# Rectifying Classifier Chains for Multi-Label Classification

**Robin Senge**[1], **Juan José del Coz**[2] **and Eyke Hüllermeier**[1]
[1]:Department of Mathematics and Computer Science, University of Marburg, Germany
[2]:Artificial Intelligence Center, University of Oviedo at Gijón, Spain

## Abstract

Classifier chains have recently been proposed as an appealing method for tackling the multi-label classification task. In addition to several empirical studies showing its state-of-the-art performance, especially when being used in its ensemble variant, there are also some first results on theoretical properties of classifier chains. Continuing along this line, we analyze the influence of a potential pitfall of the learning process, namely the discrepancy between the feature spaces used in training and testing: While true class labels are used as supplementary attributes for training the binary models along the chain, the same models need to rely on estimations of these labels at prediction time. We elucidate under which circumstances the attribute noise thus created can affect the overall prediction performance. As a result of our findings, we propose two modifications of classifier chains that are meant to overcome this problem. Experimentally, we show that our variants are indeed able to produce better results in cases where the original chaining process is likely to fail.

## 1 Introduction

Multi-label classification (MLC) has attracted increasing attention in the machine learning community during the past few years. Apart from being interesting theoretically, this is largely due to its practical relevance in many domains, including text classification, media content tagging and bioinformatics, just to mention a few. The goal in MLC is to induce a model that assigns a *subset* of labels to each example, rather than a single one as in multi-class classification. For instance, in a news website, a multi-label classifier can automatically attach several labels—usually called tags in this context—to every article; the tags can be helpful for searching related news or for briefly informing users about their content.

Current research on MLC is largely driven by the idea that optimal predictive performance can only be achieved by modeling and exploiting *statistical dependencies* between labels. Roughly speaking, if the relevance of one label may depend on the relevance of others, then labels should be predicted *simultaneously* and not *separately*. This is the main argument against simple *decomposition techniques* such as binary relevance (BR) learning, which splits the original multi-label task into several independent binary classification problems, one for each label.

Until now, several methods for capturing label dependence have been proposed in the literature. They can be categorized according to two major properties: (i) the size of the subsets of labels for which dependencies are modeled and (ii) the type of label dependence they seek to capture. Looking at the first property, there are methods that only consider pairwise relations between labels [5; 6; 14; 19] and approaches that take into account correlations among larger label subsets [12; 13; 17]; the latter include those that consider the influence of all labels simultaneously [2; 8; 11]. Regarding the second criterion, it has been proposed to distinguish between the modeling of *conditional* and *unconditional label dependence* [3; 4], depending on whether the dependence is conditioned on an instance [3; 11; 13; 16] or describing a kind of global correlation in the label space [2; 8; 19].

In this paper, we focus on a method called *classifier chains* (CC) [13]. This method enjoys great popularity, even though it has been introduced only lately. As its name suggests, CC selects an order on the label set—a *chain* of labels—and trains a binary classifier for each label in this order. The difference with respect to BR is that the feature space used to induce each classifier is extended by the previous labels in the chain. These labels are treated as additional attributes, with the goal to model conditional dependence between a label and its predecessors. CC performs particularly well when being used in an ensemble framework, usually denoted as *ensemble of classifier chains* (ECC), which reduces the influence of the label order.

Our study aims at gaining a deeper understanding of CC's learning process. More specifically, we address an issue that, despite having been noticed [4], has not been picked out as an important theme so far: Since information about preceding labels is only available for training, this information has to be replaced by estimations (coming from the corresponding classifiers) at prediction time. As a result, CC has to deal with a specific type of attribute noise: While a classifier is learnt on "clean" training data, including the true values of preceding labels, it is applied on "noisy" test data, in which true labels are replaced by possibly incorrect predictions. Obviously, this type of noise may affect the performance of each classifier in the chain. More importantly, since each classifier relies on its predecessors, a single false prediction might be propagated and possibly even reinforced along the whole chain.

The contribution of this paper is twofold. First, we analyze the above problem of classifier chains in more detail. Using both synthetic and real data sets, we design experiments in order to reveal those factors that influence the effect of error propagation in CC. Second, we propose and

evaluate modifications of the original CC method that are intended to overcome this problem.

The rest of the paper is organized as follows. The next section introduces the setting of MLC more formally, and Section 3 explains the classifier chains method. Section 4 is devoted to a deeper discussion of the aforementioned pitfalls of CC, along with some first experiments for illustration purposes.[1] In Section 5, we introduce modifications of CC and propose a method called *nested stacking*. An empirical study, in which we experimentally compare this method with the original CC approach, is presented in Section 6. The paper ends with a couple of concluding remarks in Section 7.

## 2 Multi-Label Classification

Let $\mathcal{L} = \{\lambda_1, \lambda_2, \ldots, \lambda_m\}$ be a finite and non-empty set of class labels, and let $\mathcal{X}$ be an instance space. We consider a MLC task with a training set $S = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_n, \boldsymbol{y}_n)\}$, generated independently according to a probability distribution $\mathbf{P}(\mathbf{X}, \mathbf{Y})$ on $\mathcal{X} \times \mathcal{Y}$. Here, $\mathcal{Y}$ is the set of possible label combinations, i.e., the power set of $\mathcal{L}$. To ease notation, we define $\boldsymbol{y}_i$ as a binary vector $\boldsymbol{y}_i = (y_{i,1}, y_{i,2}, \ldots, y_{i,m})$, in which $y_{i,j} = 1$ indicates the presence (relevance) and $y_{i,j} = 0$ the absence (irrelevance) of $\lambda_j$ in the labeling of $\boldsymbol{x}_i$. Under this convention, the output space is given by $\mathcal{Y} = \{0, 1\}^m$. The goal in MLC is to induce from $S$ a hypothesis $\boldsymbol{h} : \mathcal{X} \longrightarrow \mathcal{Y}$ that correctly predicts the subset of relevant labels for unlabeled query instances $\boldsymbol{x}$.

The most straightforward and arguably simplest approach to tackle the MLC problem is *binary relevance* (BR) learning. The BR method reduces a given multi-label problem with $m$ labels to $m$ *binary classification* problems. More precisely, $m$ hypotheses $h_1, h_2, \ldots, h_m$ are induced, each of them being responsible for predicting the relevance of one label, using $\mathcal{X}$ as an input space:

$$h_j : \mathcal{X} \longrightarrow \{0, 1\} \tag{1}$$

In this way, the labels are predicted independently of each other and no label dependencies are taken into account.

In spite of its simplicity and the strong assumption of label independence, it has been shown theoretically and empirically that BR performs quite strong in terms of decomposable loss functions [3], including the well-known *Hamming loss*:

$$L_H(\boldsymbol{y}, \boldsymbol{h}(\boldsymbol{x})) = \frac{1}{m} \sum_{i=1}^{m} [\![ y_i \neq h_i(\boldsymbol{x}) ]\!] \tag{2}$$

The Hamming loss averages the standard 0/1 classification error over the $m$ labels and hence corresponds to the proportion of labels whose relevance is incorrectly predicted. Thus, if one of the labels is predicted incorrectly, this accounts for an error of $\frac{1}{m}$. Another extension of the standard 0/1 classification loss is the *subset 0/1 loss*:

$$L_{ZO}(\boldsymbol{y}, \boldsymbol{h}(\boldsymbol{x})) = [\![ \boldsymbol{y} \neq \boldsymbol{h}(\boldsymbol{x}) ]\!] \tag{3}$$

Obviously, this measure is more drastic and already treats a mistake on a single label as a complete failure. The necessity to exploit label dependencies in order to minimize the generalization error in terms of the subset 0/1 loss has been shown in [3].

[1] This section is partly based on [15]

## 3 Classifier Chains

While following a similar setup as BR, classifier chains (CC) seek to capture label dependencies. CC learns $m$ binary classifiers linked along a chain, where each classifier deals with the binary relevance problem associated with one label. In the training phase, the feature space of each classifier in the chain is extended with the actual label information of all previous labels in the chain. For instance, if the chain follows the order $\lambda_1 \rightarrow \lambda_2 \rightarrow \ldots \rightarrow \lambda_m$, then the classifier $h_j$ responsible for predicting the relevance of $\lambda_j$ is of the form

$$h_j : \mathcal{X} \times \{0, 1\}^{j-1} \longrightarrow \{0, 1\} \ . \tag{4}$$

The training data for this classifier consists of instances $(\boldsymbol{x}_i, y_{i,1}, \ldots, y_{i,j-1})$ labeled with $y_{i,j}$, that is, original training instances $\boldsymbol{x}_i$ supplemented by the relevance of the labels $\lambda_1, \ldots, \lambda_{j-1}$ preceding $\lambda_j$ in the chain.

At prediction time, when a new instance $\boldsymbol{x}$ needs to be labeled, a label subset $\boldsymbol{y} = (y_1, \ldots, y_m)$ is produced by successively querying each classifier $h_j$. Note, however, that the inputs of these classifiers are not well-defined, since the supplementary attributes $y_{i,1}, \ldots, y_{i,j-1}$ are not available. These missing values are therefore replaced by their respective predictions: $y_1$ used by $h_2$ as an additional input is replaced by $\hat{y}_1 = h_1(\boldsymbol{x})$, $y_2$ used by $h_3$ as an additional input is replaced by $\hat{y}_2 = h_2(\boldsymbol{x}, \hat{y}_1)$, and so forth. Thus, the prediction $\boldsymbol{y}$ is of the form

$$\boldsymbol{y} = \big( h_1(\boldsymbol{x}),\, h_2(\boldsymbol{x}, h_1(\boldsymbol{x})), \ldots \big)$$

Realizing that the order of labels in the chain may influence the performance of the classifier, and that an optimal order is hard to anticipate, the authors in [13] propose the use of an ensemble of CC classifiers. This approach combines the predictions of different random orders and, moreover, uses a different sample of the training data to train each member of the ensemble. *Ensembles of classifier chains* (ECC) have been shown to increase predictive performance over CC by effectively using a simple voting scheme to aggregate predicted relevance sets of the individual CCs: For each label $\lambda_j$, the proportion $\hat{w}_j$ of classifiers predicting $y_j = 1$ is calculated. Relevance of $\lambda_j$ is then predicted by using a threshold $t$, that is, $\hat{y}_j = [\![ \hat{w}_j \geq t ]\!]$.

## 4 The Problem of Attribute Noise in Classifier Chains

The learning process of CC violates a key assumption of supervised learning, namely the assumption that the training data is representative of the test data in the sense of being identically distributed. This assumption does not hold for the chained classifiers in CC: While using the *true* label data $y_j$ as input attributes during the training phase, this information is replaced by *estimations* $\hat{y}_j$ at prediction time. Needless to say, $y_j$ and $\hat{y}_j$ are not guaranteed to follow the same distribution; on the contrary, unless the classifiers produce perfect predictions, these distributions are likely to differ in practice (in particular, note that the $\hat{y}_j$ are deterministic predictions whereas the $y_j$ normally follow a non-degenerate probability distribution).

From the point of view of the classifier $h_j$, which uses the labels $y_1, \ldots, y_{j-1}$ as additional attributes, this problem can be seen as a problem of *attribute noise*. More specifically, we are facing the "clean training data vs. noisy test data" case, which is one of four possible noise scenarios that have been studied quite extensively in [20]. For CC,

this problem appears to be vital: Could it be that the additional label information, which is exactly what CC seeks to exploit in order to gain in performance (compared to BR), eventually turns out to be a source of impairment? Or, stated differently, could the additional label information perhaps be harmful rather than useful?

This question is difficult to answer in general. In particular, there are several factors involved, notably the following:

- *The length of the chain*: The larger the number $j-1$ of preceding classifiers in the chain, the higher is the potential level of attribute noise for a classifier $h_j$. For example, if prediction errors occur independently of each other with probability $\epsilon$, then the probability of a noise-free input is only $(1-\epsilon)^{j-1}$. More realistically, one may assume that the probability of a mistake is not constant but will increase with the level of attribute noise in the input. Then, due to the recursive structure of CC, the probability of a mistake will be reinforced and increase even more rapidly along the chain.
- *The order of the chain*: Since some labels might be inherently more difficult to predict than others, the order of the chain will play a role, too. In particular, it would be advantageous to put simpler labels in the beginning and harder ones more toward the end of the chain.
- *The accuracy of the binary classifiers*: The level of attribute noise is in direct correspondence with the accuracy of the binary classifiers along the chain. More specifically, these classifiers determine the input distributions in the test phase. If they are perfect, then the training distribution equals the test distribution, and there is no problem. Otherwise, however, the distributions will differ.
- *The dependency among labels*: Perhaps most interestingly, a (strong enough) dependence between labels is a prerequisite for both, an improvement and a deterioration through chaining. In fact, CC cannot gain (compared to BR) in case of no label dependency. In that case, however, it is also unlikely to loose, because a classifier $h_j$ will most likely[2] ignore the attributes $y_1, \ldots, y_{j-1}$. Otherwise, in case of pronounced label dependence, it will rely on these attributes, and whether or not this is advantageous will depend on the other factors above.

In the following, we present two experimental studies that are meant to illustrate the above issues. Based on our discussion so far and these experiments, two modifications of CC will then be introduced in the next sections, both of them with the aim to alleviate the problems outlined above.

### 4.1 First Experiment

Our intuition is that attribute noise in the test phase can produce a propagation of errors through the chain, thereby affecting the performance of the classifiers depending on their position in the chain. More specifically, we expect classifiers in the beginning of the chain to systematically perform better than classifiers toward the end. In order to verify this conjecture, we perform the following simple experiment: We train a CC classifier on 500 randomly generated label orders. Then, for each label order and each position, we compute the performance of the classifier on that position in terms of the relative increase of classification error compared to BR. Finally, these errors are averaged *position-wise* (not label-wise). For this experiment, we used three standard MLC benchmark data sets whose properties are summarized in Table 1 (shown in Section 5).

[2]The possibility to ignore parts of the input information does of course also depend on the type of base classifier used.

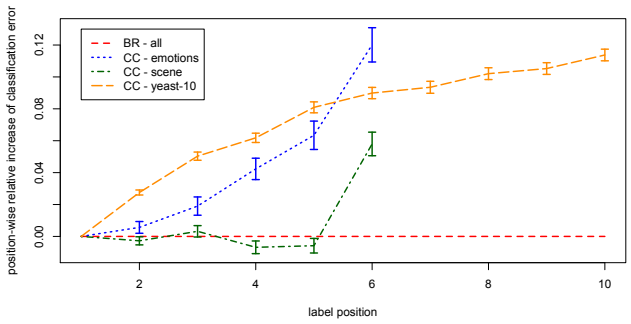


Figure 1: Results of the first experiment: position-wise relative increase of classification error (mean plus standard error bars). The *yeast-10* data set used here is a reduced yeast data set containing only the ten most frequent labels and their instances.

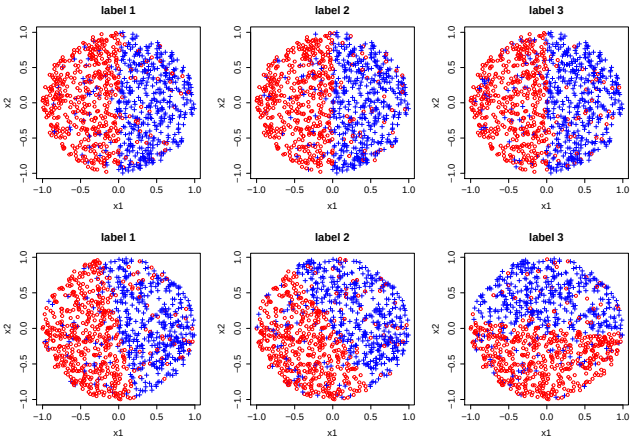


Figure 2: Example of synthetic data: the top three labels are generated using $\tau = 0$, the three at the bottom with $\tau = 1$.

The results in Figure 1 clearly confirm our expectations. In two cases, CC starts to loose immediately, and the loss increases with the position. In the third case, CC is able to gain on the first positions but starts to loose again later on.

### 4.2 Second Experiment

In a second experiment, we use a synthetic setup that was proposed in [4] to analyze the influence of label dependence. The input space $\mathcal{X}$ is two-dimensional and the underlying decision boundary for each label is linear in these inputs. More precisely, the model for each label is defined as follows:

$$h_j(\boldsymbol{x}) = \begin{cases} 1 & a_{j,1}x_1 + a_{j,2}x_2 \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

The input values are drawn randomly from the unit circle. The parameters $a_{j,1}$ and $a_{j,2}$ for the $j$-th label are set to

$$a_{j,1} = 1 - \tau r_1, \quad a_{j,2} = \tau r_2 \quad , \tag{6}$$

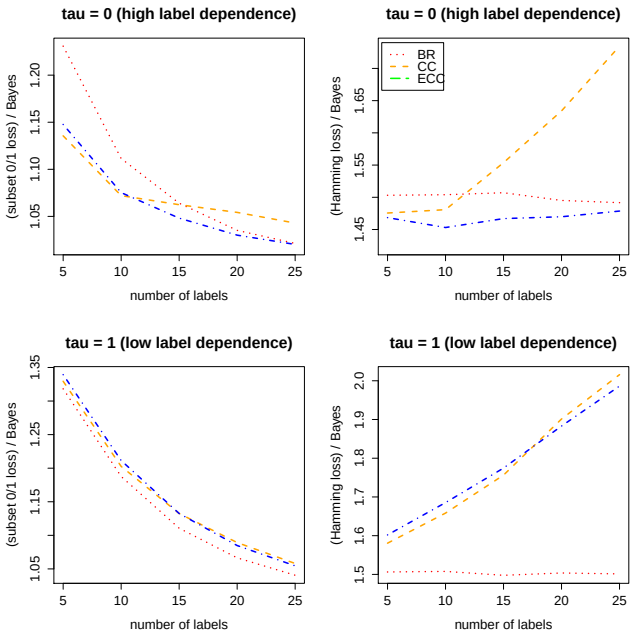


Figure 3: Results of the second experiment for $\tau = 0$ (top—high label dependence) and $\tau = 1$ (bottom—low label dependence).

with $r_1$ and $r_2$ randomly chosen from the unit interval. Additionally, random noise is introduced for each label by independently reversing a label with probability $\pi = 0.1$. Obviously, the level of label dependence can be controlled by the parameter $\tau$. Figure 2 shows two example data sets with three labels. The first one (pictures on the top) is generated with $\tau = 0$, the second one (bottom) with $\tau = 1$. As can be seen, the label dependence is quite strong in the first case, where the model parameters (6) are the same for each label. For the second case, the model parameters are different for each label. There is still label dependence, but certainly less pronounced.

For different label cardinalities $m \in \{5, 10, 15, 20, 25\}$, we run 10 repetitions of the following experiment: We created 10 different random model parameter sets (two for each label) and generated 10 different training sets, each consisting of 50 instances. For each training set, a model is learnt and evaluated (in terms of Hamming and subset 0/1 loss) on an additional data set comprising 1000 instances.

Figure 3 summarizes the results in terms of the average loss divided by the corresponding Bayes loss (which can be computed since the data generating process is known); thus, the optimum value is always 1. Apart from BR and CC, we already include the performance curve for the method to be introduced in the next section (NS); this should be ignored for now. Comparing BR and CC, the big picture is quite similar to the previous experiment: The performance of CC tends to decrease relative to BR with an increasing number of labels. In the case of low label dependence, this can already be seen for only five labels. The case of high label dependence is more interesting: While CC seems to gain from exploiting the dependency for a small to moderate number of labels, it cannot extend this gain to more than 15 labels.

# 5 Nested Stacking

A first very simple idea to mitigate the problem of attribute noise in CC is to let a classifier $h_j$ use predicted labels $\hat{y}_1, \ldots, \hat{y}_{j-1}$ as supplementary attributes for training instead of the true labels $y_1, \ldots, y_{j-1}$. This way, one could make sure that the data distribution is the same for training and testing. Or, stated differently, the situation faced by a classifier during training does indeed equal the one it will encounter later on at prediction time. Since then a classifier is trained on the predictions of other classifiers, this approach fits the stacked generalization learning paradigm [18], also simply known as *stacking*.

## 5.1 Stacking versus Nested Stacking

The idea of stacking has already been used in the context of MLC by Godbole and Sharawagi [8]. In the learning phase, their method builds a stack of two groups of classifiers. The first one is formed by the standard BR classifiers: $\boldsymbol{h}^1(\boldsymbol{x}) = (h_1^1(\boldsymbol{x}), \ldots, h_m^1(\boldsymbol{x}))$. On a second level, also called meta-level, another group of binary models (again one for each label) is learnt, but these classifiers consider an augmented feature space that includes the binary outputs of all models of the first level: $\boldsymbol{h}^2(\boldsymbol{x}, \boldsymbol{y}') = (h_1^2(\boldsymbol{x}, \boldsymbol{y}'), \ldots, h_m^2(\boldsymbol{x}, \boldsymbol{y}'))$, where $\boldsymbol{y}' = \boldsymbol{h}^1(\boldsymbol{x})$. The idea is to capture label dependencies by learning their relationships in the meta-level step. In the test phase, the final predictions are the outputs of the meta-level classifiers, $\boldsymbol{h}^2(\boldsymbol{x})$, using the outputs of $\boldsymbol{h}^1(\boldsymbol{x})$ exclusively to obtain the values of the augmented feature space.

Mimicking the chain structure of CC, our variant of stacking is a *nested* one: Instead of a two-level architecture as in standard stacking, we obtain a nested hierarchy of stacked (meta-)classifiers. Hence, we call it *nested stacking* (NS). Moreover, each of these classifiers is only trained on a subset of the predictions of other classifiers. Like in CC, $m$ models need to be trained in total, while $2m$ models are trained in standard stacking.

## 5.2 Out-of-Sample versus Within-Sample Training

To make sure that the distribution of the labels $\hat{y}_1, \ldots, \hat{y}_{j-1}$, which are used as supplementary attributes by the classifier $h_j$, is indeed the same at training and prediction time, these labels should be produced by means of an out-of-sample prediction procedure. For example, an internal leave-one-out cross validation procedure could be implemented for this purpose.

Needless to say, a procedure of that kind is computationally complex, even for classifiers that can be trained and "detrained" incrementally (such as incremental and decremental support vector machines [1]). In our current version of NS, we therefore implement a simple within-sample strategy. In several experimental studies, we found this strategy to perform almost as good as out-of-sample training, while being significantly faster. In fact, methods such as logistic regression, which are not overly flexible, are hardly influenced by excluding or including a single example.

## 5.3 A First Experiment

To get a first impression of the performance of NS, we return to the experiment in Section 4.2. As can be seen in Figure 3, NS does indeed gain in comparison to CC with an increasing number of labels; only if the labels are few, CC is still a bit better. This tendency is more pronounced

in the case of strong label dependency, whereas the differences are rather small if label dependence is low.

To explain the competitive performance of CC if the number of labels is small, note that replacing "clean" training data $y_1, \ldots, y_{j-1}$ by possibly more noisy data $\hat{y}_1, \ldots, \hat{y}_{j-1}$, as done by NS, may not only have the positive effect of making the training data more authentic. In fact, it may also make the problem of learning $h_j$ more difficult (because the dependency $y_1, \ldots, y_{j-1} \rightarrow y_j$ might be "easier" than the dependency $\hat{y}_1, \ldots, \hat{y}_{j-1} \rightarrow y_j$). Apparently, this effect plays an important role if the number of labels is small, whereas the positive effect dominates for longer label chains.

### 5.4 Subset Correction

Our second modification is motivated by the observation that the number of label combinations that are commonly observed in MLC data sets is only a tiny fraction of the total number $|\mathcal{Y}| = 2^m$ of possible subsets; see Table 1, which reports the value $|\mathcal{Y}_D|2^{-m}$, where $\mathcal{Y}_D$ is the set of unique label combinations contained in the data $D$, as the "observation rate" in the last column. Moreover, if a label combination $\boldsymbol{y}$ has an occurrence probability of $\epsilon > 0$, then the probability that it has never be seen in a data set of size $n$ reduces to $(1-\epsilon)^n$. Thus, by contraposition, one may argue that such a label combination is indeed unlikely to exist at all (at least for large enough $n$).

Our idea of "subset correction", therefore, is to restrict a learner to the prediction of label combinations whose existence is testified by the (training) data. More precisely, let $\mathcal{Y}_S$ denote the set of label subsets $\boldsymbol{y}$ that have be seen in the training data $S$. Then, given a prediction $\hat{\boldsymbol{y}}$ produced by a classifier $\boldsymbol{h}$, this prediction is replaced by the "most similar" subset $\boldsymbol{y}^* \in \mathcal{Y}_S$:

$$\boldsymbol{y}^* \in \operatorname*{argmin}_{\boldsymbol{y}' \in \mathcal{Y}_S} L_H(\hat{\boldsymbol{y}}, \boldsymbol{y}') \tag{7}$$

Thus, $\boldsymbol{y}^*$ is eventually returned as a prediction instead of $\hat{\boldsymbol{y}}$. If the minimum in (7) is not unique, those label combinations with higher frequency in the training data are preferred.

In principle, the Hamming loss could of course be replaced by other MLC loss functions in (7). Its use here is mainly motivated by the fact, that it is used for a similar purpose, namely decoding, in the framework of *error correcting output codes* (ECOC). As such, it has been applied in multi-class classification [**?**] and lately also in MLC [9; 7].

## 6 Nested Stacking vs. Classifier Chains

In this section, we compare NS and CC, both with and without subset correction, on real MLC benchmark data. As can be seen in Table 1, the data sets differ quite significantly in terms of the number of attributes, examples, labels, cardinality (number of labels per example) and the observation rate.

Logistic regression was used as a base learner for binary prediction in all MLC methods [10]. Unlike [13], we do not apply any threshold selection procedure; instead, we simply used $t = 0.5$ for deciding the relevance of a label. In fact, our goal is to study the behavior of CC and NS without the influence of other factors that may bias the results.

Since CC's main goal is to detect conditional label dependence, we used example-based metrics for evaluation. In addition to Hamming and subset 0/1 loss introduced earlier, we also applied the $F_1$ and Jaccard index defined, respectively, as follows (note that these are accuracy measures instead of loss functions):

$$F_1(\boldsymbol{y}, \boldsymbol{h}(\boldsymbol{x})) = \frac{2\sum_{i=1}^m [\![y_i = 1 \text{ and } h_i(\boldsymbol{x}) = 1]\!]}{\sum_{i=1}^m ([\![y_i = 1]\!] + [\![h_i(\boldsymbol{x}) = 1]\!])} \tag{8}$$

$$Jaccard(\boldsymbol{y}, \boldsymbol{h}(\boldsymbol{x})) = \frac{\sum_{i=1}^m [\![y_i = 1 \text{ and } h_i(\boldsymbol{x}) = 1]\!]}{\sum_{i=1}^m [\![y_i = 1 \text{ or } h_i(\boldsymbol{x}) = 1]\!]} \tag{9}$$

The value for a test set is defined as the average over all instances. The scores reported in Tables 2 and 3 were estimated by means of 10-fold cross-validation, repeated three times. We used a paired t-test for establishing statistical significance on each data set.

Table 4: The effect of subset correction in terms of statistical significance. The corresponsing loss/accuracy values can be found in Tables 2-3. $\uparrow\uparrow$ ($\downarrow\downarrow$) means that $\text{NS}_{SC}$ ($\text{CC}_{SC}$) is significantly better (worse) than NS (CC) at level $p < 0.01$ ($\uparrow$ and $\downarrow$ at level $p < 0.05$) in a paired t-test.

NS vs. $\text{NS}_{SC}$

| no. | $m$ | Hamming | Subset 0/1 | Jaccard | $F_1$ |
|---|---|---|---|---|---|
| 1 | 159 | $\uparrow\uparrow$ | $\uparrow\uparrow$ | $\uparrow\uparrow$ | $\uparrow\uparrow$ |
| 2 | 6 | | $\uparrow\uparrow$ | $\uparrow\uparrow$ | $\uparrow\uparrow$ |
| 3 | 53 | $\uparrow\uparrow$ | $\uparrow\uparrow$ | $\uparrow\uparrow$ | $\uparrow\uparrow$ |
| 4 | 27 | | $\uparrow\uparrow$ | | |
| 5 | 5 | $\downarrow\downarrow$ | $\uparrow\uparrow$ | $\uparrow\uparrow$ | $\uparrow\uparrow$ |
| 6 | 101 | $\uparrow\uparrow$ | $\uparrow\uparrow$ | | $\downarrow\downarrow$ |
| 7 | 45 | $\uparrow\uparrow$ | $\uparrow\uparrow$ | $\uparrow$ | $\downarrow\downarrow$ |
| 8 | 7 | $\downarrow\downarrow$ | $\uparrow\uparrow$ | $\uparrow\uparrow$ | $\uparrow\uparrow$ |
| 9 | 6 | $\downarrow\downarrow$ | $\uparrow\uparrow$ | $\uparrow\uparrow$ | $\uparrow\uparrow$ |
| 10 | 22 | $\uparrow\uparrow$ | $\uparrow\uparrow$ | $\uparrow\uparrow$ | |
| 11 | 14 | $\downarrow\downarrow$ | $\uparrow\uparrow$ | $\uparrow\uparrow$ | $\uparrow\uparrow$ |

CC vs. $\text{CC}_{SC}$

| no. | $m$ | Hamming | Subset 0/1 | Jaccard | $F_1$ |
|---|---|---|---|---|---|
| 1 | 159 | $\uparrow\uparrow$ | $\uparrow\uparrow$ | $\uparrow\uparrow$ | $\uparrow\uparrow$ |
| 2 | 6 | | $\uparrow$ | | |
| 3 | 53 | $\uparrow\uparrow$ | $\uparrow\uparrow$ | $\uparrow\uparrow$ | $\uparrow\uparrow$ |
| 4 | 27 | | $\uparrow\uparrow$ | | |
| 5 | 5 | | | | |
| 6 | 101 | $\uparrow\uparrow$ | $\uparrow\uparrow$ | $\uparrow\uparrow$ | $\uparrow\uparrow$ |
| 7 | 45 | $\uparrow\uparrow$ | $\uparrow\uparrow$ | | $\downarrow\downarrow$ |
| 8 | 7 | $\downarrow\downarrow$ | $\uparrow\uparrow$ | $\uparrow\uparrow$ | $\uparrow\uparrow$ |
| 9 | 6 | $\downarrow$ | $\uparrow\uparrow$ | | $\downarrow\downarrow$ |
| 10 | 22 | $\uparrow\uparrow$ | $\uparrow\uparrow$ | $\uparrow\uparrow$ | |
| 11 | 14 | | $\uparrow\uparrow$ | $\uparrow\uparrow$ | $\uparrow\uparrow$ |

Looking at the comparison between CC and NS (without subset correction) as shown in Table 2), the first thing to mention is the strong performance of NS in terms of Hamming loss (8 significant wins and 3 losses). In terms of their properties, the three data sets on which NS looses do indeed seem to be favorable for CC: Since slashdot, medical and genbase all have a rather low Hamming loss, the danger of error propagation is limited. Thus, the results are completely in agreement with our expectations.

For Jaccard and F1, the picture is not as clear. In both cases, NS wins 6 times. Again, like for Hamming loss, NS

Table 1: Properties of the data sets used in the experiments.

| no. | Data set | Attributes | Examples | Labels | Cardinality | Observation Rate |
|---|---|---|---|---|---|---|
| 1 | bibtex | 1836 | 7395 | 159 | 2.40 | 3.9E-45 |
| 2 | emotions | 72 | 593 | 6 | 1.87 | 4.0E-1 |
| 3 | enron | 1001 | 1702 | 53 | 3.38 | 8.3E-14 |
| 4 | genbase | 1185 | 662 | 27 | 1.25 | 2.3E-7 |
| 5 | image | 135 | 2000 | 5 | 1.24 | 6.0E-1 |
| 6 | mediamill | 120 | 5000 | 101 | 4.27 | 2.5E-27 |
| 7 | medical | 1449 | 978 | 45 | 1.25 | 2.6E-12 |
| 8 | reuters | 243 | 7119 | 7 | 1.24 | 1.9E-1 |
| 9 | scene | 294 | 2407 | 6 | 1.07 | 2.3E-1 |
| 10 | slashdot | 1079 | 3782 | 22 | 1.18 | 3.7E-5 |
| 11 | yeast | 103 | 2417 | 14 | 4.24 | 1.2E-2 |

Table 2: Experimental results of NS and CC on benchmark data sets. $\uparrow\uparrow$ ($\downarrow\downarrow$) means that NS is significantly better (worse) than CC at level $p < 0.01$ ($\uparrow$ and $\downarrow$ at level $p < 0.05$) in a paired t-test.

| | | $F_1$ | | | JACCARD INDEX | | |
|---|---|---|---|---|---|---|---|
| no. | $m$ | CC | NS | | CC | NS | |
| 1 | 159 | 0.1697±.0071 | 0.1747±.0077 | $\uparrow\uparrow$ | 0.1098±.0060 | 0.1133±.0064 | $\uparrow\uparrow$ |
| 2 | 6 | 0.5883±.0534 | 0.6028±.0500 | $\uparrow$ | 0.5003±.0521 | 0.5144±.0514 | $\uparrow$ |
| 3 | 53 | 0.3483±.0191 | 0.3729±.0214 | $\uparrow\uparrow$ | 0.2474±.0163 | 0.2693±.0178 | $\uparrow\uparrow$ |
| 4 | 27 | 0.9863±.0090 | 0.9854±.0085 | $\downarrow$ | 0.9804±.0115 | 0.9789±.0109 | $\downarrow$ |
| 5 | 5 | 0.5556±.0284 | 0.4780±.0299 | $\downarrow\downarrow$ | 0.5196±.0271 | 0.4460±.0278 | $\downarrow\downarrow$ |
| 6 | 101 | 0.5326±.0054 | 0.5619±.0053 | $\uparrow\uparrow$ | 0.4280±.0052 | 0.4459±.0052 | $\uparrow\uparrow$ |
| 7 | 45 | 0.6462±.0331 | 0.6444±.0340 | | 0.5828±.0343 | 0.5804±.0356 | |
| 8 | 7 | 0.8599±.0128 | 0.8570±.0116 | $\downarrow\downarrow$ | 0.8336±.0138 | 0.8302±.0129 | $\downarrow\downarrow$ |
| 9 | 6 | 0.5969±.0403 | 0.6031±.0348 | | 0.5745±.0405 | 0.5766±.0344 | |
| 10 | 22 | 0.3278±.0185 | 0.3259±.0186 | | 0.2747±.0176 | 0.2726±.0180 | |
| 11 | 14 | 0.5836±.0182 | 0.6068±.0172 | $\uparrow\uparrow$ | 0.4848±.0198 | 0.4990±.0183 | $\uparrow\uparrow$ |

| | | HAMMING LOSS | | | SUBSET 0/1 LOSS | | |
|---|---|---|---|---|---|---|---|
| no. | $m$ | CC | NS | | CC | NS | |
| 1 | 159 | 0.0724±.0020 | 0.0672±.0016 | $\uparrow\uparrow$ | 0.9837±.0052 | 0.9833±.0052 | |
| 2 | 6 | 0.2367±.0268 | 0.2169±.0253 | $\uparrow\uparrow$ | 0.7578±.0575 | 0.7477±.0633 | |
| 3 | 53 | 0.1233±.0051 | 0.1050±.0051 | $\uparrow\uparrow$ | 0.9565±.0135 | 0.9510±.0133 | $\uparrow$ |
| 4 | 27 | 0.0019±.0011 | 0.0020±.0010 | $\downarrow$ | 0.0408±.0211 | 0.0443±.0213 | $\downarrow$ |
| 5 | 5 | 0.2104±.0127 | 0.1962±.0119 | $\uparrow\uparrow$ | 0.5857±.0269 | 0.6468±.0249 | $\downarrow\downarrow$ |
| 6 | 101 | 0.0303±.0004 | 0.0291±.0004 | $\uparrow\uparrow$ | 0.8752±.0049 | 0.8969±.0048 | $\downarrow\downarrow$ |
| 7 | 45 | 0.0248±.0031 | 0.0249±.0031 | | 0.5890±.0425 | 0.5934±.0463 | |
| 8 | 7 | 0.0506±.0046 | 0.0483±.0043 | $\uparrow\uparrow$ | 0.2454±.0173 | 0.2499±.0175 | $\downarrow$ |
| 9 | 6 | 0.1470±.0143 | 0.1397±.0124 | $\uparrow\uparrow$ | 0.4918±.0434 | 0.5019±.0355 | $\downarrow$ |
| 10 | 22 | 0.0908±.0027 | 0.0913±.0028 | $\downarrow$ | 0.8652±.0185 | 0.8678±.0198 | |
| 11 | 14 | 0.2242±.0093 | 0.2069±.0087 | $\uparrow\uparrow$ | 0.8104±.0229 | 0.8469±.0231 | $\downarrow\downarrow$ |

outperforms CC on data sets with many labels (bibtex, enron, mediamill) or a relatively high Hamming loss (yeast), whereas CC is better for data sets with only a few labels (image, reuters) or with high accuracy (genbase).

The picture for CC and NS with subset correction (denoted $\text{CC}_{SC}$ and $\text{NS}_{SC}$, respectively) is quite similar (Table 3), although the performance differences tend to decrease in absolute size. On subset 0/1 loss, for which the original CC performs quite strong and typically outperforms NS, the corrected version $\text{NS}_{SC}$ even achieves 3 significant wins over $\text{CC}_{SC}$.

To analyze the effect of subset correction in more detail, Table 4 provides a summary of a comparison of Table 2 and Table 3. Interestingly enough, subset correction yields improvements on almost every experiment, regardless of the performance measure, and most of these improvements are even significant. More specifically, counting the number of significant wins, subset correction appears to be most beneficial for subset 0/1 loss and least beneficial for Hamming loss. In fact, for Hamming loss, subset correction looses for data sets with only a few labels (reuters, scene, yeast and image) and a relatively high observation rate. Comparing NS and CC, the former seems to benefit even more from subset correction than the latter, except for Hamming loss, on which NS is already strong in its basic version. In terms of subset 0/1 loss, however, significant improvements can be seen on every single data set. In light of the simplicity of the idea, these effects of subset correction are certainly striking.

## 7 Conclusions

This paper has thrown a critical look at the classifier chains method for multi-label classification, which has been adopted quite quickly by the MLC community and is now commonly used as a baseline when it comes to comparing methods for exploiting label dependency. Notwith-

Table 3: Experimental results of $\text{NS}_{SC}$ and $\text{CC}_{SC}$ on benchmark data sets. $\uparrow\uparrow$ ($\downarrow\downarrow$) means that $\text{NS}_{SC}$ is significantly better (worse) than $\text{CC}_{SC}$ at level $p < 0.01$ ($\uparrow$ and $\downarrow$ at level $p < 0.05$) in a paired t-test.

| | | $F_1$ | | | JACCARD INDEX | | |
|---|---|---|---|---|---|---|---|
| no. | $m$ | $\text{CC}_{SC}$ | $\text{NS}_{SC}$ | | $\text{CC}_{SC}$ | $\text{NS}_{SC}$ | |
| 1 | 159 | 0.2026±.0119 | 0.2090±.0113 | $\uparrow\uparrow$ | 0.1528±.0099 | 0.1582±.0100 | $\uparrow\uparrow$ |
| 2 | 6 | 0.5905±.5905 | 0.6132±.6132 | $\uparrow\uparrow$ | 0.5027±.0521 | 0.5239±.0525 | $\uparrow\uparrow$ |
| 3 | 53 | 0.3843±.3843 | 0.4016±.4016 | $\uparrow\uparrow$ | 0.2821±.0190 | 0.3005±.0238 | $\uparrow\uparrow$ |
| 4 | 27 | 0.9843±.9843 | 0.9838±.9838 | | 0.9807±.0129 | 0.9802±.0125 | |
| 5 | 5 | 0.5557±.5557 | 0.5315±.5315 | $\downarrow\downarrow$ | 0.5197±.0272 | 0.4972±.0304 | $\downarrow\downarrow$ |
| 6 | 101 | 0.5328±.0054 | 0.5610±.0052 | $\uparrow\uparrow$ | 0.4282±.0052 | 0.4457±.0050 | $\uparrow\uparrow$ |
| 7 | 45 | 0.6220±.6220 | 0.6231±.6231 | | 0.5898±.0435 | 0.5900±.0460 | |
| 8 | 7 | 0.8624±.8624 | 0.8639±.8639 | | 0.8367±.0142 | 0.8382±.0126 | |
| 9 | 6 | 0.5921±.5921 | 0.6105±.6105 | $\uparrow\uparrow$ | 0.5739±.0423 | 0.5873±.0370 | $\uparrow\uparrow$ |
| 10 | 22 | 0.3271±.3271 | 0.3248±.3248 | | 0.2843±.0186 | 0.2818±.0202 | |
| 11 | 14 | 0.5889±.5889 | 0.6141±.6141 | $\uparrow\uparrow$ | 0.4890±.0200 | 0.5104±.0200 | $\uparrow\uparrow$ |

| | | HAMMING LOSS | | | SUBSET 0/1 LOSS | | |
|---|---|---|---|---|---|---|---|
| no. | $m$ | $\text{CC}_{SC}$ | $\text{NS}_{SC}$ | | $\text{CC}_{SC}$ | $\text{NS}_{SC}$ | |
| 1 | 159 | 0.0282±.0008 | 0.0270±.0006 | $\uparrow\uparrow$ | 0.9592±.0080 | 0.9568±.0082 | $\uparrow$ |
| 2 | 6 | 0.2363±.0268 | 0.2190±.0266 | $\uparrow\uparrow$ | 0.7555±.0581 | 0.7404±.0652 | $\uparrow$ |
| 3 | 53 | 0.0819±.0023 | 0.0766±.0030 | $\uparrow\uparrow$ | 0.9491±.0130 | 0.9346±.0156 | $\uparrow\uparrow$ |
| 4 | 27 | 0.0019±.0012 | 0.0019±.0012 | | 0.0332±.0176 | 0.0337±.0172 | |
| 5 | 5 | 0.2104±.0127 | 0.2199±.0140 | $\downarrow\downarrow$ | 0.5855±.0270 | 0.6027±.0277 | $\downarrow\downarrow$ |
| 6 | 101 | 0.0302±.0004 | 0.0291±.0003 | $\uparrow\uparrow$ | 0.8750±.0049 | 0.8925±.0051 | $\downarrow\downarrow$ |
| 7 | 45 | 0.0210±.0025 | 0.0210±.0027 | | 0.5017±.0465 | 0.5037±.0514 | |
| 8 | 7 | 0.0513±.0049 | 0.0506±.0042 | | 0.2403±.0177 | 0.2391±.0167 | |
| 9 | 6 | 0.1479±.0147 | 0.1441±.0130 | $\uparrow$ | 0.4802±.0449 | 0.4815±.0386 | |
| 10 | 22 | 0.0840±.0026 | 0.0842±.0028 | | 0.8348±.0186 | 0.8380±.0201 | |
| 11 | 14 | 0.2243±.0093 | 0.2089±.0097 | $\uparrow\uparrow$ | 0.8073±.0230 | 0.8097±.0237 | |

standing the appeal of the method and the plausibility of its basic idea, we have argued that, at second sight, the chaining of classifiers begs an important flaw: A binary classifier that has learnt to rely on the values of previous labels in the chain might be misled when these values are replaced by possibly erroneous estimations at prediction time. The classification errors produced because of this attribute noise may subsequently be propagated or even reinforced along the entire chain. Roughly speaking, what looks as a gift at training time may turn out to become a handicap in prediction.

Our results have shown that the problem of error propagation is highly relevant, and that it may strongly impair the performance of CC. In order to avoid this problem, the method of nested stacking proposed in this paper uses predicted instead of observed label relevances as additional attribute values in the training phase. Our experimental studies clearly confirm that, although NS does not consistently outperform CC, it seems to have advantages for those data sets on which error propagation becomes an issue, namely data sets with many labels or low (label-wise) prediction accuracy.

There are several lines of future work. First, it is of course desirable to complement this study by meaningful theoretical results supporting our claims. Second, it would be interesting to investigate to what extent the problem of attribute noise also applies to the probabilistic variant of classifier chains introduced in [3]. Last but not least, given the interesting effects that are produced by the simple idea of subset correction, this approach seems to be worth further investigation, all the more as it is completely general and not limited to specific MLC methods such as those considered in this paper.

## References


[1] Gert Cauwenberghs and Tomaso Poggio. Incremental and decremental support vector machine learning. *Proc. NIPS*, pages 409–415, 2001.

[2] W. Cheng and E. Hüllermeier. Combining instance-based learning and logistic regression for multilabel classification. *Machine Learning*, 76(2-3):211–225, 2009.

[3] K. Dembczyński, W. Cheng, and E. Hüllermeier. Bayes optimal multilabel classification via probabilistic classifier chains. In *ICML*, pages 279–286, 2010.

[4] K. Dembczyński, W. Waegeman, W. Cheng, and E. Hüllermeier. On label dependence and loss minimization in multi-label classification. *Machine Learning*, To appear, 2012.

[5] A. Elisseeff and J. Weston. A Kernel Method for Multi-Labelled Classification. In *ACM Conf. on Research and Develop. in Infor. Retrieval*, pages 274–281, 2005.

[6] J. Fürnkranz, E. Hüllermeier, E.L. Mencía, and K. Brinker. Multilabel classification via calibrated label ranking. *Machine Learning*, 73:133–153, 2008.

[7] Johannes Fürnkranz and Sang-Hyeun Park. Error-correcting output codes as a transformation from multi-class to multi-label prediction. In *Proc. Discovery Science*, pages 254–267. 2012.

[8] S. Godbole and S. Sarawagi. Discriminative methods for multi-labeled classification. In *Pacific-Asia Conf. on Know. Disc. and Data Mining*, pages 22–30, 2004.

[9] Tomasz Kajdanowicz and Przemysław Kazienko. Multi-label classification using error correcting out-

put codes. *International Journal of Applied Mathematics and Computer Science*, 22(4):829–840, 2012.

[10] C.-J. Lin, R. C. Weng, and S. S. Keerthi. Trust region Newton method for logistic regression. *Journal of Machine Learning Research*, 9(Apr):627–650, 2008.

[11] E. Montañés, J. R. Quevedo, and J. J. del Coz. Aggregating independent and dependent models to learn multi-label classifiers. In *Proc. ECML/PKDD*, 2011.

[12] J. Read, B. Pfahringer, and G. Holmes. Multi-label classification using ensembles of pruned sets. In *IEEE Int. Conf. on Data Mining*, pages 995–1000. IEEE, 2008.

[13] J. Read, B. Pfahringer, G. Holmes, and E. Frank. Classifier chains for multi-label classification. *Machine Learning*, 85(3):333–359, 2011.

[14] R. E. Schapire and Y. Singer. Boostexter: A boosting-based system for text categorization. In *Machine Learning*, pages 135–168, 2000.

[15] Robin Senge, Juan Jos del Coz, and Eyke Hüllermeier. On the problem of error propagation in classifier chains for multi-label classification. In *Conference of the German Classification Society*, 2012.

[16] G. Tsoumakas, I. Katakis, and I. Vlahavas. Mining multi-label data. In *Data Mining and Knowledge Discovery Handbook*, pages 667–685. 2010.

[17] G. Tsoumakas and I. Vlahavas. Random k-Labelsets: An Ensemble Method for Multilabel Classification. In *Proc. ECML/PKDD*, LNCS, pages 406–417. Springer, 2007.

[18] D. H. Wolpert. Stacked generalization. *Neural Networks*, 5:214–259, 1992.

[19] M.-L. Zhang and Z.-H. Zhou. Multilabel neural networks with applications to functional genomics and text categorization. *IEEE Trans. on Knowl. and Data Eng.*, 18:1338–1351, 2006.

[20] X. Zhu and X. Wu. Class noise vs. attribute noise: a quantitative study of their impacts. *Artificial Intelligence Review*, 22(3):177–210, 2004.