Using Tensor Products to Detect Unconditional Label Dependence in Multilabel Classifications

Jorge Díez*, Juan José del Coz, Oscar Luaces, Antonio Bahamonde

Artificial Intelligence Center. University of Oviedo at Gijón, 33204 Asturias, Spain
http://www.aic.uniovi.es

Abstract

Multilabel (ML) classification tasks consist of assigning a set of labels to each input. It is well known that detecting label dependencies is crucial in order to improve the performance in ML problems. In this paper, we study a new kernel approach to take into account unconditional label dependence between labels. The aim is to improve the performance measured by a micro-averaged loss function. The core idea is to transform a ML task into a binary classification problem whose inputs are drawn from a tensor space of the original input space and a representation of the labels. In this joint feature space we define a kernel to explicitly involve both labels and object descriptions. In addition to the theoretical contributions, the experimental results of this study provide an interesting conclusion: the performance in terms of Hamming Loss can be improved when unconditional label dependence is considered, as our method does. We report a thoroughly experimentation carried out with real world domains and several synthetic datasets devised to analyze the effect of exploiting label dependence in scenarios with different degrees of dependency.

Keywords: Multilabel, Label Dependence, Tensor products, Kernel methods

*Corresponding author

Email addresses: jdiez@aic.uniovi.es (Jorge Díez), juanjo@aic.uniovi.es (Juan José del Coz), oluaces@aic.uniovi.es (Oscar Luaces), antonio@aic.uniovi.es (Antonio Bahamonde)

1. Introduction

In *multilabel* (ML) classification tasks the aim is to assign, to each instance, more than one class or label instead of a single one; the so-called *relevant* labels. This is the case of text categorization where items have to be tagged for future retrieval. News or other kind of documents are annotated with more than one label according to different points of view. Other application fields include semantic annotation of images and video, functional genomics, music categorization into emotions and directed marketing. Tsoumakas *et al.* in [27, 28] have made a detailed presentation of ML classification and their applications.

Probably one of the most popular approaches to tackle a ML classification task is *Binary Relevance* (BR). It is the simplest one, but very effective for some loss measures. Each label is classified as relevant or irrelevant by a binary classifier independently learned for each label. Notice that BR does not consider any relation between labels, making the learning assumption that labels are independent. On the other hand, *proper* ML strategies try to take advantage of correlation or interdependency between labels. The presence or absence of a label in the set assigned to an instance depends on the feature values of the instance, but it also depends on the relevance of the remaining labels. For example, it is very likely that labels "NBA" and "Los Angeles Lakers" appear frequently together as tags for some videos of a sports website because there is a strong relationship between both labels.

In this sense, two types of label dependence can be taken into account, namely conditional and unconditional label dependence. See [5] for a complete discussion of label dependence in ML classification. The difference between both types of dependency is that conditional dependence captures the dependence among labels given a particular example, whereas unconditional dependence is a kind of global dependence, independent of any concrete observation. In the previous example, it seems that the dependence between labels "Los Angeles Lakers" and "NBA" is mostly unconditional, no matter the actual content of the video, if it is related to "Los Angeles Lakers", it is also related to the "NBA"

because the former is a franchise of the latter.

This paper presents a method based on capturing label dependence. The main goal is that the algorithm takes into account, not only the descriptions of the objects, but also the dependency between each pair of labels. Lately, many proposed ML classifiers exploit label dependence, see for instance [17, 22, 23]. Our approach is different from these methods cited previously because combines together two factors: 1) it exploits unconditional label dependence instead of conditional dependence which has been more studied in recent years, and 2) it tackles ML classification as a structured output prediction problem [8, 26]. Our algorithm is a kernel method and the key element is a kernel to capture unconditional label dependence. It is well known that kernel functions [24] allow to efficiently represent input instances in a feature space, different from the original input space. The proposed method incorporates a kernel over the space of labels, which it is combined with the kernel applied over the input instances.

The performance of ML classifiers can be measured using very different loss functions. When ML predictions are bipartitions of the set of labels (relevant and irrelevant labels), loss functions are computed comparing subsets and averaging scores. Taxonomically, loss measures can be divided into two groups: example-based and label-based. The latter group includes also two versions depending on how averages are computed: macro-averaged and micro-averaged loss functions. Theoretically macro-averaged measures can be optimized following a decomposition approach, like BR. Another important characteristic of our method is that it is designed to optimize a complete family of loss measures for ML classification, namely micro-averaged loss functions. Our approach can be applied to optimize any micro-average loss function whenever exists an algorithm able to optimize such measure for binary classification.

Thus, the main contribution of this work is to propose a kernel-based ML learner that is devised to induce classifiers aimed at improving the performance in terms of a micro-averaged measure. Our proposal is based on two ideas: i) to make a reduction from a ML task into a binary classification problem

whose inputs are drawn from a tensor space of the original input space and a representation of the label space, and ii) to define a kernel that explicitly incorporates both labels and object descriptions, in an attempt to exploit unconditional label dependence. Despite this approach could be applicable for any micro-averaged measure, we experimentally study its behavior in terms on Hamming Loss, whose macro-averaged and micro-averaged versions coincide. Interestingly, the experiments show that our proposal takes advantage of considering unconditional label dependence and obtains better Hamming scores compared to making the assumption of label independence. This result is interesting because it answers some questions raised in recent studies [5, 6] about if exploiting label dependence could help to improve Hamming loss performance. It also corroborates the results reported in [11], in which the authors investigate whether Hamming scores can be improved if the distribution of test categories is known *a priori*. Here we use a different approach, we introduce additional prior information in the form of a kernel for labels, but the conclusion is the same: Hamming scores can be improved in some cases. Additionally, we show that our kernel can be plugged into the algorithm proposed in [11] boosting its performance in terms of Hamming loss.

After a formal presentation of ML learning tasks and hypotheses, we discuss in detail label-based loss functions. Then, we derive a kernel method to optimize micro-averaged measures. The paper ends reporting some experiments devised to show the performance of such approach.

2. Multilabel Classification

Let $\mathscr{L}$ be a finite and non-empty set of labels $\{l_1, \ldots, l_m\}$, let $\mathcal{X}$ be an input space, and let $\mathcal{Y}$ be the output space, the set of subsets of labels. A ML classification task can be represented 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 $\boldsymbol{y}_i$ drawn from an unknown distribution. We identify the output space $\mathcal{Y}$ with vectors of dimension m with

Table 1: Description of the most important symbols used in the paper

D	Original ML training set	$\tilde{D}$	Transformed training set from D
$\boldsymbol{x}$	Input instance	$\boldsymbol{y}_i$	Labels associated with $\boldsymbol{x}$
$\mathbb{X}$	Matrix of input spaces $\boldsymbol{x}$ of D	$\mathbb{Y}$	Matrix of labels $\boldsymbol{y}$ of D
n	Number of instances in D	m	Number of labels in D
$\mathcal{X}$	Input space	$\mathcal{Y}$	Output space
p	Dimension of $\mathcal{X}$	$\mathscr{L}$	Set of labels
h	Hypothesis (a ML classifier)	l	A label from $\mathscr{L}$
$\rightsquigarrow$	.. is transformed into ...	$\boldsymbol{e}_l$	Vectorial representation of l
$\otimes$	Kronecker product	$\delta_{i,l}$	Kronecker delta
$\mathbb{I}$	Identity matrix	$\boldsymbol{W}$	Parameter vector of $\tilde{h}$
l_k	Vector column of label k in $\mathbb{Y}$	ρ	Correlation coefficient
$\mu(v)$	Mean of vector v	$\sigma(v)$	Standard deviation of vector v

components in $\{0, 1\}$. In this sense, for $\boldsymbol{y} \in \mathcal{Y}$,

$$l \in \boldsymbol{y} \quad \text{and} \quad \boldsymbol{y}[l] = 1$$

will appear interchangeably and will be considered equivalent. In the following, we assume that the input space is a subset of the Euclidean space of dimension p,

$$\mathcal{X} \subset \mathbb{R}^p.$$

Then, we will refer to a learning task as a couple of matrices

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

of n rows and p and m columns respectively.

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

Definition 1. *A* ML hypothesis *is a function h from the input space to the output space, set of subsets (power set) of labels $\mathscr{P}(\mathscr{L})$; in symbols,*

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

In other words, $h(\boldsymbol{x})[l] = 1$ means that label l is included in the set of predictions of the hypothesis h for input $\boldsymbol{x}$.

2.1. Loss Functions for Multilabel Classification

ML classifiers can be evaluated from different points of view. The predictions can be considered as either as a bipartition or as a ranking of the set of labels. In this paper the performance of ML classifiers will be evaluated as a bipartition. Thus, loss functions must compare subsets of labels.

Usually these measures can be divided into two groups [28]. The *example-based* measures compute the average differences of the actual and the predicted sets of labels over all examples. On the other hand, the *label-based* measures decompose the evaluation into separate evaluations for each label. There are two options here, averaging the measure label-wise (usually called *macro-average*), or concatenating all label predictions and computing a single value over all of them, the so-called *micro-averaged* version of a measure.

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:

	$\boldsymbol{y}[l] = 1$	$\boldsymbol{y}[l] = 0$
$h(\boldsymbol{x})[l] = 1$	$a(\boldsymbol{x}, l)$	$b(\boldsymbol{x}, l)$
$h(\boldsymbol{x})[l] = 0$	$c(\boldsymbol{x}, l)$	$d(\boldsymbol{x}, l)$

(4)

Each of the functions in this matrix represents one of the four possible states for a predicted label: a true positive (a), a false positive (b), a false negative (c) and a true negative (d). They have a value of 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 for example $\boldsymbol{x}$ includes

the truly relevant label l. Furthermore, for a particular pair $(\boldsymbol{x}, l)$ only one of the entries of the matrix is 1, the rest are 0.

The measures based on labels are now defined as follows. Throughout these definitions, h is a ML hypothesis (3) and

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

is a test set. Additionally, we use the following aggregations of contingency matrices:

$$\begin{array}{ll} A_l = \sum_{\boldsymbol{x}'} a(\boldsymbol{x}', l) & A = \sum_{(\boldsymbol{x}', l)} a(\boldsymbol{x}', l) \\ B_l = \sum_{\boldsymbol{x}'} b(\boldsymbol{x}', l) & B = \sum_{(\boldsymbol{x}', l)} b(\boldsymbol{x}', l) \\ C_l = \sum_{\boldsymbol{x}'} a(\boldsymbol{x}', l) & C = \sum_{(\boldsymbol{x}', l)} c(\boldsymbol{x}', l) \\ D_l = \sum_{\boldsymbol{x}'} b(\boldsymbol{x}', l) & D = \sum_{(\boldsymbol{x}', l)} d(\boldsymbol{x}', l) \end{array}$$

Definition 2. *The* Recall *is defined as the proportion of truly relevant labels that are included in predictions. The macro- and micro-averaged version are computed as follows.*

$$R^{ma} = \frac{1}{m} \sum_{l=1}^{m} \frac{A_l}{A_l + C_l},$$

$$R^{mi} = \frac{A}{A + C}.$$

Definition 3. *The* Precision *is defined as the proportion of predicted labels that are truly relevant. Macro and micro versions are defined by*

$$P^{ma} = \frac{1}{m} \sum_{l=1}^{m} \frac{A_l}{A_l + B_l},$$

$$P^{mi} = \frac{A}{A + B}.$$

The trade-off between *Precision* and *Recall* is formalized by their *harmonic* mean. So, in general, the F_β $(\beta \geq 0)$ is defined by

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

Definition 4. *The macro and micro* F_β $(\beta \geq 0)$ *are defined by*

$$F_\beta^{ma} = \frac{1}{m} \sum_{l=1}^{m} \frac{(1 + \beta^2) A_l}{(1 + \beta^2) A_l + B_l + \beta^2 C_l}$$

$$F_\beta^{mi} = \frac{(1+\beta^2)A}{(1+\beta^2)A + B + \beta^2 C}.$$

Notice that *Precision*, *Recall* and F-measure are not loss function, but score functions, the higher the better. F_1 is the most frequently used F-measure.

Other performance measures for ML classifiers can also be specified using the contingency matrices (4). This is the case of *Hamming loss* defined as follows.

Definition 5. Hamming loss *measures the proportion of misclassifications. The macro-averaged version is given by*

$$Hl^{ma} = \frac{1}{m}\sum_{l=1}^{m} \frac{B_l + C_l}{A_l + B_l + C_l + D_l}.$$

Taking into account that the sum of the components of contingency matrices of (4) is 1, the macro-averaged Hamming loss can be written as

$$\begin{aligned} Hl^{ma} &= \frac{1}{m}\sum_{l=1}^{m} \frac{B_l + C_l}{n'} = \frac{B + C}{m \cdot n'} \\ &= \frac{B + C}{A + B + C + D} = Hl^{mi}. \end{aligned} \tag{5}$$

That is to say, the Hamming loss is a measure that has the same value in their macro- and micro-averaged versions. Thus, a method that optimizes one of them, will optimize both.

3. A Reduction Framework for Micro-averaged Loss Functions

Let D (1) be a ML classification task and let L be a loss function, like those discussed above, defined for a pair of lists of subsets of labels:

$$L\big((\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n), (\hat{\boldsymbol{y}}_1, \ldots, \hat{\boldsymbol{y}}_n)\big) = L(\boldsymbol{Y}, \hat{\boldsymbol{Y}}),$$

where $\boldsymbol{Y}$ and $\hat{\boldsymbol{Y}}$ are just short names for the lists of subsets of labels $\boldsymbol{y}_i$ and $\hat{\boldsymbol{y}}_i$ respectively. If lower values of L are preferable to higher ones, like in case of Hamming Loss, then for a list of inputs $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$, the optimal predictions are given by a hypothesis

$$h_L^*(\boldsymbol{X}) = \operatorname*{argmin}_{\hat{\boldsymbol{Y}}} \sum_{\boldsymbol{Y}} \Pr(\boldsymbol{Y}|\boldsymbol{X}) \cdot L(\boldsymbol{Y}, \hat{\boldsymbol{Y}}). \tag{6}$$

In order to optimize micro-averaged measures, our approach is based on the reduction of the original ML classification task to one binary problem. This reduction was already introduced in [24].

The key observation is that the sum extended to inputs and labels can be converted into a sum of a new kind of examples indexed by inputs and labels. In symbols,

$$\boldsymbol{x} \rightsquigarrow \{(\boldsymbol{x}, l) : l = 1, \ldots, m\}.$$

That is, each example $\boldsymbol{x}$ is transformed into a set of new examples, one for each label l. Thus, our original ML learning task D (1) is transformed into a new binary classification task $\tilde{D}$:

$$\tilde{D} = \big\{\big((\boldsymbol{x}, l), [\![l \in \boldsymbol{y}]\!]\big) \colon (\boldsymbol{x}, \boldsymbol{y}) \in D, l \in \mathscr{L}\big\}. \tag{7}$$

Here the value of $[\![q]\!]$ for a predicate q is 1 when it is true, and 0 otherwise.

Starting from a ML task D, $\tilde{D}$ is a binary classification task. Moreover, there is a bijection between hypotheses for D and $\tilde{D}$:

$$h : \mathcal{X} \longrightarrow \{0,1\}^m \;\cong\; \tilde{h} : \mathcal{X} \times \mathscr{L} \longrightarrow \{0,1\}),$$

because every prediction made by h corresponds to exactly one prediction made by $\tilde{h}$, and the other way around. This correspondence is obviously given that

$$h(\boldsymbol{x})[l] = \tilde{h}(\boldsymbol{x}, l)$$

for inputs $\boldsymbol{x} \in \mathcal{X}$ and labels $l \in \mathscr{L}$. And, according to the definitions of Section 2.1, we have that

Proposition 1. *A ML hypothesis h is optimal for a micro-averaged loss function L if and only if $\tilde{h}$ is optimal for the binary version of L.*

Notice that the optimality of h depends on $\tilde{h}$ being optimal for the binary version of L. This means that we should employ a learner that takes into account label interdependencies, considering together the predictions for all labels. However, if $\tilde{h}$ is induced assuming label independence, which usually does not

hold in ML classification problems, the corresponding hypothesis h will not be optimal.

The consequence of this result is that from a theoretical point of view, now we could obtain, for instance, optimal micro-averaged F_1 multilabel classifiers if we learn optimal binary classifiers for that performance measure. However, it is not straightforward to do that in practice.

Firstly, in a SVM approach, we must define a kernel suitable for capturing the similarities of pairs $(\boldsymbol{x}, l)$ of entries of $\tilde{D}$ built from an ML task D as in (7). In the next section, we describe a kernel method that is able to exploit unconditional label dependence in the context of the reduction framework described here.

4. A Kernel Method to Optimize Micro-Averaged Measures

In this section, the core of the paper, we are going to present a family of kernels aiming to improve the performance measured by micro-averaged loss functions.

4.1. A Tensor Space for ML classification

Remember that our goal is to learn an hypothesis $\tilde{h}$ that somehow take into account both the description of the objects and label information. So we need to combine these two elements.

Let us consider the linear properties desired for a hypothesis $\tilde{h}$ learned from the binary task $\tilde{D}$ derived from the original ML task D. We distinguish between the two arguments of $\tilde{h}(\boldsymbol{x}, l)$. With respect to the label $l \in \mathscr{L}$, we could hope to have the following *additive* property. If l and l' are labels assigned to an input $\boldsymbol{x}$, then the union $\{l, l'\}$ must be included in the predictions for $\boldsymbol{x}$. Somehow, this condition could be expressed linearly if labels were represented in a vectorial space. Thus, the first step is to represent the labels in a vectorial space.

The simplest representation is to use a vectorial space of m dimensions and to map each label l to the canonical base indexed by l. In symbols,

$$e : \mathscr{L} \longrightarrow \mathbb{R}^m;\ e(l) = \boldsymbol{e}_l = (\delta_{i,l} : i = 1, \ldots, m). \tag{8}$$

Given a label l we obtain a vector, $\boldsymbol{e}_l$, of m dimensions in which just one of the components is 1 and the rest are 0. The components of the vector $\boldsymbol{e}_l$ are the Kronecker's delta: $\delta_{i,l} = [\![i = l]\!]$. This is the simplest representation but there could be other ones, more complex, as we shall see below.

Factorizing the hypothesis $\tilde{h}$ through e, the additive property for labels means that, for a fixed input $\boldsymbol{x}$, $\tilde{h}(\boldsymbol{x}, \cdot)$ is linear.

On the other hand, for a fixed label l, we usually search for a linear hypothesis $\tilde{h}(\cdot, l)$ in $\mathbb{R}^p$. In other words, we are searching for *bilinear* hypotheses

$$\tilde{h} : \mathbb{R}^p \times \mathbb{R}^m \longrightarrow \mathbb{R}.$$

That is, for any $\boldsymbol{x} \in \mathbb{R}^p$, the map $\tilde{h}(\boldsymbol{x}, \boldsymbol{e}_l)$ is a linear map from $\mathbb{R}^m$ to $\mathbb{R}$, and for any representation of a label through e, $\boldsymbol{e}_l \in \mathbb{R}^m$, the map from $\mathbb{R}^p$ to $\mathbb{R}$ is also linear.

Mathematically, this is equivalent to ask linearity for the extension of $\tilde{h}$ to the *tensor product* of two Hilbert spaces. The universal property of tensor products can be stated as follows

$$\text{Bilinear}(\mathbb{R}^p \times \mathbb{R}^m, \mathbb{R}) \cong \text{Linear}(\mathbb{R}^p \otimes \mathbb{R}^m, \mathbb{R}).$$

Then, finally, we are looking for a hypothesis built from a linear map from the tensor product. To ease the reading, we overload the name $\tilde{h}$ for the hypothesis defined from the tensor product space

$$\begin{array}{ccccc} \mathcal{X} \times \mathscr{L} & \xrightarrow{\mathbb{I} \times e} & \mathbb{R}^p \times \mathbb{R}^m & \xrightarrow{in} & \mathbb{R}^p \otimes \mathbb{R}^m \\ & & & \searrow^{\tilde{h}} & \downarrow{\scriptstyle \tilde{h}} \\ & & & & \mathbb{R} \end{array}$$

where $\mathbb{I}$ is the identity and in is the inclusion in the tensor product. Thus, the hypothesis learned from a ML task will have the form

$$\tilde{h}(\boldsymbol{x}, l) = \langle \boldsymbol{W}, \boldsymbol{x} \otimes \boldsymbol{e}_l \rangle.$$

Thus, to obtain our hypothesis $\tilde{h}$ we just need to learn the vector parameter $\boldsymbol{W}$ whose dimension is $p \times m$.

4.2. Kernels to Learn ML Classifiers

In the tensor space described above, the natural kernels are given by *tensor kernels*; that is, the product of kernels of the two factor spaces: the input space and the label space:

$$K^{\otimes}\big((\boldsymbol{x},l),(\boldsymbol{x}',l')\big) = k^{\mathbb{X}}(\boldsymbol{x},\boldsymbol{x}')\cdot k^{\mathbb{Y}}(l,l'). \tag{9}$$

Using the linear kernel in both spaces and the vectorial representation e for the labels defined in (8), we have that

$$\begin{aligned} K^{\otimes}\big((\boldsymbol{x},l),(\boldsymbol{x}',l')\big) &= k^{\mathbb{X}}_L(\boldsymbol{x},\boldsymbol{x}')\cdot k^{\mathbb{Y}}_L(\boldsymbol{e}_l,\boldsymbol{e}_{l'}) \\ &= \langle \boldsymbol{x},\boldsymbol{x}'\rangle \cdot \langle \boldsymbol{e}_l,\boldsymbol{e}_{l'}\rangle \\ &= \langle \boldsymbol{x},\boldsymbol{x}'\rangle \cdot \delta_{l,l'} \\ &= \langle \boldsymbol{x},\boldsymbol{x}'\rangle \cdot [\![l=l']\!]. \end{aligned} \tag{10}$$

To illustrate the previous formal framework, we may use the Kronecker's product of vectors to represent the tensor product. Thus, the Kronecker's product of $\boldsymbol{x}\in\mathcal{X}\subset\mathbb{R}^p$, and $\boldsymbol{e}_l\in\mathbb{R}^m$, for a label $l\in\mathscr{L}$, is defined by

$$\boldsymbol{x}\otimes\boldsymbol{e}_l = (\boldsymbol{x}\cdot\delta_{1,l},\ldots,\boldsymbol{x}\cdot\delta_{m,l})\in(\mathbb{R}^p)^m\cong\mathbb{R}^p\otimes\mathbb{R}^m.$$

The interpretation of these products in the learning task (2) is that the input matrix $\tilde{\mathbb{X}}$ in $\tilde{D}$ (7) is now the Kronecker's product of the original input matrix $\mathbb{X}$ and the identiy matrix of dimension m.

$$\tilde{\mathbb{X}} = \mathbb{X}\otimes\mathbb{I}_m. \tag{11}$$

In this context, the hypothesis $\tilde{h}$ is defined by a vector

$$\boldsymbol{W} = (\boldsymbol{w}_1,\ldots,\boldsymbol{w}_m)\in\mathbb{R}^{p\times m},\ \boldsymbol{w}_i\in\mathbb{R}^p. \tag{12}$$

This means that the algorithm will learn all together the m parameter vectors, $\boldsymbol{w}_i$, one corresponding for each label. This may allow the learner to capture the relationships between all labels.

In order to predict the relevance of a particular label l we have that

$$\begin{aligned} \tilde{h}(\boldsymbol{x}, l) &= \langle \boldsymbol{W}, \boldsymbol{x} \otimes \boldsymbol{e}_l \rangle \\ &= \sum_{i=1}^{m} \langle \boldsymbol{w}_i, \boldsymbol{x} \rangle \cdot \delta_{i,l} = \langle \boldsymbol{w}_l, \boldsymbol{x} \rangle. \end{aligned} \tag{13}$$

Notice that in the preceding derivation, we lost the *intercept terms* frequently used in the hypotheses learned by SVMs. In order to use standard software, we add a new constant feature for all vectors $\boldsymbol{x}$. This trick recovers the intercept terms in linear kernels. We only use this kernel in the input space $\mathcal{X}$. Thus, in practice,

$$\mathcal{X} \subset \mathbb{R}^{p+1}. \tag{14}$$

4.3. Correlation Kernel for Labels

The problem of using the identity matrix in (11), i.e., to use (8) as the vectorial space to represent the labels and the linear kernel $\langle \boldsymbol{e}_l, \boldsymbol{e}_{l'} \rangle = \delta_{l,l'}$ in (10) and (13), is that the algorithm can not capture the relationship between labels. In other words, we are assuming that the labels are independent, like BR does.

However, the previous framework allows to easily overcome this drawback. In fact, it is as simple as plugging another kernel for labels in $K^{\otimes}$. This paper proposes and analyzes a kernel for labels that captures unconditional dependencies among them. In the experimental section at the end of the paper we will use the correlation coefficient between labels as a kernel. Thus, if $\mathbb{Y}$ is the matrix of the labels (see Section 2) and l_k is the k-th column that represents the corresponding label, the kernel can be given by

$$k_{\rho}^{\mathbb{Y}}(l_i, l_j) = \rho(l_i, l_j). \tag{15}$$

in which ρ computes the correlation between label l_i and label l_j.

In order to consider this kernel, once we have a training set with matrix of labels $\mathbb{Y}$, we only need to redefine a new representation map instead of e (8). Thus, each label is represented by the standardization of the corresponding column in $\mathbb{Y}$:

$$\psi_{\mathbb{Y}}(l_k) = \frac{l_k - \mu(l_k)}{\sigma(l_k)\sqrt{n}}, \tag{16}$$

in which μ and σ represents respectively the mean and the standard deviation of l_k. We call $\mathbb{Y}_\psi$ the matrix of labels transformed by ψ. Then, the inner product is the correlation:

$$\langle \psi_{\mathbb{Y}}(l_i), \psi_{\mathbb{Y}}(l_j) \rangle = K_\rho^{\mathbb{Y}}(l_i, l_j).$$

Finally, we have the tensor correlation kernel

$$K_\rho^{\otimes}\big((\boldsymbol{x}, l_i), (\boldsymbol{x}', l_j)\big) = \langle \boldsymbol{x}, \boldsymbol{x}' \rangle \cdot K_\rho^{\mathbb{Y}}(l_i, l_j). \tag{17}$$

Notice that while this kernel is characterized by the correlation matrix, the kernel defined in (10) is built from the identity matrix. This means, that when we use the identity as the similarity matrix of labels, labels are considered independent among them. In the following, we will refer to the algorithms using these kernels as $\text{SVM}_\rho^{\otimes}$ and $\text{SVM}_I^{\otimes}$ respectively.

The matrix representation of the input space $\tilde{\mathbb{X}}$ in the learning task $\tilde{D}$, according to (16) is now

$$\tilde{\mathbb{X}} = \mathbb{X} \otimes (\mathbb{Y}_\psi)'.$$

Notice that it is not necessary to compute explicitly this matrix. The kernel (17) does this job implicitly for us. Of course, this is also the case of the matrix (11) and the kernel (10).

5. Related Work

Multilabel classification has received the attention of many contributions with different approaches. In [9] and [24] the approach was an extension of multiclass classification. Other extensions have been done from different settings; this is the case of Bayesian learners [33], nearest neighbors [34], logistic regression [15], decision trees [29] and ensembles [14].

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 [3, 17, 18] and Classifier Chains (CC) [23].

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 form the learning task. This includes [9] and [20].

There are other approaches that aim to explicitly optimize a given loss function. Usually, they are based on theoretical results that inspire heuristic implementations to improve the scores of ML learners, see [4], [19] and [20]. In this group we could include also approaches that perform some sort of inference, for instance, those that optimize subset 0/1 loss using Probabilistic Classifier Chains [3, 6, 13, 21], and [8], a method that treats a ML task as a special case of structured-output prediction and makes joint predictions without suffering from intractability of the inference problem.

On the other hand, this paper is related to those that employ a combined feature representation. This is the case of [12, 26] that uses joint features maps, a combined feature representation of inputs and outputs, and, from a different perspective [32]. With regard to former algorithms, the main novelty of our method is the use of a specific correlation kernel defined over the label space. The use of tensor kernels is also highlighted in [10] to incorporate samples from multiple sources into a joint kernel defined feature space. They are also related to *joint kernels* in [31], which were introduced to deal with learning tasks where the output has very high dimension. The tensor product representation is also used in Maximum Margin Regression (MMR) [25], a maximum margin framework that can be applied to multiclass (vector valued) classification, i.e. ML classification tasks, to learn a multi-variate score function in the tensor product space.

Other related works can be found in [19], where the authors present a *reverse* classification where the aim was to optimize macro-averaged loss functions; also, closely related to our approach is [11], which proposes an efficient algorithm named M3L to tackle multi label problems. Their approach uses the tensor product to build the prediction function and incorporates prior knowledge in the form of the distribution of the labels. The main differences with our method is that 1) we use a standard SVM implementation while they propose a different formulation, specially adapted for optimizing Hamming loss, and 2)

we define a kernel that captures unconditional label dependences. We included this algorithm in the experimental comparison described below.

6. Experimental Results

In this section we report a number of experiments carried to show the practical performance of the kernel method based on correlations that was presented in Section 4.3. The goal of these experiments was to show if our approach can benefit when some level of unconditional label dependence is present. In order to study this aspect, we used a measure, previously defined in [16], to quantify unconditional dependence that is given by the average of the correlation of labels (in absolute value) weighted by the number of common examples. In symbols,

$$\text{dependency}(\mathbb{Y}) = \frac{\sum_{i<j} \text{abs}(\rho(l_i, l_j))|l_i \cap l_j|}{\sum_{i<j} |l_i \cap l_j|}. \tag{18}$$

As our target loss function we could employ any micro-averaged loss function L described in Section 2.1. Recall that in order to optimize L in D we must use a binary classifier able to optimize the binary version of L in $\tilde{D}$. In these experiments we focused on Hamming loss, which only requires to optimize the accuracy in $\tilde{D}$ and can be achieved by using a standard SVM implementation.

We modified a general-purpose SVM to implement the kernels presented in Section 4. We used LibSVM [1] adding the tensor kernel in two versions. In the part of $\boldsymbol{x}$-inputs the linear kernel was employed in all cases after adding a new column (14) with a constant value set to 1. To deal with label similarities we used the two kernels discussed above: the one based on the identity matrix (10) and the correlation kernel (17). We named these algorithms $\text{SVM}_I^{\otimes}$ and $\text{SVM}_\rho^{\otimes}$, respectively. We compared the results of our approach to those of BR, CC (both using SVM as their base classifier) and M3L, discussed in Section 5. This last method, like ours, was devised to take advantage of dependency information between labels by means of a similarity matrix. Thus, we used two versions of the algorithm, M3L_I and M3L_ρ, using again the identity and the correlation matrix, respectively.

Notice that $\text{SVM}_I^{\otimes}$ and BR can be considered similar approaches in the sense that both, in order to build the model, assume independence between labels (or, stated differently, have a lack of explicit information about the dependency between labels). The difference is that $\text{SVM}_I^{\otimes}$ makes a global regularization for the whole model, while the BR method regularizes each classifier independently.

The goal of these experiments was to study whether exploiting unconditional dependence could help to improve Hamming scores. To check such hypothesis, we made a thoroughly comparison on a collection of benchmark and synthetic datasets.

6.1. Results over Benchmark Datasets

Table 2 shows the main characteristics of the benchmark datasets used in the experiments. The most important one is the level of unconditional label dependence (18). All these datasets were download from MULAN repository [30] but one, *slashdot*, that was obtained from LAMDA repository [35]. Whenever there was available a split in train and test, we used it; in the other cases, we made a random split separating 80% of the examples for training and the rest for testing. The dataset *mediamill* is available in MULAN; however, we used the abbreviated version of [2] with 5000 examples to speed up the experimentation.

When the value of a label in a training set was constant (always present or absent), we removed that label from the dataset. This was done to be able to compute the correlations ensuring that we are not going to find null standard deviations. We added the number of remaining labels to the name of the datasets, to indicate which were pruned. This is the case of *enron(52), medical(38), slashdot(20)* and *mediamill(87)*.

In order to adjust the regularization parameter C of all the classifiers considered, half of the training set was used for learning the models and the other half was used as the *validation* set. Once an optimal C value was found, we used it on the whole training set, and the model so obtained was applied to the test set. The range of C values explored was $\{10^i : i = -2, -1, 0, 1\}$. Notice that the kernels used, (10) and (17), by M3L and $\text{SVM}^{\otimes}$ do not have additional

Table 2: Summary of benchmark datasets. For each dataset the dependency measured using (18) is reported in the last column. The datasets are sorted by this value in ascending order

Dataset	|Train|	|Test|	feats.	labels	card.	dep.
slashdot(20)	3025	757	1079	20	1.18	0.05
enron(52)	1123	579	1001	52	3.39	0.12
scene	1211	1196	294	6	1.06	0.14
cal500	402	100	68	174	26.17	0.14
medical(38)	333	645	1449	38	1.26	0.22
mediamill(87)	4000	1000	120	87	4.27	0.23
yeast	1500	917	103	14	4.23	0.25
emotions	391	202	72	6	1.81	0.27
reuters	5695	1424	243	7	1.24	0.31

parameters.

Table 3 reports the Hamming scores obtained by all the algorithms over the benchmark datasets. At first glance, their results are quite similar for all of them, except CC. In fact, comparing their scores by means of a Friedman-Nemenyi test [7] the differences between BR, M3L$_I$, M3L$_\rho$, SVM$_I^\otimes$ and SVM$_\rho^\otimes$ are not significant at all. We do think that this fact makes sense considering the properties of the algorithms compared and the characteristics of the benchmark datasets used in the experiments. In this kind of experiment, which combines datasets with low and high label dependence, it would be abnormal that one of the methods outperforms the rest. The reason is that some of the methods are well suited for those tasks in which the labels tend to be independent (BR, M3L$_I$ and SVM$_I^\otimes$), other for domains in which the labels show unconditional label dependence (M3L$_\rho$ and SVM$_\rho^\otimes$) and the last method (CC) for datasets with conditional label dependence. Thus, it is logical that none of the methods statistically outperforms the rest for all datasets. Each method should perform better in those datasets that fit to its properties regarding label dependence.

Table 3: Average Hamming scores achieved by SVM$^{\otimes}$, M3L, BR and CC. SVM$^{\otimes}$, M3L both used without and with the correlation matrix on real world datasets. The score of the best method for each dataset is shown in bold

Dataset	BR	M3L$_I$	SVM$_I^{\otimes}$	M3L$_\rho$	SVM$_\rho^{\otimes}$	CC
slashdot(20)	**0.0400**	0.0417	0.0417	0.0417	0.0414	0.0548
enron(52)	**0.0475**	**0.0475**	0.0481	0.0497	0.0483	0.0517
scene	**0.1012**	0.1063	0.1048	0.1062	0.1045	0.1158
cal500	0.1357	**0.1348**	0.1357	**0.1348**	0.1349	0.1819
medical(38)	0.0130	**0.0126**	0.0133	0.0133	0.0141	0.0131
mediamill(87)	**0.0325**	0.0330	0.0330	0.0331	0.0330	0.0335
yeast	0.1995	0.1990	0.1989	**0.1974**	0.1975	0.2110
emotions	0.2162	0.2112	0.2096	0.2162	**0.2021**	0.2376
reuters	0.0462	0.0458	0.0463	0.0456	0.0467	**0.0450**

Therefore, the Hamming scores reported must be analyzed taking into account the level of dependency of each dataset. Interestingly, the conclusions of such analysis are in line with the expected behavior of each method. It seems that when label dependence is very low BR performs better than the rest of the methods. For instance, in the three datasets with lowest label dependence (*enron*, *scene* and *slashdot*) the winner is BR. But when label dependence starts to increase, other methods take the lead. And particularly when the label dependence is high, the performance of M3L$_\rho$ and SVM$_\rho^{\otimes}$ is better. It is worth noting that M3L$_\rho$ and/or SVM$_\rho^{\otimes}$ outperform BR in the three datasets with largest label dependence (*emotions*, *reuters* and *yeast*). In the case of *reuters* dataset the absolute winner is CC, a method that performs poorly for most of the datasets. This seems to suggest that there is some level of conditional label dependence in such dataset.

Thus, these results are promising but do not totally support (statistically) our initial hypothesis that the difference in performance would be higher and

favorable to the versions using the correlation kernel when the dependency is large. On the other hand, these results may not be conclusive since the number of datasets was rather low to generalize the behavior of the algorithms. A larger collection of datasets, with different degrees of label dependence, would be needed.

6.2. Performance on Synthetic Multilabel Classification Tasks

In order to elucidate the relation between the unconditional dependence and the performance of $\text{SVM}_\rho^\otimes$ and M3L_ρ we generated a collection of synthetic datasets of different sizes and degrees of dependency. For this purpose, we used a synthetic dataset generator [16] based on a genetic algorithm to obtain ML datasets with specific characteristics. The aim of the generator was to produce a dataset with a given *cardinality* and *dependency*.

We built 108 datasets (each one composed by a training, a validation and a testing set) by varying four parameters. In all cases the input space was $\mathbb{R}^{p+1}$ with $p \in \{10, 25, 50, 100\}$, see (14). The size m of the set of labels varied in $\{100, 150, 200\}$. The target cardinalities (average number of labels per example) ranged in $\{2.5, 3.5, 4.5\}$. The desired values for dependency were in [0.05..0.65]. The generator draws a set of 400 points in $[0, 1]^p$ that will be used to build the training set. Then, it searches for m hyperplanes such that the set of labels so obtained fits as much as possible the cardinality and dependency required. Notice that it is nearly impossible to obtain a dataset fulfilling some combinations of these parameters. Thus, it can only obtains an approximate solution. After obtaining those hyperplanes, the validation and testing sets are generated with 400 and 600 examples respectively.

We used these synthetic datasets to check the performance of $\text{SVM}_I^\otimes$ and M3L_I with respect to the versions using the correlation kernel, $\text{SVM}_\rho^\otimes$ and M3L_ρ. After adjusting the C parameter using train and validation sets, the performance was computed on the test set. In this case, C ranged in $\{10^i : i = -1, 0, 1, 2, 3\}$.

We show the average Hamming scores achieved by $\text{SVM}_\rho^\otimes$ and $\text{SVM}_I^\otimes$ in Table 4 and Figure 1, where the datasets are grouped by the number of labels

Table 4: Average Hamming scores (expressed as percentages) achieved by SVM$^{\otimes}$ using I and ρ matrices for different number of labels and dimensions (p) of the input space. Each score is the average of 9 datasets. The last band shows the number of times that the correlation kernel helps to improve the performance of the algorithm SVM$^{\otimes}$

		SVM$^{\otimes}$ scores (% Hamming)			
	labels	$p = 10$	$p = 25$	$p = 50$	$p = 100$
I	100	0.85	1.32	1.77	2.54
	150	0.82	1.30	1.52	2.00
	200	0.81	1.13	1.38	1.78
	Average	0.83	1.25	1.56	2.11
ρ	100	0.76	1.20	1.64	2.38
	150	0.74	1.20	1.42	1.92
	200	0.77	1.07	1.31	1.68
	Average	0.76	1.15	1.46	1.99
ρ wins	100	8	6	7	9
	150	9	6	7	7
	200	8	7	6	7
	Total (out of 27)	25	19	20	23

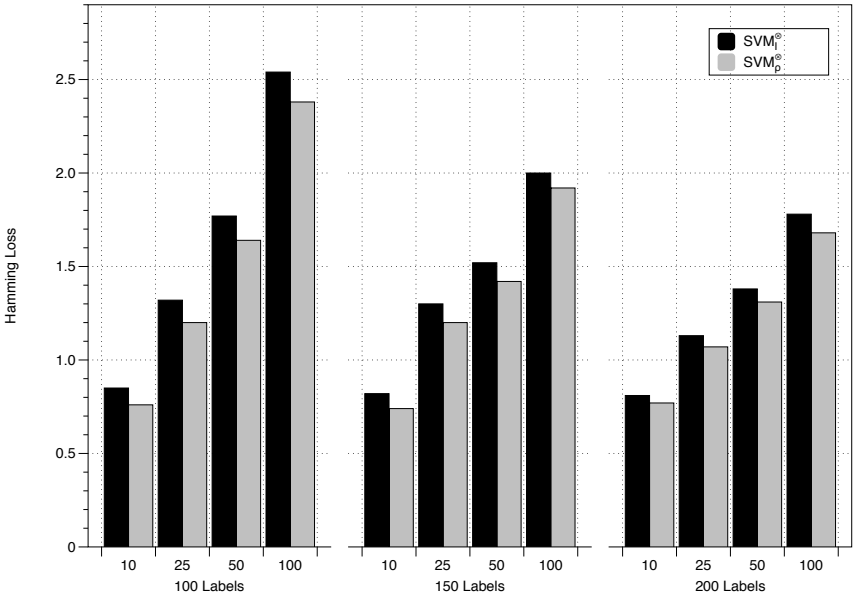

Figure 1: SVM$_\rho^{\otimes}$ and SVM$_I^{\otimes}$ average Hamming scores over artificial datasets when the number of labels varies (100,150 and 200). For each case, four different dimensions (10, 25, 50 and 100) of the input space were analyzed. The scores are expressed as percentages

(m) and input space dimension (p). In general, we observe that the use of the correlation kernel (17) improves the performance when m and p increase. This is shown in the last band of Table 4, which reveals the number of times the version using the correlation kernel wins over the version using the identity matrix. As we can observe, SVM$_\rho^{\otimes}$ outperforms SVM$_I^{\otimes}$ for every combination of the parameters studied. In fact, SVM$_\rho^{\otimes}$ wins in 87 cases out of 108. Moreover, it seems that the differences between both are greater in favor of SVM$_\rho^{\otimes}$ when the learning tasks become harder, when the size of the input space and the number of labels increase. In such cases, the role played by correlations appears to be decisive.

But we also need to study these results taken into account the level of unconditional dependence present. The advantage of using the correlation kernel analyzing this factor can be easily noticed by observing Figure 2 and Figure 3,

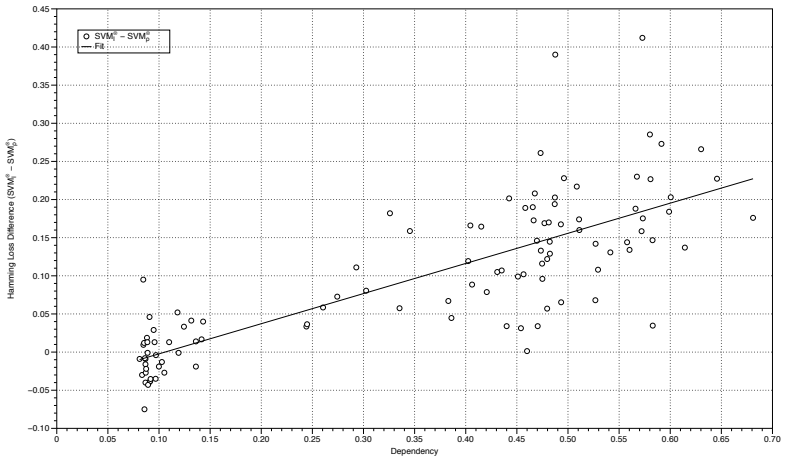

Figure 2: $\text{SVM}_\rho^\otimes$ vs. $\text{SVM}_I^\otimes$. For each synthetic dataset we represent a point whose X coordinate is the dependency of the training set and whose Y coordinate represents the benefit of using the correlation kernel in terms of Hamming score

which depict the increase in performance (difference between Hamming scores) obtained when using the correlation kernel with respect to the use of the identity matrix for both $\text{SVM}^\otimes$ and M3L, respectively. For instance, in Figure 2 we can observe that when unconditional dependence is low, it seems that $\text{SVM}_I^\otimes$ performs slightly better that $\text{SVM}_\rho^\otimes$. But, when the dependency increases, $\text{SVM}_\rho^\otimes$ is the clear winner. A similar behavior happens in the case of M3L, see Figure 3. These pictures show that the benefits of using the matrix ρ increase proportionally to the dependency. In Figure 4 we compared M3L_ρ and $\text{SVM}_\rho^\otimes$ and we can observe that the performance of the latter is better than the former and that the difference between both algorithms increases with the dependency too.

To statistically confirm these conclusions, we compare M3L_I, M3L_ρ, $\text{SVM}_I^\otimes$ and $\text{SVM}_\rho^\otimes$ by means of a Friedman-Nemenyi test [7] on the scores obtained over the synthetic datasets. The test, which is graphically summarized in Figure 5, found significant differences between using or not the correlation kernel both for $\text{SVM}^\otimes$ and for M3L. It also found that our approach $\text{SVM}_\rho^\otimes$ is significantly better than the rest of algorithms, although the differences were small. In

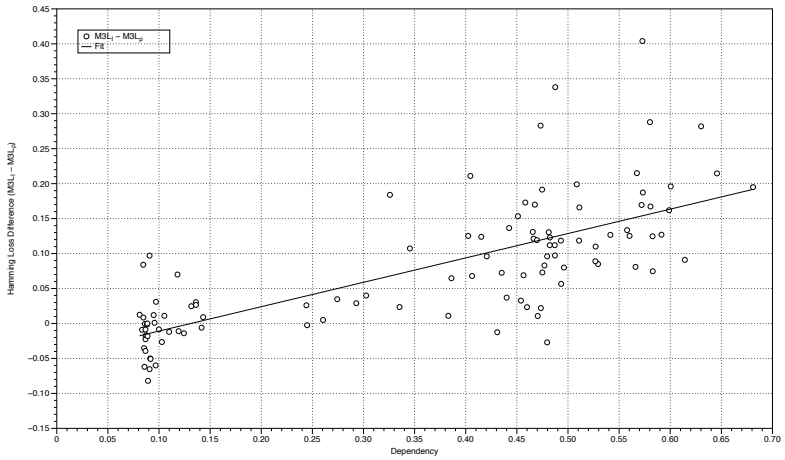

Figure 3: M3L$_\rho$ vs. M3L$_I$. For each synthetic dataset we represent a point whose X coordinate is the dependency of the training set and whose Y coordinate represents the benefit of using the correlation kernel in terms of Hamming score

addition, we performed a Wilcoxon signed-rank test between SVM$^{\otimes}_{\rho}$ and M3L$_\rho$, obtaining that the difference is significant with $p < 0.01$ in favor of our approach.

7. Conclusions

This paper presents a new kernel-based approach to handle ML classification tasks when the aim is to optimize a loss function defined by micro-averaging a measure based on labels. The core idea is to transform a ML task into a binary classification problem whose inputs are drawn from a tensor space of the original input space and a representation of the labels. In fact, the goal is to exploit the correlation between the labels because they are normally not assigned independently of each other; instead, they present statistical dependencies. From a learning point of view, these relationships constitute a crucial source of information, in addition to that coming from the mere description of the objects. Here, we propose tensor kernels for combining both sources of information, and a correlation kernel for taking advantage of such label correlations.

The experiments reported let us conclude that our approach is able to ben-

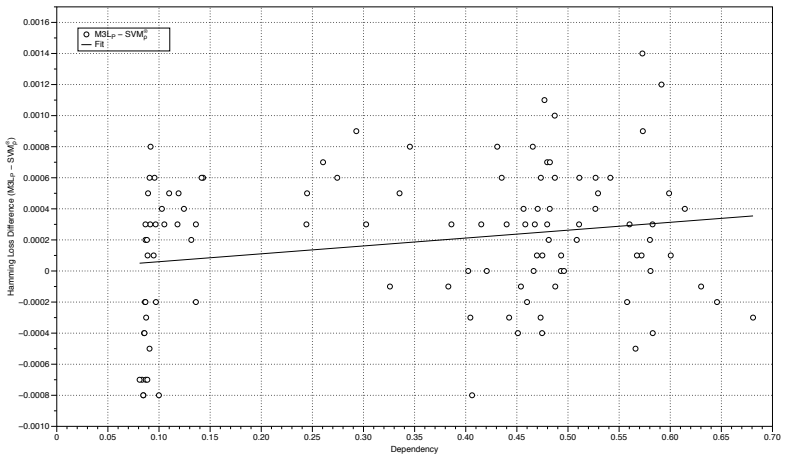

Figure 4: $\text{SVM}^{\otimes}_{\rho}$ vs. M3L_{ρ}. For each synthetic dataset we represent a point whose X coordinate is the dependency of the training set and whose Y coordinate represents the benefit of using the correlation kernel in terms of Hamming score

efit form capturing unconditional label dependence. Even in the case of Hamming loss, which theoretically can be optimized assuming independence between labels, like BR does, our method can improve such performance when the unconditional dependence increases. This is a quite interesting result because it answers the questions raised in [6] whether exploiting label dependence could help to improve Hamming loss performance. In their discussion, they conclude that the improvements obtained by some ML classifiers (Probabilistic Classifier Chains in their case) are due to employ a much richer hypothesis space in comparison to the one used by BR. Here, our approach enriches its hypothesis space by considering label correlations through a kernel and outperforms those methods that assume label independence whenever such correlations are present.

Acknowledgments

The research reported here is supported in part under grant TIN2011-23558 from the MICINN (Ministerio de Economía y Competitividad, Spain). We would also like to acknowledge all those people who generously shared the

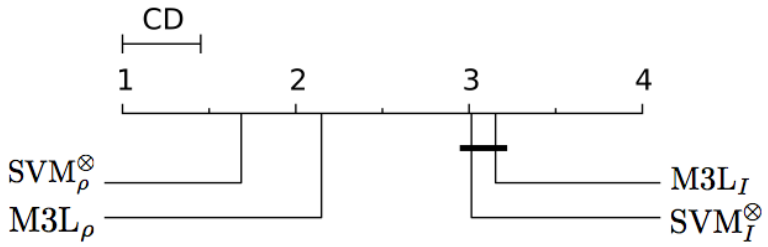

Figure 5: Results of a Friedman-Nemenyi test (confidence level of 95%) using the scores obtained by $\text{SVM}_I^\otimes$, $\text{SVM}_\rho^\otimes$, M3L_I and M3L_ρ. The scale from 1 to 4 shows the average rank of each method regarding its performance. The bold line links those algorithms whose differences in performance are not statistically significant

datasets and software used in this paper, as well as the anonymous reviewers who undoubtedly contributed to improve the quality of this paper.

References

[1] C.C. Chang, C.J. Lin, LIBSVM: A library for support vector machines, ACM Transactions on Intelligent Systems and Technology 2 (2011) 27:1–27:27.

[2] W. Cheng, E. Hüllermeier, Combining instance-based learning and logistic regression for multilabel classification, Machine Learning 76 (2009) 211–225.

[3] K. Dembczyński, W. Cheng, E. Hüllermeier, Bayes optimal multilabel classification via probabilistic classifier chains, Proceedings of the 27th International Conference on Machine Learning (ICML) (2010).

[4] K. Dembczyński, W. Waegeman, W. Cheng, E. Hüllermeier, An exact algorithm for f-measure maximization, in: Proceedings of the Neural Information Processing Systems (NIPS).

[5] K. Dembczyński, W. Waegeman, W. Cheng, E. Hüllermeier, On label dependence and loss minimization in multi-label classification, Machine Learning 88 (2012) 5–45.

[6] K. Dembczyński, W. Waegeman, E. Hüllermeier, An analysis of chaining in multi-label classification, in: ECAI 2012, pp. 294–299.

[7] J. Demšar, Statistical Comparisons of Classifiers over Multiple Data Sets, Journal of Machine Learning Research 7 (2006) 1–30.

[8] J.R. Doppa, J. Yu, C. Ma, A. Fern, P. Tadepalli, Hc-search for multi-label prediction: An empirical study, in: Proceedings of AAAI Conference on Artificial Intelligence (AAAI).

[9] A. Elisseeff, J. Weston, A kernel method for multi-labelled classification, in: In Advances in Neural Information Processing Systems 14, MIT Press, 2001, pp. 681–687.

[10] D. Hardoon, J. Shawe-Taylor, Decomposing the tensor kernel support vector machine for neuroscience data with structured labels, Machine learning 79 (2010) 29–46.

[11] B. Hariharan, S. Vishwanathan, M. Varma, Efficient max-margin multi-label classification with applications to zero-shot learning, Machine Learning 88 (2012) 127–155.

[12] T. Joachims, T. Finley, C. Yu, Cutting-plane training of structural svms, Machine Learning 77 (2009) 27–59.

[13] A. Kumar, S. Vembu, A.K. Menon, C. Elkan, Beam search algorithms for multilabel learning, Mach. Learn. 92 (2013) 65–89.

[14] P. Li, H. Li, M. Wu, Multi-label ensemble based on variable pairwise constraint projection, Information Sciences 222 (2013) 269 – 281.

[15] H. Liu, S. Zhang, X. Wu, Mlslr: Multilabel learning via sparse logistic regression, Information Sciences 281 (2014) 310–320.

[16] Ó. Luaces, J. Díez, J. Barranquero, J.J. del Coz, A. Bahamonde, Binary relevance efficacy for multilabel classification, Progress in Artificial Intelligence 4 (2012) 303–313.

[17] E. Montañés, R. Senge, J. Barranquero, J. Quevedo, J.J. del Coz, E. Hüllermeier, Dependent binary relevance models for multi-label classification, Pattern Recognition 47 (2014) 1494 – 1508.

[18] E. Montañés, J.R. Quevedo, J.J. del Coz, Aggregating independent and dependent models to learn multi-label classifiers, Machine Learning and Knowledge Discovery in Databases (2011) 484–500.

[19] J. Petterson, T. Caetano, Reverse multi-label learning, Advances in Neural Information Processing Systems 23 (2010) 1912—1920.

[20] J.R. Quevedo, O. Luaces, A. Bahamonde, Multilabel classifiers with a probabilistic thresholding strategy, Pattern Recognition 45 (2012) 876–883.

[21] J. Read, L. Martino, D. Luengo, Efficient monte carlo methods for multi-dimensional learning with classifier chains, Pattern Recognition 47 (2014) 1535 – 1546.

[22] J. Read, L. Martino, P.M. Olmos, D. Luengo, Scalable multi-output label prediction: From classifier chains to classifier trellises, Pattern Recognition 48 (2015) 2096–2109.

[23] J. Read, B. Pfahringer, G. Holmes, E. Frank, Classifier chains for multi-label classification, Machine Learning 85 (2011) 333–359.

[24] R. Schapire, Y. Singer, Boostexter: A boosting-based system for text categorization, Machine learning 39 (2000) 135–168.

[25] S. Szedmák, J. Shawe-Taylor, E. Parrado-Hernández, Learning via linear operators: Maximum margin regression; multiclass and multiview learning at one-class complexity, Technical Report, PASCAL, Southampton, UK., 2006.

[26] I. Tsochantaridis, T. Joachims, T. Hofmann, Y. Altun, Large margin methods for structured and interdependent output variables, Journal of Machine Learning Research 6 (2006) 1453.

[27] G. Tsoumakas, I. Katakis, Multi Label Classification: An Overview, International Journal of Data Warehousing and Mining 3 (2007) 1–13.

[28] G. Tsoumakas, I. Katakis, I. Vlahavas, Mining Multilabel Data, In O. Maimon and L. Rokach (Ed.), Data Mining and Knowledge Discovery Handbook, Springer (2010).

[29] G. Tsoumakas, I. Katakis, I. Vlahavas, Random k-Labelsets for Multi-Label Classification, IEEE Transactions on Knowledge Discovery and Data Engineering (2010).

[30] G.T. Tsoumakas, E. Spyromitros-Xioufis, J. Vilcek, I. Vlahavas, Mulan: A Java library for multi-label learning, Journal of Machine Learning Research 12 (2011) 2411–2414.

[31] J. Weston, B. Schölkopf, O. Bousquet, Joint kernel maps, Computational Intelligence and Bioinspired Systems (2005) 135–168.

[32] J. Yu, Y. Rui, Y.Y. Tang, D. Tao, High-order distance-based multiview stochastic learning in image classification, Cybernetics, IEEE Transactions on 44 (2014) 2431–2442.

[33] M.L. Zhang, J.M. Peña, V. Robles, Feature selection for multi-label naive bayes classification, Information Sciences 179 (2009) 3218–3229.

[34] M.L. Zhang, Z. Zhou, ML-KNN: A Lazy Learning Approach to Multi-label Learning, Pattern Recognition 40 (2007) 2038–2048.

[35] Z. Zhou, Learning And Mining from DatA (LAMDA). http://lamda.nju.edu.cn/data.ashx, 2010.