additional optimization variable. For the square loss, this leads to the following optimization task:

$$\underset{\substack{\mathbf{c} \in \mathcal{C}^n \\ f_1, \ldots, f_{|\mathcal{C}|} \in \mathcal{H}_k}}{\operatorname{argmin}} \sum_{h=1}^{|\mathcal{C}|} \left( \sum_{i=1}^{n} \big(p_h(\mathbf{c})_i - f_h(\mathbf{x}_i)\big)^2 + \lambda ||f_h||^2_{\mathcal{H}_k} \right).$$

Hence, one is again given a mixed-integer programming problem; the key problem is to find an appropriate assignment for the integer variable $\mathbf{c}$ 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 $\mathbf{c}$, since changing the class membership of a *single* training instance leads to the modification of *two* of the induced classification models $f_1, \ldots, f_{|\mathcal{C}|}$.

The optimization problem for the unsupervised extension of the one-vs-all multi-class framework can be re-written in the form

$$\underset{\mathbf{c} \in \mathcal{C}^n}{\operatorname{argmin}} \, Q(\mathbf{c}) \tag{9}$$

with

$$Q(\mathbf{c}) = \sum_{h=1}^{|\mathcal{C}|} F(p_h(\mathbf{c})), \tag{10}$$

where $F$ is defined via (7). Hence, the unsupervised multi-class extension can be considered as task of finding an appropriate local minimum for the objective function $Q(\mathbf{c})$. 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 constraints, along with computational shortcuts for assessing the quality of intermediate candidate solutions.

## III. 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 [17]).

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 $S(\mathbf{c}, j, d)$, where $\mathbf{c} \in \mathcal{C}^n$, $j \in \{1, \ldots, n\}$, and $d \in \mathcal{C}$, to denote the value of the objective function $Q$ defined in (10) for a cluster label vector, whose entries are equal to those of $\mathbf{c}$ except that the label of the $j$th 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 $\mathbf{c} \in \mathcal{C}^n$ for $n$ data points, we next consider the search algorithms for solving the clustering problems.

### A. 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 by Gieseke *et al.* [11]. In our experiments, we use a similar algorithm modified for multi-class clustering as a baseline method. The modification can take advantage of the computational shortcuts presented in Appendix, and it is therefore computationally as efficient as the other methods proposed in this paper.

**Algorithm 1** Stochastic Descent

```
 1: Initialize $\mathbf{c} \in \mathcal{C}^n$ randomly
 2: loop
 3:     $b \leftarrow$ True
 4:     for $j = 1, \ldots, n$ do
 5:         $d \leftarrow \mathrm{argmin}_{d \in \mathcal{C}}\, S(\mathbf{c}, j, d)$
 6:         if $c_j \neq d$ then
 7:             $c_j \leftarrow d$
 8:             $b \leftarrow$ False
 9:         end if
10:     end for
11:     if $b$ then                    ▷ Stop if local optimum found
12:         break
13:     end if
14: end loop
```

Fig. 1. Stochastic Descent

**Algorithm 2** Steepest Descent

```
1: Initialize $\mathbf{c} \in \mathcal{C}^n$ randomly
2: loop
3:     $j, d \leftarrow \mathrm{argmin}_{j \in \{1,\ldots,n\}, d \in \mathcal{C}}\, S(\mathbf{c}, j, d)$
4:     if $c_j = d$ then
5:         break
6:     else
7:         $c_j \leftarrow d$
8:     end if
9: end loop
```

Fig. 2. Steepest Descent

Another framework is the basic *steepest descent search* (see Algorithm 2) for the multi-class 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.

### B. 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).

**Algorithm 3** STEEPEST DESCENT WITH SHAKING

```
1: Initialize $\mathbf{c} \in \mathcal{C}^n$ randomly
2: for $i = 0, \ldots, s$ do
3:     for $d \in \mathcal{C}$ do
4:         $\alpha \leftarrow \frac{n}{2^i|\mathcal{C}|} + \frac{n}{|\mathcal{C}|} - |\{h \mid c_h = d\}|$
5:         for $j = 1, \ldots, \alpha$ do
6:             $j \leftarrow \text{argmin}_{j \in \{1,\ldots,n\}, d \neq c_j} S(\mathbf{c}, j, d)$
7:             $c_j \leftarrow d$
8:         end for
9:     end for
10: end for
```

Fig. 3. Steepest Descent with Shaking

*1) General Idea:* 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 $d$.

*2) Exchanging Class Labels:* The number

$$\alpha = \frac{n}{2^i|\mathcal{C}|} + \frac{n}{|\mathcal{C}|} - |\{h \mid c_h = d\}| \tag{11}$$

of points claimed by the clusters firstly depends on the round of the outer loop. During the first iterations, all clusters claim a large number of points, but the number decreases exponentially with respect to the iteration index of the loop. This dependence is encoded in the first term of (11) via the exponential factor in the nominator. 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 (11), which is positive for smaller than average size clusters and negative for the larger ones. Note that this definition of $\alpha$ 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.

*3) Shaking Meta-Heuristic:* 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 $s = 20$.

The behavior of the proposed method is demonstrated in Figure 4 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.

### *C. Efficient Optimization via Matrix-Based Updates*

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

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 sketch:* 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 $S(\mathbf{c}, j, d)$, 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.

## IV. 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 several baseline methods.

### *A. Methods*

As baselines we employ the cutting plane multi-class MMC method (CPMMC) [12], K-means clustering initialized with k-means++ [18], Gaussian mixture models fitted using expectation maximization with full covariance structure (GMM) [19], and spectral clustering using 10-nearest neighbors graph as similarity graph (SC) [20]. Additionally, to demonstrate the importance of the shaking heuristic, we also provide results

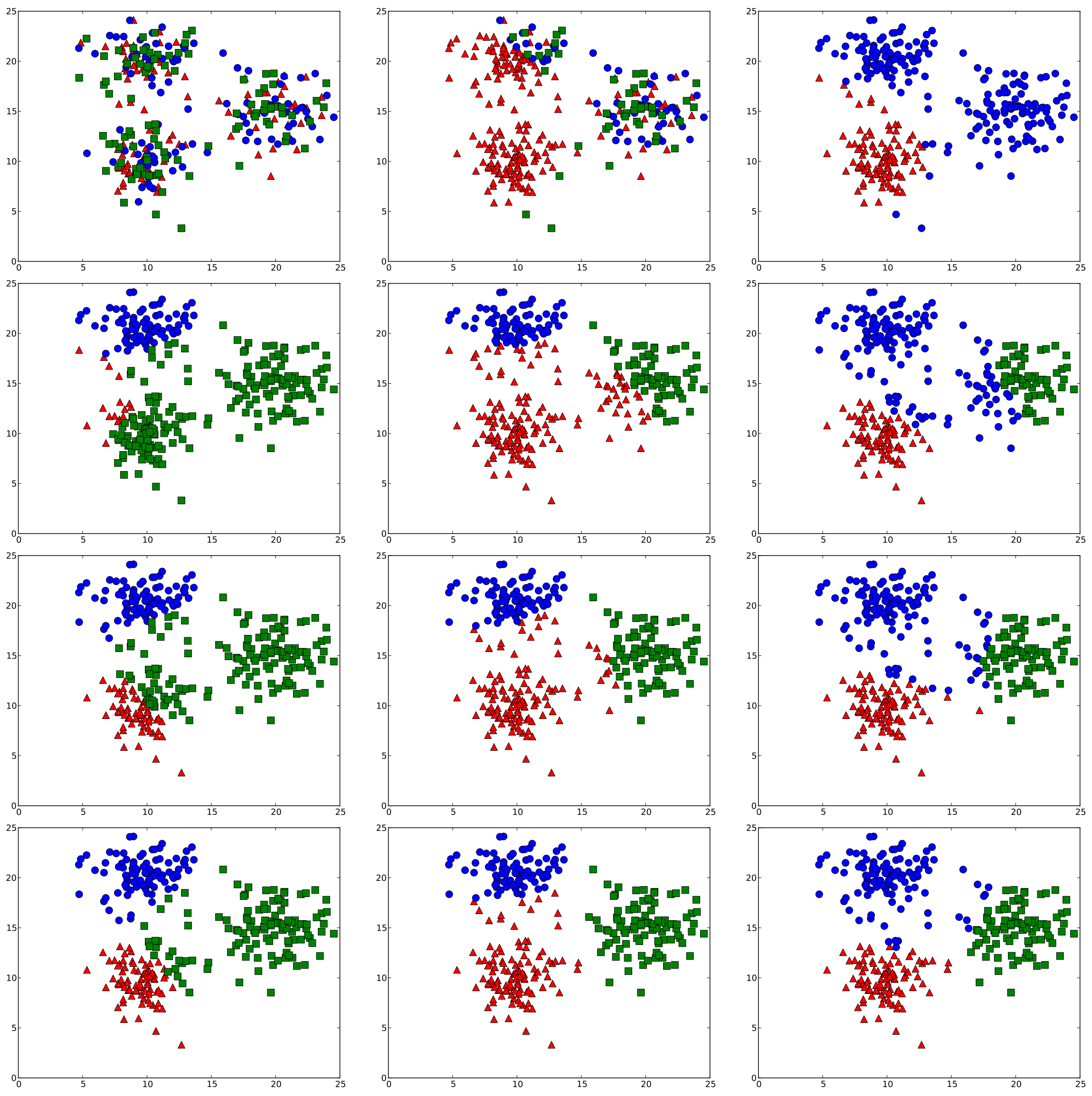

Fig. 4. Demonstration of the shaking meta-algorithm on three Gaussian clusters. The first image contains the initial random assignments of the cluster labels. The next three images correspond to the first round of the outermost loop of during which each class at a time claims a large number of data points. 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.

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 the binary MMC method of [11] to the multi-class setting.

UMC-RLS, Steep-RLS, Stoc-RLS, and CPMMC are implemented using the Python Numpy and Scipy libraries. Further, 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 [12], the method can be straightforwardly kernelized by running it on a feature representation generated by eigen decomposing the kernel matrix (see, e.g., [21] for a detailed discussion). The 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.* [12] 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 machine learning library[2].

## B. Data Sets

We perform experiments on eight tasks that represent a wide variety of application domains (see Table I). Iris, Letter and USPS are standard benchmarks from the UCI repository. From

[2]http://scikit-learn.org

TABLE I
COMPARISON OF THE CLUSTERING METHODS. MEAN ARI AND THE ONE STANDARD DEVIATION BASED ON TEN REPETITIONS ARE PROVIDED.

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

Letter, we choose the first 4 classes in the data. We split the USPS to two tasks, USPS 1-4 and USPS 5-8, which both contain 4 classes from the original data. We use the full COIL image recognition data set [22], as well as two subsets, COIL 1-4 and COIL 5-8. Moons is a well-known artificial benchmark data set with non-linear structure. Letter and USPS data sets are sub-sampled so that they have at most 500 examples, in order to allow for running the experiments for 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.

### C. *Clustering Performance*

We estimate the clustering performance of the compared methods using the Adjusted Rand Index (ARI) [23]. 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 [12], [24], we choose the regularization parameter $\lambda$ and kernel width $\sigma$ 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 $\lambda$ is chosen from grid $\{2^{-10}, 2^{-9} \dots 2^{-1}\}$. Kernel width $\sigma$ is chosen from the grid $\{0.1\sigma_0, 0.2\sigma_0, \dots, \sigma_0\}$, where $\sigma_0$ 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 [12], [24], we set the error tolerance parameters $\alpha$ and $\epsilon$ for CPMMC both to 0.01. The parameter $l$ of CPMMC was set to 10, based on preliminary experiments.

Table I 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.

TABLE II
COMPARISON OF THE CLUSTERING METHODS. THE MAXIMUM ARI OUT OF 10 REPETITIONS IS PROVIDED.

| dataset | 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** |

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 meta-heuristics. The results are presented in Table II. 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 II) to the mean ARI out of 10 repetitions for UMC-RLS (Table I) 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.

### D. *Runtime*

Finally, we explore also experimentally the scalability of the proposed method. In Figure 5 we have plotted the number of steepest descent steps executed while running UMC-RLS for varying sized random subsets of the COIL dataset 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. The computational bottleneck 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.

## V. 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 [25].

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 [26], which will decrease the complexity of performing the decomposition to linear, 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 [27], [28]. In the future, we also intend to investigate the potential of these methods for setting the hype-parameter values.

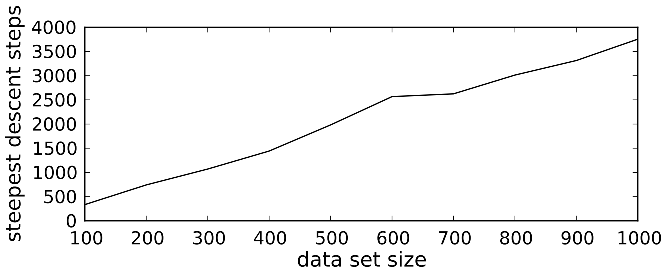

Fig. 5. Number of steepest descent steps required by UMC-RLS as a function of the data set size.

## VI. 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.* [10] or Gieseke *et al.* [11]. 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 III 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.

### A. Initialization of Cache Memories

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

**Lemma 1.**
$$F(\mathbf{y}) = 1 - \mathbf{y}^{\mathrm{T}}\mathbf{V}\mathbf{\Lambda}\widetilde{\mathbf{\Lambda}}\mathbf{V}^{\mathrm{T}}\mathbf{y}$$

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

$$\begin{aligned}
F(\mathbf{y}) &= (\mathbf{y} - \mathbf{KGy})^{\mathrm{T}}(\mathbf{y} - \mathbf{KGy}) + \lambda\mathbf{y}^{\mathrm{T}}\mathbf{GKGy} \\
&= \mathbf{y}^{\mathrm{T}}(\mathbf{I} - \mathbf{KG} - \mathbf{GK} + \mathbf{GKKG} + \lambda\mathbf{GKG})\,\mathbf{y} \\
&= \mathbf{y}^{\mathrm{T}}\mathbf{V}\left(\mathbf{I} - 2\mathbf{\Lambda}\widetilde{\mathbf{\Lambda}} + \mathbf{\Lambda}^2\widetilde{\mathbf{\Lambda}}^2 + \lambda\mathbf{\Lambda}\widetilde{\mathbf{\Lambda}}^2\right)\mathbf{V}^{\mathrm{T}}\mathbf{y} \\
&= \mathbf{y}^{\mathrm{T}}\mathbf{V}\left(\mathbf{I} + (-2\mathbf{I} + \widetilde{\mathbf{\Lambda}}\mathbf{\Lambda} + \lambda\widetilde{\mathbf{\Lambda}})\mathbf{\Lambda}\widetilde{\mathbf{\Lambda}}\right)\mathbf{V}^{\mathrm{T}}\mathbf{y} \\
&= \mathbf{y}^{\mathrm{T}}\mathbf{V}\left(\mathbf{I} + (-2\mathbf{I} + \widetilde{\mathbf{\Lambda}}(\mathbf{\Lambda} + \lambda\mathbf{I}))\mathbf{\Lambda}\widetilde{\mathbf{\Lambda}}\right)\mathbf{V}^{\mathrm{T}}\mathbf{y} \\
&= \mathbf{y}^{\mathrm{T}}\mathbf{V}\left(\mathbf{I} + (-2\mathbf{I} + \mathbf{I})\mathbf{\Lambda}\widetilde{\mathbf{\Lambda}}\right)\mathbf{V}^{\mathrm{T}}\mathbf{y} \\
&= \mathbf{y}^{\mathrm{T}}\mathbf{V}\left(\mathbf{I} - \mathbf{\Lambda}\widetilde{\mathbf{\Lambda}}\right)\mathbf{V}^{\mathrm{T}}\mathbf{y} \\
&= 1 - \mathbf{y}^{\mathrm{T}}\mathbf{V}\mathbf{\Lambda}\widetilde{\mathbf{\Lambda}}\mathbf{V}^{\mathrm{T}}\mathbf{y}.
\end{aligned}$$

■

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.[3] It is 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.* [12]); 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 $\mathbf{K} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{\mathrm{T}}$, where $\mathbf{V} \in \mathbb{R}^{n\times r}$, $\mathbf{\Lambda} \in \mathbb{R}^{r\times r}$, and $r$ is the rank of the kernel matrix, as well as an initial vector of class labels $\mathbf{c} \in \mathcal{C}^n$. 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*
$$\mathbf{R} = \mathbf{V}\mathbf{\Lambda}\widetilde{\mathbf{\Lambda}}\mathbf{V}^{\mathrm{T}}, \qquad (12)$$
- *the vectors $\mathbf{R}p_c(\mathbf{c}), \forall c \in \mathcal{C}$,*
- *as well as the values $Q(\mathbf{c})$ and $F(p_h(\mathbf{c})), \forall h \in \mathcal{C}$.*

[3]In the compact decomposition, the matrix $\mathbf{V} \in \mathbb{R}^{n\times r}$ contains the $r$ eigenvectors and $\mathbf{\Lambda} \in \mathbb{R}^{r\times r}$ is a diagonal matrix containing the $r$ nonzero eigenvalues.

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

### B. Computation of the Steepest Descent Directions

The next lemma concerns the efficient computation of $S(\mathbf{c}, j, d)$, 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 short-cuts.

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

*Proof:* Let $\mathbf{y} \in \{-1, 1\}^n$ and let us denote $\hat{\mathbf{y}} = \mathbf{y} - 2y_j\mathbf{e}^j$, that is, $\mathbf{y}$ and $\hat{\mathbf{y}}$ are two $\pm 1$-valued vectors differing from each other only by the $j$th entry. Moreover, we denote $\mathbf{t} = \mathbf{Ry}$ and $\overline{\{j\}} = \{1, \ldots, n\} \setminus \{j\}$. Then, continuing from (12), if we already know $F(\mathbf{y})$, $\mathbf{R}$, and $\mathbf{t}$, the value of $F$ for $\hat{\mathbf{y}}$ can be computed from

$$\begin{aligned}
&F(\hat{\mathbf{y}}) \\
&= 1 - \hat{\mathbf{y}}^{\mathrm{T}}\mathbf{R}\hat{\mathbf{y}} \\
&= 1 - \hat{y}_j\mathbf{R}_{j,j}\hat{y}_j - 2\hat{y}_j\mathbf{R}_{j,\overline{\{j\}}}\mathbf{y}_{\overline{\{j\}}} - \mathbf{y}^{\mathrm{T}}_{\overline{\{j\}}}\mathbf{R}_{\overline{\{j\}},\overline{\{j\}}}\mathbf{y}_{\overline{\{j\}}} \\
&= 1 - y_j\mathbf{R}_{j,j}y_j - 2\hat{y}_j\mathbf{R}_{j,\overline{\{j\}}}\mathbf{y}_{\overline{\{j\}}} - \mathbf{y}^{\mathrm{T}}_{\overline{\{j\}}}\mathbf{R}_{\overline{\{j\}},\overline{\{j\}}}\mathbf{y}_{\overline{\{j\}}} \\
&= 1 - \mathbf{y}^{\mathrm{T}}\mathbf{Ry} - 2(\hat{y}_j - y_j)^{\mathrm{T}}\mathbf{R}_{j,\overline{\{j\}}}\mathbf{y}_{\overline{\{j\}}} \\
&= F(\mathbf{y}) - 2(\hat{y}_j - y_j)^{\mathrm{T}}(t_j - \mathbf{R}_{j,j}y_j) \\
&= F(\mathbf{y}) + 4y_j(t_j - \mathbf{R}_{j,j}y_j) \\
&= F(\mathbf{y}) + 4y_jt_j - 4\mathbf{R}_{j,j}.
\end{aligned}$$

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

$$D(\mathbf{y}, j) = F(\hat{\mathbf{y}}) - F(\mathbf{y}) = 4y_jt_j - 4\mathbf{R}_{j,j}. \qquad (13)$$

Putting together (8), (10), and (13), $S(\mathbf{c}, j, d)$ can be formally written as

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

The formula contains the objective value adjustments of both the old cluster $c_j$, and the new one $d$ of the aggregate objective function (10) 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 $\mathbf{c}$ in $\mathcal{O}(1)$ time. ■

### C. 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 $\mathbf{R}p_{c_j}(\mathbf{c})$ is stored in memory, the vector $\mathbf{R}(p_{c_j}(\mathbf{c}) - 2\mathbf{e}^j)$ can be obtained from

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

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

## D. Runtime Proof

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

*Proof of Theorem 1:* As shown in Lemma 2, the evaluation of $S(\mathbf{c}, j, d)$ 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 $S(\mathbf{c}, j, d)$, each with a constant time complexity, and the initialization of the caches requires $\mathcal{O}(n^2 r)$ time. The proof follows. ∎

## Acknowledgment

This work has been supported in part by the Academy of Finland (Tapio Pahikkala, grant 134020) and by funds of the *Deutsche Forschungsgemeinschaft* (DFG) (Fabian Gieseke, grant KR 3695). The authors would like to thank the anonymous reviewers for valuable comments and suggestions on an early version of this work.

## References

[1] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, 2nd ed. Springer, 2009.

[2] A. Jain and R. Dubes, *Algorithms for clustering data*. Prentice Hall, 1988.

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

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

[5] L. Xu, J. Neufeld, B. Larson, and D. Schuurmans, "Maximum margin clustering," in *Advances in Neural Information Processing Systems 17*, L. K. Saul, Y. Weiss, and L. Bottou, Eds. MIT Press, 2005, pp. 1537–1544.

[6] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[7] H. Valizadegan and R. Jin, "Generalized maximum margin clustering and unsupervised kernel learning," in *Advances in Neural Information Processing Systems 19*, B. Schölkopf, J. C. Platt, and T. Hoffman, Eds. MIT Press, 2007, pp. 1417–1424.

[8] B. Zhao, F. Wang, and C. Zhang, "Efficient maximum margin clustering via cutting plane algorithm," in *Proceedings of the SIAM International Conference on Data Mining*. Society for Industrial and Applied Mathematics, 2008, pp. 751–762.

[9] Y.-F. Li, I. Tsang, J. Kwok, and Z.-H. Zhou, "Tighter and convex maximum margin clustering," in *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics*, ser. JMLR: Workshop and Conference Proceedings, D. van Dyk and M. Welling, Eds., vol. 5. JMLR, 2009, pp. 344–351.

[10] K. Zhang, I. Tsang, and J. Kwok, "Maximum margin clustering made practical," in *Proceedings of the 24th International Conference on Machine Learning*, ser. ACM International Conference Proceeding Series, Z. Ghahramani, Ed., vol. 227. ACM, 2007, pp. 1119–1126.

[11] F. Gieseke, T. Pahikkala, and O. Kramer, "Fast evolutionary maximum margin clustering," in *Proceedings of the 26th International Conference on Machine Learning*, ser. ACM International Conference Proceeding Series, L. Bottou and M. Littman, Eds., vol. 382. ACM, June 2009, pp. 361–368.

[12] B. Zhao, F. Wang, and C. Zhang, "Efficient multiclass maximum margin clustering," in *Proceedings of the 25th international conference on Machine learning*, ser. ACM International Conference Proceeding Series, W. W. Cohen, A. McCallum, and S. T. Roweis, Eds., vol. 307. ACM, 2008, pp. 1248–1255.

[13] L. Xu and D. Schuurmans, "Unsupervised and semi-supervised multiclass support vector machines," in *Proceedings of the 20th national conference on Artificial intelligence*, A. Cohn, Ed. AAAI Press, 2005, pp. 904–910.

[14] R. Rifkin and A. Klautau, "In defense of one-vs-all classification," *Journal of Machine Learning Research*, vol. 5, pp. 101–141, 2004.

[15] R. Rifkin, G. Yeo, and T. Poggio, "Regularized least-squares classification," in *Advances in Learning Theory: Methods, Models and Applications*, ser. NATO Science Series III: Computer and System Sciences, J. Suykens, G. Horvath, S. Basu, C. Micchelli, and J. Vandewalle, Eds. Amsterdam: IOS Press, 2003, vol. 190, ch. 7, pp. 131–154.

[16] G. Kimeldorf and G. Wahba, "Some results on Tchebycheffian spline functions," *Journal of Mathematical Analysis and Applications*, vol. 33, no. 1, pp. 82–95, 1971.

[17] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

[18] D. Arthur and S. Vassilvitskii, "k-means++: the advantages of careful seeding," in *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, N. Bansal, K. Pruhs, and C. Stein, Eds. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035.

[19] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society, Series B*, vol. 39, no. 1, pp. 1–38, 1977.

[20] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 888–905, Aug. 2000.

[21] B. Schölkopf, S. Mika, C. Burges, P. Knirsch, K.-R. Müller, G. Rätsch, and A. Smola, "Input space versus feature space in kernel-based methods," *IEEE Transactions On Neural Networks*, vol. 10, no. 5, pp. 1000–1017, 1999.

[22] S. Nene, S. Nayar, and H. Murase, "Columbia object image library (coil-100)," Department of Computer Science, Columbia University, Tech. Rep., 1996.

[23] L. Hubert and P. Arabie, "Comparing partitions," *Journal of Classification*, vol. 2, no. 1, pp. 193–218, 1985.

[24] F. Wang, B. Zhao, and C. Zhang, "Linear time maximum margin clustering," *IEEE Transactions on Neural Networks*, vol. 21, no. 2, pp. 319–332, 2010.

[25] W. Waegeman, J. Verwaeren, B. Slabbinck, and B. De Baets, "Supervised learning algorithms for multi-class classification problems with partial class memberships," *Fuzzy Sets and Systems*, vol. 184, no. 1, pp. 106–125, 2011.

[26] C. K. I. Williams and M. Seeger, "Using the nyström method to speed up kernel machines," in *Advances in Neural Information Processing Systems 13*, T. K. Leen, T. G. Dietterich, and V. Tresp, Eds. MIT Press, 2001, pp. 682–688.

[27] M. Halkidi, Y. Batistakis, and M. Vazirgiannis, "On clustering validation techniques," *J. Intell. Inf. Syst.*, vol. 17, no. 2-3, pp. 107–145, Dec. 2001.

[28] Q. Zhao, "Cluster validity in clustering methods," Ph.D. dissertation, University of Eastern Finland, 2012.