Multi-Dimensional Classification with Super-Classes

Jesse Read, Concha Bielza, *Member, IEEE,* and Pedro Larrañaga, *Member, IEEE,*

Abstract—The multi-dimensional classification problem is a generalization of the recently-popularized task of multi-label classification, where each data instance is associated with multiple class variables. There has been relatively little research carried out specific to multi-dimensional classification and, although one of the core goals is similar (modeling dependencies among classes), there are important differences; namely a higher number of possible classifications. In this paper we present method for multi-dimensional classification, drawing from the most relevant multi-label research, and combining it with important novel developments. Using a fast method to model the conditional dependence between class variables, we form super-class partitions and use them to build multi-dimensional learners, learning each super-class as an ordinary class, and thus explicitly modeling class dependencies. Additionally, we present a mechanism to deal with the many class values inherent to super-classes, and thus make learning efficient. To investigate the effectiveness of this approach we carry out an empirical evaluation on a range of multi-dimensional datasets, under different evaluation metrics, and in comparison with high-performing existing multi-dimensional approaches from the literature. Analysis of results shows that our approach offers important performance gains over competing methods, while also exhibiting tractable running time.

Index Terms—Multi-dimensional classification, problem transformation

1 Introduction

The goal of *multi-dimensional classification* is to assign each data instance to *multiple* classes. This contrasts with the traditional task of classification which involves assigning each instance to a single class. The recently popularised task of multi-label classification (see [16], [22] for overviews) can be viewed as a particular case of the multi-dimensional problem that only involves binary classes, considered as *labels* that can be turned on (1) or off (0) for any data instance. Multi-label classification can be applied to a variety of real-world problems, but there are many others only suitable for multi-dimensional classification. For example, an image can be multi-labelled with a set of concepts (beach, forest, etc.), but other non-binary information such as the month, season, number of objects present, or the type of subject, are best represented in a context which allows for multiple classes of multiple values; i.e., multi-dimensional classification.

As in multi-label classification, a fundamental goal of multi-dimensional learning is modelling the relationships (dependencies) between classes and dealing with the computational complexity that this entails. If classes are completely unrelated, it should suffice to create a separate (single-dimensional) independent model for each class. However, this is unlikely to occur.

Although all multi-label problems can be considered as multi-dimensional problems, the reverse is not true, and there are some crucial differences meaning that much multi-label research is not directly applicable.

Quantitatively speaking, for d class variables of K possible values each[1], there are K^d possible class assignments in the multi-dimensional setting, compared to 2^d possible label assignments in a multi-label problem. There is often also a qualitative difference in the class distribution. Typically in multi-label classification, each class label is used to indicate "relevant" / "not relevant". Of many possible labels, any particular label will be *not* relevant most of the time. In other words, multi-labelling is sparse and imbalanced. On the other hand, consider a binary class in a multi-dimensional problem indicating "male" / "female"; clearly (specific prior-knowledge of the problem aside), we expect an even balance of both classes. Note that while of course this class label is valid for multi-label data, the type of distribution it entails is less typical of one, being more like a 'category' than a 'label' [16].

In this study, we investigate some existing techniques from the multi-label literature and combine them with novel developments suitable for the multi-dimensional domain. The core contributions of this work are a novel method for combining classes into super-classes based on conditional dependencies between classes, and a mechanism to make the resulting problem tractable

- *J. Read is with the Universidad Carlos III de Madrid, Leganés, Madrid 28911, Spain. E-mail: jesse@tsc.uc3m.es.*
- *C. Bielza and P. Larrañaga are with the Universidad Politécnica de Madrid, Boadilla del Monte, Madrid 28660, Spain. E-mail: {mcbielza; pedro.larranaga}@fi.upm.es.*

Manuscript received 12 Dec. 2012; revised 9 Oct. 2013; accepted 18 Oct. 2013. Date of publication 24 Oct. 2013; date of current version 9 July 2014.
Recommended for acceptance by V. S.-M. Tseng.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier 10.1109/TKDE.2013.167

1. This is a simplification. As we explain shortly, each class variable can take a different number of values.

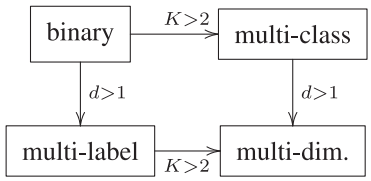


Fig. 1. Relationship between different classification paradigms, where d is the number of class variables and K is the number of values each of these variables may take.

for multi-dimensional learning settings by reducing the number of distinct super-class values. Furthermore, our approach reveals conditional dependencies among classes and their relative strength (which may be of interest in data analysis), is equally applicable to multi-label data and, because it can take any base classifier, it is very flexible and can be adapted to a wider range of problems than other methods in the literature.

The rest of the paper is organised as follows. First, we review multi-dimensional classification and classifiers (Section 2). We then introduce our super-class classifier (Section 3), a filtering mechanism to make this tractable (Section 4), and combine these into an ensemble approach (Section 5). We then carry out an experimental evaluation and discuss the relative performance of our classifiers (Section 6), and finally, we draw conclusions and discuss future work (Section 7).

2 Multi-dimensional Classification

In multi-dimensional classification, we have a number of training examples from which we wish to build a classifier (i.e., some function) that associates multiple class values with each data instance.

The data instance is represented by a vector of m values $\mathbf{x} = (x_1, \ldots, x_m)$, each drawn from some input domain $\mathcal{X}_1 \times \cdots \times \mathcal{X}_m$.

The classes associated with each data instance are represented as a vector of d values $\mathbf{y} = (y_1, \ldots, y_d)$ from the domain $\mathcal{Y}_1 \times \cdots \times \mathcal{Y}_d$ where each $\mathcal{Y}_j = \{1, \ldots, K_j\}$ is the set of possible values for the jth class variable. In the traditional task of (single-dimensional) *multi-class* classification, there is only one such variable associated with a data instance, i.e., $(\mathbf{x}, y)$ where $y \in \mathcal{Y}$. In *multi-label* classification, each $|\mathcal{Y}_j| = 2$ (there are only two classes) for all $j = 1, \ldots, d$ (i.e., binary classification where a label is either relevant or not). In the *multi-dimensional* case, each $|\mathcal{Y}_j| = K_j$ for any positive integer K_j.

It is important to reiterate that we refer to a *class variable* as a target or output variable; a kind of multi-dimensional label that can take a number of class *values* (a label, as we refer to it, is basically a binary class variable).

Fig. 1 displays the relationship between the different classification paradigms in terms of d class variables of K possible values each. Table 1 exhibits a toy multi-dimensional problem where $d = 3$.

In multi-dimensional learning we assume a set of training data of N labelled examples $D = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{N}$, where $\mathbf{y}^{(i)}$ is the class vector assignment of the ith example and $y_j^{(i)}$ is the value of the jth class assigned to the ith example.

TABLE 1
Multi-Dimensional Problem of N Examples and $d = 3$ Class Variables

	X_1	X_2	X_3	X_4	X_5	Y_1 sex	Y_2 prof.	Y_3 income
(1)	1	22	0.8	1.8	0	1	1	1
(2)	0	20	0.7	3.5	1	2	1	1
(3)	0	39	0.1	1.2	0	1	2	3
(4)	1	43	0.5	3.2	1	2	3	3
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
(N)	1	42	0.6	1.3	1	1	2	2
$\tilde{\mathbf{x}}$	0	35	0.8	3.3	1	?	?	?

Suppose that $\mathcal{Y}_1 = \{M, F\}$, $\mathcal{Y}_2 = \{\texttt{student}, \texttt{doctor}, \texttt{pilot}\}$, and $\mathcal{Y}_3 = \{\texttt{low}, \texttt{med}, \texttt{high}\}$ such that $\mathbf{y} = (2, 1, 3)$ is a female student on a high income. Each instance is described in this case by five variables $X_1, \ldots, X_5$. The goal is to learn to assign class values to test instances $\tilde{\mathbf{x}}$.

We seek to build a classifier $\mathbf{h}$ that assigns each instance $\mathbf{x}$ a vector $\mathbf{y}$ of class values:

$$\begin{aligned} \mathbf{h}: \mathcal{X}_1 \times \cdots \times \mathcal{X}_m &\rightarrow \mathcal{Y}_1 \times \cdots \times \mathcal{Y}_d \\ \mathbf{x} &\mapsto \mathbf{y}, \end{aligned}$$

where $\mathbf{h}$ is usually composed of a number of single-dimensional classifiers $h_1, h_2, \ldots$ (hence the bold notation). Thus, classifier $\mathbf{h}$ outputs prediction vector $\hat{\mathbf{y}}$ for any test instance $\tilde{\mathbf{x}}$:

$$(\hat{y}_1, \ldots, \hat{y}_d) = \hat{\mathbf{y}} = \mathbf{h}(\tilde{\mathbf{x}}).$$

2.1 Multi-Dimensional Classifiers

A straightforward method for multi-dimensional classification is the *independent classifiers* method (IC); where one single-dimensional multi-class classifier is used for each class variable. Hence, IC trains d classifiers $\mathbf{h} := (h_1, \ldots, h_d)$, where each

$$h_j: \mathcal{X}_1 \times \cdots \times \mathcal{X}_m \rightarrow \mathcal{Y}_j$$

is a standard classifier that learns to associate one of the values $y_j \in \mathcal{Y}_j$ to each data instance. The main problem with IC is that it does not model class dependencies, and its accuracy suffers as a result [16], [19], [23], [25].

To overcome the limitation of IC, [19] put forth the idea (in a multi-label context) of *classifier chains* (CC). As in IC, d classifiers are used, but linked in a chain such that each classifier learns the association of that label given the instance *and* the previous label associations in the chain, such that:

$$h_j: \mathcal{X}_1 \times \cdots \times \mathcal{X}_m \times \mathcal{Y}_1 \times \cdots \times \mathcal{Y}_{j-1} \rightarrow \mathcal{Y}_j$$

i.e., $\hat{y}_j = h_j(\tilde{\mathbf{x}}, \hat{y}_1, \ldots, \hat{y}_{j-1})$ for any test instance $\tilde{\mathbf{x}}$ (classifiers are evaluated in order $h_1, \ldots, h_d$).

This method has demonstrated high performance in multi-label domains and is directly applicable to multi-dimensional classification.

A few Bayesian classifier-chain methods have appeared recently in the literature. A Bayes-optimal classifier chain was presented by [6]; however this method is intractable for many real-world problems because it explores all 2^d paths of the chain (K^d paths if it were used in a multi-dimensional setting). In [25] a Bayesian network approach is followed according to the dependency relations between the target variables. This network is learned as a tree structure d times (where the root node of the jth tree is the jth class variable). Like other chain-based methods, the predictive performance of this tree depends on the order of the nodes.

An alternative offered in the multi-label literature to chain-based learning is the so-called *label powerset* method; which we shall refer to as the *class powerset* method (CP) since that is more fitting for the multi-dimensional context. This method considers all possible label combinations (i.e., the powerset) as the set of values of a single class. In practice, it predicts any combination of the training set as an approximation of the full space:

$$h{:}\mathcal{X}_1 \times \cdots \times \mathcal{X}_m \rightarrow \text{DISTINCT}\{\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(N)}\} \approx \mathcal{Y}_1 \times \cdots \times \mathcal{Y}_d$$

In other words, the output space is the Cartesian product of the class spaces, approximated in practice by the distinct class-combinations in the training set.

Because this method models label dependencies, it often outperforms IC, but it is usually far too computationally complex for practical application [19], [23] (even more so in the multi-dimensional setting where it models up to K^d class combinations). Additionally, CP easily suffers from class imbalance and overfitting (by not being able to predict class combinations it has not seen in the training data). Several multi-label approaches have been introduced to address these issues, particularly that of running time, such as RAkEL [23] which creates an ensemble of random label subsets, and EPS [17] which eliminates some of the less-frequently occurring label combinations prior to training. RAKEL could be adapted to multi-dimensional classification (with changes to its label-voting ensemble process) but can no longer compete with more modern methods (as shown in a recent empirical evaluation by [15]); its label subsets are arbitrary, without leveraging label-dependency information. EPS's label-based (binary-only) 'subset' method is unsuitable for multi-dimensional classification since it is based on finding subsets $\mathbf{y}' \subseteq \mathbf{y} \Leftrightarrow \mathbf{y}' \wedge \mathbf{y} = \mathbf{y}'$; a concept that does not translate outside of the binary context.

Compared to the volume and variety of multi-label classification, there is relatively little work specific to multi-dimensional classification. We have already mentioned the relatively recent Bayesian-network approach of [25]. Another Bayesian approach to multi-dimensional classification is [2], extended in [3] using Markov blankets; dependencies are modelled among all input and class variables. In [20], 'predictive clustering' decision trees are used. These trees are built with a standard top-down induction of decision trees, but use a difference variance function, so that the tree can make multi-dimensional classifications at the leaves. A newer ensemble version of this approach is presented in [14] which has proved highly competitive in multi-label classification, as reported by [15]. In [24], decision rules are adapted to make multi-dimensional predictions.

These methods are of the so-called *algorithm adaptation* type (adapting probabilistic classifiers, decision trees and decision rules, respectively); which often excel in certain domains but are less flexible than *problem transformation* such as IC, CC and CP, which can take any base classifier, and thus easily be adapted to the problem at hand. For example, Support Vector Machines have been shown to perform very well on many multi-label problems [19].

In the following sections we present the components of a multi-dimensional problem transformation method that we propose, called a *Super-class Classifier*. Specifically:

1) Section 3: creating super-classes; Section 3.1: based on conditional dependency information;
2) Section 4: a filter mechanism to make learning super-classes more efficient and deal the issue of sparsity; and
3) Section 5: a multi-dimensional ensemble process for this classifier.

This method explicitly models class dependencies without incurring intractable complexity and, unlike many existing multi-label methods, it models only the strongest conditional dependencies. As we show in later sections, it proves very competitive.

3 A Super-Class Classifier for Multi-dimensional Classification

It is already clear that independent classifiers (IC) do not leverage class-dependency information, and therefore this approach can yield poor accuracy. In essence, IC assumes $p(\mathbf{y}|\mathbf{x}) = \prod_{j=1}^{d} p(y_j|\mathbf{x})$, which is clearly violated in the presence of class dependencies.

In light of this, authors in the multi-label literature have turned to approaches which explicitly model label dependencies, usually approximations of CP, e.g., [17], [19], [23]. However, these methods superficially tackle CP's disadvantages: its time complexity and tendency to overfit the data.

Relatively little of the literature has challenged the assumptions that class dependencies are 1) incomplete and 2) unequal (although [12], for example, addresses the issue of 'local' rather than global correlations). These two assumptions are almost certainly valid for most real-world scenarios. Not all classes are always dependent on all other classes (as assumed by CP), and not with the same strength. This has important implications in building a classifier. Moreover, we must take into account that the training data is drawn from an unknown distribution. Therefore, dependencies in the data may be inaccurate, (especially in smaller training sets) and therefore it may even be at best unproductive and at worst positively harmful to include them in the model.

We can demonstrate this issue with some toy examples. Suppose we have a dataset of examples $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N}$; instances associated with $d = 3$ target variables Y_1, Y_2, Y_3,

each of $K = 3$ classes, i.e., $|\mathcal{Y}_1| = |\mathcal{Y}_1| = |\mathcal{Y}_3| = 3$. IC models

$$h_1 : \mathcal{X}_1, \ldots, \mathcal{X}_m \rightarrow \{1, 2, 3\}$$
$$h_2 : \mathcal{X}_1, \ldots, \mathcal{X}_m \rightarrow \{1, 2, 3\}$$
$$h_3 : \mathcal{X}_1, \ldots, \mathcal{X}_m \rightarrow \{1, 2, 3\}.$$

CP, on the other hand, models

$$\mathbf{h}: \mathcal{X}_1, \ldots, \mathcal{X}_m \rightarrow \text{DISTINCT}\{\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(N)}\}.$$

Which is the better model? In an extreme case where $|\text{DISTINCT}\{\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(N)}\}| = 3$, CP is in most cases the better option: using N examples to learn 3 classes is likely more easier than using the same number of examples to learn $3 \times 3 = 9$ classes (as does IC). However, at the other extreme where $|\text{DISTINCT}\{\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(N)}\}| = K^d = 27$, CP must learn 27 classes from N examples. IC in this case (which still only learns 9 classes in total) is far more likely to be the best model, especially for small N.

Rather than simply deciding between CP and IC for a particularly problem, we investigate the issue further. The distribution of our data could be such that (in our toy example) the dependency between class variables Y_1 and Y_2 is very high (let's say $p(Y_1 = 1, Y_2 = 1) = 0.5$ and $p(Y_1 = 3, Y_2 = 3) = 0.5$), whereas $\mathcal{Y}_3$ is independent of Y_1, Y_2 ($p(Y_3|Y_1, Y_2) \approx p(Y_3)$) with uniformly distributed values. In this case the ideal model will be:

$$\begin{aligned}\mathbf{h}_{1,2} : \mathcal{X}_1, \ldots, \mathcal{X}_m &\rightarrow \text{DISTINCT}\{\mathbf{y}^{(1)}_{1,2}, \ldots, \mathbf{y}^{(N)}_{1,2}\} \\ &= \{(1, 1), (3, 3)\} \approx \mathcal{Y}_1 \times, \mathcal{Y}_2 \\ h_3 : \mathcal{X}_1, \ldots, \mathcal{X}_m &\rightarrow \{1, 2, 3\} = \mathcal{Y}_3,\end{aligned}$$

where $\mathbf{y}_{1,2} \equiv (y_1, y_2)$. This model uses N examples to learn 2 bidimensional class values and 3 single-dimensional class values. This is a *super-class* classifier. We can define generally:

$$\mathbf{h}_\theta := (\mathbf{h}_{S_1}, \ldots, \mathbf{h}_{S_{|\theta|}}),$$

where θ is a *partition* of classes

$$\theta = \{S_1, \ldots, S_{|\theta|}\}$$

that takes these into account. In the above example the partition of classes is

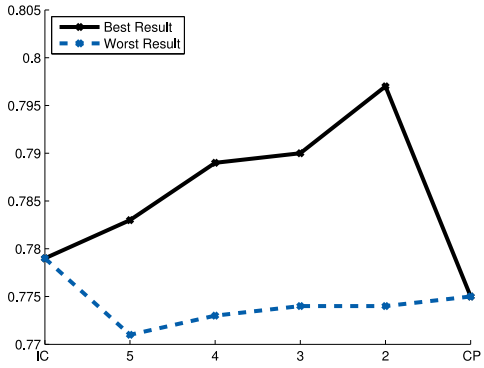
$$\theta = \{(1, 2), (3)\}.$$

The space of any super-class $S \in \theta$ is:

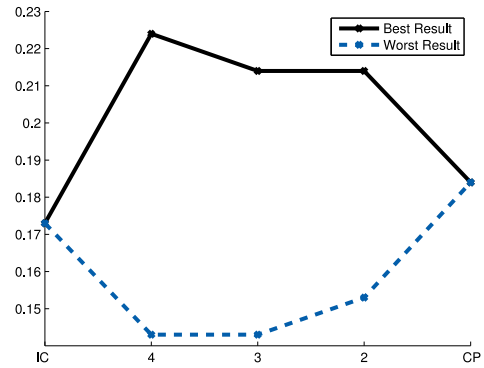
$$\begin{aligned}\mathcal{Y}_S &= \text{DISTINCT}\{\mathbf{y}^{(1)}_S, \ldots, \mathbf{y}^{(N)}_S\} \\ &\approx \mathcal{Y}_{S_1} \times \ldots \times \mathcal{Y}_{S_{|S|}}.\end{aligned}$$

Thus S_j can be considered an ordinary multi-dimensional class, and can be learned with any off-the-shelf multi-class classifier. Another way of seeing it, is that the set of super-classes can be learned by any off-the-shelf multi-dimensional classifier (e.g., IC) with $|\theta|$ classes. A super-class classifier with a good partition should perform better than both IC and CP.

Fig. 2 illustrates our case for super-classes with respect to real-world data. We see that the best performance is obtained for these datasets neither for IC nor CP, but rather for some partition of super-classes. Specifically, the



(a) Music Data ($d = 6$), CLASS ACCURACY



(b) Parkinson's Data ($d = 5$), EXAMPLE ACCURACY

Fig. 2. Best and worst predictive performances for the Music (top) and Parkinson's (bottom) data for $|\theta| = d, \ldots, 1$ classes (ordered by complexity), where $|\theta| = d$ is equivalent to IC, and $|\theta| = 1$ (super) class is equivalent to CP. The values for all possible combinations were obtained for a single train/test split using SVMs as a base classifier, with the best and worst displayed on the graph. Note that there is, of course, only a single combination possible with $|\theta| = 1$ and $|\theta| = d$. See Section 6.1 for details on CLASS ACCURACY and EXAMPLE ACCURACY.

best performance on the Music data is obtained with $|\theta| = 2$ super-classes. The best performance for Parkinson's is obtained for $|\theta| = 4$.

However, it is also clear that just trying to determine a good *number* of super-classes for the partition is not enough. Rather, it is fundamental to choose a good partition if we hope to achieve better accuracy than just using either IC or CP. If we choose a partition randomly, it could perform worse than both these methods.

Hence, the first objective for creating a super-class classifier is to find a *good* partition. The main obstacle is the sheer size of the space of possible partitions, which for d classes is the dth Bell number B_d, where $B_1 = 1$ and

$$B_d = \sum_{k=0}^{d-1} \binom{d-1}{k} B_k \quad (1)$$

for d classes. That is, 203 possible partitions for the relatively modest dimension of $d = 6$ and already 115,975 for $d = 10$.

In the following, we describe a way to score partitions in multi-dimensional data by measuring conditional class

dependencies, and then how we use this score to search through the space of possible partitions.

3.1 Modelling class dependencies

There are two types of class dependence. We can consider *unconditional dependence* which looks at the probability of one class given another irrespective of the associated data instances, i.e., if:

$$p(Y_j, Y_k) \neq p(Y_j)p(Y_k)$$

then there is unconditional dependence between the jth and kth classes. There is *conditional dependence* between these classes given $\mathbf{x}$ if:

$$p(Y_j, Y_k|\mathbf{x}) \neq p(Y_j|\mathbf{x})p(Y_k|\mathbf{x})$$

which can be measured by learning from the data instances. A good review of dependence (in the multi-label context) is given in [7].

In the multi-label literature, there already exist approaches for creating partitions θ by measuring unconditional dependence. [21] computes a chi-squared (χ^2) score from the relative frequencies of pairs of classes and proposes a χ^2-dependency ensemble where a large number of labelset partitions are generated randomly, and a score is computed for each partition by summing the χ^2 score for all pairs in the same set, and subtracting from it the score of all pairs in different sets:

$$\pi(\theta): = \Big(\sum_{j,k|\exists S:\{j,k\}\subset S} \chi^2_{j,k}\Big) - \Big(\sum_{q,r|\nexists S:\{q,r\}\subset S} \chi^2_{q,r}\Big). \quad (2)$$

The indices j, k represent all pairs found together in some set $S \in \theta$, and q, r are all pairs of labels in separate sets (there are $\frac{d(d-1)}{2}$ pairs in total). The top M partitions (a user parameter) are then used to build an ensemble. This method can evaluate labelset partitions very rapidly because it does not rely on building and evaluating internal models. However, this means the method only measures *unconditional* label dependence. While conditional and unconditional dependence may be related, there is no guarantee that they are [7], and ultimately it is conditional dependence which is more relevant to classification accuracy, since that is where the data instance dimension is considered.

The authors of [21] also present a method for combining labels based on conditional dependence; beginning with `IC` and then iteratively joining the most dependent pair of labels (again using Eq. (2)) but this time builds and evaluates the model and accepts it iff its predictive performance improves on that of the previously accepted model. Because this method takes into account the instance space (when it builds the classifier) it gives an indirect measure of conditional dependence. The main problem with this approach is that it is too slow. In the worst case, this method builds B_d possible models (see Eq. (1)) before arriving at a `CP` model (a single partition), although, since it searches in a greedy fashion, it is unlikely to reach the optimal partition at all. Furthermore, there is no definitive evidence that even a model with a good partition could compete with an ensemble of several `CP`-based models (such as [17], [23]) in the literature.

A related multi-label approach is offered by [11], where linear correlation coefficients (rather than χ^2) are measured between each pair of classes, and classes are split up into two groups: independent classes, which are trained using `IC`, and dependent classes which are trained all-together by a single `CC` classifier. This method assumes that each label is either independent, or correlated with all other non-independent labels, and does not consider conditional dependence.

All these methods are multi-label approaches that do not deal with multi-dimensional learning and the problems associated with it, namely a potentially much higher number of class-value combinations which makes it more difficult to get a good estimate of any measure of correlation from finite data.

For constructing a Bayesian network, [26] provides an efficient way of measuring pairwise *conditional* dependence in multi-label data, based on the fact that maximising the likelihood of the data is equivalent to minimising the mutual information between the data instances and the error. We modify this measurement strategy for multi-dimensional data; formulated as follows.

Given training examples $\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^N$ and `IC` classifier $\mathbf{h}$; we can obtain the vector of errors for the ith example as:

$$\begin{aligned} \boldsymbol{\epsilon}^{(i)} &= \mathbf{I}(\mathbf{y}^{(i)}, \mathbf{h}(\mathbf{x}^{(i)})) \quad (3)\\ (\epsilon_1^{(i)}, \ldots, \epsilon_d^{(i)}) &= (I(y_1^{(i)}, h_1(\mathbf{x}^{(i)})), \ldots, I(y_d^{(i)}, h_d(\mathbf{x}^{(i)}))), \end{aligned}$$

where $\boldsymbol{\epsilon}^{(i)} \in \{0, 1\}^d$ ($I(a, b)$ is an indicator function returning 1 if $a = b$ and 0 if $a \neq b$). We can then say that the jth and kth classes are *conditionally dependent* iff the errors ϵ_j and ϵ_k are *not* independent on each other.

The original Bayesian-network inspired multi-label method as presented by [26] measures three types of errors for each label: false positive (1 predicted instead of 0), false negative (0 predicted instead of 1), or the correct label predicted (no error). This means an error space of 3×3 for each pair of labels j, k. In a multi-dimensional setting, this would correspond an error space of $K_j \times K_k$ (including the non-error, i.e., the correct classification). This means many degrees of freedom and, clearly, without very large amounts of data, it will be difficult to get good estimates of the dependency between classes. To make the method more appropriate for multi-dimensional data we instead consider the three types of errors $e = 1, 2, 3$ for each pair of class values j from Y_j, and k from Y_k, and calculate the measured and expected frequencies for each e as

e	measured frq. $f_e(j,k)$	expected frq. $E_e(j,k)$
0	$\frac{1}{N}\sum_{i=1}^N \epsilon_j^{(i)}\epsilon_k^{(i)}$	$\frac{1}{N}\sum_{i=1}^N \epsilon_j^{(i)} \cdot \frac{1}{N}\sum_{i=1}^N \epsilon_k^{(i)}$
1	$\frac{1}{N}\sum_{i=1}^N \epsilon_j^{(i)} \oplus \epsilon_k^{(i)}$	$1 - (E_0(j,k) + E_2(j,k))$
2	$\frac{1}{N}\sum_{i=1}^N \neg\epsilon_j^{(i)}\neg\epsilon_k^{(i)}$	$\frac{1}{N}\sum_{i=1}^N \neg\epsilon_j^{(i)} \cdot \frac{1}{N}\sum_{i=1}^N \neg\epsilon_k^{(i)}$

where $\oplus$ is the logical exclusive OR operation, and $\neg$ is the logical negation.

Algorithm 1 CONDDEP($\mathbf{h}, \tilde{\mathcal{D}}$) Obtaining a conditional dependency matrix

Input:

- trained IC classifier $\mathbf{h} = (h_1, \ldots, h_d)$
- test instances $\tilde{\mathcal{D}} = \{(\tilde{\mathbf{x}}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{\tilde{N}}$ (with true classifications); $\tilde{N} = |\tilde{\mathcal{D}}|$

Algorithm:

- For $i = 1, \ldots, \tilde{N}$:
  1) calculate each error $\boldsymbol{\epsilon}^{(i)} = \mathbf{I}(\mathbf{y}^{(i)}, \mathbf{h}(\tilde{\mathbf{x}}^{(i)}))$ as in Eq. (3)
- For all pairs $\{(j,k)\}; j < k, j \in \mathcal{Y}_j, k \in \mathcal{Y}_k$:
  1) calculate $\bar{\chi}^2_{j,k}$, as in Eq. (4)
  2) offset $\bar{\chi}^2_{j,k} \leftarrow \bar{\chi}^2_{j,k} - \bar{\chi}^2_{\mathrm{C}}$ as in Eq. (5)

Output:

- $\bar{\boldsymbol{\chi}}^2$ the matrix of all pairwise conditionally dependent significance values

We can then calculate the conditional-dependence chi-squared statistic for these three types of errors:

$$\bar{\chi}^2_{j,k} = \sum_{e \in \{0,1,2\}} \frac{(f_e(j,k) - E_e(j,k))^2}{E_e(j,k)}, \tag{4}$$

where we add the bar to the notation to distinguish the fact that this is a conditional dependence score (unlike Eq. (2) from [21]).

Finally, similarly to [21], we offset each statistic with the critical value:

$$\bar{\chi}^2_{j,k} \leftarrow \bar{\chi}^2_{j,k} - \bar{\chi}^2_{\mathrm{C}}, \tag{5}$$

where we use $\bar{\chi}^2_{\mathrm{C}}$ as the critical value for two degrees of freedom with a p-value of 0.10. If $\bar{\chi}^2_{j,k} > 0$ then the class values j and k can be considered conditionally dependent.

Algorithm 1 describes the process of creating a matrix of pairwise conditionally-dependent significance values $\bar{\boldsymbol{\chi}}^2$ for all class pairs.

Given the $\bar{\boldsymbol{\chi}}^2$ statistic for all of class pairs, we can calculate a score for any class-set partition $\theta = \{S_1, \ldots, S_{|\theta|}\}$ like in Eq. (2).

The score we have obtained here is based on the *conditional* label dependence, as we have taken into account the input space. This calculation is much faster than other methods that measure conditional class dependencies such as building and evaluating a model for each θ (as in [21]).

This calculation is fast enough to be able to try many different partitions, but the potentially huge number of possible partitions means that it will still be infeasible to search through them iteratively in many cases. On the other hand, a random search (as suggested by [21] for unconditional dependencies) does not take advantage of the relationship between partitions: if $\theta = \{(0,3),(2),(1)\}$ is good, then it makes sense to next try, for example, $\theta' = \{(0,3,2),(1)\}$ or $\theta' = \{(0,3),(2,1)\}$.

Therefore, we use a *simulated annealing* scheme [13]; starting with a random partition θ, and progressively mutating it over a series of steps, and gradually reducing the probability for "uphill" moves. To mutate a partition $\theta = \{S_1, \ldots, S_{|\theta|}\}$ into θ' (i.e., a Markov step within the partition space) with $p(\theta'|\theta)$, we select some $j \in \mathcal{U}\{1, \ldots, d\}$ and some $l \in \mathcal{U}\{1, \ldots, |\theta|\}$; if $j \in S_l$ then we move it into a new set $\{j\}$, else we move j into the existing set S_l.

Simulated annealing is not guaranteed to find a global optimum, but in our experience it will usually find a good local optimum in this scenario. Furthermore, through empirical exploration we found that usually there are many (although quite different) good possible partitions, and that it is much more effective to make use of several good partitions than to invest in trying to find a single good one that may be slightly better. This leads to the introduction of an ensemble scheme, which we present in Section 5.

The most expensive part of Algorithm 1 requires $3(d(d-1)/2)$ operations over the error data (each of the 3 types of error is assessed for all pairwise combinations. Due to the computational simplicity of these operations, even for relatively large values of d (100 or so), this will not present an obstacle (and only requires storing $(d(d-1)/2)$ values) compared with the relatively much higher complexity in building a classifier.

As an optional second part to the algorithm, we can fine tune the partition θ with internal validation; based on the idea that an actual trained model will provide the most accurate gauge of final performance. We already explained that internal building and validation is too slow to explore the partition space in most cases, but we can assume that our simulated annealing scheme brings us close to a maximum (or at least a good local maximum), and uses a much smaller number of iterations to 'fine tune' the partition. In this phase we mutate the set in the same way as before, but this time we use the internal train/test split to build and evaluate the model, and we always accept it if it is better than the previous.

Algorithm 2 details the full algorithm for creating a super-class classifier (SC) from a given training set. Any multi-dimensional classifier can be used to learn the super-classes as if they were ordinary classes. IC is an obvious option, but any multi-dimensional method can be applied, such as a classifier chain (CC). $T + T'$ is the total number of partitions we look at. If $T' > 0$ (in the second step using internal validation), then we denote this as SC$'$.

The super-class method should perform better than either IC or CP. However, because of our method's close relation to CP (training several classes as a single class) it may suffer from some of the same problems, depending on the size of the super-classes, such as overfitting, and running-time issues. In the following section we introduce a filter for any multi-dimensional dataset (or class-subset thereof) which improves both the predictive and time performance of CP-like methods, and makes super-class classifiers more applicable to many real-world problems.

4 A Nearest-Neighbour Replacement Filter for Multi-Dimensional Training Data

A major issue with super-classes, particularly in the multi-dimensional domain, is the number of possible values they can take. Combining two classes will create up to $K_j \times K_k$ possible values. This means that some of the disadvantages

Algorithm 2 Super-class classifier (SC) construction; producing $\mathbf{h}_\theta$ parameterised by θ

Input:

- A training set $\mathcal{D} = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{N}$
- A function $p(\theta'|\theta)$ which mutates θ
- A function $\pi(\theta)$ which evaluates θ according to $\bar{\chi}^2$, see Eq. (2)
- A function $q(\mathbf{h}_\theta, \tilde{\mathcal{D}})$ which evaluates $\mathbf{h}_\theta$ on $\tilde{\mathcal{D}}$ (required only when $T' > 0$)

Algorithm:

- Create internal train $\mathcal{D}_{\text{train}} \subset \mathcal{D}$ and test $\mathcal{D}_{\text{test}} \subset \mathcal{D}$ sets (where $\mathcal{D} = \mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{test}}$)
- Train a standard IC classifier $\mathbf{h}$ on $\mathcal{D}_{\text{train}}$
- Create matrix $\bar{\chi}^2 = \text{CONDDEP}(\mathbf{h}, \mathcal{D}_{\text{test}})$
- Randomly generate an initial partition θ
- for $t = 1, \ldots, T$:
  1) $\theta' \sim p(\theta'|\theta)$
  2) draw a uniform random number $u \sim \mathcal{U}([0, 1])$
  3) if $\min\left[1, \exp(|\pi(\theta') - \pi(\theta)|\frac{1}{t})\right] > u$:
     - $\theta \leftarrow \theta'$ // accept
- if $T' > 0$
  1) Train $\mathbf{h}_\theta$ on $\mathcal{D}_{\text{train}}$
- for $t = 1, \ldots, T'$:
  1) $\theta' \sim p(\theta|\theta_t)$
  2) Train $\mathbf{h}_{\theta'}$ on $\mathcal{D}_{\text{train}}$
  3) if $q(\mathbf{h}_{\theta'}, \mathcal{D}_{\text{test}}) > q(\mathbf{h}_\theta, \mathcal{D}_{\text{test}})$:
     - $\mathbf{h}_\theta \leftarrow \mathbf{h}_{\theta'}$ // accept
- Train $\mathbf{h}_\theta$ on $\mathcal{D}$

Output:

- Super-class classifier $\mathbf{h}_\theta$

pertaining to CP are still relevant: fewer examples per value which leads to higher complexity, overfitting, and difficulty in learning a concept.

To make training super-classes feasible, we use a *nearest-neighbour replacement* filter (NNR) to reduce the number of values associated with each class in the training data. A related mechanism was introduced in [17] called "pruning and subsampling", but this is only suitable for multi-label data. Here we develop a more advanced version which is suitable for multi-dimensional data.

The NNR filter can be applied on any multi-dimensional dataset $\mathcal{D}$, or a column-wise class subset of this original dataset, i.e., having classes $S \subseteq \{1, \ldots, d\}$. However, for simplicity and generality, we just refer to $\mathcal{D}$ in the following explanation.

The idea of NNR is straightforward: identify all *p-infrequent* class-values and replace them with their *n*-most-frequent nearest neighbours.

Definition 1. *The frequency of a class-value* $\mathbf{y}$ *in* $\mathcal{D}$ *is:*

$$\sum_{i=1}^{N} I(\mathbf{y}, \mathbf{y}^{(i)}).$$

Definition 2. *A class value* $\mathbf{y}$ *is p-infrequent in* $\mathcal{D}$ *if:*

$$\sum_{i=1}^{N} I(\mathbf{y}, \mathbf{y}^{(i)}) \leq p.$$

Algorithm 3 outlines the NNR filter. Basically, it replaces any examples $(\mathbf{x}, \mathbf{y})$ that have an infrequent $\mathbf{y}$, with examples $(\mathbf{x}, \mathbf{y}_1), \ldots, (\mathbf{x}, \mathbf{y}_n)$ where each $\mathbf{y}_i$ *is* frequent in the data and has a Hamming distance from $\mathbf{y}$ by at most 1. Each of these new examples is given a weight of $\frac{1}{n}$ (not shown in the pseudo-code).

This means that noise is introduced at a cost of reducing the number of class values. However, we expect classification to improve, since we introduce only one bit of noise for each new example created and, as we explain in the following section, some noise is not necessarily a problem and can even be beneficial. That is to say, the increased learning ability of the classifier by having a higher ratio of data instances to class values will counteract the small amount of noise while at the same time, the complexity of the classifier is greatly reduced.

Fig. 3 shows the effect of p and n in practice. With p, the number of instances stays relatively constant, whereas the number of class values drops rapidly (and with it – running time). In this case, accuracy stays constant or increases until around $p = 7$ and best results are obtained between $p = 3$ and $p = 6$. Any value $n > 0$ (for fixed $p = 3$) exceeds the original accuracy, whereas the effect on running time is less influential: It is negligibly increased until $n \geq 4$ (thereupon the maximum number of possible neighbours is reached). In this example it is clear that NNR is beneficial, both in terms of speed and accuracy.

It is possible to choose p to control the number of distinct class values (that have a frequency of 1). And thus enforce a maximum complexity of this number instead of $O(\min(N, K^d))$. However, in practice, we have not found the need for this, since even small values of p will greatly reduce the number of classes and thus the running time of whichever base classifier is used.

Recall that in the super-class context, we have a partition of the class space $\theta = \{S_1, \ldots, S_{|\theta|}\}$ where each S represents a set of classes. Then we define the dataset containing only these classes as $\mathcal{D}_S = \{(\mathbf{x}^{(i)}, \mathbf{y}_S^{(i)})\}_{i=1}^{N}$. This is the dataset used to build $\mathbf{h}_S$. Before training each individual super-class classifier we first pass the data through NNR prior to training; such that for each $S \in \theta$:

$$h_{\mathbf{s}}{:}\mathcal{X}_1, \ldots, \mathcal{X}_m \rightarrow \text{DISTINCT}\{\text{NNR}(\{\mathbf{y}_S^{(1)}, \ldots, \mathbf{y}_S^{(N')}\}, p, n)\} \approx \mathcal{Y}_{S_1} \times \ldots \times \mathcal{Y}_{S_{|S|}}, \quad (6)$$

where $\text{NNR}(\mathcal{D}, p, n)$ is the set of all $\mathbf{y}$ from a dataset processed by NNR (Algorithm 3). N' is the number of examples in the output dataset D', not necessarily the same as N. Given some super-class classifier $\mathbf{h}_\theta = \mathbf{h}_{S_1}, \ldots, \mathbf{h}_{S_{|\theta|}}$ and data $\mathcal{D}$, we train each $\mathbf{h}_S$ on data $\text{NNR}(\mathcal{D}_S, p, n)$.

There are possible scenarios where NNR will appear to have difficulty. Referring back to the toy dataset in Table 1, imagine that we have a super-class $Y_{2,3}$ (modelling income and profession together as a single variable). If there is a single entry {student, high}, NNR with $p = 3, n = 1$ will replace it with a new example, such as {student, low}. This has clearly introduced the wrong concept. However

Algorithm 3 The NNR filter (NNR). Examples with p-infrequent class combinations are replaced with their n-most-frequent nearest neighbours in a dataset; where DIST is a Hamming distance function: $\text{DIST}(\mathbf{y}', \mathbf{y}) = \sum_{j=1}^{d} I(y'_j, y_j)$.

Input:

- A dataset $\mathcal{D}$
- p parameter defining the p-frequency
- n parameter defining the number of nearest neighbours to use

Algorithm:

1) build $\phi(\mathbf{y}) :=$ a map which returns the frequency of $\mathbf{y}$ in $\mathcal{D}$
2) $\mathcal{D}' \leftarrow \{\}$
3) `for` $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}$:
   - `if` $\phi(\mathbf{y}) \leq p$ (i.e., $\mathbf{y}$ is p-infrequent in $\mathcal{D}$):
     - $V = (\mathbf{y}'_1, \ldots, \mathbf{y}'_o)$ where $\forall \mathbf{y}' : \exists (\mathbf{x}', \mathbf{y}') \in \mathcal{D}, \text{DIST}(\mathbf{y}, \mathbf{y}') \leq 1$ (is in $\mathcal{D}$ and like $\mathbf{y}$) and where $\forall 1 \leq q < r \leq o : \phi(\mathbf{y}'_q) > \phi(\mathbf{y}_r)$ (sorted by frequency)
     - $V \leftarrow V_{1:n}$ take the top n elements
     - $\forall \mathbf{y} \in V : \mathcal{D}' \leftarrow \mathcal{D}' \cup (\mathbf{x}, \mathbf{y}')$
   - `else`:
     - $\mathcal{D}' \leftarrow \mathcal{D}' \cup (\mathbf{x}, \mathbf{y})$

Output:

- Dataset $\mathcal{D}'$

we probably have lots of examples for {`student`, `low`} (we are guaranteed to have at least p), and the wrong information will just become noise. If we, for some reason, had many examples of {`student`, `high`}, it would not have been pruned in the first place!

If a rare kind of instance occurs in the test data, it is still possible to make a correct classification by way of a voting scheme. We introduce such a scheme in the following section, based on the principles of the well-known bootstrap aggregation (*Bagging*) procedure [4].

As a final remark on NNR, note that it is more suited to super-class partitions than not having partitions: the more dependent class variables are on each other the more likely the super-class values are to pertain to a few core combinations, and the super-class partitions are based precisely upon class dependence.

5 An Ensemble of Super-Class Classifiers

Ensembles are known for increasing the power of base classifiers, and have been used prolifically in the multi-label literature (e.g., [18], [23], [25]). They are also ideal for reducing overfitting when the base classifier is particularly affected by relatively small variations in the training data. This is the case in our super-class methods, with respect to class dependencies ($\bar{\chi}^2$), as well as our NNR filter with respect to the frequency of certain class-value combinations. In particular, we cannot expect a single super-class partition to represent all the class dependencies in a dataset; however an ensemble of these, each with a slightly different

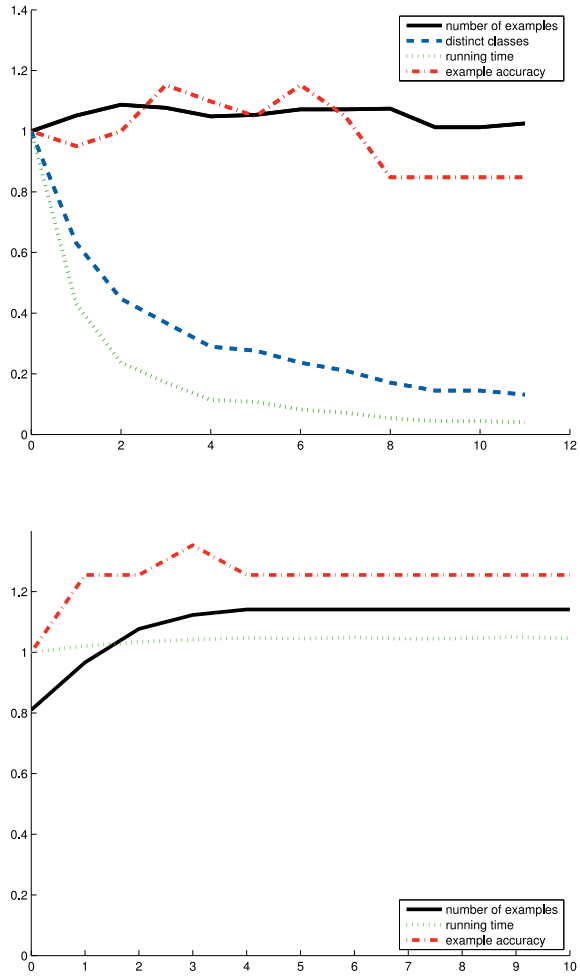


Fig. 3. Top: NNR on the Parkinson's dataset for varying values of p (horizontal axis), for $n = 2$. At $p = 0$ NNR is disabled; all other values (accuracy, running time, etc.) are plotted as *proportional* to those obtained at $p = 0$. In this example, we consider a single super-class of all 5 class variables. Bottom: same, but for varying values of n (horizontal axis), for $p = 3$.

partition, can arrive much closer to this goal. Hence, we look at *ensembles* of super-classes classifiers (ESC).

A Bagging ensemble [4] involves creating M new training sets; each training set is formed by sampling with replacement from the original training data N' times (typically $N' = N$, but not necessarily so). That is to say, some examples will probably be duplicated in the new dataset of N' examples. NNR, as described in the previous section, already samples with replacement (whenever class combinations are infrequent) and thus we already benefit from the advantages associated with Bagging. For this reason, we only take a random cut of the original training set for each ensemble member *without* replacement, knowing that NNR will duplicate some examples. We take specifically 67%; but note that this number is not directly in relation with the 63.2% expected number of duplicate examples under Bagging where $N = N'$, rather it is our experience that this number (or thereabouts) tends to yield approximately $N \approx N'$ in practice, as in Fig. 3.

It also common in ensembles to introduce variation into the individual models. In our case, we use a different random seed (and thus start with a different initial partition) for each super-class; and in NNR we use model parameter

$p \sim \mathcal{U}\{1, \ldots, 5\}$ (a random value between 1 and 5 inclusive for each model).

Each model of a multi-dimensional ensemble classifier returns a probability mass distribution for each jth class, for any test instance $\tilde{\mathbf{x}}$. For the mth model and the jth class, we get a vector, which in the probabilistic case is

$$\mathbf{w}_j^{(m)} = \big(p(Y_j = 1|\tilde{\mathbf{x}}), \ldots, p(Y_j = K_j|\tilde{\mathbf{x}})\big)$$

such that $w_{j,v}^{(m)} \equiv p(Y_j = v|\tilde{\mathbf{x}})$ (or approximation thereof, if the base classifier is not probabilistic), i.e., the probability that the jth class takes value $v \in \{1, \ldots, K_j\}$ according to the mth model. As the final classification for the jth class for a test instance, we simply assign the value which $m = 1, \ldots, M$ models, to predict

$$\hat{y}_j = \underset{v=1,\ldots,K_j}{\operatorname{argmax}} \sum_{m=1}^{M} w_{j,v}^{(m)}.$$

This voting process is particularly helpful for offsetting effects of any noise introduced by NNR in our superclass scheme. Although NNR may purge a low-frequency example from the training data, the ensemble can recover this combination for a test instance by votes for any of its parts. Continuing the example at the end of Section 4, {student, high} could be recovered by strong voting for combinations involving *either* of these class values.

That said, this prediction-phase is a generic procedure for any multi-dimensional ensemble method. It is similar to probabilistic method that [1] found to work well for Bagging ensembles in a single-dimensional context.

6 Experiments

We conduct an empirical evaluation on a range of real-world datasets, comparing our methods with the baseline independent classifiers (IC) as well as competitive methods from the literature, under two contrasting measures of predictive performance, and an analysis of running times.

We used the MEKA framework (http://meka.sourceforge.net): an open-source Java framework based on the WEKA framework for machine learning [10], adding support for multi-label and multi-dimensional classification and evaluation. The source code for all methods in this evaluation will be made available within MEKA. We also used the CLUS framework for one of the algorithms from the literature, although we ported these results into MEKA's evaluation.

In all experiments we randomise the order of instances in the datasets and carry out 5-fold cross-validation.

6.1 Evaluation Metrics

We evaluate the predictive performance of methods using two metrics; *example accuracy*, which considers a vector of class values as a single classification (that can either be fully correct or incorrect):

$$\text{EXAMPLE ACCURACY} = \frac{1}{N} \sum_{i=1}^{N} I(\hat{\mathbf{y}}^{(i)}, \mathbf{y}^{(i)})$$

TABLE 2
Sample of Multi-Dimensional Datasets and Their Associated Statistics: Number of Examples N, Number of Classes d, Number of Values per Target K, and Number of Attributes m

	N	d	K	m
Solar Flare	323	3	5	$10x$
Bridges	107	5	2–6	$7x$
Thyroid	9172	7	2–5	$7n$, $20b$, $1x$
Parkinson's	488	5	3	$18n$, $1b$, $39x$
Music	593	6	2	$72n$
Scene	2407	6	2	$294n$
Yeast	2417	14	2	$103n$
Enron	1702	53	2	$1001b$
TMC07	28596	22	2	$500b$

n, b, and x indicate numeric, binary, and nominal attributes, respectively.

We have separated the multi-label datasets (where K = 2) with a horizontal line.

and *class accuracy*, which is the average accuracy of each class (scored separately):

$$\text{CLASS ACCURACY} = \frac{1}{d} \sum_{j=1}^{d} \frac{1}{N} \sum_{i=1}^{N} I(\hat{y}_j^{(i)}, y_j^{(i)}),$$

where $I(a, b)$ is the indicator function as used also earlier.

A relatively high result for EXAMPLE ACCURACY means that class dependencies are being taken into account. On the other hand, a relatively high result for CLASS ACCURACY means that each dimension is being predicted well individually, but the combinations of all predicted classes may contain conflicting results. Generally, we would expect IC to perform relatively better under CLASS ACCURACY, since EXAMPLE ACCURACY tends to reward CP-like methods. This is exactly what we saw in Fig. 2.

Many evaluation metrics commonly used in multi-label evaluation (such as F-measure metrics) are not suitable for the multi-dimensional domain where outputs are not necessarily binary, nor can be thought of as being 'retrieved' or not.

We also consider running time (training testing time) in seconds. All experiments are run on Intel Xeon 3.16GHz CPUs allowing up to 2GB of RAM in each case.

6.2 Datasets

Table 2 displays the datasets we use. Solar Flare (categorising solar flares), Bridges (estimating bridge properties from certain constraints) and Thyroid (estimating types of thyroid problems given patient attributes) are from the UCI collection [9]. Parkinson's (determining the classes of disabilities incurred by Parkinson's patients) was used by [3]. Unfortunately, there are not yet many publicly available standardised multi-dimensional datasets, so we boost our collections with some of the datasets most commonly used in the multi-label literature: Music (labelling tracks with emotions), Scene (labelling images with scene concepts), Yeast (genes are associated with multiple biological functions), Enron (Labelled e-mail messages from the *Enron* corpus), and TMC07 (aviation reports diagnosed with multiple problems); used and described in, for example, [6], [17], [19], [23].

The three target attributes of Solar Flare correspond to types of solar flares seen in a 24 hour period. In Bridges, bridge design properties are predicted based on specification properties. Thyroid and Parkinson's are medical datasets. In Music, pieces of music are associated with various emotions. Scene is an image annotation problem. Yeast is a biological dataset where genes are associated with (potentially multiple) biological functions.

6.3 Methods and Parameters

From the novel material in this paper we setup the following methods:

1) ECP: an ensemble of 10 class-powerset classifiers, using NNR for tractability
2) ESC: an ensemble of 10 super-class classifiers with NNR, $T = 1000, T' = 0$
3) ESC′: an ensemble of 10 super-class classifiers with NNR, $T = 1000, T' = 10$

all with $p \sim \mathcal{U}\{1, \ldots, 5\}$ and $n = 2$ for each instantiation of NNR, with a cut of 67%.

We compare to [25]'s ensembles of Bayesian classifier chains (EBCC); [14]'s EPCT: a Bagging ensemble of predictive clustering decision trees; and to ECC (ensembles of classifier chains) from [19]. Additionally, we compare to some well-known methods from the multi-label literature (where appropriate, on the multi-label datasets): [23]'s RAkEL and [5]'s Instance-Based Logistic-Regression method (IBLR).

A variant of EPCT and ECC (which we mentioned in Section 2.1) were recently rated among the highest performing in the multi-label literature by the extensive empirical evaluation of [15]. As it happens, both methods are directly applicable to multi-dimensional data, and thus make an ideal comparison for our experiments. EBCC is one of the few methods focussing exclusively on multi-dimensional data.

Note that [14]'s PCT method performs best under a random forest paradigm, at least in the comparison of [15]. We use a standard bagging scheme so as to compare directly with the other ensemble methods in comparison (which are all bagging schemes). In any case, we found that the difference between bagging and random forest is marginal compared to difference between the different methods used in our comparison (see the results and following discussion).

Additionally, by comparing to ECP, we will be able to see if our super-class methods (ESC and ESC′) are justifiable; and furthermore, ECP provides a good approximation of the baseline class powerset method CP, which is otherwise not a viable option due to its computational complexity.

We use both Support Vector Machines (SVMs) and Naive Bayes as the base multi-class classifier[2] on all problem-transformation methods. Note that this naturally excludes the algorithm-adaptation methods EPCT (decision trees) and IBLR (instance-based logistic regression).

2. Using the implementations provided in the WEKA framework with default parameters; binary SVMs are made multi-class capable with a pair-wise implementation – it is recommended to tune parameters for maximum performance.

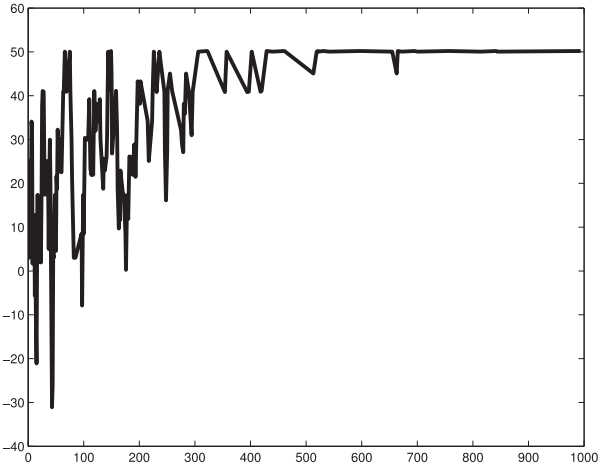


Fig. 4. Score (see Eq. (2)) of the partition in selection at each step $t = 1, \ldots, T$ of our simulated-annealing scheme, on the Music data. Note that the partition space is explored more liberally at first, before settling in to a maximum.

We use $M = 10$ models in each ensemble (as found to work well in, for example, in [19]), except for EBCC where the authors specifically recommend using d models (where the j-th class node is the root in the j-th model). We use the probabilistic voting scheme described in Section 5.

6.4 Results

Tables 3 and 5 display the mean results for predictive performance with the rank for each method per dataset, and their average rank over all datasets, for multi-dimensional and multi-label datasets respectively. We conducted the Nemenyi test [8] (with a significance level of $p = 0.1$) on these rankings, and display the results as $a \succ b$, indicating that algorithm a is found to have statistically better performance than algorithm b. Table 4 displays the average running times of all methods.

Fig. 4 shows the progress of one of ESC's model's simulated annealing search on a particular run on Music. We remark that ESC′ updated the partition twice out of the $T' = 10$ additional internal validation steps.

Table 6 illustrates some of the individual contributions of the different steps in our approach.

Table 7 shows the partitions found by a SC and SC′ model for five cross-validation folds of the data, on a selection of datasets.

6.5 Discussion

Both ESC and ESC′ obtain the best average ranks over all evaluation measures, and they consistently outperform the baseline IC and the competing methods (an improvement which is statistically significant in several cases).

ECP is arguably the third-strongest method. This tells us that our NNR mechanism works well, although our super-class methods provide better – and, at least for ESC, faster – performance. The exceptions to this are on Solar, Music and Parkinson's; but of these, two are much faster under ESC than ECP.

Our methods regularly outperform methods from the literature (ECC, EBCC, EPCT) on all but two datasets; EBCC is best on Bridges and EPCT is best on Thyroid. The

TABLE 3
Average Results for Multi-Dimensional Datasets Over 5-fold Cross-Validation with Rankings

(a) EXAMPLE ACCURACY - SVMS

Dataset	BR	ECC	EBCC	EPCT	ECP	ESC	ESC′
Solar	0.817 (2)	0.808 (6)	0.817 (2)	0.796 (7)	0.820 (⋆)	0.817 (2)	0.817 (2)
Bridges	0.186 (3)	0.159 (5)	0.168 (4)	0.120 (7)	0.131 (6)	0.187 (2)	0.206 (⋆)
Thyroid	0.781 (7)	0.785 (2)	0.782 (5)	0.944 (⋆)	0.782 (5)	0.784 (3)	0.784 (3)
Parkinson's	0.164 (6)	0.172 (5)	0.154 (7)	0.201 (4)	0.224 (2)	0.234 (⋆)	0.215 (3)
avg. rank	4.50	4.50	4.50	4.75	3.50	2.00	2.25

(b) CLASS ACCURACY - SVMS

Dataset	BR	ECC	EBCC	EPCT	ECP	ESC	ESC′
Solar	0.919 (4)	0.917 (6)	0.919 (4)	0.912 (7)	0.924 (⋆)	0.923 (2)	0.923 (2)
Bridges	0.705 (⋆)	0.688 (5)	0.689 (4)	0.678 (6)	0.648 (7)	0.705 (⋆)	0.703 (3)
Thyroid	0.966 (2)	0.966 (2)	0.966 (2)	0.990 (⋆)	0.966 (2)	0.966 (2)	0.966 (2)
Parkinson's	0.677 (6)	0.696 (5)	0.670 (7)	0.702 (3)	0.702 (3)	0.718 (⋆)	0.715 (2)
avg. rank	3.25	4.50	4.25	4.25	3.25	1.50	2.25

(c) EXAMPLE ACCURACY - NAIVE BAYES

Dataset	BR	ECC	EBCC	EPCT	ECP	ESC	ESC′
Solar	0.777 (6)	0.786 (5)	0.774 (7)	0.796 (⋆)	0.792 (2)	0.789 (3)	0.789 (3)
Bridges	0.197 (5)	0.216 (⋆)	0.216 (⋆)	0.120 (7)	0.149 (6)	0.216 (⋆)	0.216 (⋆)
Thyroid	0.587 (6)	0.625 (5)	0.581 (7)	0.944 (⋆)	0.821 (2)	0.818 (4)	0.819 (3)
Parkinson's	0.195 (6)	0.199 (5)	0.193 (7)	0.201 (4)	0.209 (3)	0.211 (⋆)	0.211 (⋆)
avg. rank	5.75	4.00	5.50	3.25	3.25	2.25	2.00

(d) CLASS ACCURACY - NAIVE BAYES

Dataset	BR	ECC	EBCC	EPCT	ECP	ESC	ESC′
Solar	0.893 (5)	0.888 (6)	0.878 (7)	0.912 (⋆)	0.911 (2)	0.910 (3)	0.905 (4)
Bridges	0.718 (2)	0.712 (3)	0.722 (⋆)	0.678 (4)	0.663 (5)	0.663 (5)	0.663 (5)
Thyroid	0.925 (6)	0.935 (5)	0.923 (7)	0.990 (⋆)	0.970 (2)	0.969 (4)	0.970 (2)
Parkinson's	0.677 (5)	0.676 (6)	0.642 (7)	0.702 (⋆)	0.700 (2)	0.696 (4)	0.697 (3)
avg. rank	4.50	5.00	5.50	1.75	2.75	4.00	3.50

Best values are marked with a (⋆).

TABLE 4
Average Running Times, in Seconds, for All Problem-Transformation Methods Under SVMs Except Enron (C4.5 Decision Trees) and TMC07 (Naive Bayes); and EPCT

	IC	ECC	EBCC	EPCT	ECP	ESC	ESC′
Solar	0.2	1.1	1.3	*0.1*	0.8	4.3	17.8
Bridges	0.3	2.1	2.5	*0.1*	2.7	17.2	75.0
Thyroid	35.3	189.2	196.8	*3.7*	136.2	155.5	730.7
Parkinson's	2.0	10.1	17.3	*0.4*	65.5	19.8	91.3
Music	*0.3*	2.8	0.5	1.2	30.1	8.2	65.9
Scene	*17.2*	54.5	90.2	19.3	23.2	52.5	195.7
Yeast	*9.4*	79.0	28.1	130.0	325.2	119.5	457.6
Enron	*95.2*	613.5	203.2	102.3	5902.7	3400.2	21610.7
TMC07	*33.3*	299.0	1185.5	152.5	538.8	869.4	5091.9

Fastest times are marked in italics.

classification paradigm has a big effect on predictive performance depending on the problem domain. Decision trees are clearly the better option for the Thyroid data, and Naive Bayes on Bridges. The algorithm adaptation methods (EPCT, IBLR) perform relatively strongly against the transformation methods when the latter employ Naive Bayes; as is to be expected. If SVMs are not used in the problem-transformation methods, other methods such as EPCT appear much more attractive, particularly in view of its good time performance. This aside, other trends between Tables 3 (SVMs) and 5 (Naive Bayes) are the same: the ESC methods perform strongly, and consistently outperform the classifier-chains methods.

As a side note, it is interesting to see the difference between classifier-chain and super-class methods on several of the datasets (e.g., Enron, TMC07) under Naive Bayes (the chain methods performing noticeably poorer).

ESC′ improves on ESC in some cases; thus the extra computational time invested in fine-tuning the class partitions by this method can offer some benefit. On the other hand, ESC′ does not always improve on ESC, for example on the Parkinson's data and under CLASS ACCURACY

TABLE 5
Average Results for Multi-Label Datasets over 5-fold Cross-Validation with Rankings

(a) EXAMPLE ACCURACY - SVMS

Dataset	BR	ECC	EBCC	EPCT	RAKEL	IBLR	ECP	ESC	ESC′
Music	0.270 (9)	0.324 (5)	0.323 (6)	0.313 (8)	0.345 (2)	0.316 (7)	0.343 (3)	0.338 (4)	0.356 (⋆)
Scene	0.527 (9)	0.641 (4)	0.567 (7)	0.565 (8)	0.606 (6)	0.637 (5)	0.698 (3)	0.702 (2)	0.705 (⋆)
Yeast	0.149 (9)	0.204 (4)	0.177 (7)	0.153 (8)	0.184 (6)	0.199 (5)	0.248 (3)	0.256 (⋆)	0.256 (⋆)
Enron	0.108 (8)	0.112 (6)	0.117 (5)	0.124 (4)	0.109 (7)	0.084 (9)	0.166 (⋆)	0.149 (2)	0.149 (2)
TMC07	0.287 (6)	0.302 (2)	DNF	0.249 (7)	0.290 (4)	0.233 (8)	0.301 (3)	0.288 (5)	0.306 (⋆)
avg. rank	8.20	4.20	6.25	7.00	5.00	6.80	2.60	2.80	1.20

Nemenyi signif.: ECP≻BR; ESC≻BR; ESC′≻BR; ESC′≻EBCC; ESC′≻EPCT; ESC′≻IBLR;

(b) CLASS ACCURACY - SVMS

Dataset	BR	ECC	EBCC	EPCT	RAKEL	IBLR	ECP	ESC	ESC′
Music	0.809 (7)	0.811 (6)	0.814 (3)	0.806 (9)	0.818 (⋆)	0.812 (4)	0.809 (7)	0.812 (4)	0.816 (2)
Scene	0.894 (8)	0.905 (5)	0.893 (9)	0.900 (7)	0.903 (6)	0.910 (4)	0.917 (⋆)	0.916 (2)	0.916 (2)
Yeast	0.801 (2)	0.798 (5)	0.793 (8)	0.789 (9)	0.797 (6)	0.799 (4)	0.796 (7)	0.801 (2)	0.802 (⋆)
Enron	0.939 (7)	0.947 (5)	0.939 (7)	0.951 (3)	0.940 (6)	0.939 (7)	0.950 (4)	0.953 (⋆)	0.953 (⋆)
TMC07	0.942 (⋆)	0.872 (8)	DNF	0.931 (6)	0.939 (2)	0.929 (7)	0.937 (3)	0.935 (5)	0.936 (4)
avg. rank	5.00	5.80	6.75	6.80	4.20	5.20	4.40	2.80	2.00

(c) EXAMPLE ACCURACY - NAIVE BAYES

Dataset	BR	ECC	EBCC	EPCT	RAKEL	IBLR	ECP	ESC′	ESC
Music	0.189 (9)	0.214 (7)	0.211 (8)	0.313 (2)	0.248 (6)	0.316 (⋆)	0.287 (4)	0.250 (5)	0.294 (3)
Scene	0.171 (9)	0.179 (7)	0.173 (8)	0.565 (2)	0.524 (6)	0.637 (⋆)	0.558 (3)	0.549 (5)	0.550 (4)
Yeast	0.096 (9)	0.113 (6)	0.101 (8)	0.153 (5)	0.110 (7)	0.199 (3)	0.205 (2)	0.196 (4)	0.206 (⋆)
Enron	0.002 (9)	0.005 (7)	0.003 (8)	0.124 (4)	0.009 (6)	0.084 (5)	0.166 (⋆)	0.153 (3)	0.154 (2)
TMC07	0.121 (7)	0.119 (8)	0.116 (9)	0.249 (4)	0.212 (6)	0.233 (5)	0.309 (⋆)	0.308 (2)	0.308 (2)
avg. rank	8.60	7.00	8.20	3.40	6.20	3.00	2.20	3.80	2.40

Nemenyi signif.: EPCT≻BR; IBLR≻BR; IBLR≻EBCC; ECP≻BR; ECP≻EBCC; ESC≻BR; ESC≻EBCC;

(d) CLASS ACCURACY - NAIVE BAYES

Dataset	BR	ECC	EBCC	EPCT	RAKEL	IBLR	ECP	ESC	ESC′
Music	0.743 (9)	0.749 (7)	0.746 (8)	0.806 (2)	0.771 (6)	0.812 (⋆)	0.779 (3)	0.778 (4)	0.778 (4)
Scene	0.759 (9)	0.764 (7)	0.760 (8)	0.900 (2)	0.877 (3)	0.910 (⋆)	0.870 (4)	0.867 (5)	0.867 (5)
Yeast	0.699 (8)	0.702 (7)	0.694 (9)	0.789 (2)	0.739 (6)	0.799 (⋆)	0.774 (3)	0.773 (4)	0.769 (5)
Enron	0.809 (8)	0.824 (7)	0.809 (8)	0.951 (⋆)	0.926 (6)	0.939 (5)	0.948 (2)	0.944 (3)	0.943 (4)
TMC07	0.879 (7)	0.872 (9)	0.874 (8)	0.931 (4)	0.921 (6)	0.929 (5)	0.937 (⋆)	0.936 (3)	0.937 (⋆)
avg. rank	8.20	7.40	8.20	2.20	5.40	2.60	2.60	3.80	3.80

Nemenyi signif.: EPCT≻BR; EPCT≻ECC; EPCT≻EBCC; IBLR≻BR; IBLR≻EBCC; ECP≻BR; ECP≻EBCC;

Best values are marked with a (⋆).

(the latter case is not surprising since the extra internal step of ESC′ is set to maximise EXAMPLE ACCURACY, although this could obviously be changed). Using internal folds of cross-validation (rather than a simple train/test split) may give improved results for ESC′, although this would lead to longer training times, and this method is already the slowest overall. It is arguable that the partition choice of ESC is sufficient and, for most real-world applications, ESC′ is probably not worth the extra computational expense. This is also clear from Table 7: the partition found by SC is typically minimally changed by the second-stage internal-validation iterations of SC′. This is of course a positive result; it tells us that our CONDDEP method is an efficient and effective way to model conditional dependencies and create partitions based upon them.

EPCT obtains very fast running times, since each ensemble model is only a single tree model for all classes. The difference is particularly noticeable on the larger datasets.

ESC can perform slower than ECP, however, the fact it can also be faster is impressive considering the extra overhead carried out by ESC: it builds an IC classifier internally for each ensemble model, calculates all pairwise dependency significance values and uses them to score 1000 random partitions. It is clear, then, that our super-class approach can have a significant effect in reducing time complexity. On Yeast we see that the time taken by ESC is only a third of that of ECP, and on Enron we also see an important reduction.

Table 6 provides a detailed view of the individual contributions to our final approach. We see that a random partition is worse in this case than just using CP (as expected). However, forming good partitions recovers this lost accuracy, while at the same time being more efficient. In fact, even after adding $T' = 10$ extra internal-validation iterations SC is still (marginally) faster than CP – although not in an ensemble. Again we see that perhaps these T' extra iterations are not actually worth the computational

TABLE 6
Individual Contribution of Different Aspects of Our Approach Compared to Baseline IC: CP (No Partitions), $\text{SC}_{T=0}$ (Random Partitions, NNR filter), $\text{SC}_{T=1000}$ (Partitions Created Under the Simulated Annealing Scheme of $T = 1000$ Iterations), $\text{SC}^{T'=10}_{T=1000}$(Additionally Fine-Tuned with $T' = 10$ Iterations of Internal Validation), and Finally in an Ensemble: ESC′ (also in Tables 3 and 5)

Music

	Example Acc.	Class Acc.	Time (s)
IC	0.270 ± 0.061	0.809 ± 0.015	0.315 ± 0.128
CP	0.345 ± 0.072	0.803 ± 0.022	4.882 ± 0.608
$\text{SC}_{T=0}$	0.287 ± 0.072	0.784 ± 0.019	0.560 ± 0.167
$\text{SC}_{T=1000}$	0.346 ± 0.072	0.797 ± 0.023	0.515 ± 0.139
$\text{SC}^{T'=10}_{T=1000}$	0.351 ± 0.084	0.805 ± 0.024	3.448 ± 0.565
ESC′	0.356 ± 0.032	0.816 ± 0.011	65.232 ± 5.768

Parkinson's

	Example Acc.	Class Acc.	Time (s)
IC	0.164 ± 0.023	0.677 ± 0.012	1.995 ± 0.269
CP	0.211 ± 0.025	0.699 ± 0.030	37.039 ± 6.380
$\text{SC}_{T=0}$	0.143 ± 0.051	0.672 ± 0.016	9.491 ± 0.835
$\text{SC}_{T=1000}$	0.207 ± 0.038	0.688 ± 0.027	8.998 ± 1.761
$\text{SC}^{T'=10}_{T=1000}$	0.209 ± 0.030	0.690 ± 0.024	34.351 ± 6.525
ESC′	0.215 ± 0.031	0.718 ± 0.015	91.328 ± 6.125

TABLE 7
Partitions Discovered in Each of Five Folds at Steps for SC (After $T = 1000$ Iterations of Simulated Annealing) and SC′ (After an Additional $T' = 10$ Iterations of Internal Validation) Models

Music

Partition θ of SC	$\lvert\theta\rvert$	Partition θ of SC′	$\lvert\theta\rvert$
(1 2 5) (0) (3) (4)	4	(1 5) (4) (0) (3) (2)	5
(0 5) (2 3 4) (1)	3	(0 1 5) (2 3 4)	2
(4) (0 2 5) (1) (3)	4	(4) (2 5) (0) (1) (3)	5
(3) (0 5) (2 4) (1)	4	(2 4) (0 5) (3) (1)	4
(1 2 5) (0) (4) (3)	4	(1 2 5) (4) (0) (3)	4

Parkinson's

Partition θ of SC	$\lvert\theta\rvert$	Partition θ of SC′	$\lvert\theta\rvert$
(0 1 2 3 4)	1	(0 1 3 4) (2)	2
(0 1 2 3 4)	1	(0 2 3 4) (1)	2
(1 2 3 4) (0)	2	(1 2 4) (0 3)	2
(0 1 2 3 4)	1	(0 1 3 4) (2)	2
(0 1 2 3 4)	1	(1 3 4) (0 2)	2

Yeast

Partition θ of SC	$\lvert\theta\rvert$
(0 1) (2 3 4 5 6 7 8 9 10) (11 12) (13)	4
(2 3 4 5 6 7 8) (0 1) (9 10 11 12) (13)	4
(13) (0 1 9 10 11 12) (2 3 4 5 6 7 8)	3
(0 1 9 10) (2 3 4 5 6 7 8 11 12) (13)	3
(0 1) (9 10) (2 3 4 5 11 12) (6 7 8) (13)	5

Partition θ of SC′	$\lvert\theta\rvert$
(0 1 12) (2 3 4 5 6 7 8 9 10) (11 13)	3
(2 3 4 5 6 7 8) (0 1 12 13) (9 10 11)	3
(12) (0 1 10 11) (2 3 4 5 6 7 8 9) (13)	4
(0 1 9 10 13) (2 3 4 5 6 7 8 11 12)	2
(0) (1 9 10) (2 3 4 5 8 12) (6 11) (7 13)	5

expenditure, particularly reflecting upon the minimal difference between ESC and ESC′ in Table 3, with the former being much faster.

An additional advantage of our SC methods is that they provide an indication of relationships between classes. Note, for example, how for Music in Table 7 class variables 1, 2 and 5 are grouped together. These correspond in the data to labels happy-pleased, relaxing-calm, angry-aggressive. Also, 2 and 4 (corresponding to relaxing-calm and sad-lonely) often occur together, whereas 3 (quiet-still) often occurs alone. We can speculate that (1 2 5) is based on a mutually exclusive relation (with 5), whereas (2 4) is a strong co-occurrence relation. In Yeast, labels 2, 3, 4, and 5 are inseparable throughout (this could be interpreted by a domain specialist).

7 Conclusion

We presented a method for multi-dimensional classification which creates "super-classes" from a partition in the original set of classes. This is done by using conditional dependence information to efficiently and effectively searching space of possible partitions. In order to make this even more efficient, we presented a filter mechanism to reduce the number of class-combinations in each super-class training set prior to training. This is an important adaptation for working with multi-dimensional data. Finally we created an ensemble of super-class classifiers, and carried out an experimental evaluation on a variety of multi-dimensional data with state-of-the-art methods from the literature. Our methods convincingly performed best overall, and also exhibited competitive running time performance. Additionally our results facilitated an analysis of class conditional dependencies.

Acknowledgments

This work has been partly supported in part by the Spanish Government through projects COMONSENS (CSD2008-00010), TIN2010-20900-C04-04, Consolider Ingenio 2010-CSD2007-00018 and in part by the Cajal Blue Brain of the Spanish Ministry of Economy and Competitiveness (MINECO). The authors also thank the anonymous reviewers for their detailed comments that have lead to the improvement of this manuscript.

References

[1] E. Bauer and R. Kohavi, "An empirical comparison of voting classification algorithms: Bagging, boosting, and variants," *Mach. Learn.*, vol. 36, no. 1–2, pp. 105–139, 1999.
[2] C. Bielza, G. Li, and P. Larrañaga, "Multi-dimensional classification with Bayesian networks," *Int. J. Approx. Reason.*, vol. 52, no. 6, pp. 705–727, Sept. 2011.
[3] H. Borchani, C. Bielza, P. Martínez-Martín, and P. Larrañaga, "Markov blanket-based approach for learning multi-dimensional Bayesian network classifiers: An application to predict the European Quality of Life-5 Dimensions (EQ-5D) from the 39-item Parkinson's Disease questionnaire (PDQ-39)," *J. Biomed. Inform.*, vol. 45, no. 6, pp. 1175–1184, 2012.
[4] L. Breiman, "Bagging predictors," *Mach. Learn.*, vol. 24, no. 2, pp. 123–140, 1996.
[5] W. Cheng and E. Hüllermeier, "Combining instance-based learning and logistic regression for multilabel classification," *Mach. Learn.*, vol. 76, no. 2–3, pp. 211–225, 2009.

[6] K. Dembczyński, W. Cheng, and E. Hüllermeier, "Bayes optimal multilabel classification via probabilistic classifier chains," in *Proc. 27th ICML* , Haifa, Israel, 2010, pp. 279–286.
[7] K. Dembczyński, W. Waegeman, W. Cheng, and E. Hüllermeier, "On label dependence in multi-label classification," in *Workshop Proc. Learn. Multi-Label Data*, Haifa, Israel, 2010, pp. 5–12.
[8] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, Jan. 2006.
[9] A. Andrew. (2010). *Frank and Arthur Asuncion. UCI machine learning repository* [Online]. Available: http://archive.ics.uci.edu/ml
[10] M. Hall *et al.*, "The WEKA data mining software: An update," *SIGKDD Explor.*, vol. 11, no. 1, pp. 10–18, 2009.
[11] P. Hernández-Leal, F. Orihuela-Espina, E. Sucar, and E. F. Morales, "Hybrid binary-chain multi-label classifiers," in *Proc. 6th European Workshop Probabilistic Graphical Models*, 2012.
[12] S.-J. Huang and Z.-H. Zhou, "Multi-label learning by exploiting label correlations locally," in *Proc. Conf. AAAI*, Toronto, ON, Canada, Jul. 2012.
[13] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, May 1983.
[14] D. Kocev, C. Vens, J. Struyf, and S. Dzeroski, "Ensembles of multi-objective decision trees," in *Proc. 18th ECML* (Lecture Notes in Computer Science), Warsaw, Poland, 2007, pp. 624–631.
[15] G. Madjarov, D. Kocev, D. Gjorgjevikj, and S. Dzeroski, "An extensive experimental comparison of methods for multi-label learning," *Pattern Recognit.*, vol. 45, no. 9, pp. 3084–3104, 2012.
[16] J. Read, "Scalable Multi-label Classification," PhD thesis, Univ. Waikato, Hamilton, New Zealand, 2010.
[17] J. Read, B. Pfahringer, and G. Holmes, "Multi-label classification using ensembles of pruned sets," in *Proc. 8th IEEE ICDM*, Pisa, Italy, 2008, pp. 995–1000.
[18] J. Read, B. Pfahringer, G. Holmes, and E. Frank, "Classifier chains for multi-label classification," in *Proc. 20th ECML*, Bled, Slovenia, 2009, pp. 254–269.
[19] J. Read, B. Pfahringer, G. Holmes, and E. Frank, "Classifier chains for multi-label classification," *Mach. Learn.*, vol. 85, no. 3, pp. 333–359, 2011.
[20] J. Struyf, S. Dzeroski, H. Blockeel, and A. Clare, "Hierarchical multi-classification with predictive clustering trees in functional genomics," in *Proc. Workshop Comput. Meth. Bioinformatics 12th Portuguese Conf. Artif. Intell.*, Covilhã, Portugal, 2005, pp. 272–283.
[21] L. Tenenboim, L. Rokach, and B. Shapira, "Identification of label dependencies for multi-label classification," in *Proc. 2nd Int. Workshop Learn. MLD ICML/COLT*, Haifa, Israel, 2010.
[22] G. Tsoumakas, I. Katakis, and I. P. Vlahavas, "Mining multi-label data," in *Data Mining and Knowledge Discovery Handbook*, O. Maimon and L. Rokach, Eds. Berlin, Germany: Springer, 2010.
[23] G. Tsoumakas and I. P. Vlahavas, "Random k-labelsets: An ensemble method for multilabel classification," in *Proc. 18th ECML*, Warsaw, Poland, 2007, pp. 406–417.
[24] B. Ženko and S. Džeroski, "Learning classification rules for multiple target attributes," in *Proc. 12th PAKDD*, Osaka, Japan, 2008, pp. 454–465.
[25] J. H. Zaragoza, E. Sucar, E. F. Morales, C. Bielza, and P. Larrañaga, "Bayesian chain classifiers for multidimensional classification," in *Proc. 24th IJCAI*, 2011, pp. 2192–2197.
[26] M.-L. Zhang and K. Zhang, "Multi-label learning by exploiting label dependency," in *Proc. 16th ACM SIGKDD Int. Conf. KDD*, Washington, DC, USA, 2010, pp. 999–1008.

Jesse Read is a Lecturer (until recently Post-Doctoral Researcher) in the Department of Signal Theory and Communications, Carlos III University of Madrid (UC3M), Spain. He received the bachelor's degree in computer science with first class honours from the University of Waikato, New Zealand, in 2005, and also obtained the Ph.D. degree from the same university in 2010, in the Machine Learning Group. His current research interests include multi-label classification, and learning in large-scale and data-stream contexts.

Concha Bielza received the M.S. degree in mathematics from the Complutense University of Madrid, Spain, in 1989 and the Ph.D. degree in computer science from the Technical University of Madrid (Universidad Politecnica de Madrid, UPM), in 1996. Since 2010, she has been a Full Professor of Statistics and Operations Research in the Department of Artificial Intelligence, UPM. Her current research interests include areas of probabilistic graphical models, decision analysis, metaheuristics for optimization, data mining, classification models, and real applications, mainly biomedicine and neuroscience. She has over 70 refereed publications and is currently involved in the Human Brain Project, one of the two European FET Flagship projects funded over the next 10 years. She is a member of the IEEE.

Pedro Larrañaga is a Full Professor in Computer Science and Artificial Intelligence at the Universidad Politecnica de Madrid (UPM), where he leads the Computational Intelligence Group. His current research interests include fields of Bayesian networks, estimation of distribution algorithms, multi-label classification, regularization and data streams, with applications in bioinformatics, biomedicine and neuroscience. He has supervised more than 20 Ph.D. students, published more than 100 papers in international journals and participated in more than 30 projects in collaboration with the industry. He has also participated in more than 50 projects granted by public institutions, the most recent one being the Human Brain Project, selected as one of the two European Flagships for the period 2013–2023. From 2007-2010, he was an Expert Manager of the Computer Technology Area, and Deputy Directorate of Research Projects of the Spanish Ministry of Science and Innovation. He is Fellow of the European Artificial Intelligence Society (ECCAI). He is a member of the IEEE.