



DCSVM: fast multi-class classification using support vector machines

Duleep Rathgamage Don¹ · Ionut E. Iacob¹

Received: 8 November 2018 / Accepted: 10 July 2019 / Published online: 19 July 2019
© Springer-Verlag GmbH Germany, part of Springer Nature 2019

Abstract

Using binary classification techniques to perform multi-class classification of data is still of great practical interest due to the robustness and simplicity of binary classifiers. These techniques produce a single multi-class classification decision based on many binary decisions. Our work relies on the simple observation that as dimensionality increases so does the data sparsity and, consequently, a single binary classifier may separate multiple classes. Therefore, we claim that the number of binary decisions can be significantly reduced. We present Divide and Conquer Support Vector Machines (DCSVM), an efficient algorithm for multi-class classification using Support Vector Machines. DCSVM is a divide and conquer algorithm which relies on data sparsity in high dimensional space and performs a smart partitioning of the whole training data set into disjoint subsets that are easily separable. A single prediction performed between two partitions eliminates at once one or more classes in one partition, leaving only a reduced number of candidate classes for subsequent steps. The algorithm continues recursively, reducing the number of classes at each step, until a final binary decision is made between the last two classes left in the competition. In the best case scenario, our algorithm makes a final decision between k classes in $O(\log k)$ decision steps and in the worst case scenario DCSVM makes a final decision in $k-1$ steps, which is not worse than the existent techniques.

Keywords Multiclass classification · SVM · Divide and conquer

Mathematics Subject Classification 62H30 · 68T10

1 Introduction

Machine learning classification is the task of performing data categorization based on previously seen data categories (labels or classes). When only two classes are present it is called binary classification and for more than two classes it is called multiclass classification. Binary classification is inherently simpler to perform and decomposing a multiclass classification problem into a number of binary classification problems (called transformation to binary) has been a popular method for producing multiclass classification [12, 16]. The transformation to binary methods inherit the robustness and simplicity of the binary methods but suffer, in general, from producing many pairs of binary classifiers and typically

using a large number of binary decisions for producing a single multiclass classification decision. In this article we propose a new method of performing multiclass classification using transformation to binary with a significantly reduced number of binary decisions. Our method relies on the key observation that in high dimensional space the sparsity of the data may increase significantly. As a consequence, a single binary classifier may accurately perform classification for more than the two classes the classifier was designed for. We show how to detect such practical situations and efficiently exploit them to perform multiclass classification with less binary decision.

According to Wikipedia [29], the curse of dimensionality refers to various phenomena that arise when analyzing and organizing data in high-dimensional spaces (often with hundreds or thousands of dimensions) that do not occur in low-dimensional settings such as the three-dimensional physical space. The expression was introduced by Richard E. Bellman in a highly acclaimed article considering problems in dynamic optimization [2, 3]. In essence, as dimensionality increases, the volume of the space increases rapidly, and the

✉ Ionut E. Iacob
ieiacob@GeorgiaSouthern.edu

Duleep Rathgamage Don
dr04108@GeorgiaSouthern.edu

¹ Department of Mathematical Sciences, Georgia Southern University, Statesboro, USA

available data become sparser and sparser. In general, this sparsity is problematic for any method that requires statistical significance. In order to obtain a statistically sound and reliable result, the amount of data needed to support the result often grows exponentially with the dimensionality, which would prevent common data processing techniques from being efficient.

Since its introduction, the Support Vector Machines (SVM) [7] has quickly become a popular tool for classification which has attracted a lot of interest in the machine learning community. However, SVM is primarily a binary classification tool. The multiclass classification with SVM is still an ongoing research problem (see, for example, [4, 24, 25, 30] for some recent work and [14, 20] for real-life applications). We present an SVM-based multi-class classification method that exploits the curse of dimensionality to efficiently perform classification of highly dimensional data.

Our method, the Divide and Conquer SVM (DCSVM) algorithm, exploits the curse of dimensionality and the power of SVM binary classifiers to produce faster multiclass classifications. The algorithm's idea is described next using the example in Fig. 1. The figure shows 6 classes (1—red, 2—blue, 3—green, 4—black, 5—orange, and 6—maroon) of two-dimensional points and a linear SVM separation of classes 1 and 2 (the line that separates the points in these classes). It happens that the SVM model for classifying classes 1 and 2 completely separates the points in classes 4 (which takes class 2's side) and 6 (which takes class 1's side). Moreover, the classifier does a relatively good job classifying most points of the class 5 as class 2

(with relatively few points classified as 1) and a poor job on classifying the points of class 3 (as the points in this class are classified about half as 1's and the other half as 2's). With DCSVM we use the SVM classifier for classes 1 and 2 for a candidate of an unknown class: if the classifier predicts 1, then we next decide between classes 1, 6, 3, and 5; if the classifier predicts class 2, then we next decide between classes 2, 4, 3, and 5. Notice that in either case one or more classes are eliminated, and we are left to predict a fewer number of classes. That is, a multi-class classification problem of a smaller size (less classes). The algorithm then proceeds recursively on the smaller problem. In the best case scenario at each step half of the k classes will be eliminated and the algorithm will finish in $\lceil \log k \rceil$ steps. Notice that, in the above scenario, classes 2 and 4 are completely separated from classes 1 and 6, whereas classes 3 and 5 are not clearly on one side or the other of the separation line. For this reason, classes 3 and 5 are part of the next decision step, regardless of the prediction of the first classifier.

However, there is a significant difference between classes 3 and 5. While class 3 is almost divided in half by the separation line, class 5 can be predicted as “2” with a relatively small error. In DCSVM we use a threshold value θ to indicate the maximum classification error accepted in order to consider a class on one side or the other of a separation line. For instance, let us consider that only 2% of the points of class 5 are on the same side as class 1. With the threshold value set to 0.02, DCSVM will separate classes 1, 3, and 6 (when 1 is predicted) from classes 2, 3, 4, and 5 (when 2 is predicted). A higher threshold value will produce a better separation of classes (less overlapping) and less classes to process in the subsequent steps. This comes at the price of possibly sacrificing the accuracy of the final prediction.

Clearly, the method presented in the example above is suitable for multiclass classification using a binary classifier, in general. Our choice of using SVM is based on the SVM algorithm's remarkable power in producing accurate binary classification. Our main contribution is producing a multiclass classification decision faster than the previously proposed methods, based on performing significantly less binary decisions. We do so without sacrificing the accuracy of the previously proposed methods. However, the accuracy of each binary decision and how fast a binary decision is produced depend solely on the performance of the binary classifier being used. Of course, using binary classifiers for multiclass classification has the obvious disadvantage of having to create a large number of classifiers ($k(k-1)/2$ for k classes), this process is highly parallelizable. A major challenge for our method was deciding which classifiers to use and in which order to achieve best results. For this purpose, we introduce some measures (Definitions 3 and 4) in Sect. 3.1 and explain how to use these measures to achieve our goals.

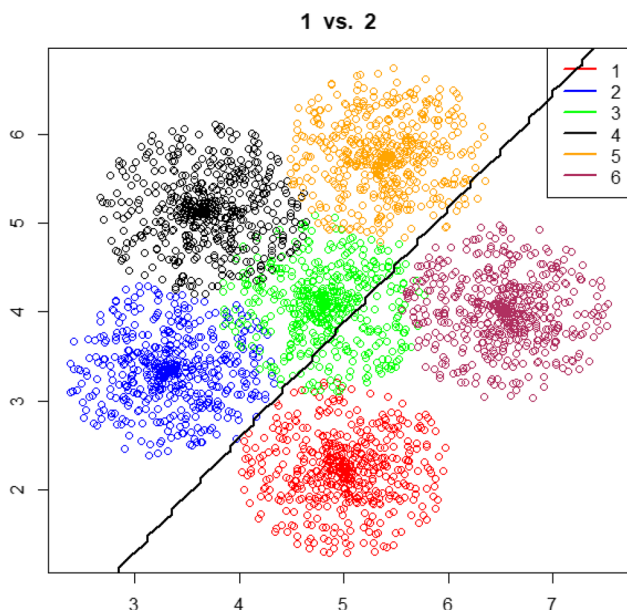


Fig. 1 Binary SVM classifier for classes 1 and 2 out of a dataset of six classes

The content of this article is organized as follows. We give a brief description of binary classification with SVM and related work on using SVM for multi-class classification in Sect. 2. DCSVM is described in detail in Sect. 3 and experimental results (including performance comparisons with one-versus-one approach) are given in Sect. 4. We conclude in Sect. 5.

2 Preliminaries and related work

Support Vector Machines (SVM) [7] was primary developed as a tool for the binary classification problem by finding a separation hyperplane for the classes in feature space. If such a plane cannot be find, the “separating plane” requirement is softened and a maximal margin separation is produced instead. Formally, the problem of finding a maximal margin separation can be stated as a quadratic optimization problem. Given a set of L training vectors $\mathbf{x}_i \in \mathbb{R}^d, i = 1 \dots L$ with labels $y_i \in \{-1, 1\}$ and a feature space projection $\Phi : \mathbb{R}^d \rightarrow H$, the SVM method consists in finding the solution of the following:

$$\begin{aligned} \min_{\mathbf{w} \in H, b \in \mathbb{R}, \xi_i \in \mathbb{R}} \quad & \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{i=1}^L \xi_i \\ \text{subject to} \quad & y_i (\langle \mathbf{w}, \Phi(\mathbf{x}_i) \rangle + b) \geq 1 - \xi_i \\ & \xi_i \geq 0, i = 1 \dots L \end{aligned}$$

where $\mathbf{w} \in H$ is the weights vector and $C \in \mathbb{R}^+$ is a cost regularization constant. The corresponding decision function is:

$$f(\mathbf{x}) = \text{sign}(\langle \mathbf{w}, \Phi(\mathbf{x}) \rangle + b)$$

An interesting property of the method is that the dot product can be represented by a kernel function:

$$k(\mathbf{x}, \mathbf{x}') = \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle$$

which is computationally much less expensive than actually projecting \mathbf{x} and \mathbf{x}' into the feature space H .

In the case of multiple classes, the problem formulation becomes more complicated and inherently more difficult to address. Given a set of L training vectors $\mathbf{x}_i \in \mathbb{R}^d, i = 1 \dots L$ with labels $y_i \in \{1, \dots, k\}$, one must find a way to distinguish between k classes.

Several approaches were proposed, which can be grouped into direct methods (a single optimization problem formulation for multi-class classification) and indirect methods (using multiple binary classifiers to produce multi-class classification). Many of the indirect methods were introduced, in fact, as methods for multi-class classification using binary classifiers, in general. They are not limited to the SVM method.

A comparison [16] of these methods of multi-class classification using binary SVM classifiers shows that

one-versus-one method and its Directed Acyclic Graph improvement are more suitable for practical use.

2.1 Direct formulation of multi-class classification

Direct formulations to distinguish between k classes in a single optimization problem were given in [6, 8, 27, 28] or, more recently, in [15, 30]. Each of these formulations has a single objective function for training all k -binary SVMs simultaneously and maximize the margins from each class to the remaining ones. The decision function then chooses the “best classified” class.

For instance, Crammer et al. in [8] solve the following optimization problem for k classes:

$$\begin{aligned} \min_{\mathbf{w}_m \in H, \xi_i \in \mathbb{R}} \quad & \frac{1}{2} \sum_{m=1}^k \mathbf{w}_m^T \mathbf{w}_m + C \sum_{i=1}^L \xi_i \\ \text{subject to} \quad & \langle \mathbf{w}_{y_i}, \Phi(\mathbf{x}_i) \rangle - \langle \mathbf{w}_t, \Phi(\mathbf{x}_i) \rangle \geq 1 - \delta_{y_i,t} - \xi_i \\ & \xi_i \geq 0, i = 1 \dots L, t = 1 \dots k \end{aligned} \tag{1}$$

where $\delta_{i,j}$ is the Kronecker delta function. The corresponding decision function is:

$$\text{argmax}_m f_m(\mathbf{x}) = \text{argmax}_m \langle \mathbf{w}_m, \Phi(\mathbf{x}) \rangle$$

The original formulation addresses the classification without taking into account the bias terms b_i (for each of the k classes). These can be easily included in the formulation using additional constraints (see, for instance, [16]). Crammer’s formulation is among the most compact optimization problem formulations for multi-class classification problem.

A common issue of the single optimization problem formulations for multi-class classification is the large number of variables involved. For instance, (1), although a compact formulation, includes $L \times k$ variables (not taking into account b_i ’s, if included), which yields large computation complexity. In [15], Crammer’s formulation is extended by relaxing its constraints and subsequently solving a single l -variable quadratic programming problem for multi-class classification.

2.2 One-versus-rest approach

The one-versus-rest approach [5, 26, 27] is an indirect method relying on binary classifiers as follows. For each class $t \in \{1, \dots, k\}$ a binary classifier f_t is created between class t (as positive examples in the training set) and all the other classes, $\{1, \dots, t - 1, t + 1, \dots, k\}$ (all as negative examples in the training set). The corresponding decision function is then:

$$f(\mathbf{x}) = \text{argmax}_{1 \leq t \leq k} f_t(\mathbf{x})$$

That is, the class label is determined by the binary classifier that gives maximum output value (the winner among all

classifiers). A well-known shortcoming of the one-versus-rest approach is the highly imbalanced training set for each binary classifier (the more classes, the bigger the imbalance). Assuming equal number of training examples for all classes, the ratio of positive to negative examples for each binary classifier is $1/(k-1)$. The symmetry of the original problem is lost and the classification results may be dramatically affected (especially for sparse classes).

2.3 One-versus-one approach

The one-versus-one approach ([11, 18, 19, 22] or the improvement by Platt et al. [23]) aims to remove the imbalance problem of the one-versus-rest method by training binary classifiers strictly with data in the two classifier's classes. For each pair of classes, $s, t \in \{1, \dots, k\}$ a binary classifier $f_{s,t}$ is created. This classifier is trained using all data in class s as positive examples and all data in class t as negative examples, hence all balanced binary classifiers. Each binary classifier is the result of a smaller optimization problem, at the cost of producing $k(k-1)/2$ classifiers. The corresponding decision function is based on majority voting. All classifiers $f_{s,t}$ are used on an input data item \mathbf{x} and each class appears in exactly $k-1$ classifiers, hence an opportunity for up to $k-1$ votes out of the $k(k-1)/2$ binary classification rounds. The class with the majority of votes is the winner.

An improvement on the number of voting rounds was originally proposed by Platt et al. in [23] and more recently also used in [1, 17]. Their method, called Directed Acyclic Graph SVM (DAGSVM), forms a decision-graph structure for the testing phase and it takes exactly $k-1$ individual voting rounds to decide the label of a data item \mathbf{x} . In a nutshell, DAGSVM uses one binary classifier at the time and subsequently removes the losing class from all subsequent classifications. There is no particular criterion on the order of using each binary classifier in this process.

3 Divide and conquer SVM (DCSVM)

As noted in the introduction and illustrated in Fig. 1, the key idea is that any binary classifier may, in practice, separate more than two classes. Which raises a natural question: which classes are separated (and with what accuracy) by each binary classifier? DCSVM combines the one-versus-one method's simplicity of producing balanced, fast binary classifiers with the classification speed of the DAGSVM's decision graph. The essential difference consists of producing the most efficient decision tree capable of delivering the decision in at most $k-1$ steps in the worst case scenario, or $O(\log k)$ steps in the best case scenario.

3.1 DCSVM training

Let us introduce some notations and then we will proceed to the formal description of the algorithm. Given a data set D of k classes (labels) where to each data item $\mathbf{x} \in D$ has been assigned a label $l \in \{1, \dots, k\}$, we want to construct a decision function $dcsvm : D \rightarrow \{1, \dots, k\}$ so that $dcsvm(\mathbf{x}) = l$, where l is the corresponding label of $\mathbf{x} \in D$. As usual, by considering a split $D = R \cup T$ of the data set D into two disjoint sets R (the training set) and T (the test set), we will be using the data in R to construct our decision function $dcsvm()$ and then the data in T to measure its accuracy. Furthermore, we consider $R = R_1 \cup R_2 \cup \dots \cup R_k$ as an union of disjoint sets R_l , where each $x \in R_l$ has label $l, l = 1, \dots, k$. (Similarly, we consider $T = T_1 \cup T_2 \cup \dots \cup T_k$ as a union of disjoint sets T_l , where each $x \in T_l$ has label $l, l = 1, \dots, k$.)

Let $svm_{i,j} : D \rightarrow \{i, j\}$, be a SVM binary classifier created using the training set $R_i \cup R_j$, $i < j$ and $i = 1, \dots, k-1, j = 2, \dots, k$. There are $k(k-1)/2$ such one-versus-one binary classifiers. We must clearly specify that the $svm()$ decision function we consider here is not the ideal one, but the practical one, likely affected by misclassification errors. That is, for some $\mathbf{x} \in R_i \cup T_i$, we may have that $svm_{i,j}(\mathbf{x}) = j$.

Our goal is to create the $dcsvm()$ decision function that uses a minimal number of binary decisions for k -classes classification, while not sacrificing the classification accuracy. While seeking to achieve this goal, we were exploring metrics from Decision Trees classification (such as Gini impurity, Information gain, etc). However, our situation is different. In most cases, DCSVM decision trees do not separate a set of classes into disjoint subsets. Some classes may appear in both subtrees rooted at the same decision node. We hence tried to identify other metrics that would help reaching our goal.

We start by defining a few measures we use in the process of identifying the shortest path to a multi-class classification decision.

Definition 1 (Class Predictions Likelihoods) The class predictions likelihoods of a SVM binary classifier $svm_{i,j}(\cdot)$ for a label $l \in \{1, \dots, k\}$, denoted respectively as $C_{i,j}(l, i)$ and $C_{i,j}(l, j)$, are:

$$C_{i,j}(l, i) = \frac{|\{\mathbf{x} \in R_l \mid svm_{i,j}(\mathbf{x}) = i\}|}{|R_l|}$$

$$C_{i,j}(l, j) = \frac{|\{\mathbf{x} \in R_l \mid svm_{i,j}(\mathbf{x}) = j\}|}{|R_l|} = 1 - C_{i,j}(l, i)$$

Each class prediction likelihood represents the expected outcome likelihood for i or j when a binary classifier $svm_{i,j}(\cdot)$ is used for prediction on all data items in R_l . These

likelihoods are computed for each binary classifier and each class in the training data set.

All pairs of likelihood predictions, for every binary classifier $svm_{i,j}(\cdot)$ and classes are stored in a table, as follows.

Definition 2 (All-Predictions Table) We arrange all classes predictions likelihoods in rows (corresponding to each binary classifier $svm_{i,j}$) and columns (corresponding to each class $1, \dots, k$) to form a table \mathcal{T} where each entry is given by a pair of predictions likelihoods as follows:

$$\mathcal{T}[svm_{i,j}, l] = (C_{i,j}(l, i), C_{i,j}(l, j))$$

Figure 2 shows the *All-Predictions Table* computed for the *glass* data set in [9]. The data set contains 6 classes, labeled as 1, 2, 3, 5, 6, and 7 (notice that label 4 is skipped in the original *glass* data set). Each row corresponds to a binary classifier $svm_{1,2}, \dots, svm_{6,7}$ and the columns correspond to the class labels. Each table cell contains a pair of likelihood predictions (as percentages) for the row classifier and class column. For instance, $C_{1,2}(1, 1) = 100\%$, $C_{1,2}(1, 2) = 0\%$ and $C_{1,6}(2, 1) = 91.8\%$, $C_{1,6}(2, 6) = 8.2\%$.

The table gives also a general overview of how good the binary classifiers are for separating other classes. As in the example described in Sect. 1, by setting precision goal (called *precision threshold* and denoted by θ) one can enforce each binary classifier to perform with some accuracy, if possible. For instance, choosing $\theta = 5\%$ would require a precision of at least $1 - \theta = 95\%$, which in turn would allow “svm 1 vs. 7” (row 5 in Fig. 2) separate only classes 1, 5, and 7. For $\theta = 10\%$, the same binary classifier would separate classes 1, 3, 5, and 7. We must emphasize that while θ is clearly related to the multiclass classification accuracy, it does not indicate a guaranteed accuracy overall. θ can be considered an expected accuracy only for

some classes: namely for those that can be decided with that accuracy by some binary classifiers.

We formally define next two measures for the quality of the classification of each $svm_{i,j}(\cdot)$. The impurity index measures how good the binary classifier is for classifying all classes as i or j for a given precision threshold θ . In a nutshell, a class l is classified as “definitely” i by $svm_{i,j}(\cdot)$ if $C_{i,j}(l, i) \geq 1 - \theta$; as “definitely” j if $C_{i,j}(l, j) \geq 1 - \theta$; otherwise, it is classified as “undecided” i or j . The impurity index counts how many “undecided” decisions a binary classifier produces. The lower the index, the better the separation. The balance index measures how “balanced” a separation is in terms of the number of classes predicted as i and j . The larger the index, the better.

Definition 3 (SVM Impurity and Balance Indexes) For an accuracy threshold θ of a SVM classifier $svm_{i,j}(\cdot)$, we define:

- the impurity index, denoted as $\mathcal{P}_{i,j}(\theta)$, as:

$$\mathcal{P}_{i,j}(\theta) = \left(\sum_{l=1}^k (\chi_{\theta}(C_{i,j}(l, i)) + \chi_{\theta}(C_{i,j}(l, j))) \right) - k$$

- the balance index, denoted as $\mathcal{B}_{i,j}(\theta)$, as:

$$\mathcal{B}_{i,j}(\theta) = \min \left(k - \sum_{l=1}^k \chi_{\theta}(C_{i,j}(l, j)), k - \sum_{l=1}^k \chi_{\theta}(C_{i,j}(l, i)) \right)$$

where χ_{θ} is the step function:

$$\chi_{\theta}(x) = \begin{cases} 1 & \text{if } x > \theta \\ 0 & \text{if } x \leq \theta \end{cases}$$

For instance, the impurity index for row $svm_{1,6}$ and threshold $\theta = 0.05$ in Fig. 2 is:

	svm	1	2	3	5	6	7
1	1 vs. 2	1=100%; 2=0%	1=0%; 2=100%	1=0%; 2=100%	1=0%; 2=100%	1=0%; 2=100%	1=0%; 2=100%
2	1 vs. 3	1=100%; 3=0%	1=59.02%; 3=40.98%	1=0%; 3=100%	1=58.33%; 3=41.67%	1=14.29%; 3=85.71%	1=9.09%; 3=90.91%
3	1 vs. 5	1=100%; 5=0%	1=86.89%; 5=13.11%	1=100%; 5=0%	1=0%; 5=100%	1=42.86%; 5=57.14%	1=4.55%; 5=95.45%
4	1 vs. 6	1=100%; 6=0%	1=91.8%; 6=8.2%	1=100%; 6=0%	1=50%; 6=50%	1=0%; 6=100%	1=22.73%; 6=77.27%
5	1 vs. 7	1=100%; 7=0%	1=86.89%; 7=13.11%	1=93.33%; 7=6.67%	1=0%; 7=100%	1=14.29%; 7=85.71%	1=0%; 7=100%
6	2 vs. 3	2=100%; 3=0%	2=100%; 3=0%	2=0%; 3=100%	2=91.67%; 3=8.33%	2=71.43%; 3=28.57%	2=90.91%; 3=9.09%
7	2 vs. 5	2=100%; 5=0%	2=100%; 5=0%	2=100%; 5=0%	2=0%; 5=100%	2=28.57%; 5=71.43%	2=0%; 5=100%
8	2 vs. 6	2=100%; 6=0%	2=100%; 6=0%	2=66.67%; 6=33.33%	2=91.67%; 6=8.33%	2=0%; 6=100%	2=22.73%; 6=77.27%
9	2 vs. 7	2=100%; 7=0%	2=100%; 7=0%	2=100%; 7=0%	2=91.67%; 7=8.33%	2=28.57%; 7=71.43%	2=0%; 7=100%
10	3 vs. 5	3=40.74%; 5=59.26%	3=93.44%; 5=6.56%	3=100%; 5=0%	3=0%; 5=100%	3=0%; 5=100%	3=0%; 5=100%
11	3 vs. 6	3=100%; 6=0%	3=100%; 6=0%	3=100%; 6=0%	3=50%; 6=50%	3=0%; 6=100%	3=100%; 6=0%
12	3 vs. 7	3=100%; 7=0%	3=96.72%; 7=3.28%	3=100%; 7=0%	3=33.33%; 7=66.67%	3=28.57%; 7=71.43%	3=0%; 7=100%
13	5 vs. 6	5=100%; 6=0%	5=100%; 6=0%	5=100%; 6=0%	5=100%; 6=0%	5=0%; 6=100%	5=0%; 6=100%
14	5 vs. 7	5=100%; 7=0%	5=100%; 7=0%	5=100%; 7=0%	5=100%; 7=0%	5=0%; 7=100%	5=0%; 7=100%
15	6 vs. 7	6=83.33%; 7=16.67%	6=91.8%; 7=8.2%	6=86.67%; 7=13.33%	6=66.67%; 7=33.33%	6=100%; 7=0%	6=0%; 7=100%

Fig. 2 The *All-Predictions* table for the *glass* data set in [9]

$$\mathcal{P}_{1,6}(0.05) = ((1 + 0) + (1 + 1) + (1 + 0) + (1 + 1) + (0 + 1) + (1 + 1)) - 6 = 3$$

and indicates that 3 of the classes (namely 2, 5, and 7) are undecided when the required precision is at least $\theta = 5\%$.

For accuracy threshold $\theta = 0.05$, the balance index for row $svm_{1,2}$ in Fig. 2 is $\mathcal{B}_{1,2} = 1$ and for row $svm_{5,6}$ is $\mathcal{B}_{5,6} = 2$.

The SVM score, defined next, is a measure of the precision of the binary classifier $svm_{i,j}(\cdot)$ for classifying classes i and j . The higher the score the better the classifier precision.

Definition 4 (SVM Score) The score of a SVM classifier $svm_{i,j}(\cdot)$, denoted as $\mathcal{S}_{i,j}$, is

$$\mathcal{S}_{i,j} = \frac{\mathcal{C}_{i,j}(i, i) + \mathcal{C}_{i,j}(j, j)}{2}$$

For instance, the table in Fig. 2 shows that

$$\mathcal{S}_{1,2} = \frac{\mathcal{C}_{1,2}(1, 1) + \mathcal{C}_{1,2}(2, 2)}{2} = \frac{100\% + 100\%}{2} = 100\%.$$

Algorithm 1 describes the DCSVM training and proceeds as follows. In the main procedure, TRAINDCSVM, the SVM binary classifiers for all class pairs are trained (line 4) and the predictions likelihoods are stored in the predictions table (line 5). The decision function $dcsvm$ is created as an empty tree (line 8) and then recursively populated in the DCSVM-SUBTREE procedure (line 10). The recursion procedure creates a left and/or a right node at each step (lines 12 and 19, respectively) or may stop with creating a left and/or a right label (lines 9 and 16, respectively). Each new node is associated to the binary $svm_{i,j}$ that is the decider at that node (line 5), or with a class label if an end node (lines 9 and 16).

An important part of the DCSVM-SUBTREE procedure is choosing the “optimal” svm from a current predictions likelihoods table (line 4). For this purpose, we use the SVM Impurity Index, Balance, and Score from Definitions 3 and 4, respectively. The order these measures are used may influence the decision tree shape and precision.

Algorithm 1 DCSVM training

```

1: procedure TRAINDCSVM( $R_1, \dots, R_k, \theta$ ) ▷ Creates DCSVM classifier
2: Input:  $R = R_1, \dots, R_k$ : data set,  $\theta$ : accuracy threshold
3: Output:  $dcsvm()$ 
4:    $svm_{i,j} \leftarrow$  train SVM with  $R_i \cup R_j$ ,  $i = 1, \dots, k - 1$ ,  $j = 2, \dots, k$ ,  $i < j$ 
5:    $\mathcal{T}[svm_{i,j}, l] = (\mathcal{C}_{i,j}(l, i), \mathcal{C}_{i,j}(l, j))$ , for all  $svm_{i,j}$ ,  $l = 1, \dots, k$ 
6:   //Recursively construct a binary decision tree
7:   //with each node associated with a  $svm_{i,j}$  binary classifier
8:    $dcsvm \leftarrow$  empty binary tree
9:    $dcsvm.root \leftarrow$  new tree node
10:  DCSVM-SUBTREE( $dcsvm.root, \mathcal{T}, \theta$ )
11:  return  $dcsvm$  ▷ Returns the decision tree
12: end procedure
13: procedure DCSVM-SUBTREE( $pnode, \mathcal{T}, \theta$ ) ▷ Creates subtree routed at  $pnode$ 
14: Input:  $pnode$ : current parent node,  $\mathcal{T}$ : current predictions table,  $\theta$ : accuracy threshold
15: Output: recursively constructs sub-tree rooted at  $pnode$ 
16:    $svm_{i,j} \leftarrow$  optimal  $svm$  in  $\mathcal{T}$ , for given  $\theta$ 
17:    $pnode[svm] \leftarrow svm_{i,j}$ 
18:    $list_i \leftarrow$  classes labeled as  $i$  or undecided by  $svm_{i,j}$ 
19:    $list_j \leftarrow$  classes labeled as  $j$  or undecided by  $svm_{i,j}$ 
20:   if  $length(list_i) = 1$  then ▷ reached a leaf
21:      $pnode.leftnode \leftarrow$  tree-node(label in  $list_i$ )
22:   else
23:      $\mathcal{T}.left \leftarrow \mathcal{T}$  minus  $svm_{m,n}$ ,  $m \in list_j$  or  $n \in list_j$  rows, and columns of classes
     not in  $list_i$ 
24:      $pnode.left \leftarrow$  new tree node
25:     DCSVM-SUBTREE( $pnode.left, \mathcal{T}.left, \theta$ )
26:   end if
27:   if  $length(list_j) = 1$  then ▷ reached a leaf
28:      $pnode.rightnode \leftarrow$  tree-node(label in  $list_j$ )
29:   else
30:      $\mathcal{T}.right \leftarrow \mathcal{T}$  minus  $svm_{m,n}$ ,  $m \in list_i$  or  $n \in list_i$  rows, and columns of classes
     not in  $list_j$ 
31:      $pnode.right \leftarrow$  new tree node
32:     DCSVM-SUBTREE( $pnode.right, \mathcal{T}.right, \theta$ )
33:   end if
34: end procedure

```

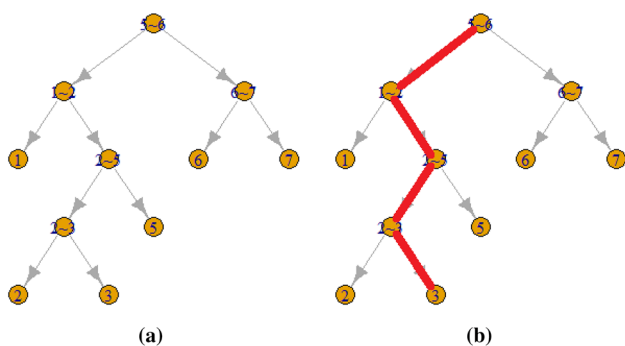


Fig. 3 DCSVM decision tree (a) with a decision process example (b), for the *glass* data set

If Score is used then the Impurity and Balance Indexes are used to break a tie, the resulting tree favors accuracy over the speed of decisions (may yield bushier trees). If Impurity and Balance Indexes are used first, then Score, if a tie, the resulting tree may be more balanced. The decision speed is favored while possibly sacrificing accuracy.

A *dcsvm* decision tree for the *glass* data set is shown in Fig. 3. Clearly, the algorithm may produce highly unbalanced *dcsvm* decision trees (when some classes are decided faster than others) or very balanced decision trees (when most of class labels are leaves situated at about same depth). Regardless of outcome, the following result is almost immediate.

Proposition 1 (Worst case scenario) The *dcsvm* decision tree constructed in Algorithm 1 has depth at most $k-1$.

Proof The lists of classes labeled i and j (lines 6, 7 in DCSVM-SUBTREE procedure) contain at least one label each: i or j , respectively. Once a class column is removed from \mathcal{T} at some tree node n , it will not appear again in a node or leaf

in the subtree rooted at that node n . Hence with each recursion the number of classes decreases by at least one (lines 11, 18) from k to 2, ending the recursion with a left or a right label node in lines 9 or 16, respectively.

Notice that a scenario where each *dcsvm* decision tree label has depth k is possible in practice: when no $svm_{i,j}$ binary classifier is a good separator for classes other than i and j (and therefore at each node only classes i and j are separated, while the other are undecided and will appear in both left and right branches). We call this the worst case scenario, for obvious reasons. The opposite case scenario is also possible in practice: each $svm_{i,j}$ separates all classes into two disjoint lists of about same lengths. The *dcsvm* decision tree is also very balanced in this case, but a lot smaller.

Proposition 2 (Best case scenario) The *dcsvm* decision tree constructed in Algorithm 1 when each $svm_{i,j}$ produced balanced, disjoint separation between all classes has depth $\lceil \log k \rceil$.

Proof Clearly, this is a case scenario where at each recursion step a node is created such that half of the classes are assigned to the left subtree and the other half to the right subtree. This produces a balanced binary tree (not necessarily complete) with k leaves. The tree has 1, then at each level the number of nodes doubles: 1, 2, 4, 8, etc. At the maximum depth h , there are exactly 2^h leaves (nodes) if the tree is complete. Since our tree is balanced (one level must be complete before moving to the next) but not necessary complete:

$$2^{h-1} < k \leq 2^h \Rightarrow h - 1 < \log k \leq h \Rightarrow \lceil \log k \rceil = h$$

Hence the depth is $\lceil \log k \rceil$.

Algorithm 2 DCSVM classifier

```

1: procedure DCSVMCLASSIFY(dcsvm, x)                                ▷ Produces DCSVM classification
2: Input: dcsvm: decision tree; x: data item
3: Output: Label of data item x
4:   node ← dcsvm.root
5:   while node not a leaf do                                         ▷ Visits the decision tree nodes towards a leaf
6:     svmi,j ← node[svm]                                           ▷ Retrieves the svm associated to current node
7:     label ← svmi,j(x)
8:     if label =  $i$  then
9:       node ← node.left
10:    else
11:      node ← node.right
12:    end if
13:  end while
14:  return label of node                                             ▷ Returns the leaf label
15: end procedure
    
```

The DCSVM classifier Algorithm 2 relies on the *dcsvm* decision tree produced by Algorithm 1 to take any data item \mathbf{x} and predict its label. The algorithm starts at the decision tree root node (line 4) then each node’s associated *svm* predicts the path to follow (lines 6–12) until a leaf node is reached. The label of the leaf node is the DCSVM’s prediction (line 14) for the input data item \mathbf{x} . An example of a prediction path in a *dcsvm* tree is illustrated in Fig. 3b.

Propositions 1 and 2 directly justify the following result.

Theorem 1 The Algorithm 2 performs multi-class classification of any data item \mathbf{x} in at most $k-1$ binary decisions steps (in the worst case scenario) and at most $\lceil \log k \rceil$ binary decision steps (in the best case scenario).

We illustrate next how the *dcsvm* decision tree is created and how a prediction is computed using a working example.

3.2 A working example

We use the *glass* data set [9] to illustrate DCSVM at work. This data set contains 6 classes, labeled 1, 2, 3, 5, 6, and 7 (notice there is no label 4). Consequently, $6 * (6 - 1) / 2 = 15$ binary *svm* classifiers are created and then the “all predictions likelihoods” table \mathcal{T} is computed (Fig. 2). Let us choose the accuracy threshold $\theta = 0$, for simplicity. That is, a class l is classified by an *svm* _{i,j} as only i if *svm* _{i,j} predicts that all data items in R_l have class i ; l is classified as only j if *svm* _{i,j} predicts that all data items in R_l have class j ; else, l is undecided and will appear on both sides of the decision tree node associated with *svm* _{i,j} .

We have chosen the “glass” data set as the working examples mostly based on its size: not trivially small, not artificial, and not too big to have a visual representation of the method so that the reader can follow. The “glass” data set is not necessary the best case scenario (Fig. 3 shows a decision tree that is not perfectly balanced) but also not the worst case scenario. We consider it has the advantage of showing both sides.

We follow next the DCSVM-SUBTREE procedure in Algorithm 1 and construct the *dcsvm* decision tree. Notice that $S_{i,j} = 100\%$ for all *svm* _{i,j} , so score does not matter for choosing the optimal *svm* _{i,j} in line 4. The choice will be solely based on the impurity and balance indexes. Table 1 shows all values for these measures for the initial predictions likelihoods table. The table shows rows 1, 13, and 14 as candidates with minimum impurity indexes. Then a tie between rows 13 and 14 as the winners among these. Row 13 comes first and hence *svm*_{5,6} is selected as the root node. Figure 3a shows the full decision tree, with *svm*_{5,6} as the root node. Subsequently, *svm*_{5,6} labels classes 1, 2, 3, and 5 as

Table 1 *svm* optimality measures for *glass* data set and the initial *All-Predictions* table

	<i>svm</i> _{i,j}	$\mathcal{P}_{i,j}(0)$	$\mathcal{B}_{i,j}(0)$	$\mathcal{S}_{i,j}(\%)$
1.	<i>svm</i> _{1,2}	0	1	100
2.	<i>svm</i> _{1,3}	4	1	100
3.	<i>svm</i> _{1,5}	3	1	100
4.	<i>svm</i> _{1,6}	3	1	100
5.	<i>svm</i> _{1,7}	3	1	100
6.	<i>svm</i> _{2,3}	3	1	100
7.	<i>svm</i> _{2,5}	1	2	100
8.	<i>svm</i> _{2,6}	3	1	100
9.	<i>svm</i> _{2,7}	2	1	100
10.	<i>svm</i> _{3,5}	2	1	100
11.	<i>svm</i> _{3,6}	1	1	100
12.	<i>svm</i> _{3,7}	3	1	100
13.	<i>svm</i> _{5,6}	0	2	100
14.	<i>svm</i> _{5,7}	0	2	100
15.	<i>svm</i> _{6,7}	4	1	100

“5” (left), and classes 6 and 7 as “6” (right). The algorithm continues recursively with classes {1, 2, 3, 5} to the left, and classes {6, 7} to the right. The right branch will be completed immediately with one more tree node (for *svm*_{6,7}) and two corresponding leaf nodes (for labels 6 and 7).

For the left branch the algorithm will proceed with a reduced *All-Predictions* table: rows 4, 5, 8, 9, 11, 12, 13, 14, and 15 and columns for classes 6 and 7 are removed. The optimality measures will be subsequently computed for all *svm* and classes still in competition (1, 2, 3, and 5) in the left branch. The corresponding measures are given in Table 2 (for an easier identification, the indices in the first column are kept the same as the original indices in the *All-Predict* table in Fig. 2). There is a tie between *svm*_{1,2} and *svm*_{2,5}, and *svm*_{1,2} is being used first. A node is consequently created, with a leaf as a left child. The rest of the tree is subsequently created in the same manner.

Table 2 Optimality measures in the second step of creating the decision tree in Fig. 3b

	<i>svm</i> _{i,j}	$\mathcal{P}_{i,j}(0)$	$\mathcal{B}_{i,j}(0)$	$\mathcal{S}_{i,j}(\%)$
1.	<i>svm</i> _{1,2}	0	1	100
2.	<i>svm</i> _{1,3}	2	1	100
3.	<i>svm</i> _{1,5}	1	1	100
6.	<i>svm</i> _{2,3}	1	1	100
7.	<i>svm</i> _{2,5}	0	1	100
10.	<i>svm</i> _{3,5}	2	1	100

4 Experimental results

We implemented DCSVM in R v3.4.3 using the *e1071* library [10], running on Windows 10, 64-bit Intel Core i7 CPU @3.40GHz, 16GB RAM. For testing, we used 14 data sets from the UCI repository [9] (as listed in Table 3). We performed three sets of experiments: (1) multi-class prediction accuracy comparison, (2) prediction performance in terms of speed (time and number of binary decisions) and resources (number of support vectors), and (3) DCSVM performance comparisons for different data sets and accuracy threshold parameter values. For the first set of experiments we compared three multi-class predictors: the built-in multi-class SVM (from the *e1071* library), our R implementation of one-versus-one, and the R implementation of DCSVM. For a fair comparison, in the second set of experiments we compared only the R implementations of one-versus-one and DCSVM. The built-in multi-class SVM would benefit of the inherent speed of native code it relies on. Finally, the third set of experiments focused on the DCSVM’s R implementation performance and fine tuning.

4.1 Accuracies comparison: built-in multi-class SVM, one-versus-one, and DCSVM

The main goal of DCSVM is to improve multi-class prediction performance while not sacrificing the prediction accuracy. The first experimental results compare multi-class prediction accuracy of: (1) built-in SVM multi-class prediction (in the *e1071* package), (2) one-versus-one implementation in R, and (3) DCSVM implementation in R. For the experiment, we used cross-validation with 80% data for training and 20% for testing, for each data set. We ran 10 trials and

averaged the results. The results are displayed in Fig. 4 and show no significant differences between the three methods.

Next, we performed the Wilcoxon signed rank test on sequences of accuracies from each trial and compared accuracies of DCSVM vs. built-in SVM (BI), and of DCSVM vs. one-versus-one implementation in R (OvsO). The p-value of the results, for each data set, are presented in Table 4. The results in Table 4 show that there is no compelling evidence to conclude the median of the accuracies of the DCSVM trials differ from the median of one-versus-one

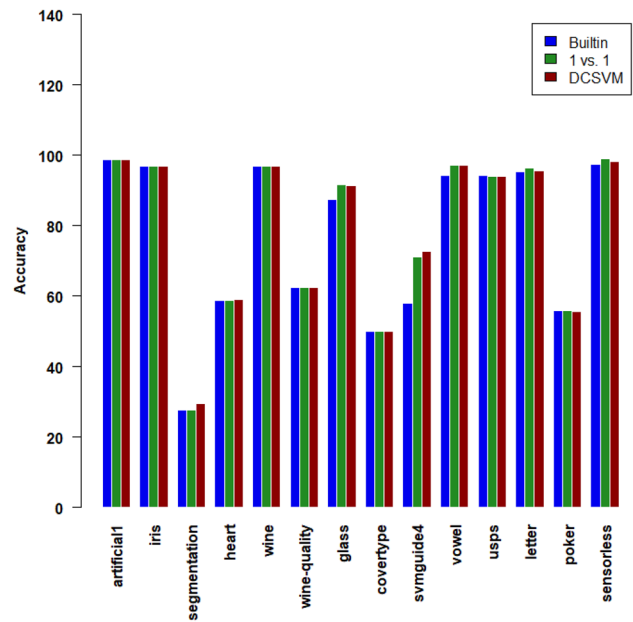


Fig. 4 Multi-class prediction accuracy comparison: built-in SVM, one-versus-one, and DCSVM

Table 3 Data sets

No	Dataset	Classes
1.	artificial	6
2.	iris	3
3.	segmentation	7
4.	heart	5
5.	wine	3
6.	wine-quality	6
7.	glass	6
8.	coverytype	7
9.	svmguide4	6
10.	vowel	11
11.	usps	10
12.	letter	26
13.	poker	10
14.	sensorless	11

Table 4 p values from Wilcoxon signed rank test

No	Dataset	BI – DCSVM	OvsO – DCSVM
1.	artificial	0.472211684046012	1
2.	iris	1	1
3.	segmentation	0.371093369522698	0.371093369522698
4.	heart	0.58621368107314	1
5.	wine	0.850106739138526	1
6.	wine-quality	0.93216149611896	1
7.	glass	0.0491095391972045	1
8.	coverytype	1	1
9.	svmguide4	0.00585709908528413	0.0567594463860135
10.	vowel	0.019769406440668	1
11.	usps	0.915345283110766	0.160967196971152
12.	letter	0.00588927004181748	0.396438915257121
13.	poker	0.461450987833361	1
14.	sensorless	0.00592153702414871	0.722674457315873

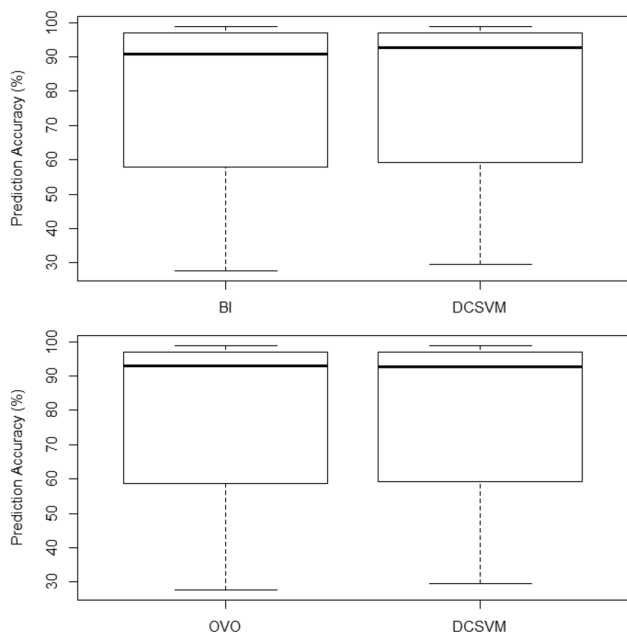


Fig. 5 Prediction accuracies comparison for all data sets

method (the null hypothesis is not rejected, according to values in the “OvsO—DCSVM” column of the table). Interestingly, there are likely differences in the median when comparing DCSVM and the built-in method for a few data sets (as usual, we consider the null hypothesis being rejected when p value < 0.05). For instance, one of the smallest p value corresponds to the “svmguide4” data set: 0.00585709908528413. However, as Fig. 4 shows, all these instances correspond to situations where DCSVM’s average accuracy is better than the built-in method’s accuracy (including the case of the “svmguide4” data set). These conclusions (and data presented in Table 4) are consistent with the prediction accuracies over all data sets in our experiments in Fig. 5: on median, DCSVM performs similarly to the one-versus-one implementation and slightly better than the built-in method.

4.2 Prediction performance comparison

For this purpose, we compared the R implementations of one-versus-one method and DCSVM. We analyzed prediction performance in three aspects: the average number of support vectors, the average number of binary decisions, and time. The number of support vectors used was computed by summing up all support vectors from every binary decider, over all steps of binary decisions until the multi-class prediction was achieved. The number of such support vectors is clearly proportional not only to the number of decision steps (which are illustrated separately), but also to the configuration of data separated by each binary classifier. The

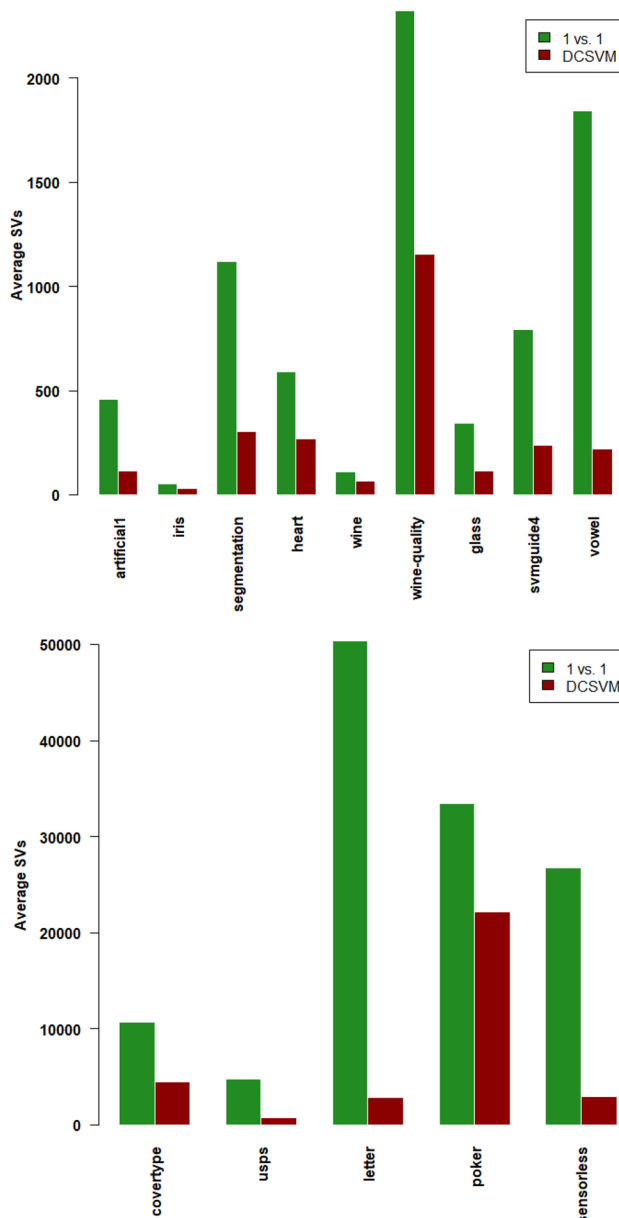


Fig. 6 Average number of Support Vectors for multi-class predictions

corresponding performance results are presented in Figs. 6, 7, and 8, respectively. Due to large variations in size between the data sets we used, we split the data sets into two size-balanced groups and displayed each graph side-by-side for each group. DCSVM significantly outperforms one-versus-one, clearly being much less computationally intensive (number of support vectors for prediction) and faster (number of binary decisions and prediction times).

From the first two sets of experimental results we can conclude already that DCSVM achieved the goal of being a faster multi-class predictor without sacrificing prediction accuracy. This conclusion is also supported by the overall comparison (for all data sets in our experiments) of the

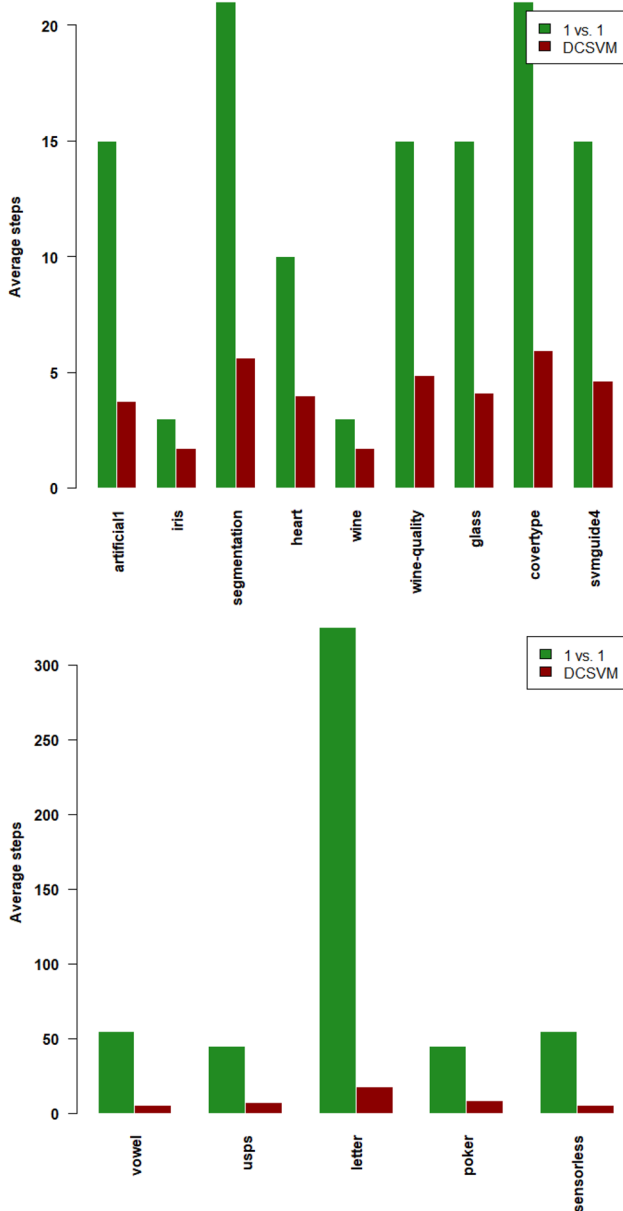


Fig. 7 Average number of binary decisions for multi-class predictions

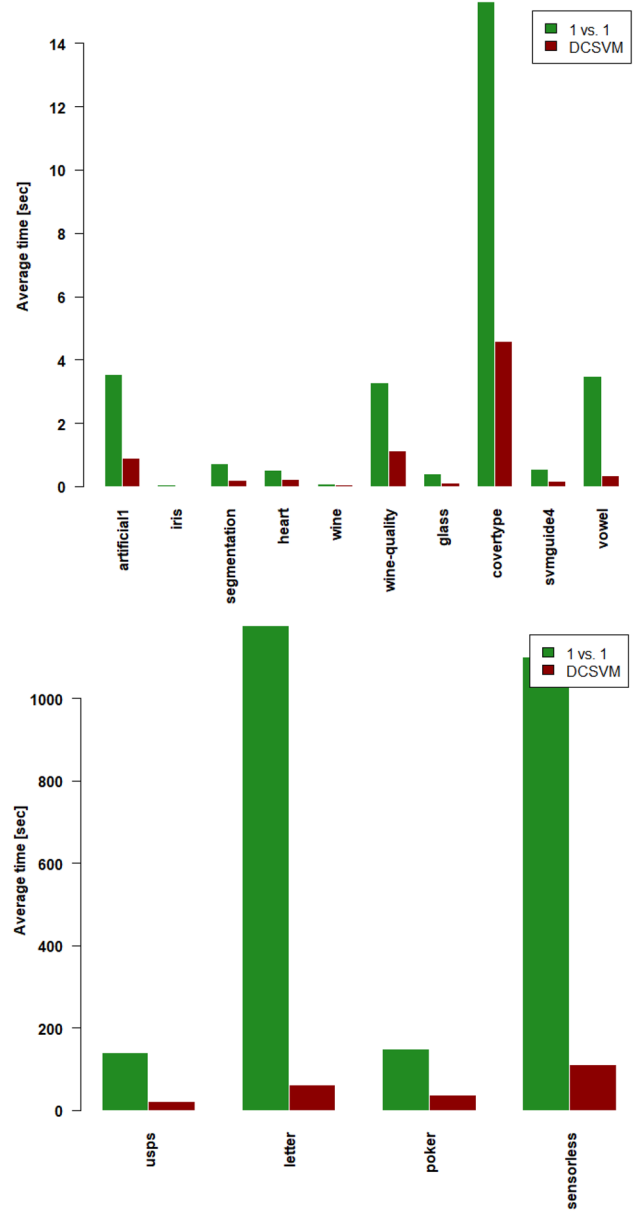


Fig. 8 Average prediction times for multi-class predictions

number of support vectors used for prediction, as illustrated in Fig. 9.

4.3 DCSVM performance fine tuning

In this set of experiments we analyze in close detail DCSVM’s performance in terms of the accuracy threshold parameter. Figure 10 shows the trade-off between accuracy (top) and the average prediction steps (bottom) with various threshold values. Clearly, the accuracy threshold parameter permits a trade-off between accuracy and speed. However, this is largely data dependent. The more separable the data is, the less influence the threshold has on speed. For less

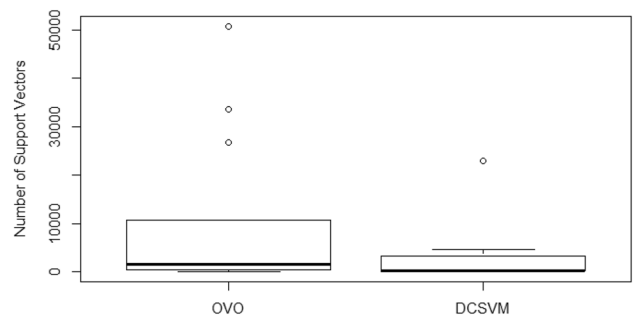


Fig. 9 Total number of Support Vectors for multi-class predictions (aggregated over all data sets and for one-versus-one and DCSVM methods)

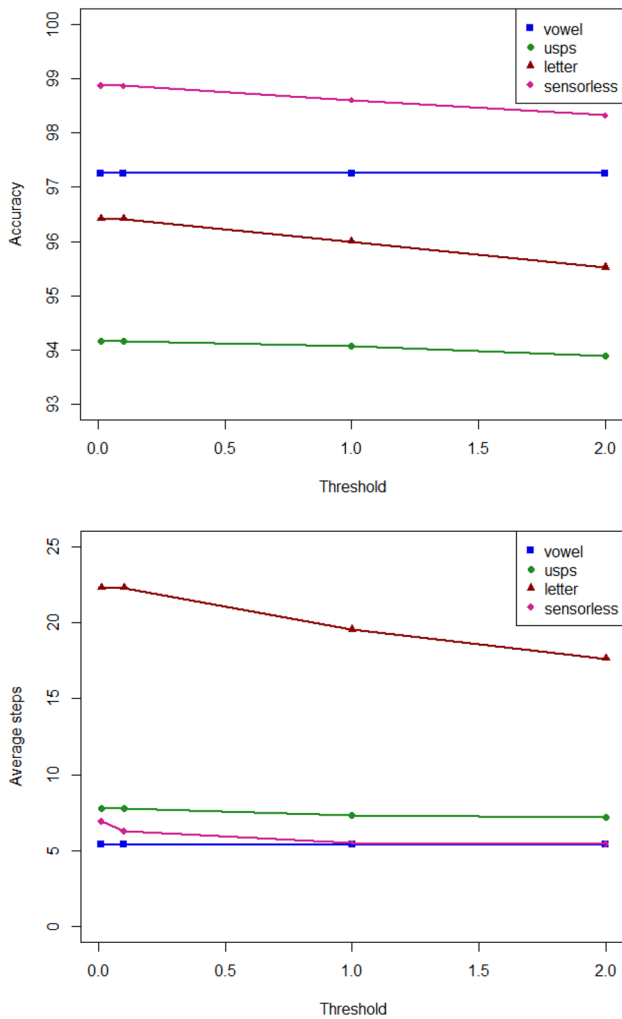


Fig. 10 Accuracy and the average number of prediction steps for different thresholds

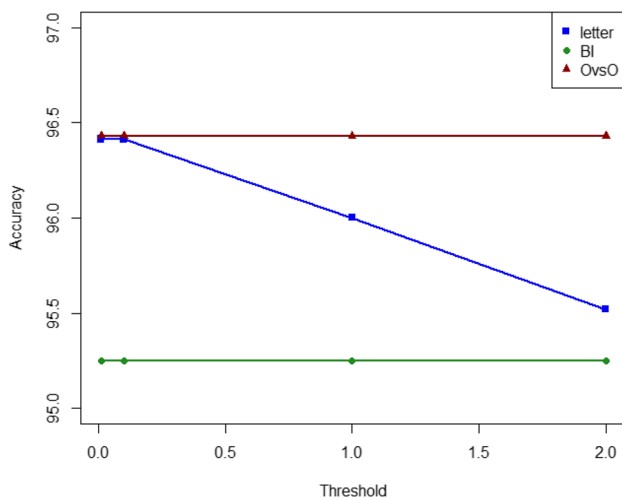


Fig. 11 Accuracy for predicting “letter” with each method, for different split thresholds

separable data (such as the *letter* data set), fine adjustment of the threshold permits trade-off between prediction accuracy and prediction speed. This is not the case for the *vowel* data set, which is highly separable: changes in the threshold influence neither the accuracy of prediction nor the average number of prediction steps.

Figure 11 shows how DCSVM accuracy compares to other multi-class classification methods (BI, built-in; OvsO, one-against-one) for various threshold values. For less separable data (such as *letter*) DCSVM’s accuracy drops sharply with the threshold (starting at some small threshold value) compared to the accuracy of one-against-one method, which we found to perform better than the built-in method. The built-in and one-against-one methods do not depend on the threshold value, of course. They are shown on the same graph for comparison purpose. However, it is interesting to notice that by increasing the threshold the prediction accuracy of DCSVM on *letter* data sets decreases from a comparable value with one-versus-one method’s accuracy (which performs best on this data set) to the accuracy of the built-in method. With a threshold value $\theta = 2\%$ the prediction accuracy of DCSVM is still above the accuracy of the built-in method (for the *letter* data set).

Table 5 shows side-by-side accuracies of multi-class classification using (1) built-in (BI), (2) one-against-one (OvsO), and (3) DCSVM (for a few threshold values θ). DCSVM performs very well in terms of accuracy (compared to the other methods) for all data sets, for threshold values $\theta \in \{2\%, 1\%, 0.1\%, 0.01\%\}$ (the smaller the threshold, the better the accuracy, in general). A larger threshold θ may increase the prediction speed (Table 6) and reduce the computation effort (Table 7). Interesting to notice: Table 6 shows that in all cases displayed in the table the number of decision steps is less than $k-1$, where k is the number of classes in the respective data set. In terms of the number of SVM binary decisions DCSVM outperforms (even when the threshold θ is not very small) one-against-one and its improvement DAGSVM, which reaches multi-class prediction after $k-1$ steps. In this context it worth mentioning that recent work on improving DAGSVM performance [1, 13, 21] focus on combining DAGSVM with other methods or learning the optimal DAG structure for computational efficiency, not on reducing the number of binary decisions. Moreover, previous work ([16] and more recently [1, 13, 21]) indicates that DAGSVM performs similarly to one-against-one in terms of accuracy and all recent efforts were concentrated on improving its computational efficiency.

The *All-Predictions* table in Fig. 2 collects all information used by DCSVM to construct its multi-class prediction strategy (the *dcsvm* decision tree in Algorithm 1). The same information can be used to predict how much separation can be achieved for different threshold values. For instance,

Table 5 Prediction accuracies for different split thresholds

No	Dataset	BI	OvsO	DCSVM			
				$\theta = 2$	$\theta = 1$	$\theta = 0.1$	$\theta = 0.01$
1	artificial1	98.85	98.76	98.70	98.70	98.76	98.76
2	iris	96.97	96.97	96.97	96.97	96.97	96.97
3	segmentation	27.71	27.71	29.44	29.44	29.44	29.44
4	heart	58.82	58.82	59.12	59.12	59.12	59.12
5	wine	96.97	96.97	96.97	96.97	96.97	96.97
6	wine-quality	62.33	62.33	62.39	62.39	62.39	62.39
7	glass	87.55	91.70	91.29	91.29	91.29	91.29
8	covertypes	49.95	49.95	49.95	49.95	49.95	49.95
9	svmguide4	57.88	71.21	72.73	72.73	72.73	72.73
10	vowel	94.34	97.26	97.26	97.26	97.26	97.26
11	usps	94.17	93.94	93.89	94.08	94.17	94.17
12	letter	95.25	96.43	95.52	96.00	96.41	96.41
13	poker	55.94	55.96	55.56	55.83	55.96	55.96
14	sensorless	97.46	98.87	98.32	98.60	98.86	98.87

Table 6 DCSVM average number of steps per decision, for different split thresholds vs. “1 vs. 1” and DAGSVM number of steps per decision

No	Dataset	$\theta = 2$	$\theta = 1$	$\theta = 0.1$	$\theta = 0.01$	1 vs. 1	DAGSVM
1	artificial1	3.76	3.75	3.67	3.67	15	5
2	iris	1.71	1.71	1.71	1.71	3	2
3	segmentation	5.63	5.63	5.63	5.63	21	6
4	heart	4.00	4.00	4.00	4.00	10	4
5	wine	1.69	1.69	1.69	1.69	3	2
6	wine-quality	4.85	4.86	4.87	4.87	15	5
7	glass	4.09	4.12	4.12	4.12	15	5
8	covertypes	5.93	5.93	5.93	5.93	21	6
9	svmguide4	4.63	4.88	4.88	4.88	15	5
10	vowel	5.41	5.41	5.41	5.41	55	10
11	usps	7.16	7.29	7.80	7.80	45	9
12	letter	17.63	19.56	22.29	22.29	325	25
13	poker	8.36	8.40	8.43	8.43	45	9
14	sensorless	5.44	5.49	6.28	6.93	55	10

for the *glass* data set *All-Predictions* table in Fig. 2 and for a threshold value $\theta = 2\%$ there are 58 entries in the table where the percentage of predicting one class or the other is at least $100 - \theta = 98\%$ (out of a total of $15 \times 6 = 90$ entries in the table). The percentage $58/90 = 64.4\%$ is a good indicator of purity for DCSVM with threshold $\theta = 2\%$: the higher the percentage, the more separation is produced at each step and hence a shallow decision tree. Figure 12 shows the class separation percentages for threshold values $0 \leq \theta \leq 5$ and four data sets (*letter*, *vowel*, *usps*, and *sensorless*). Intuitively, as threshold increases so does the separation percentage. The *letter* and *usps* data sets display an almost linear increase of separation with threshold. *sensorless* displays a sharp increase for small threshold values, then it tends to flatten, that is, not much gain for significant increase in threshold (and hence possibly less accuracy). Lastly, *vowel* displays a

step-like behavior: not much gain in separation until threshold value reaches approx $\theta = 2.3\%$, a steep increase until θ approaches 3%, then nothing much happens again. One can use these indicators to decide the trade-off between speed and accuracy of predictions.

5 Conclusion and future work

In this paper we present DCSVM, a fast algorithm for multi-class classification using Support Vector Machines. Our method relies on dividing the whole training data set into two partitions that are easily separable by a single binary classifier. Then, a prediction between the two training set partitions would eliminate one or more classes at the time. The algorithm continues recursively until a final

Table 7 DCSVM: Average support vectors per decision, for different split thresholds

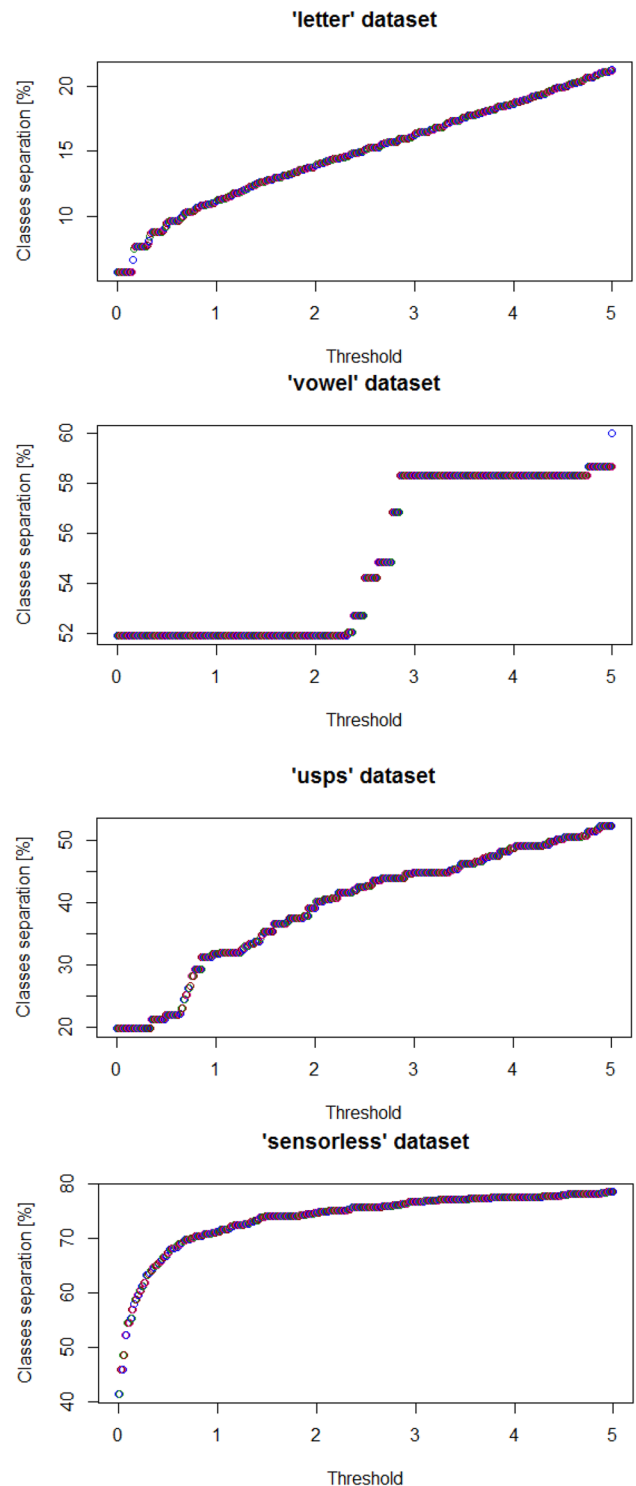
No	Dataset	$\theta = 2$	$\theta = 1$	$\theta = 0.1$	$\theta = 0.01$
1	artificial1	115.17	117.29	127.22	127.22
2	iris	32.47	32.47	32.47	32.47
3	segmentation	305.49	305.49	305.49	305.49
4	heart	270.18	270.18	270.18	270.18
5	wine	66.41	66.41	66.41	66.41
6	wine-quality	1154.49	1155.55	1157.13	1157.13
7	glass	112.14	114.37	114.37	114.37
8	covertype	4528.47	4528.47	4528.47	4528.47
9	svmguide4	236.58	245.71	245.71	245.71
10	vowel	218.36	218.36	218.36	218.36
11	usps	785.21	798.41	846.84	846.84
12	letter	2822.54	3110.42	3307.19	3307.19
13	poker	22166.49	22735.92	22803.01	22807.62
14	sensorless	2977.89	3057.77	3404.91	3735.30

binary decision is made between the last two classes left in the competition. Our algorithm performs consistently better than the existent methods on average. In the best case scenario, our algorithm makes a final decision between k classes in $O(\log k)$ decision steps between different partitions of the training data set. In the worst case scenario, DCSVM makes a final decision in $k-1$ steps, which is not worse than the existent techniques.

The SVM divide and conquer technique we present for multi-class classification can be easily used with any binary classifier. It is rather a consequence of increasing data sparsity with the dimensionality of the space, which can be exploited, in general, in favor of producing fast multi-class classification using binary classifiers. Our experimental results on a few popular data sets show the applicability of the method.

A clear limitation of using binary classifiers to produce multiclass classification is the large number of binary classifiers to be created (quadratic in the number of classes). This can be very computationally expensive for large number of classes. However, this is a general problem for multiclass classifiers, in general. In the case of using binary classifiers the limitation can be partially overcome by using parallelization (binary classifiers can be independently created, in parallel).

Our choice of using SVMs as binary classifiers relies on the fact that SVM binary classifiers produce optimal separation hyperplanes (the underlying optimization problem finds a global minimum, rather than a local minimum). It would be interesting, as future work, to compare with techniques that produce direct multiclass classification (using

**Fig. 12** Number of separated classes for different thresholds

neural networks, decision trees, k -nearest neighbours, etc) rather than relying on binary classifiers. Also, possibly adapting some decision trees metrics (Gini impurity,

Information gain) for producing better DCSVM decision trees would be another direction to explore in future work. Yet, other directions to further investigate would be related to the precision threshold θ : a data-driven procedure for its selection and the possible benefits of using variable θ values on different decision tree branches (the shallow ones). The former would help practitioners understand trade-offs between different values, while the latter may improve classification accuracy for some classes with the price of a small increase of tree-branch depth (hence more decision steps).

Acknowledgements We would like to express our appreciation to the anonymous reviewers for their numerous suggestions that improved the quality of this paper. We owe our reviewers not only corrections and improvements but also some interesting future directions to investigate.

References

- Agarwal N, Balasubramanian VN, Jawahar C (2018) Improving multiclass classification by deep networks using DAGSVM and Triplet Loss. *Pattern Recogn Lett* 112:184–190. <https://doi.org/10.1016/j.patrec.2018.06.034>. <http://www.sciencedirect.com/science/article/pii/S0167865518302745>
- Bellman R (2003) *Dynamic programming*. Dover books on computer science series. Dover Publications, New York
- Bellman R, Corporation R (1957) *Dynamic programming*. Rand corporation research study. Princeton University Press, Princeton
- Bishwas AK, Mani A, Palade V (2017) An all-pair approach for big data multiclass classification with quantum SVM. *CoRR abs/1704.07664*
- Bottou L, Cortes C, Denker J, Drucker H, Guyon I, Jackel L, Lecun Y, Muller U, Sackinger E, Simard P, Vapnik V (1994) Comparison of classifier methods: a case study in handwritten digit recognition. In: IAPR (ed) *Proceedings of the international conference on pattern recognition*, Jerusalem, vol II, pp 77–82, IEEE
- Bredensteiner EJ, Bennett KP (1999) Multicategory classification by support vector machines. *Comput Optim Appl* 12(1–3):53–79
- Cortes C, Vapnik V (1995) Support-vector networks. *Mach Learn* 20:273–297
- Crammer K, Singer Y (2002) On the algorithmic implementation of multiclass kernel-based vector machines. *J Mach Learn Res* 2:265–292
- Dheeru D, Taniskidou EK (2017) UCI machine learning repository. <http://archive.ics.uci.edu/ml>
- Dimitriadou E, Hornik K, Leisch F, Meyer D, Weingessel A (2018) e1071: Misc functions of the department of statistics (e1071), TU Wien. R package version 1.7-0. <http://cran.r-project.org/package=e1071>. Accessed Nov 2018
- Friedman JH (1996) Another approach to polychotomous classification. Tech. rep., Department of Statistics, Stanford University. <http://www-stat.stanford.edu/~jhf/ftp/poly.ps.Z>. Accessed Nov 2018
- Galar M, Fernández A, Barrenechea E, Bustince H, Herrera F (2011) An overview of ensemble methods for binary classifiers in multi-class problems: experimental study on one-vs-one and one-vs-all schemes. *Pattern Recogn* 44(8):1761–1776
- Gao T, Koller D (2011) Discriminative learning of relaxed hierarchy for large-scale visual recognition, pp 2072–2079. <https://doi.org/10.1109/ICCV.2011.6126481>. (Cited By 100)
- Hajikhodaverdikhan P, Nazari M, Mohsenizadeh M, Shamsirband S, Chau K wing (2018) Earthquake prediction with meteorological data by particle filter-based support vector regression. *Eng Appl Comput Fluid Mech* 12(1):679–688
- He X, Wang Z, Jin C, Zheng Y, Xue X (2012) A simplified multi-class support vector machine with reduced dual optimization. *Pattern Recogn Lett* 33:71–82
- Hsu CW, Lin CJ (2002) A comparison of methods for multiclass support vector machines. *IEEE Trans Neural Netw* 13(2):415–425
- Joutsijoki H, Juhola M (2012) DAGSVM vs. DAGKNN: an experimental case study with benthic macroinvertebrate dataset. In: Perner P (ed) *Machine learning and data mining in pattern recognition*. Springer, Berlin, Heidelberg, pp 439–453
- Knerr S, Personnaz L, Dreyfus G (1990) Single-layer learning revisited: a stepwise procedure for building and training a neural network. In: Fogelman Soulié F, Héroult J (eds) *Neurocomputing: algorithms, architectures and applications, NATO ASI Series*, vol F68, pp 41–50. Springer
- Kreßel UHG (1999) *Advances in kernel methods*. chap. Pairwise classification and support vector machines. MIT Press, Cambridge, pp 255–268
- Moazenzadeh R, Mohammadi B, Shamsirband S, Chau KW (2018) Coupling a firefly algorithm with support vector regression to predict evaporation in northern Iran. *Eng Appl Comput Fluid Mech* 12(1):584–597
- Panda P, Roy K (2016) Attention tree: learning hierarchies of visual features for large-scale image recognition. *CoRR abs/1608.00611*. <http://arxiv.org/abs/1608.00611>
- Park SH, Fürnkranz J (2007) Efficient pairwise classification. In: Kok JN, Koronacki J, Mantaras RLD, Matwin S, Mladenić D, Skowron A (eds) *Machine learning: ECML 2007*, pp 658–665. Springer, Berlin, Heidelberg
- Platt JC, Cristianini N, Shawe-Taylor J (1999) Large margin dags for multiclass classification. In: *Proceedings of the 12th international conference on neural information processing systems, NIPS'99*, pp 547–553. MIT Press, Cambridge
- Rosales-Perez A, Garcia S, Terashima-Marin H, Coello CAC, Herrera F (2018) Mc2esvm: Multiclass classification based on cooperative evolution of support vector machines. *IEEE Comput Intell Mag* 13(2):18–29
- Silva-Palacios D, Ferri C, Ramírez-Quintana MJ (2018) Probabilistic class hierarchies for multiclass classification. *J Comput Sci* 26:254–263
- Szedmak S, Shawe-Taylor J, Saunders C, Hardoon D (2004) Multiclass classification by l1 norm support vector machine. In: *Pattern recognition and machine learning in computer vision workshop*
- Vapnik V (1998) *Statistical learning theory*. Wiley, New York
- Weston J W C (1999) Multi-class support vector machines. In: *Proceedings of the European symposium on artificial neural networks ESANN99*, pp 219–224
- Wikipedia: Curse of dimensionality (2018) https://en.wikipedia.org/wiki/Curse_of_dimensionality. Accessed Nov 2018
- Xu J, Liu X, Huo Z, Deng C, Nie F, Huang H (2017) Multi-class support vector machine via maximizing multi-class margins. In: *Proceedings of the 26th international joint conference on artificial intelligence*, pp 3154–3160. AAAI Press

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.