Synthetic Datasets for Sound Experimental Evaluation of Multilabel Classifiers

Oscar Luaces, Jorge Díez, Juan José del Coz, José Barranquero, and Antonio Bahamonde

Artificial Intelligence Center. University of Oviedo at Gijón, Asturias, Spain
www.aic.uniovi.es

Abstract. Multilabel classification is a very active area of research. However, the experimental studies have only available a small collection of benchmark datasets. Moreover, the handy data frequently do not capture the essential features of multilabel classification tasks. In this paper we present a generator of synthetic datasets implemented using a genetic algorithm. The datasets generated are able to reproduce a wide variety of situations. The paper reports a thoroughly experimentation devoted to assess the features of these datasets.

1 Introduction

Multilabel (ML) classification has received the attention of many contributions from different points of view. In [15] and [6] the approach was an extension of multiclass classification. Other learning algorithms have been done from different settings; this is the case of nearest neighbors [19], decision trees [17], logistic regression and nearest neighbors [3], and Bayesian learners [18,1].

There is other interesting group of learners, those based on the chain rule, which try to mixture both inputs and labels in the learning process. In this group are [12], [4], and [8]. Another successful approach consists in learning a ranking of labels for each instance and then, if necessary, produce a bipartition using a threshold that can be a fixed value or a variable learned from the learning task. This includes [6] and [10].

Finally, there are other approaches that aim to explicitly optimize a given loss function. Usually, they are theoretical results that inspire heuristic implementations to improve the scores of classifiers, see [4], [5], [9] and [10].

Despite the proliferation of theoretical proposals, the experimental studies have to confront the lack of rich collections of benchmarks. There are just a few publicly available datasets and they cover a small and biased proportion of the huge possibilities of ML datasets.

In this paper, we present a generator of synthetic datasets able to reproduce a wide variety of situations. It is available at www.aic.uniovi.es/ml_generator. The next section gives a formal presentation of ML learning tasks and hypotheses. Then, we review the characteristics of the available datasets in the most popular repository, MULAN [16]. The generator presented here is a genetic algorithm.

Section 4 details both the theoretical motivation and some implementation issues. To close the paper, the fifth section reports a thoroughly experimentation devoted to assess the features of datasets generated.

2 Multilabel Classifiers and Datasets

Let $\mathscr{L}$ be a finite and non-empty set of labels $\{l_1, \ldots, l_L\}$, let $\mathcal{X}$ be an input space, and let $\mathcal{Y}$ be the output space defined as the set of subsets of labels.

Definition 1. *A* ML classification task *is given by a dataset*

$$D = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_n, \boldsymbol{y}_n)\} \subset \mathcal{X} \times \mathcal{Y} \tag{1}$$

of pairs of inputs $\boldsymbol{x}_i \in \mathcal{X}$ and subsets of labels $\boldsymbol{y}_i$ as outputs.

The set of labels assigned to each input is usually called the set of *relevant* labels of the input entry.

Sometimes, when the input space is an Euclidean space of p dimensions, we will refer to the learning task by a couple of matrices

$$D \equiv (\mathbb{X}, \mathbb{Y}) \tag{2}$$

of n rows and p and L columns respectively.

The goal of a ML classification task D is to induce a hypothesis defined as follows.

Definition 2. *A* ML hypothesis *is a function h from the input space to the output space, the power set of labels $\mathscr{P}(L)$; in symbols,*

$$h : \mathcal{X} \longrightarrow \mathcal{Y} = \mathscr{P}(\mathscr{L}) = \{0, 1\}^L. \tag{3}$$

Hence, $h(\boldsymbol{x})$ is the set of relevant labels predicted by h. Sometimes, we use $h(\mathbb{X}) = \mathbb{Y}$ to mean that the predictions of h applied to an input set represented by a matrix $\mathbb{X}$ are a set of labels codified by a matrix $\mathbb{Y}$.

Binary Relevance (BR) is a straightforward approach to handle a ML classification task. It is the simplest strategy, but it is very effective. Each label is classified as relevant or irrelevant by a binary classifier learned for each label independently.

ML classifiers can be evaluated from different points of view. The predictions can be considered as a bipartition or, alternatively, as a ranking of the set of labels. In this paper the performance of ML classifiers will be evaluated measuring the accuracy of the bipartitions produced. Thus, loss functions must compare subsets of labels; a common setting is to use Information Retrieval metrics to perform these comparisons.

Usually these measures can be divided in two groups [16]. The *example-based* measures compute the average differences of the actual and the predicted sets of labels over all examples. The *label-based* measures decompose the evaluation for

each label. There are two options here, averaging the measure label-wise (usually called *macro-average*), or computing a single value concatenating all predictions, the so-called *micro-average* version of a measure.

Notice that the macro averages can be trivially optimized following a BR strategy. Thus we will concentrate on micro averages. The results can be trivially extended to deal with example based loss functions.

The performance measures of IR in this context can be read as follows. The *Recall (R)* is defined as the proportion of truly relevant labels that are included in the predicted labels. On the other hand, the *Precision (P)* is defined as the proportion of predicted labels that are truly relevant. Frequently, the performance is quantified using a trade-off between both measures: the F-measures formalized by the harmonic average. The F_β ($\beta \geq 0$) is defined by

$$F_\beta = \frac{(1+\beta^2)P \cdot R}{\beta^2 P + R}.$$

The most frequently used F-measure is F_1.

For further reference, let us recall the formal definitions of these measures. For a prediction of a multilabel hypothesis $h(\boldsymbol{x})$ and a subset of *truly relevant* labels $\boldsymbol{y} \subset \mathscr{L}$, for each label $l \in \mathscr{L}$ we can compute the following contingency matrix:

$$\begin{array}{c|cc} & \boldsymbol{y}[l]=1 & \boldsymbol{y}[l]=0 \\ \hline h(\boldsymbol{x})[l]=1 & a(\boldsymbol{x},l) & b(\boldsymbol{x},l) \\ h(\boldsymbol{x})[l]=0 & c(\boldsymbol{x},l) & d(\boldsymbol{x},l). \end{array} \tag{4}$$

Each entry (a, b, c, d) is 1 when the predicates of the corresponding row and column are both true, otherwise the value is 0. Notice for instance, that $a(\boldsymbol{x}, l)$ is 1 only when the prediction of h includes the truly relevant label l. Only one of the entries of the matrix is 1, the rest are 0.

To define the micro averaged F_β ($\beta \geq 0$), let

$$D' = \{(\boldsymbol{x'}_1, \boldsymbol{y}'_1), \ldots, (\boldsymbol{x'}_{n'}, \boldsymbol{y}'_{n'})\}$$

be a test set.

Definition 3. *The* micro averaged F_1 *is given by*

$$F_\beta^{mi} = \frac{(1+\beta^2)\sum_{(\boldsymbol{x}',l)} a(\boldsymbol{x}',l)}{(1+\beta^2)\sum_{(\boldsymbol{x}',l)} a(\boldsymbol{x}',l) + \sum_{(\boldsymbol{x}',l)} b(\boldsymbol{x}',l) + \beta^2 \sum_{(\boldsymbol{x}',l)} c(\boldsymbol{x}',l)}. \tag{5}$$

Other performance measures for ML classifiers can also be defined. The *Hamming loss* is very popular and it is defined as the proportion of misclassifications. Notice that, as for macro averages, this measure can also be optimized using a simple BR strategy.

2.1 Dataset Descriptions

ML datasets (see (Eq. 1 and 2)) can be described using a number of features. Some are shared with other types of learning tasks; this is the case of the number

of examples (n) and the dimensionality of the inputs (p). However, there are other features that are unique to multi-label classification, they consider the number of possible labels (L) and their distribution among the dataset. In this paper we will use the *cardinality*, *density*, and *dependency* defined as follows, see [16].

The *cardinality* of a dataset is defined as the average number of labels per example.

$$cardinality(\mathbb{Y}) = \frac{\sum_{i=1}^{n} \sum_{j=1}^{L} \mathbb{Y}(i,j)}{n}. \tag{6}$$

The proportion of ones in the matrix $\mathbb{Y}$ of labels is called the *density* of the dataset and it is the cardinality divided by the number of possible labels,

$$density(\mathbb{Y}) = \frac{cardinality(\mathbb{Y})}{L}. \tag{7}$$

The *dependency* is more subtle. There is no universal definition for that. In fact, from a Bayesian point of view, there are two possible kinds of dependency: the conditional and the unconditional dependency [4,1,18]. In fact, considering the columns of the matrix $(\mathbb{X}, \mathbb{Y})$ (Eq. 2) as variables, the Bayesian network that represents all these variables may revel some dependencies.

To express in equations these concepts, let l_i be the i^{th} column of matrix $\mathbb{Y}$ that represents the corresponding label. There are *conditional* dependency between labels whenever

$$\Pr(l_1, \ldots, l_L | \boldsymbol{X}) \neq \prod_{i=1}^{L} \Pr(l_i | \boldsymbol{X}).$$

This means that there are disjoint subsets of labels such that

$$\Pr((l_j : j \in J) | \boldsymbol{X}) \neq \Pr((l_j : j \in J) | \boldsymbol{X}, (l_i : i \in I)).$$

In [7], these relationships between input variables and labels are represented graphically and used to build a method for feature selection for ML learning tasks.

On the other hand, when the reference to input variables of the above equations can be skipped, the dependency between labels, if there is any, is *unconditional*. In this paper we measure this kind of dependency as the average of the correlation of labels weighted by the number of common examples. In symbols,

$$dependency(\mathbb{Y}) = \frac{\sum_{i<j} \rho(l_i, l_j) |l_i \cap l_j|}{\sum_{i<j} |l_i \cap l_j|}. \tag{8}$$

3 Benchmarks in ML Classification Tasks

Strictly speaking, there are no generators of synthetic ML learning tasks. The approach presented in [14,11] is mainly concerned with streaming data and can hardly be used to obtain ML datasets with a realistic combination of features.

The few publicly available datasets are gathered in some repositories. The most popular is MULAN [16]. In Table 1 we report the main features of all datasets of MULAN.

It is quite striking the low density of these datasets. The median is just 2.21% (expressed as a percentage). Notice that a hypothesis predicting no labels for any input, will have a very low percentage of misclassifications. This is a good reason to neglect the Hamming loss as an appropriate performance measure.

The low density is due to the low cardinality. Half of the datasets have a cardinality below 2.85. In other words, those ML datasets are not far away from being single-label multi-class datasets. Additionally, the unconditional dependency, measured by (Eq. 8), is very low. The labels do not seem to be very related.

4 Genetic Generator

In this section we describe the genetic algorithm implemented to search for ML datasets with a set of target features selected by the user. The goal is to obtain, for each desired combination of features, three datasets: train, validation and test, with approximately the same feature values. They will be represented by matrices as in (Eq. 2).

In all cases, the input space will be a hypercube

$$\mathcal{X} = [0,1]^p \subset \mathbb{R}^p. \tag{9}$$

The requisites of the search set by the user include

number of labels (L)	number of test examples	
number of training examples (n)	cardinality (c_t)	(10)
number of validation examples	dependency (d_t).	

All parameters but cardinality and dependency are somehow structural and can be easily fulfilled. Thus, the generator starts with a set of inputs drawn from a uniform distribution in $\mathcal{X}$; let $\mathbb{X}$ be the matrix of input instances for the training set. The core idea of the generator presented here is that it searches for a *hypothesis* to classify the inputs in $\mathbb{X}$ and obtain a ML task with cardinality and dependency as close as possible to those specified by the user.

Once the hypothesis is found, the *validation* and *test* datasets are built; their input instances are independently drawn with uniform distribution again, from the input space $\mathcal{X}$.

Thus, the focus of the generator are the hypotheses for ML classification. The building blocks of these hypotheses are hyperplanes that split the input space in a positive and a negative region. In fact a set of hyperplanes may define a linear classifier or a nonlinear one. We are going to give the details in the next subsections.

Table 1. Description of MULAN datasets

dataset	samples	features	labels	cardinality	dependency	density
bookmarks	87856	2150	208	2.03	0.10	0.98%
CAL500	502	68	174	26.04	0.14	14.97%
bibtex	7395	1836	159	2.40	0.15	1.51%
bow	43907	100	101	4.38	0.22	4.33%
mpeg	43907	320	101	4.38	0.22	4.33%
Corel5k	5000	499	374	3.52	0.15	0.94%
Corel16k001	13766	500	153	2.86	0.14	1.87%
Corel16k002	13761	500	164	2.88	0.14	1.76%
Corel16k003	13760	500	154	2.83	0.13	1.84%
Corel16k004	13837	500	162	2.84	0.14	1.75%
Corel16k005	13847	500	160	2.86	0.13	1.79%
Corel16k006	13859	502	160	2.88	0.14	1.80%
Corel16k007	13915	500	174	2.89	0.14	1.66%
Corel16k008	13864	500	168	2.88	0.14	1.72%
Corel16k009	13884	500	173	2.93	0.13	1.69%
Corel16k010	13618	500	144	2.82	0.13	1.96%
delicious	16105	500	983	19.02	0.11	1.93%
emotions	593	72	6	1.87	0.28	31.14%
enron	1702	1001	53	3.38	0.12	6.37%
eurlex-dc	19348	5000	412	1.29	0.21	0.31%
eurlex-sm	19348	8792	201	0.02	0.76	0.01%
eurlex-ed	19348	1208	3993	5.31	0.11	0.13%
genbase	662	1186	27	1.25	0.54	4.64%
mediamill	43907	120	101	4.38	0.22	4.33%
medical	978	1449	45	1.25	0.18	2.77%
rcv1subset1	6000	47236	101	2.88	0.21	2.85%
rcv1subset2	6000	47236	101	2.63	0.22	2.61%
rcv1subset3	6000	47236	101	2.61	0.22	2.59%
rcv1subset4	6000	47229	101	2.48	0.23	2.46%
rcv1subset5	6000	47235	101	2.64	0.20	2.62%
scene	2407	294	6	1.07	0.11	17.90%
tmc2007	28596	49060	22	2.16	0.10	9.81%
tmc2007-500	28596	500	22	2.22	0.11	10.09%
yeast	2417	103	14	4.24	0.25	30.26%
Medians	13763.5	500	148.5	2.85	0.14	2.21%

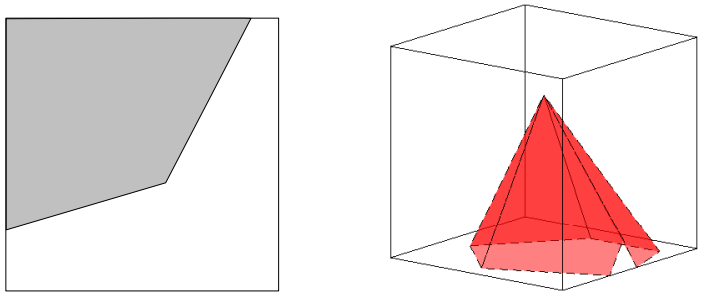

Fig. 1. Pyramids in two and three dimensions. The interior of these polyhedrons are the regions where labels will be assigned to the input instances

4.1 Building Blocks

Let

$$\boldsymbol{x}_0 \in \mathcal{X} = [0,1]^p, \quad \boldsymbol{w}_0 \in [-1,1]^p.$$

The hyperplane that passes through $\boldsymbol{x}_0$ and is perpendicular to $\boldsymbol{w}_0$ splits the input training set $\mathbb{X}$ into two subsets

$$\mathbb{X}^+ = \{\boldsymbol{x} : \langle \boldsymbol{w}_0, \boldsymbol{x} - \boldsymbol{x}_0 \rangle \geq 0\}, \quad \mathbb{X}^- = \{\boldsymbol{x} : \langle \boldsymbol{w}_0, \boldsymbol{x} - \boldsymbol{x}_0 \rangle < 0\}.$$

When the purpose is to build linearly separable datasets, the genetic algorithm randomly constructs a first generation of linear hypotheses. Each individual is characterized by a collection of hyperplanes h_l

$$\{h_l \equiv (\boldsymbol{x}_l, \boldsymbol{w}_l) : l = 1, \ldots, L\};$$

one for each label.

On the other hand, to build a nonlinear hypothesis may be a bit more tricky. Our approach is to assign relevant labels to regions of the input space defined by the intersection of several hyperplanes that share a common point. In other words, the relevant labels are geometrically defined at the interior of pyramids with a certain number of *faces*. Thus, for a given label l, we define a hypothesis h_l as follows:

$$l \in h_l(\boldsymbol{x}) \iff \langle \boldsymbol{w}_k^l, \boldsymbol{x} - \boldsymbol{x}_0^l \rangle \geq 0, \quad \forall k = 1, \ldots, \mathit{faces}. \tag{11}$$

where

$$\boldsymbol{x}_0^l \in \mathcal{X}, \boldsymbol{w}_k^l \in [-1,1]^p, \ k = 1, \ldots, \mathit{faces}. \tag{12}$$

However, if the list of vectors $\boldsymbol{w}_k^l$ (Eq. 12) is completely random, the interior of the pyramid may be empty or too small. To avoid these possibilities we force the list of $\boldsymbol{w}_k^l$ to form angles within a given range. The aim is to obtain regions like those of Figure 1.

The procedure employed to generate these pyramids is the following. First, a set of vectors $(\boldsymbol{w}_k^l : k = 1, \ldots, faces)$ is randomly drawn in $[-1, 1]^p$. Then, using the Gram-Schmidt procedure, we obtain an orthonormal basis of the linear span of vectors $\boldsymbol{w}_k^l$. Let

$$(\boldsymbol{v}_k^l : k = 1, \ldots, faces) \leftarrow \text{Gram-Schmidt}(\boldsymbol{w}_k^l : k = 1, \ldots, faces)$$

If the rank of vectors $\boldsymbol{w}_k^l$ is not *faces*, a new set is drawn. The next step redefines the vectors of the nonlinear hypothesis as follows:

$$\begin{aligned} \boldsymbol{w}_1^l &\leftarrow \boldsymbol{v}_1^l \\ \boldsymbol{w}_k^l &\leftarrow \lambda \boldsymbol{v}_1^l + \boldsymbol{v}_k^l,\ k = 2, ...faces \end{aligned} \tag{13}$$

Notice that

$$\cos(\boldsymbol{w}_1^l, \boldsymbol{w}_k^l) = \frac{\langle \boldsymbol{v}_1^l, \lambda \boldsymbol{v}_1^l + \boldsymbol{v}_k^l \rangle}{\sqrt{1 + \lambda^2}} = \frac{\lambda}{\sqrt{1 + \lambda^2}}$$

Therefore,

$$\frac{\cos^2(\boldsymbol{w}_1^l, \boldsymbol{w}_k^l)}{1 - \cos^2(\boldsymbol{w}_1^l, \boldsymbol{w}_k^l)} = \frac{1}{\tan^2(\boldsymbol{w}_1^l, \boldsymbol{w}_k^l)} = \lambda^2$$

and hence it is straightforward to fix a range for λ values if we want that, for instance,

$$\text{angle}(\boldsymbol{w}_1^l, \boldsymbol{w}_k^l) \in [50^o, 80^o], \forall k = 2, \ldots, faces.$$

Notice that then the interior angle of pyramid faces with the first one will range in $[100^o, 130^o]$. In the experiments reported at the end of the paper, the number of *faces* will be set to 5.

4.2 Conditional Dependency

To obtain a dataset with a certain degree of *conditional dependency*, we can use the following method of two steps. If L is the number of labels required, first we use the procedure described above to search for a dataset with $L/2$ labels. Let

$$D_1 = (\mathbb{X}, \mathbb{Y}_{L/2}) \tag{14}$$

be such a dataset. Thus, to obtain the rest of labels, we use the whole dataset D_1 as the input instances and search for a new collection of $L/2$ labels. In this way we have

$$D_2 = \big((\mathbb{X}, \mathbb{Y}_{L/2}), \mathbb{Y}'_{L/2}\big). \tag{15}$$

At the end, we obtain

$$D = \big(\mathbb{X}, \big[\mathbb{Y}_{L/2}\ \mathbb{Y}'_{L/2}\big]\big), \tag{16}$$

a dataset with L labels and some degree of conditional dependency.

To ensure a cardinality similar to a given amount set by the user, we divide the cardinality in two equal parts: one part for the matrix $\mathbb{Y}_{L/2}$ searching for D_1, and the rest for the second half for the matrix, $\mathbb{Y}'_{L/2}$, searching for D_2.

On the other hand, the unconditional dependency can not be guaranteed. Thus, we ask for the same amount in both searches in order to reach a similar value at the end of the process.

4.3 Genetic Stuff

Recall that individuals are hypotheses h applicable to input instances represented in a matrix $\mathbb{X}$. The evaluation of the quality of the h is measured comparing the target cardinality and dependency with the actual values of the label matrix $\mathbb{Y} = h(\mathbb{X})$. Formally,

Definition 4. *The* fitness *of a label matrix* $\mathbb{Y}$ *is the weighted sum of the absolute difference between the actual and target values of cardinality and dependency* c_t *and* d_t*. In symbols*

$$fitness(\mathbb{Y}) = \gamma|dependency(\mathbb{Y}) - d_t| + (1-\gamma)\frac{|cardinality(\mathbb{Y}) - c_t|}{10}. \quad (17)$$

In the experiments reported in the last section, we used, in all cases, the *weight* parameter in the computation of the fitness with the value $\gamma = 0.5$.

The fitness values are modified by some penalties, detailed in the following table, that try to avoid undesirable datasets.

Description	Penalty
numbers of examples with no labels (e.nl)	$(10/n) * e.nl$
number of constant labels in all examples (c.l)	$(5/L) * c.l$
repeated labels (r.l)	$(5/L) * r.l$

To select individuals to built the next generation we used *elitism* combined with a *fitness proportionate selection*, sometimes called *roulette wheel selection*. The *crossover* is a simple *one point* crossover.

On the other hand, once an individual is selected to be *mutated*, a random number of labels are selected from 1 to 25% of the number of labels available. A part of the hypotheses corresponding to these labels are replaced by new ones randomly generated. The second part are *near replicas* of other hypothesis, they are obtained by adding a gaussian noise (mean 0 and standard deviation 0.01) both to the point and the components of the director vectors (Eq. 12).

5 Experimental Results

The aim of this section is to report a number of experiments designed to evaluate the generator presented in the paper. We examine three aspects of the generator. First, we compare the cardinalities and unconditional dependencies obtained with those required in a broad range of values. In some cases, to obtain a dataset with a given pair of cardinality and dependency values is not trivial, for instance when the user selects a high dependency combined with a low cardinality, or vice versa.

Second, we assess the linearity and nonlinearity of regions of the input space that assign labels. For this purpose we compare the micro F_1 scores achieved by algorithms that learn linear and nonlinear models.

Table 2. Datasets generated with *conditional* and *unconditional* dependency. For each number of labels, the table shows the average cardinality and dependency. The number of datasets with unconditional dependency generated per number of labels is 45, thus the total number of datasets is 630: 315 linearly separable and 315 nonlinearly separable. For datasets generated with conditional dependency, 6 datasets were generated for each number of labels with a nonlinear hypotheses. The total number of datasets is then 42

	Unconditional				Conditional	
	Linear		Nonlinear			
#labels	cardi.	depend.	cardi.	depend.	cardi.	depend.
10	3.43	0.30	3.44	0.20	4.38	0.25
25	3.63	0.26	3.50	0.24	4.72	0.28
50	3.89	0.21	3.45	0.27	4.99	0.28
75	4.23	0.20	3.64	0.28	4.93	0.28
100	4.46	0.18	3.62	0.28	5.08	0.31
150	6.38	0.16	3.76	0.28	6.19	0.29
200	8.99	0.17	3.78	0.29	6.29	0.28
target	3.25	0.30	3.25	0.30	3.75	0.30

Finally, we test to what extent the generator is able to produce datasets with some degree of conditional dependency using the procedure detailed in Section 4.2. Here we compare the scores of BR strategies against those based on *Chain Classifier* (CC) that was designed to take advantage of this kind of dependency.

5.1 Testing Cardinality and Dependency Achievements

First of all, we check the degree of fulfillment of the cardinality and unconditional dependency (Eq. 8) required by the user. For this end we generated a number of datasets with the following features. In all cases the input space is the hypercube $\mathcal{X}$ of 10 dimensions (Eq. 9), and the datasets have 400 points. The size L of the set of labels varies in $\{10, 25, 50, 75, 100, 150, 200\}$.

The target cardinalities are values from 1.25 to 5.25 with a step of 0.50. The target values for dependency are in $\{0.1, 0.2, 0.3, 0.4, 0.5\}$. Both target values for cardinality and dependency were chosen to mimic and extend the ranges observed in the datasets from the MULAN repository; recall the values reported in Table 1. The aim was to cover uniformly those ranges instead of the biased distribution of the MULAN datasets.

In Table 2 we report the average cardinalities and unconditional dependencies achieved by linear and nonlinear datasets. We observe that the nonlinear datasets reach, for all number of labels, closer scores to the target values. In the linear case, the behavior of the generator depends heavily on the number of labels, for higher number of labels both cardinality and dependency are harder to reach.

Additionally, Table 2 reports the average cardinalities and unconditional dependencies achieved by datasets generated aiming to reflect conditional dependency, see Section 4.2. In this case, the labels were defined in a nonlinearly separable region, since the generation of nonlinear hypotheses perform clearly better than linear. Recall that the procedure described in Section 4.2 does not guarantee unconditional dependencies, therefore, we opted for generating datasets with a fix unconditional dependency, 0.30.

The cardinality required was divided by 2, as was mentioned in Section 4.2. Thus, we asked for values from 2.50 to 5.00 with a step of 0.50. However, the final amount obtained was far from the target values. The average cardinality should be 3.75 for each number of labels, however, the obtained values ranged from 4.38 to 6.29 depending on the number of labels. The reason is that the second stage (Eq. 15) has an input space with a large dimension and this makes the searching problem more difficult for our genetic algorithm. Curiously, the unconditional dependency was almost perfectly achieved in all cases.

5.2 ML Learning Algorithms Used in the Following Sections

For testing the linearity and the conditional dependency of the datasets generated, we used the results achieved by some learning algorithms. We compared the micro averaged F_1 scores (Definition 3), for short F_1 in the following.

We used two algorithms with two variants. The first algorithm is a simple BR (Section 2) that uses as the base learner a *LibSVM* [2].

The second multilabel learner is inspired by the *Ensembles of Classifier Chains (ECC)* [13]. We reimplemented this algorithm in the version used by [4] and later in [10]; we call this version ECC*.

Using a base learner (in this case *LibSVM* again), ECC* learns a model to estimate posterior probabilities for each label l_i using as inputs the input instances $\boldsymbol{x}$ and the *previous* labels $\{l_1, \ldots, l_{i-1}\}$:

$$\big(\boldsymbol{x}, (l_1, \ldots, l_{i-1})\big) \rightsquigarrow \Pr(l_i|\boldsymbol{x}).$$

Of course, this procedure depends on the ordering of the set of labels. Thus the labels are randomly ordered 10 times, and the final posterior probability for each label conditioned on each input instance $\boldsymbol{x}$ is given by the average of the 10 posterior probabilities so obtained.

Both algorithms, BR and ECC*, were used with a linear kernel and with a Gaussian kernel (rfb), in each case the algorithm will be denoted by BRg and ECC*g respectively. The parameters C and g (for the Gaussian kernel) were adjusted with a grid search using the validation dataset generated. The parameters could vary in

$$C \in \{10^i : i = -1, \ldots, 3\}, \quad g \in \{10^{-3}, 10^{-2}, 10^{-1}, 0.3, 0.5, 1\}.$$

The number of examples in all datasets is 400 for training, 400 for validation, and 600 for testing. Some datasets are noise free, but in other cases they have some *noise* added using a Bernoulli distribution with $p = 0.01$; that is, the labels of both training and validation sets swapped their values with a probability p.

5.3 Testing Linear and Nonlinear Separability

To evaluate the linearity and nonlinearity we used the collection of 630 datasets described in Section 5.1. Additionally, adding noise we get 630 additional datasets. Table 3 shows the F_1 scores of the algorithms presented in the previous section using linear and Gaussian kernels. In general we observe that the scores decrease as the number of labels increase.

In the linearly separable datasets, we observe that BR outperforms BRg in all cases when the datasets are noise-free. However, with noise added, the differences are smaller and for any number of labels above 50 BRg obtains better scores than the linear counterpart. The reason is that the Gaussian kernel is more robust than the linear kernel is these cases. In fact, we observe the same behavior for ECC* and ECC*g. This weakness of ECC* also appears in noise free datasets for 150 or 200 labels.

On the other hand, for nonlinear datasets, nonlinear kernels outperform linear kernels for any number of labels (with or without noise); both for BR and for ECC*. Moreover, the differences increase (in favor of Gaussian versions) the greater the number of labels.

It is worth noting here that the best BR version, BRg outperforms the best ECC* version, ECC*g. Moreover, the differences are significant through all datasets $(630 + 630)$ using a Wilcoxon two-sided signed rank test, $p < 10^{-80}$. The differences are greater with the number of labels. Somehow, the iterative procedure of ECC*g seems to propagate the classification errors with the number of labels. Figure 2 shows graphically the differences of scores achieved.

5.4 Testing Conditional Dependency

To evaluate the degree of conditional dependency embodied in the datasets generated using the procedure of Section 4.2, we compare the F_1 scores of ECC* versus BR algorithms. We used the 42 datasets whose features were reported in Table 2 and another 42 obtained by adding artificial noise. The scores are gathered in Table 4.

Since the datasets are generated with nonlinear models, we compare ECC*g against BRg. The graphical comparison is shown in Figure 3. Additionally, we used a Wilcoxon two-sided signed rank test in all cases, and we obtained the following significant differences:

Noise	Range of Labels	Significant
Free	$\{10, 25, 50\}$	ECC*g $\gg$ BRg
	$\{75, 100\}$	ECC*g $\cong$ BRg
	$\{150, 200\}$	ECC*g $\ll$ BR g
Added	$\{10, 25, 50, 75\}$	ECC*g $\gg$ BRg
	$\{100, 150, 200\}$	ECC*g $\cong$ BRg

The p-values in significant differences are always below 0.01, and above 0.60 in non significant differences.

Table 3. Testing linearity and nonlinearity for datasets with *unconditional* dependency. There are 315 linearly separable datasets and other 315 nonlinearly separable, thus a total of 630. Another collection of 630 datasets was created adding Bernoulli noise. Each cell reports the average of micro F_1 (Eq. 5) scores in the corresponding test sets

		Linear				Nonlinear			
	#labels	BR	BRg	ECC*	ECC*g	BR	BRg	ECC*	ECC*g
Noise Free	10	98.11%	97.19%	97.79%	96.82%	82.94%	87.26%	82.75%	87.02%
	25	95.66%	94.71%	94.18%	93.19%	76.11%	81.58%	74.98%	81.09%
	50	92.65%	91.09%	88.15%	87.02%	68.99%	74.68%	65.81%	73.36%
	75	89.96%	88.66%	81.86%	80.87%	63.78%	69.32%	57.41%	67.07%
	100	87.98%	86.53%	76.56%	75.72%	60.14%	65.68%	51.53%	62.46%
	150	87.35%	85.70%	69.14%	70.60%	53.66%	58.65%	39.72%	53.80%
	200	87.78%	86.12%	63.75%	69.02%	48.88%	51.36%	31.08%	46.18%
Noise Added	10	96.43%	96.00%	96.34%	95.79%	82.56%	86.85%	82.57%	86.56%
	25	91.60%	91.49%	90.59%	90.16%	74.63%	80.09%	73.81%	79.88%
	50	84.10%	84.33%	79.41%	79.98%	65.25%	71.49%	62.30%	71.09%
	75	76.95%	77.52%	68.25%	71.46%	56.62%	64.86%	51.34%	63.80%
	100	71.93%	73.30%	60.70%	66.67%	50.80%	60.09%	43.61%	58.16%
	150	68.42%	71.45%	46.44%	62.04%	39.22%	50.64%	28.41%	49.02%
	200	68.53%	71.99%	37.90%	57.20%	31.04%	43.70%	19.78%	41.27%

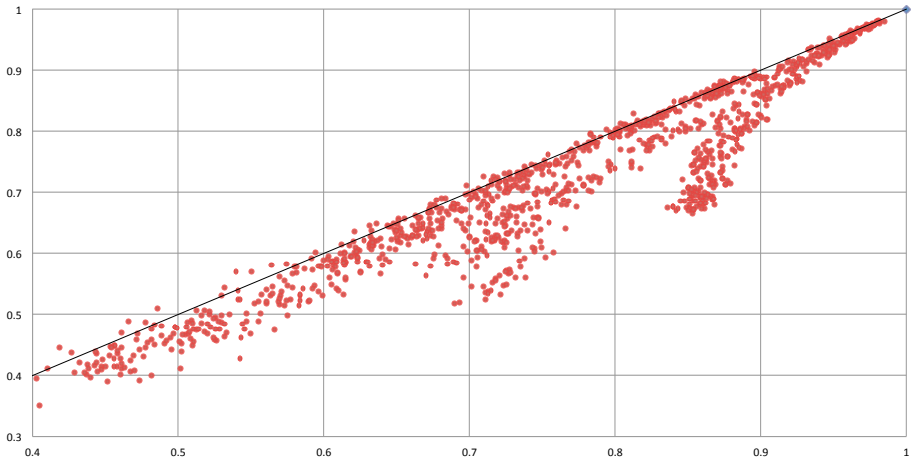

Fig. 2. Datasets with unconditional dependency. Each point is a pair of micro F_1 scores achieved in the same dataset by ECC*g and BRg. Points below the diagonal represent datasets where BRg outperforms ECC*g

Table 4. Average micro F_1 scores in test sets of datasets generated explicitly with *conditional dependency*, see Section 4.2

	Noise Free				Noise Added			
#labels	BR	BRg	ECC*	ECC*g	BR	BRg	ECC*	ECC*g
10	80.32%	83.88%	80.36%	84.46%	80.00%	83.60%	80.32%	84.11%
25	77.13%	81.12%	77.50%	81.76%	76.49%	80.59%	76.85%	81.08%
50	71.48%	76.34%	72.32%	77.46%	70.29%	75.06%	71.03%	76.68%
75	68.44%	72.29%	67.84%	72.38%	66.41%	70.68%	65.71%	71.05%
100	64.18%	69.43%	63.48%	69.68%	62.44%	67.00%	60.26%	67.37%
150	58.83%	63.31%	52.13%	62.77%	53.09%	61.55%	46.68%	60.20%
200	53.81%	58.43%	47.09%	56.87%	47.28%	54.53%	41.30%	54.96%

Roughly speaking, we see that ECC*g is better than BRg with a low number of labels. But the accumulation of errors forced by the iterative algorithm of ECC*g makes that the differences become smaller or (for noise free datasets) even that BRg outperforms ECC*g. In any case, these results are quite coherent with those reported in [13]. In that paper, the experimental results were computed with 15 datasets; only 2 of them have more than 103 labels.

However, the relevant conclusion for this paper is the success of the generator in producing datasets with the explicit intention of having some degree of conditional dependency.

6 Conclusions

Although it is a very active area of research, ML classification has too few publicly available datasets. Moreover, the available data are too sparse (they have very low density) and they exhibit very low unconditional dependency. To help on the experimental studies on ML we have presented a generator of synthetic datasets.

We have tested the features of the datasets generated in a thoroughly experimental study. We have shown that it is possible to generate sets with a broad range of features, including cardinality, and conditional and unconditional dependency.

Acknowledgements

The research reported here is supported in part under grant TIN2011-23558 from the MICINN (Ministerio de Ciencia e Innovación, Spain). We would also like to acknowledge all those people who generously shared the datasets and software used in this paper.

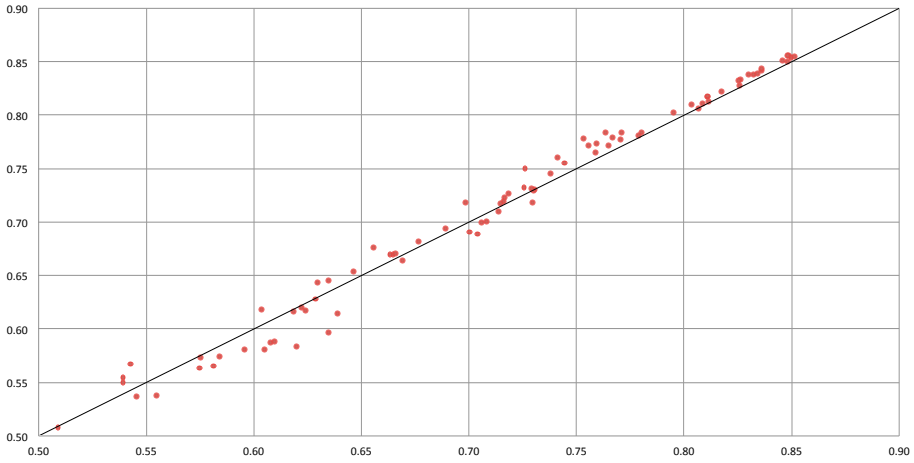

Fig. 3. Datasets with conditional dependency. Each point is a pair of Micro F_1 scores achieved in the same dataset by ECC*g and BRg. Points above the diagonal represent datasets where ECC*g outperforms BRg

References

1. Bielza, C., Li, G., Larrañaga, P.: Multi-dimensional classification with bayesian networks. International Journal of Approximate Reasoning (2011)
2. Chang, C.C., Lin, C.J.: LIBSVM: A library for support vector machines. ACM Transactions on Intelligent Systems and Technology 2, 27:1–27:27 (2011), software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`
3. Cheng, W., Hüllermeier, E.: Combining Instance-Based Learning and Logistic Regression for Multilabel Classification. Machine Learning 76(2), 211–225 (2009)
4. Dembczyński, K., Cheng, W., Hüllermeier, E.: Bayes optimal multilabel classification via probabilistic classifier chains. Proceedings of the 27th International Conference on Machine Learning (ICML) (2010)
5. Dembczyński, K., Waegeman, W., Cheng, W., Hüllermeier, E.: An exact algorithm for f-measure maximization. In: Proceedings of the Neural Information Processing Systems (NIPS) (2011)
6. Elisseeff, A., Weston, J.: A kernel method for multi-labelled classification. In: In Advances in Neural Information Processing Systems 14. pp. 681–687. MIT Press (2001)
7. Lastra, G., Luaces, O., Quevedo, J., Bahamonde, A.: Graphical feature selection for multilabel classification tasks. Lecture Notes in Computer Sciences, Proceedings of Advances in Intelligent Data Analysis X (IDA 2011) 7014, 246–257 (2011)
8. Montañés, E., Quevedo, J., del Coz, J.: Aggregating independent and dependent models to learn multi-label classifiers. Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECLM-PKDD) pp. 484–500 (2011)

9. Petterson, J., Caetano, T.: Reverse multi-label learning. Advances in Neural Information Processing Systems 23, 1912—1920 (2010)
10. Quevedo, J.R., Luaces, O., Bahamonde, A.: Multilabel classifiers with a probabilistic thresholding strategy. Pattern Recognition 45(2), 876–883 (2012)
11. Read, J., Bifet, A., Holmes, G., Pfahringer, B.: Streaming multi-label classification. JMLR Workshop and Conference Proceedings (Second Workshop on Applications of Pattern Analysis) 17, 19–25 (2011)
12. Read, J., Pfahringer, B., Holmes, G., Frank, E.: Classifier Chains for Multi-label Classification. In: Proceedings of European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD). pp. 254–269 (2009)
13. Read, J., Pfahringer, B., Holmes, G., Frank, E.: Classifier chains for multi-label classification. Machine Learning 85, 333–359 (2011)
14. Read, J., Pfahringer, B., Holmes, G.: Generating Synthetic Multi-label Data Streams. In: ECML/PKKD 2009 Workshop on Learning from Multi-label Data (MLD'09) (2009)
15. Schapire, R., Singer, Y.: Boostexter: A boosting-based system for text categorization. Machine learning 39(2), 135–168 (2000)
16. Tsoumakas, G., Katakis, I., Vlahavas, I.: Mining Multilabel Data. In O. Maimon and L. Rokach (Ed.), Data Mining and Knowledge Discovery Handbook, Springer (2010)
17. Tsoumakas, G., Katakis, I., Vlahavas, I.: Random k-Labelsets for Multi-Label Classification. IEEE Transactions on Knowledge Discovery and Data Engineering (2010)
18. Zaragoza, J., Sucar, L., Bielza, C., Larrañaga, P.: Bayesian chain classifiers for multidimensional classification. In: Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI) (2011)
19. Zhang, M.L., Zhou, Z.: ML-KNN: A Lazy Learning Approach to Multi-label Learning. Pattern Recognition 40(7), 2038–2048 (2007)