Multilabel Classification with R Package mlr

by Philipp Probst, Quay Au, Giuseppe Casalicchio, Clemens Stachl and Bernd Bischl

Abstract We implemented several multilabel classification algorithms in the machine learning package **mlr**. The implemented methods are binary relevance, classifier chains, nested stacking, dependent binary relevance and stacking, which can be used with any base learner that is accessible in **mlr**. Moreover, there is access to the multilabel classification versions of **randomForestSRC** and **rFerns**. All these methods can be easily compared by different implemented multilabel performance measures and resampling methods in the standardized **mlr** framework. In a benchmark experiment with several multilabel datasets, the performance of the different methods is evaluated.

Introduction

Multilabel classification is a classification problem where multiple target labels can be assigned to each observation instead of only one, like in multiclass classification. It can be regarded as a special case of multivariate classification or multi-target prediction problems, for which the scale of each response variable can be of any kind, for example nominal, ordinal or interval.

Originally, multilabel classification was used for text classification (McCallum, 1999; Schapire and Singer, 2000) and is now used in several applications in different research fields. For example, in image classification, a photo can belong to the classes *mountain* and *sunset* simultaneously. Zhang and Zhou (2008) and others (Boutell et al., 2004) used multilabel algorithms to classify scenes on images of natural environments. Furthermore, gene functional classifications is a popular application of multilabel learning in the field of biostatistics (Elisseeff and Weston, 2002; Zhang and Zhou, 2008). Additionally, multilabel classification is useful to categorize audio files. Music genres (Sanden and Zhang, 2011), instruments (Kursa and Wieczorkowska, 2014), bird sounds (Briggs et al., 2013) or even emotions evoked by a song (Trohidis et al., 2008) can be labeled with several categories. A song could, for example, be classified both as a *rock song* and a *ballad*.

An overview of multilabel classification was given by Tsoumakas and Katakis (2007). Two different approaches exist for multilabel classification. On the one hand, there are algorithm adaptation methods that try to adapt multiclass algorithms so they can be applied directly to the problem. On the other hand, there are problem transformation methods, which try to transform the multilabel classification into binary or multiclass classification problems.

Regarding multilabel classification software, there is the **mldr** (Charte and Charte, 2015) R package that contains some functions to get basic characteristics of specific multilabel datasets. The package is also useful for transforming multilabel datasets that are typically saved as ARFF-files (Attribute-Relation File Format) to data frames and vice versa. This is especially helpful because until now only the software packages MEKA (Read and Reutemann, 2012) and Mulan (Tsoumakas et al., 2011) were available for multilabel classification and both require multilabel datasets saved as ARFF-files to be executed. Additionally, the **mldr** package provides a function that applies the binary relevance or label powerset transformation method which transforms a multilabel dataset into several binary datasets (one for each label) or into a multiclass dataset using the set of labels for each observation as a single target label, respectively. However, there is no R package that provides a standardized interface for executing different multilabel classification algorithms. With the extension of the **mlr** package described in this paper, it will be possible to execute several multilabel classification algorithms in R with many different base learners.

In the following section of this paper, we will describe the implemented multilabel classification methods and then give a practical instruction of how to execute these algorithms in **mlr**. Finally, we present a benchmark experiment that compares the performance of all implemented methods on several datasets.

Multilabel classification methods implemented in mlr

In this section, we present multilabel classification algorithms that are implemented in the **mlr** package (Bischl et al., 2016), which is a powerful and modularized toolbox for machine learning in R. The package offers a unified interface to more than a hundred learners from the areas classification, regression, cluster analysis and survival analysis. Furthermore, the package provides functions and tools that facilitate complex workflows such as hyperparameter tuning (see, e.g., Lang et al., 2015) and

feature selection that can now also be applied to the multilabel classification methods presented in this paper. In the following, we list the algorithm adaptation methods and problem transformation methods that are currently available in **mlr**.

Algorithm adaptation methods

The **rFerns** (Kursa and Wieczorkowska, 2014) package contains an extension of the random ferns algorithm for multilabel classification. In the **randomForestSRC** (Ishwaran and Kogalur, 2016) package, multivariate classification and regression random forests can be created. In the classification case, the difference to standard random forests is that a composite normalized Gini index splitting rule is used. Multilabel classification can be achieved by using binary encoding for the labels.

Problem transformation methods

Problem transformation methods try to transform the multilabel classification problem so that a simple binary classification algorithm, the so-called base learner, can be applied.

Let n be the number of observations, let p be the number of predictor variables and let $Z = \{z_1, \ldots, z_m\}$ be the set of all labels. Observations follow an unknown probability distribution $\mathcal{P}$ on $\mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ is a $p-$dimensional input space of arbitrary measurement scales and $\mathcal{Y} = \{0,1\}^m$ is the target space. In our notation, $\mathbf{x}^{(i)} = \left(x_1^{(i)}, \ldots, x_p^{(i)}\right)^\top \in \mathcal{X}$ refers to the i-th observation and $\mathbf{x}_j = \left(x_j^{(1)}, \ldots, x_j^{(n)}\right)^\top$ refers to the j-th predictor variable, for all $i = 1, \ldots, n$ and $j = 1, \ldots, p$. The observations $\mathbf{x}^{(i)}$ are associated with their multilabel outcomes $\mathbf{y}^{(i)} = \left(y_1^{(i)}, \ldots, y_m^{(i)}\right)^\top \in \mathcal{Y}$, for all $i = 1, \ldots, n$. For all $k = 1, \ldots, m$, setting $y_k^{(i)} = 1$ indicates the relevance, i.e., the occurrence, of label z_k for observation $\mathbf{x}^{(i)}$ and setting $y_k^{(i)} = 0$ indicates the irrelevance of label z_k for observation $\mathbf{x}^{(i)}$. The set of all instances thus becomes $D = \left\{\left(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}\right), \left(\mathbf{x}^{(2)}, \mathbf{y}^{(2)}\right), \ldots, \left(\mathbf{x}^{(n)}, \mathbf{y}^{(n)}\right)\right\}$. Furthermore, $\mathbf{y}_k = \left(y_k^{(1)}, \ldots, y_k^{(n)}\right)^\top$ refers to the k-th target vector, for all $k = 1, \ldots, m$. Throughout this paper, we visualize multilabel classification problems in the form of tables ($n = 6$, $p = 3$, $m = 3$):

$$D \mathrel{\hat{=}}$$

$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{y}_1$	$\mathbf{y}_2$	$\mathbf{y}_3$
			0	0	1
			1	0	1
			1	1	0
			1	1	1
			1	1	0
			1	1	0

(1)

The entries of $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$ can be of any (valid) kind, like continuous, binary, or categorical. The table in (1) visualizes this as an empty gray background. The target variables are indicated by a red background and can only take the binary values 0 or 1.

Binary relevance

The binary relevance method (BR) is the simplest problem transformation method. BR learns a binary classifier for each label. Each classifier $C_1, \ldots, C_m$ is responsible for predicting the *relevance* of their corresponding label by a 0/1 prediction:

$$C_k : \mathcal{X} \longrightarrow \{0,1\}, \quad k = 1, \ldots, m$$

These binary prediction are then combined to a multilabel target. An unlabeled observation $\mathbf{x}^{(l)}$ is assigned the prediction $\left(C_1\left(\mathbf{x}^{(l)}\right), C_2\left(\mathbf{x}^{(l)}\right), \ldots, C_m\left(\mathbf{x}^{(l)}\right)\right)^\top$. Hence, labels are predicted independently of each other and label dependencies are not taken into account. BR has linear computational complexity with respect to the number of labels and can easily be parallelized.

Modeling label dependence

In the problem transformation setting, the arguably simplest way (Montañés et al., 2014) to model label dependence is to condition classifier models not only on $\mathcal{X}$, but also on other label information. The idea is to augment the input space $\mathcal{X}$ with information of the output space $\mathcal{Y}$, which is available in the training step. There are different ways to realize this idea of augmenting the input space. In essence, they can be distinguished in the following way:

- Should the true label information be used? (True vs. predicted label information)
- For predicting one label z_k, should all other labels augment the input space, or only a subset of labels? (Full vs. partial conditioning)

True vs. predicted label information

During the training of a classifier C_k for the label z_k, the label information of other labels are available in the training data. Consequently, these true labels can directly be used as predictors to train the classifier. Alternatively, the predictions that are produced by some classifier can be used instead of the true labels.

A classifier, which is trained on additional labels as predictors, needs those additional labels as input variables. Since these labels are not available at prediction time, they need to be predicted first. When the true label information is used to augment the feature space in the training of a classifier, the assumption that the training data and the test data should be identically distributed is violated (Senge et al., 2013). If the true label information is used in the training data and the predicted label information is used in the test data, the training data is not representative for the test data. However, experiments (Montañés et al., 2014; Senge et al., 2013) show that none of these methods should be dismissed immediately. Note that we use the superscript "true" or "pred" to emphasize that a classifier C_k^{true} or C_k^{pred} used true labels or predicted labels as additional predictors during training, respectively.

Suppose there are $n = 6$ observations with $p = 3$ predictors and $m = 3$ labels. The true label $\mathbf{y}_3$ shall be used to augment the feature space of a binary classifier C_1^{true} for label $\mathbf{y}_1$. C_1^{true} is thus trained on all predictors and the true label $\mathbf{y}_3$. The binary classification task for label $\mathbf{y}_1$ is therefore:

$$\text{Train } C_1^{\text{true}} \text{ on } \begin{array}{|ccc|c|c|}\hline x_1 & x_2 & x_3 & y_3 & y_1 \\ \hline & & & 0 & 0 \\ & & & 1 & 1 \\ & & & 0 & 1 \\ & & & 1 & 1 \\ & & & 0 & 1 \\ & & & 0 & 1 \\ \hline\end{array} \text{ to predict } \mathbf{y}_1 \tag{2}$$

For an unlabeled observation $\mathbf{x}^{(l)}$, only the three predictor variables $x_1^{(l)}, \ldots, x_3^{(l)}$ are available at prediction time. However, the classifier C_1^{true} needs a 4-dimensional observation $\left(\mathbf{x}^{(l)}, y_3^{(l)}\right)$ as input. The input $y_3^{(l)}$ therefore needs to be predicted first. A new *level-1* classifier C_3^{lvl1}, which is trained on the set $D' = \cup_{i=1}^{6} \left\{\left(\mathbf{x}^{(i)}, y_3^{(i)}\right)\right\}$, will make those predictions for $y_3^{(l)}$. The training task is:

$$\text{Train } C_3^{\text{lvl1}} \text{ on } D' \mathrel{\hat{=}} \begin{array}{|ccc|c|}\hline x_1 & x_2 & x_3 & y_3 \\ \hline & & & 1 \\ & & & 1 \\ & & & 0 \\ & & & 1 \\ & & & 0 \\ & & & 0 \\ \hline\end{array} \text{ to predict } \mathbf{y}_3 \tag{3}$$

Therefore, for a new observation $\mathbf{x}^{(l)}$, the predicted label $\hat{y}_3^{(l)}$ is obtained by using C_3^{lvl1} on $\mathbf{x}^{(l)}$. The final prediction for $y_1^{(l)}$ is then obtained by using C_1^{true} on $\left(\mathbf{x}^{(l)}, \hat{y}_3^{(l)}\right)$.

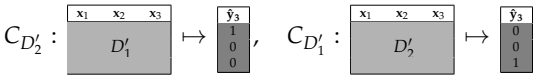
The alternative to (2) would be to use predicted labels $\hat{\mathbf{y}}_3$ instead of true labels $\mathbf{y}_3$. These labels should be produced by means of an out-of-sample prediction procedure (Senge et al., 2013). This can be done by an internal leave-one-out cross-validation procedure, which can of course be computationally intensive. Because of this, coarser resampling strategies can be used. As an example, an internal 2-fold cross-validation will be shown here. Again, let $D' = \cup_{i=1}^{6} \left\{\left(\mathbf{x}^{(i)}, y_3^{(i)}\right)\right\}$ be the set of all predictor variables with $\mathbf{y}_3$ as target variable. Using 2-fold cross-validation, the dataset D' is split into two parts $D'_1 = \cup_{i=1}^{3} \left\{\left(\mathbf{x}^{(i)}, y_3^{(i)}\right)\right\}$ and $D'_2 = \cup_{i=4}^{6} \left\{\left(\mathbf{x}^{(i)}, y_3^{(i)}\right)\right\}$:

$$\begin{array}{|ccc|c|}\hline x_1 & x_2 & x_3 & y_3 \\ \hline & D'_1 & & 1 \\ & & & 1 \\ & & & 0 \\ \hline & D'_2 & & 1 \\ & & & 0 \\ & & & 0 \\ \hline\end{array} \tag{4}$$

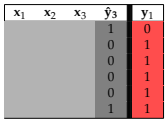
Two classifiers $C_{D'_1}$ and $C_{D'_2}$ are then trained on D'_1 and D'_2, respectively, for the prediction of $\mathbf{y}_3$:

$$\text{Train } C_{D'_1} \text{ on } \begin{array}{|ccc|c|}\hline x_1 & x_2 & x_3 & y_3 \\ \hline & D'_1 & & 1 \\ & & & 1 \\ & & & 0 \\ \hline\end{array} \text{ to predict } \mathbf{y}_3, \quad \text{Train } C_{D'_2} \text{ on } \begin{array}{|ccc|c|}\hline x_1 & x_2 & x_3 & y_3 \\ \hline & D'_2 & & 1 \\ & & & 0 \\ & & & 0 \\ \hline\end{array} \text{ to predict } \mathbf{y}_3$$

Following the cross-validation paradigm, D_1' is used as test set for the classifier $C_{D_2'}$, and D_2' is used as a test set for $C_{D_1'}$:



These predictions are merged for the final predicted label $\hat{\mathbf{y}}_3$, which is used to augment the feature space. The classifier C_1^{pred} is then trained on that augmented feature space:



(5)

The prediction phase is completely analogous to (3). It is worthwhile to mention that the level-1 classifier C_3^{lvl1}, which will be used to obtain predictions $\hat{\mathbf{y}}_3$ at prediction time, is trained on the whole set $D' = D_1' \cup D_2'$, following Simon (2007).

Full vs. partial conditioning

Recall the set of all labels $Z = \{z_1, \ldots, z_m\}$. The prediction of a label z_k can either be conditioned on all remaining labels $\{z_1, \ldots, z_{k-1}, z_{k+1}, \ldots, z_m\}$ (*full conditioning*) or just on a subset of labels (*partial conditioning*). The only method for partial conditioning, which is examined in this paper, is the chaining method. Here, labels z_k are conditioned on all previous labels $\{z_1, \ldots, z_{k-1}\}$ for all $k = 1, \ldots, m$. This sequential structure is motivated by the product rule of probability (Montañés et al., 2014):

$$P\left(\mathbf{y}^{(i)} \middle| \mathbf{x}^{(i)}\right) = \prod_{k=1}^{m} P\left(y_k^{(i)} \middle| \mathbf{x}^{(i)}, y_1^{(i)}, \ldots, y_{k-1}^{(i)}\right) \tag{6}$$

Methods that make use of this chaining structure are e.g., *classifier chains* or *nested stacking* (these methods will be discussed further below).

To sum up the discussions above: there are four ways in modeling label dependencies through conditioning labels z_k on other labels z_ℓ, $k \neq \ell$. They can be distinguished by the subset of labels, which are used for conditioning, and by the use of predicted or real labels in the training step. In Table 1 we show the four methods, which implement these ideas and describe them consequently.

	True labels	Pred. labels
Partial cond.	Classifier chains	Nested stacking
Full cond.	Dependent binary relevance	Stacking

Table 1: Distinctions in modeling label dependence and models

Classifier chains

The classifier chains (CC) method implements the idea of using partial conditioning together with the true label information. It was first introduced by Read et al. (2011). CC selects an order on the set of labels $\{z_1, \ldots, z_m\}$, which can be formally written as a bijective function (permutation):

$$\tau : \{1, \ldots, m\} \longrightarrow \{1, \ldots, m\} \tag{7}$$

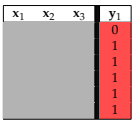
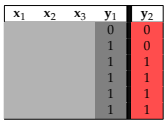
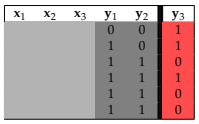
Labels will be chained along this order τ:

$$z_{\tau(1)} \rightarrow z_{\tau(2)} \rightarrow \ldots \rightarrow z_{\tau(m)} \tag{8}$$

However, for this paper the permutation shall be $\tau = id$ (only for simplicity reasons). The labels therefore follow the order $z_1 \rightarrow z_2 \rightarrow \ldots \rightarrow z_m$. In a similar fashion to the binary relevance (BR) method, CC trains m binary classifiers C_k, which are responsible for predicting their corresponding label z_k, $k = 1, \ldots, m$. The classifiers C_k are of the form

$$C_k : \mathcal{X} \times \{0,1\}^{k-1} \longrightarrow \{0,1\}, \tag{9}$$

where $\{0,1\}^0 := \emptyset$. For a classifier C_k the feature space is augmented by the true label information of all previous labels $z_1, z_2, \ldots, z_{k-1}$. Hence, the training data of C_k consists of all observations $\left(\left(\mathbf{x}^{(i)}, y_1^{(i)}, y_2^{(i)}, \ldots, y_{k-1}^{(i)}\right), y_k^{(i)}\right)$, $i = 1, \ldots, n$, with the target $y_k^{(i)}$. In the example from above, this would look like:

Train C_1 on  Train C_2 on  Train C_3 on  (10)

At prediction time, when an unlabeled observation $\mathbf{x}^{(l)}$ is labeled, a prediction $\left(\hat{y}_1^{(l)}, \ldots, \hat{y}_m^{(l)}\right)$ is obtained by successively predicting the labels along the chaining order:

$$\begin{aligned} \hat{y}_1^{(l)} &= C_1\left(\mathbf{x}^{(l)}\right) \\ \hat{y}_2^{(l)} &= C_2\left(\mathbf{x}^{(l)}, \hat{y}_1^{(l)}\right) \\ &\vdots \\ \hat{y}_m^{(l)} &= C_m\left(\mathbf{x}^{(l)}, \hat{y}_1^{(l)}, \hat{y}_2^{(l)}, \ldots, \hat{y}_{m-1}^{(l)}\right) \end{aligned} \tag{11}$$

The authors of Senge et al. (2013) summarize several factors, which have an impact on the performance of CC:

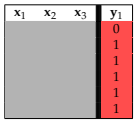
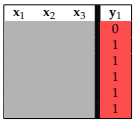
- *The length of the chain.* A high number $(k-1)$ of preceding classifiers in the chain comes with a high potential level of feature noise for the classifier C_k. One may assume that the probability of a mistake will increase with the level of feature noise in the input space. Then the probability of a mistake will be reinforced along the chain, due to the recursive structure of CC.
- *The order of the chain.* Some labels may be more difficult to predict than others. The order of a chain can therefore be important for the performance. It can be advantageous to put simple to predict labels in the beginning and harder to predict labels more towards the end of the chain. Some heuristics for finding an optimal chain ordering have been proposed in da Silva et al. (2014); Read et al. (2013). Alternatively Read et al. (2011) developed an ensemble of classifier chains, which builds many randomly ordered CC-classifiers and put them on a voting scheme for a prediction. However, these methods are not subject of this article.
- *The dependency among labels.* For an improvement of performance through chaining, there should be a dependence among labels, CC cannot gain in case of label independence. However, CC is also only likely to lose if the binary classifiers C_k cannot ignore the added features $\mathbf{y}_1, \ldots, \mathbf{y}_{k-1}$.

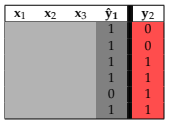
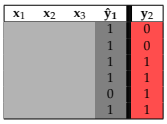
Nested stacking

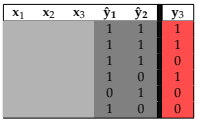
The nested stacking method (NST), first proposed in Senge et al. (2013), implements the idea of using partial conditioning together with predicted label information. NST mimicks the chaining structure of CC, but does not use real label information during training. Like in CC the chaining order shall be $\tau = id$, again for simplicity reasons. CC uses real label information $\mathbf{y}_k$ during training and predicted labels $\hat{\mathbf{y}}_\mathbf{k}$ at prediction time. However, unless the binary classifiers are perfect, it is likely that $\mathbf{y}_k$ and $\hat{\mathbf{y}}_\mathbf{k}$ do not follow the same distribution. Hence, the key assumption of supervised learning, namely that the training data should be representative for the test data, is violated by CC. Nested stacking tries to overcome this issue by using predicted labels $\hat{\mathbf{y}}_\mathbf{k}$ instead of true labels $\mathbf{y}_k$.

NST trains m binary classifiers C_k on $D_k := \cup_{i=1}^n \left\{\left(\left(\mathbf{x}^{(i)}, \hat{y}_1^{(i)}, \ldots, \hat{y}_{k-1}^{(i)}\right), y_k^{(i)}\right)\right\}$, for all $k = 1, \ldots, m$. The predicted labels should be obtained by an internal out-of-sample method (Senge et al., 2013). How these predictions are obtained was already explained in the **True vs. Predicted Label Information** chapter. The prediction phase is completely analogous to (11).

The training procedure is visualized in the following with 2-fold cross-validation as an internal out-of-sample method:

Train C_1 on  Use 2-fold CV on  to obtain  (12)

Train C_2 on  Use 2-fold CV on  to obtain  (13)

Train C_3 on  (14)

The factors which impact the performance of CC (i.e., length and order of the chain, and the dependency among labels), also impact NST, since NST mimicks the chaining method of CC.

Dependent binary relevance

The dependent binary relevance method (DBR) implements the idea of using full conditioning together with the true label information. DBR is built on two main hypotheses (Montañés et al., 2014):

(i) Taking conditional label dependencies into account is important for performing well in multilabel classification tasks.

(ii) Modeling and learning these label dependencies in an overcomplete way (take all other labels for modeling) may further improve model performance.

The first assumption is the main prerequisite for research in multilabel classification. It has been shown theoretically that simple binary relevance classifiers cannot achieve optimal performance for specific multilabel loss functions (Montañés et al., 2014). The second assumption, however, is harder to justify theoretically. Nonetheless, the practical usefulness of learning in an overcomplete way has been shown in many branches of (classical) single-label classification (e.g., ensemble methods (Dietterich, 2000)).

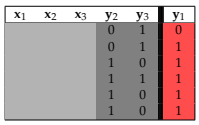
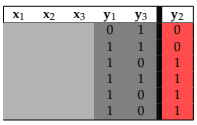
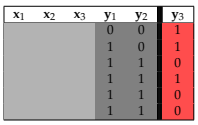
Formally, DBR trains m binary classifiers $C_1, \ldots, C_m$ (as many classifiers as labels) on the corresponding training data

$$D_k = \cup_{i=1}^{n} \left\{ \left(\left(\mathbf{x}^{(i)}, y_1^{(i)}, \ldots, y_{k-1}^{(i)}, y_{k+1}^{(i)}, \ldots, y_m^{(i)} \right), y_k^{(i)} \right) \right\}, \tag{15}$$

$k = 1, \ldots, m$. Thus, each classifier C_k is of the form

$$C_k : \mathcal{X} \times \{0,1\}^{m-1} \longrightarrow \{0,1\}.$$

Hence, for each classifier C_k the true label information of all labels except $\mathbf{y}_k$ is used as augmented features. Again, here is a visualization with the example from above:

Train C_1 on  Train C_2 on  Train C_3 on  (16)

To make these classifiers applicable, when an unlabeled instance $\mathbf{x}^{(l)}$ needs to be labeled, the help of other multilabel classifiers is needed to produce predicted labels $\hat{y}_1^{(l)}, \ldots, \hat{y}_m^{(l)}$ as additional features. The classifiers, which produce predicted labels as additional features, are called *base learners* (Montañés et al., 2014). Theoretically any multilabel classifier can be used as base learner. However, in this paper, the analysis is focused on BR as base learner only. The prediction of an unlabeled instance $\mathbf{x}^{(l)}$ formally works as follows:

(i) First level: Produce predicted labels by using the BR base learner:

$$C_{BR}\left(\mathbf{x}^{(l)}\right) = \left(\hat{y}_1^{(l)}, \ldots, \hat{y}_m^{(l)}\right)$$

(ii) Second level, which is also called meta level (Montañés et al., 2014): Produce final prediction $\hat{\hat{\mathbf{y}}}_k = \left(\hat{\hat{y}}_1^{(l)}, \ldots, \hat{\hat{y}}_m^{(l)}\right)$ by applying DBR classifiers $C_1, \ldots, C_m$:

$$\begin{aligned} C_1\left(\mathbf{x}^{(l)}, \hat{y}_2^{(l)}, \ldots, \hat{y}_m^{(l)}\right) &= \hat{\hat{y}}_1^{(l)} \\ C_2\left(\mathbf{x}^{(l)}, \hat{y}_1^{(l)}, \hat{y}_3^{(l)}, \ldots, \hat{y}_m^{(l)}\right) &= \hat{\hat{y}}_2^{(l)} \\ &\vdots \\ C_m\left(\mathbf{x}^{(l)}, \hat{y}_1^{(l)}, \ldots, \hat{y}_{m-1}^{(l)}\right) &= \hat{\hat{y}}_m^{(l)} \end{aligned}$$

Stacking

Stacking (STA) implements the last variant of Table 1, namely the use of full conditioning together with predicted label information. Stacking is short for *stacked generalization* (Wolpert, 1992) and was first proposed in the multilabel context by Godbole and Sarawagi (2004). Like in classical stacking, for each label it takes predictions of several other learners that were trained in a first step to get a new learner to make predictions for the corresponding label. Both hypotheses on which DBR is built on also apply to STA, of course.

STA trains m classifiers $C_1, \ldots, C_m$ on the corresponding training data

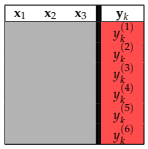
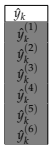
$$D_k = \cup_{i=1}^{n} \left\{ \left(\left(\mathbf{x}^{(i)}, \hat{y}_1^{(i)}, \ldots, \hat{y}_m^{(i)} \right), y_k^{(i)} \right) \right\}, k = 1, \ldots, m. \tag{17}$$

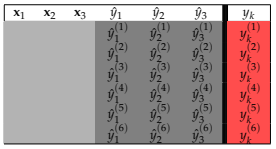
The classifiers C_k, $k = 1, \ldots, m$, are therefore of the following form:

$$C_k : \mathcal{X} \times \{0,1\}^m \longrightarrow \{0,1\}$$

Like in NST, the predicted labels should be obtained by an internal out-of-sample method (Sill et al., 2009). STA can be seen as the alternative to DBR using predicted labels (like NST is for CC). However, the classifiers $C_k, k = 1, \ldots, m$, are trained on all predicted labels $\hat{\mathbf{y}}_\mathbf{1}, \ldots, \hat{\mathbf{y}}_\mathbf{m}$ for the STA approach (in DBR the label $\mathbf{y}_k$ is left out of the augmented training set).

The training procedure is outlined in the following:

For i=1,2,3 use 2-fold CV on  to obtain  (18)

For i=1,2,3 train C_k on  (19)

Like in DBR, STA depends on a BR base learner, to produce predicted labels as additional features. Again, the use of BR as a base learner is not mandatory, but it is the proposed method in Godbole and Sarawagi (2004).

The prediction of an unlabeled instance $\mathbf{x}^{(l)}$ works almost identically to the DBR case and is illustrated here:

(i) First level. Produce predicted labels by using the BR base learner:

$$C_{BR}\left(\mathbf{x}^{(l)}\right) = \left(\hat{y}_1^{(l)}, \ldots, \hat{y}_m^{(l)}\right)$$

(ii) Meta level. Apply STA classifiers $C_1, \ldots, C_m$:

$$\begin{aligned} C_1\left(\mathbf{x}^{(l)}, \hat{y}_1^{(l)}, \ldots, \hat{y}_m^{(l)}\right) &= \hat{\hat{y}}_1^{(l)} \\ &\vdots \\ C_m\left(\mathbf{x}^{(l)}, \hat{y}_1^{(l)}, \ldots, \hat{y}_m^{(l)}\right) &= \hat{\hat{y}}_m^{(l)} \end{aligned}$$

Multilabel performance measures

Analogously to multiclass classification there exist multilabel classification performance measures. Six multilabel performance measures can be evaluated in **mlr**. These are: *Subset* $0/1$ *loss*, *hamming loss*, *accuracy*, *precision*, *recall* and F_1*-index*. Multilabel performance measures are defined on a per instance basis. The performance on a test set is the average over all instances.

Let $D_{\text{test}} = \left\{\left(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}\right), \ldots, \left(\mathbf{x}^{(n)}, \mathbf{y}^{(n)}\right)\right\}$ be a test set with $\mathbf{y}^{(i)} = \left(y_1^{(i)}, \ldots, y_m^{(i)}\right) \in \{0,1\}^m$ for all $i = 1, \ldots, n$. Performance measures quantify how good a classifier C predicts the labels $z_1, \ldots, z_n$.

(i) The subset $0/1$ loss is used to see if the predicted labels $C(\mathbf{x}^{(i)}) = \left(\hat{y}_1^{(i)}, \ldots, \hat{y}_m^{(i)}\right)$ are equal to the actual labels $\left(y_1^{(i)}, \ldots, y_m^{(i)}\right)$:

$$\text{subset}_{0/1}\left(C, \left(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right)\right) = \mathbb{1}_{\left(\mathbf{y}^{(i)} \neq C(\mathbf{x}^{(i)})\right)} := \begin{cases} 1 & \text{if } \mathbf{y}^{(i)} \neq C\left(\mathbf{x}^{(i)}\right) \\ 0 & \text{if } \mathbf{y}^{(i)} = C\left(\mathbf{x}^{(i)}\right) \end{cases}$$

The subset $0/1$ loss of a classifier C on a test set D_{test} thus becomes:

$$\text{subset}_{0/1}\left(C, D_{\text{test}}\right) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}_{\mathbf{y}^{(i)} \neq C\left(\mathbf{x}^{(i)}\right)}$$

The subset $0/1$ loss can be interpreted as the analogon of the mean misclassification error in multiclass classifications. In the multilabel case it is a rather drastic measure because it treats a mistake on a single label as a complete failure (Senge et al., 2013).

(ii) The hamming loss also takes into account observations where only some labels have been predicted correctly. It corresponds to the proportion of labels whose relevance is incorrectly predicted. For an instance $\left(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right) = \left(\mathbf{x}^{(i)}, \left(y_1^{(i)}, \ldots, y_m^{(i)}\right)\right)$ and a classifier $C\left(\mathbf{x}^{(i)}\right) = \left(\hat{y}_1^{(i)}, \ldots, \hat{y}_m^{(i)}\right)$ this is defined as:

$$\text{HammingLoss}\left(C, \left(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right)\right) = \frac{1}{m}\sum_{k=1}^{m} \mathbb{1}_{\left(y_k^{(i)} \neq \hat{y}_k^{(i)}\right)}$$

If one label is predicted incorrectly, this accounts for an error of $\frac{1}{m}$. For a test set D_{test} the hamming loss becomes:

$$\text{HammingLoss}(C, D_{\text{test}}) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{m}\sum_{k=1}^{m} \mathbb{1}_{\left(y_k^{(i)} \neq \hat{y}_k^{(i)}\right)}$$

The following measures are scores instead of loss function like the two previous ones.

(iii) The accuracy, also called Jaccard-Index, for a test set D_{test} is defined as:

$$\text{accuracy}(C, D_{\text{test}}) = \frac{1}{n}\sum_{i=1}^{n}\frac{\sum_{k=1}^{m} \mathbb{1}_{\left(y_k^{(i)}=1 \text{ and } \hat{y}_k^{(i)}=1\right)}}{\sum_{k=1}^{m} \mathbb{1}_{\left(y_k^{(i)}=1 \text{ or } \hat{y}_k^{(i)}=1\right)}}$$

(iv) The precision for a test set D_{test} is defined as:

$$\text{precision}(C, D_{\text{test}}) = \frac{1}{n}\sum_{i=1}^{n}\frac{\sum_{k=1}^{m} \mathbb{1}_{\left(y_k^{(i)}=1 \text{ and } \hat{y}_k^{(i)}=1\right)}}{\sum_{k=1}^{m} \mathbb{1}_{\left(\hat{y}_k^{(i)}=1\right)}}$$

(v) The recall for a test set D_{test} is defined as:

$$\text{recall}(C, D_{\text{test}}) = \frac{1}{n}\sum_{i=1}^{n}\frac{\sum_{k=1}^{m} \mathbb{1}_{\left(y_k^{(i)}=1 \text{ and } \hat{y}_k^{(i)}=1\right)}}{\sum_{k=1}^{m} \mathbb{1}_{\left(y_k^{(i)}=1\right)}}$$

(vi) For a test set D_{test} the F_1-index is defined as follows:

$$F_1(C, D_{\text{test}}) = \frac{1}{n}\sum_{i=1}^{n}\frac{2\sum_{k=1}^{m} \mathbb{1}_{\left(y_k^{(i)}=1 \text{ and } \hat{y}_k^{(i)}=1\right)}}{\sum_{k=1}^{m}\left(\mathbb{1}_{\left(y_k^{(i)}=1\right)} + \mathbb{1}_{\left(\hat{y}_k^{(i)}=1\right)}\right)}$$

The F_1-index is the harmonic mean of recall and precision on a per instance basis.

All these measures lie between 0 and 1. In the case of the subset 0/1 loss and the hamming loss the values should be low, in all other cases the scores should be high. Demonstrative definitions with sets instead of vectors can be seen in Charte and Charte (2015).

Implementation

In this section, we briefly describe how to perform multilabel classifications in **mlr**. We provide small code examples for better illustration. A short tutorial is also available at http://mlr-org.github.io/mlr-tutorial/release/html/multilabel/index.html. The first step is to transform the multilabel dataset into a 'data.frame' in R. The columns must consist of vectors of features and one logical vector for each label that indicates if the label is present for the observation or not. To fit a multilabel classification algorithm in **mlr**, a multilabel task has to be created, where a vector of targets corresponding to the column names of the labels has to be specified. This task is an S3 object that contains the data, the target labels and further descriptive information. In the following example, the yeast data frame is extracted from the yeast.task, which is provided by the **mlr** package. Then the 14 label names of the targets are extracted and the multilabel task is created.

```
yeast = getTaskData(yeast.task)
labels = colnames(yeast)[1:14]
yeast.task = makeMultilabelTask(id = "multi", data = yeast, target = labels)
```

Problem transformation methods

To generate a problem transformation method learner, a binary classification base learner has to be created with 'makeLearner'. A list of available learners for classifications in **mlr** can be seen at http://mlr-org.github.io/mlr-tutorial/release/html/integrated_learners/. Specific hyperparameter settings of the base learner can be set in this step through the 'par.vals' argument in 'makeLearner'. Afterwards, a learner for any problem transformation method can be created by applying the function 'makeMultilabel[...]Wrapper', where [...] has to be substituted by the desired problem transformation method. In the following example, two multilabel variants with rpart as base learner are created. The base learner is configured to output probabilities instead of discrete labels during prediction.

```
lrn = makeLearner("classif.rpart", predict.type = "prob")
multilabel.lrn1 = makeMultilabelBinaryRelevanceWrapper(lrn)
multilabel.lrn2 = makeMultilabelNestedStackingWrapper(lrn)
```

Algorithm adaptation methods

Algorithm adaptation method learners can be created directly with 'makeLearner'. The names of the specific learner can be looked up at http://mlr-org.github.io/mlr-tutorial/release/html/integrated_learners/ in the multilabel section.

```
multilabel.lrn3 = makeLearner("multilabel.rFerns")
multilabel.lrn4 = makeLearner("multilabel.randomForestSRC")
```

Train, predict and evaluate

Training and predicting on data can be done as usual in **mlr** with the functions 'train' and 'predict'. Learner and task have to be specified in 'train'; trained model and task or new data have to be specified in 'predict'.

```
mod = train(multilabel.lrn1, yeast.task, subset = 1:1500)
pred = predict(mod, task = yeast.task, subset = 1501:1600)
```

The performance of the prediction can be assessed via the function 'performance'. Measures are represented as S3 objects and multiple objects can be passed in as a list. The default measure for multilabel classification is the hamming loss (*multilabel.hamloss*). All available measures for multilabel classification can be shown by 'listMeasures' or looked up in the appendix of the tutorial page[1] (http://mlr-org.github.io/mlr-tutorial/release/html/measures/index.html).

[1] In the **mlr** package *precision* is named *positive predictive value* and *recall* is named *true positive rate*.

```
performance(pred, measures = list(multilabel.hamloss, timepredict))
multilabel.hamloss        timepredict
0.230              0.174
listMeasures("multilabel")
# [1] "multilabel.ppv" "timepredict"         "multilabel.hamloss" "multilabel.f1"
# [5] "featperc"       "multilabel.subset01" "timeboth"           "timetrain"
# [9] "multilabel.tpr" "multilabel.acc"
```

Resampling

To properly evaluate the model, a resampling strategy, for example k-fold cross-validation, should be applied. This can be done in **mlr** by using the function 'resample'. First, a description of the subsequent resampling strategy, in this case three-fold cross-validation, is defined with 'makeResampleDesc'. The resample is executed by a call to the 'resample' function. The hamming loss is calculated for the binary relevance method.

```
rdesc = makeResampleDesc(method = "CV", stratify = FALSE, iters = 3)
r = resample(learner = multilabel.lrn1, task = yeast.task, resampling = rdesc,
measures = list(multilabel.hamloss), show.info = FALSE)
r
# Resample Result
# Task: multi
# Learner: multilabel.classif.rpart
# multilabel.hamloss.aggr: 0.23
# multilabel.hamloss.mean: 0.23
# multilabel.hamloss.sd: 0.00
# Runtime: 6.36688
```

Binary performance

To calculate a binary performance measure like, e.g., the accuracy, the mean misclassification error (mmce) or the AUC for each individual label, the function 'getMultilabelBinaryPerformances' can be used. This function can be applied to a single multilabel test set prediction and also on a resampled multilabel prediction. To calculate the AUC, predicted probabilities are needed. These can be obtained by setting the argument 'predict.type = "prob"' in the 'makeLearner' function.

```
head(getMultilabelBinaryPerformances(r$pred, measures = list(acc, mmce, auc)))
#        acc.test.mean mmce.test.mean auc.test.mean
# label1     0.7389326      0.2610674     0.6801810
# label2     0.5908151      0.4091849     0.5935160
# label3     0.6512205      0.3487795     0.6631469
# label4     0.6921804      0.3078196     0.6965552
# label5     0.7517584      0.2482416     0.6748458
# label6     0.7343815      0.2656185     0.6054968
```

Parallelization

In the case of a high number of labels and larger datasets, parallelization in the training and prediction process of the multilabel methods can reduce computation time. This can be achieved by using the package parallelMap in mlr (see also the tutorial section of parallelization: http://mlr-org.github.io/mlr-tutorial/release/html/multilabel/index.html). Currently, only the binary relevance method is parallelizable, the classifier for each label is trained in parallel, as they are independent of each other. The other problem transformation methods will also be parallelizable (as far as possible) soon.

```
library(parallelMap)
parallelStartSocket(2)
lrn = makeMultilabelBinaryRelevanceWrapper("classif.rpart")
mod = train(lrn, yeast.task)
pred = predict(mod, yeast.task)
```

Benchmark experiment

In a similar fashion to Wang et al. (2014), we performed a benchmark experiment on several datasets in order to compare the performances of the different multilabel algorithms.

Datasets: In Table 2 we provide an overview of the used datasets. We retrieved most datasets from the Mulan Java library for multilabel learning[2] as well as from other benchmark experiments of multilabel classification methods. See Table 2 for article references. We uploaded all datasets to the open data platform OpenML (Casalicchio et al., 2017; Vanschoren et al., 2013), so they now can be downloaded directly from there. In some of the used datasets, sparse labels had to be removed in order to avoid problems during cross-validation. Several binary classification methods have difficulties when labels are sparse, i.e., a strongly imbalanced binary target class can lead to constant predictions for that target. That can sometimes lead to direct problems in the base learners (when training on constant class labels is simply not allowed) or, e.g., in classifier chains, when the base learner cannot handle constant features. Furthermore, one can reasonably argue that not much is to be learned for such a label. Hence, labels that appeared in less than 2% of the observations were removed. We computed *cardinality* scores (based on the remaining labels) indicating the mean number of labels assigned to each case in the respective dataset. The following description of the datasets refers to the final versions after removal of sparse labels.

- The first dataset (*birds*) consists of 645 audio recordings of 15 different vocalizing bird species (Briggs et al., 2013). Each sound can be assigned to various bird species.
- Another audio dataset (*emotions*) consists of 593 musical files with 6 clustered emotional labels (Trohidis et al., 2008) and 72 predictors. Each song can be labeled with one or more of the labels {*amazed-surprised*, *happy-pleased*, *relaxing-calm*, *quiet-still*, *sad-lonely*, *angry-fearful*}.
- The *genbase* dataset contains protein sequences that can be assigned to several classes of protein families (Diplaris et al., 2005). The entire dataset contains 1186 binary predictors.
- The *langLog*[3] dataset includes 998 textual predictors and was originally compiled in the doctorial thesis of Read (2010). It consists of 1460 text samples that can be assigned to one or more topics such as *language*, *politics*, *errors*, *humor* and *computational linguistics*.
- The UC Berkeley *enron*[4] dataset represents a subset of the original *enron*[5] dataset and consists of 1702 cases of emails with 24 labels and 1001 predictor variables (Klimt and Yang, 2004).
- A subset of the *reuters*[6] dataset includes 2000 observations for text classification (Zhang and Zhou, 2008).
- The *image*[7] benchmark dataset consists of 2000 natural scene images. Zhou and Zhang (2007) extracted 135 features for each image and made it publicly available as *processed* image dataset. Each observation can be associated with different label sets, where all possible labels are {*desert*, *mountains*, *sea*, *sunset*, *trees*}. About 22% of the images belong to more than one class. However, images belonging to three classes or more are very rare.
- The *scene* dataset is an image classification task where labels like *Beach, Mountain, Field, Urban* are assigned to each image (Boutell et al., 2004).
- The *yeast* dataset (Elisseeff and Weston, 2002) consists of micro-array expression data, as well as phylogenetic profiles of yeast, and includes 2417 genes and 103 predictors. In total, 14 different labels can be assigned to a gene, but only 13 labels were used due to label sparsity.
- Another dataset for text-classification is the *slashdot*[8] dataset (Read et al., 2011). It consists of article titles and partial blurbs. Blurbs can be assigned to several categories (e.g., *Science, News, Games*) based on word predictors.

Algorithms: We used all multilabel classification methods currently implemented in **mlr**: binary relevance (BR), classifier chains (CC), nested stacking (NST), dependent binary relevance (DBR) and stacking (STA) as well as algorithm adaption methods of the **rFerns** (RFERN) and **randomForestSRC** (RFSRC) packages. For DBR and STA the first level and meta level classifiers were equal. For CC and NST we chose random chain orders for each resample iteration.

[2] http://mulan.sourceforge.net/datasets-mlc.html
[3] http://languagelog.ldc.upenn.edu/nll/
[4] http://bailando.sims.berkeley.edu/enron_email.html
[5] http://www.cs.cmu.edu/~enron/
[6] http://lamda.nju.edu.cn/data_MIMLtext.ashx
[7] http://lamda.nju.edu.cn/data_MIMLimage.ashx
[8] http://slashdot.org

Dataset	Reference	# Inst.	# Pred.	# Labels	Cardinality
birds*	Briggs et al. (2013)	645	260	15	0.96
emotions	Trohidis et al. (2008)	593	72	6	1.87
genbase*	Diplaris et al. (2005)	662	112	16	1.20
langLog*	Read (2010)	1460	998	18	0.85
enron*	Klimt and Yang (2004)	1702	1001	24	3.12
reuters	Zhang and Zhou (2008)	2000	243	7	1.15
image	Zhou and Zhang (2007)	2000	135	5	1.24
scene	Boutell et al. (2004)	2407	294	6	1.07
yeast*	Elisseeff and Weston (2002)	2417	103	13	4.22
slashdot*	Read et al. (2011)	3782	1079	14	1.13

Table 2: Used benchmark datasets including number of instances, number of predictor, number of label and label cardinality. Datasets with an asterisk differ from the original dataset as sparse labels have been removed. The genbase dataset contained many constant factor variables, which were automatically removed by mlr.

Base Learners: We employed two different binary classification base learner for each problem transformation algorithm: random forest (rf) of the **randomForest** package (Liaw and Wiener, 2002) with ntree = 100 and adaboost (ad) from the **ada** package (Culp et al., 2012), each with standard hyperparameter settings.

Performance Measures: We used the six previously proposed performance measures. Furthermore, we calculated the reported values by means of a 10-fold cross-validation.

Code: For reproducibility, the complete code and results can be downloaded from Probst (2017). The R package **batchtools** (Bischl et al., 2015) was used for parallelization.

The results for hamming loss and F_1-index are illustrated in Figure 1. Tables 3 and 4 contain performance values with the best performing algorithms highlighted in blue. For all remaining measures one may refer to the Appendix. We did not perform any threshold tuning that would potentially improve some of the performance of the methods.

The results of the problem transformation methods in this benchmark experiment concur with the general conclusions and results in Montañés et al. (2014). The authors ran a similar benchmark study with penalized logistic regression as base learner. They concluded that, on average, DBR performs well in F_1 and accuracy. Also, CC outperform the other methods regarding the subset 0/1 loss most of the time. For the hamming loss measure they got mixed results, with no clear winner concordant to our benchmark results. As base learner, on average, adaboost performs better than random forest in our benchmark study.

Considering the measure F_1, the problem transformation methods DBR, CC, STA and NST outperform RFERN and RFSRC on most of the datasets and also almost always perform better than BR, which does not consider dependencies among the labels. RFSRC and RFERN only perform well on either precision or recall, but in order to be considered as good classifiers they should perform well on both. The generally poor performances of RFERN can be explained by the working mechanism of the algorithm which randomly chooses variables and split points at each split of a fern. Hence, it cannot deal with too many features that are useless for the prediction of the target labels.

Summary

In this paper, we describe the implementation of multilabel classification algorithms in the R package **mlr**. The problem transformation methods binary relevance, classifier chains, nested stacking, dependent binary relevance and stacking are implemented and can be used with any base learner that is accessible in **mlr**. Moreover, there is access to the multilabel classification versions of **randomForestSRC** and **RFerns**. We compare all of these methods in a benchmark experiment with several datasets and different implemented multilabel performance measures. The dependent binary relevance method performs well regarding the measures F_1 and *accuracy*. Classifier chains outperform the other methods in terms of the subset 0/1 loss most of the time. Parallelization is available for the binary relevance method and will be available soon for the other problem transformation methods. Algorithm adaptation methods and problem transformation methods that are currently not available can be incorporated in the current **mlr** framework easily. In our benchmark experiment we had to remove labels which occured too sparsely, because some algorithms crashed due to one class problems, which appeared during cross-validation. A solution to this problem and an implementation into the **mlr** framework is of great interest.

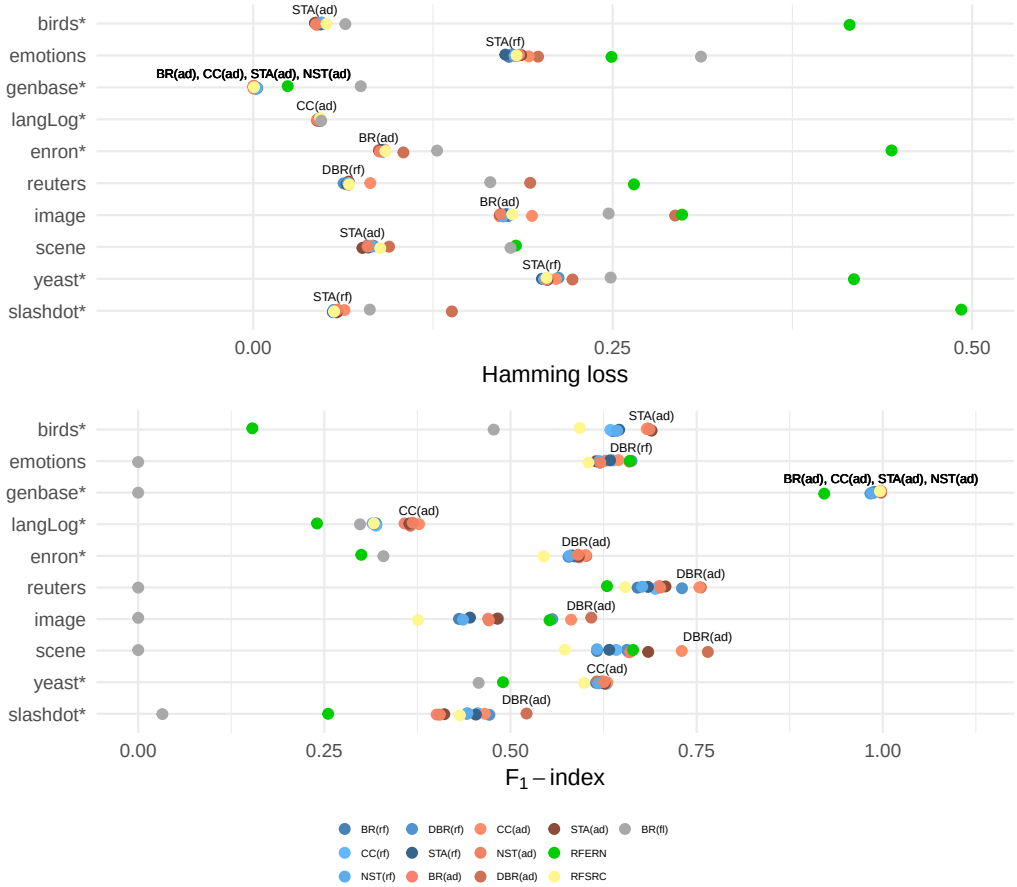


Figure 1: Results for hamming loss and F_1-index. The best performing algorithms are highlighted on the plot.

	BR(rf)	CC(rf)	NST(rf)	DBR(rf)	STA(rf)	BR(ad)	CC(ad)	NST(ad)	DBR(ad)	STA(ad)	RFERN	RFSRC	BR(fl)
birds*	0.0477	0.0479	0.0475	0.0472	0.0468	0.0442	0.0441	0.0436	0.0431	**0.0429**	0.4148	0.0510	0.0641
emotions	0.1779	0.1832	0.1818	0.1801	**0.1753**	0.181	0.1916	0.1849	0.1981	0.1863	0.2492	0.1832	0.3114
genbase*	0.0021	0.0023	0.0025	0.0027	0.0023	**0.0003**	**0.0003**	**0.0003**	0.0004	**0.0003**	0.0240	0.0006	0.0748
langLog*	0.0464	0.0465	0.0467	0.0464	0.0466	0.0451	**0.0442**	0.0446	0.0447	0.0448	0.6673	0.0466	0.0473
enron*	0.0903	0.0904	0.0902	0.0909	0.0891	**0.0874**	0.0913	0.0881	0.1045	0.0877	0.4440	0.0919	0.1279
reuters	0.0663	0.0654	0.0661	**0.0629**	0.065	0.0666	0.0814	0.0664	0.1926	0.0664	0.2648	0.0668	0.1649
image	0.1774	0.1791	0.1737	0.1761	0.1754	**0.1714**	0.1939	0.1721	0.2935	0.1717	0.2983	0.1802	0.2472
scene	0.0836	0.0809	0.0832	0.0796	0.0799	0.0791	0.0821	0.0796	0.0945	**0.076**	0.1827	0.0884	0.1790
yeast*	0.2038	0.2044	0.2023	0.2123	**0.2008**	0.2048	0.2105	0.2038	0.2221	0.2046	0.4178	0.2040	0.2486
slashdot*	0.0558	0.0560	0.0559	0.0559	**0.0554**	0.059	0.0635	0.0586	0.1382	0.0582	0.4925	0.0562	0.0811

Table 3: Hamming loss

	BR(rf)	CC(rf)	NST(rf)	DBR(rf)	STA(rf)	BR(ad)	CC(ad)	NST(ad)	DBR(ad)	STA(ad)	RFERN	RFSRC	BR(fl)
birds*	0.6369	0.6342	0.6433	0.64	0.6459	0.6835	0.683	0.6867	0.6846	**0.6895**	0.1533	0.5929	0.4774
emotions	0.6199	0.6380	0.6192	**0.6625**	0.6337	0.6274	0.6449	0.6206	0.6598	0.615	0.6603	0.6046	0.0000
genbase*	0.9885	0.9861	0.9855	0.9835	0.9861	**0.9977**	**0.9977**	**0.9977**	0.9962	**0.9977**	0.9214	0.9962	0.0000
langLog*	0.3192	0.3194	0.3148	0.3199	0.3167	0.3578	**0.3772**	0.3686	0.3653	0.3643	0.2401	0.3167	0.2979
enron*	0.5781	0.5822	0.5791	0.5866	0.5826	0.592	0.6009	0.5906	**0.6017**	0.5917	0.2996	0.5446	0.3293
reuters	0.6708	0.6944	0.6769	0.7303	0.6846	0.6997	0.7537	0.7012	**0.7556**	0.7082	0.6296	0.6541	0.0000
image	0.4308	0.4835	0.4362	0.5561	0.4456	0.47	0.5814	0.4709	**0.6085**	0.4824	0.5525	0.3757	0.0000
scene	0.6161	0.6420	0.6161	0.6563	0.6326	0.6585	0.73	0.661	**0.765**	0.685	0.6647	0.5729	0.0000
yeast*	0.6148	0.6294	0.6180	0.6195	0.6244	0.6238	**0.63**	0.6257	0.616	0.6266	0.4900	0.5991	0.4572
slashdot*	0.4415	0.4562	0.4422	0.4716	0.4535	0.4009	0.4654	0.4052	**0.5216**	0.411	0.2551	0.4320	0.0325

Table 4: F_1-index

Bibliography

B. Bischl, M. Lang, O. Mersmann, J. Rahnenführer, and C. Weihs. BatchJobs and BatchExperiments: Abstraction mechanisms for using R in batch environments. *Journal of Statistical Software*, 64(11): 1–25, 2015. doi: 10.18637/jss.v064.i11. [p12]

B. Bischl, M. Lang, L. Kotthoff, J. Schiffner, J. Richter, E. Studerus, G. Casalicchio, and Z. M. Jones. mlr: Machine learning in R. *Journal of Machine Learning Research*, 17(170):1–5, 2016. [p1]

M. R. Boutell, J. Luo, X. Shen, and C. M. Brown. Learning multi-label scene classification. *Pattern Recognition*, 37(9):1757–1771, 2004. doi: 10.1016/j.patcog.2004.03.009. [p1, 11, 12]

F. Briggs, H. Yonghong, R. Raich, et al. New methods for acoustic classification of multiple simultaneous bird species in a noisy environment. In *IEEE International Workshop on Machine Learning for Signal Processing*, pages 1–8, 2013. doi: 10.1109/mlsp.2013.6661934. [p1, 11, 12]

G. Casalicchio, J. Bossek, M. Lang, D. Kirchhoff, P. Kerschke, B. Hofner, H. Seibold, J. Vanschoren, and B. Bischl. OpenML: An R package to connect to the networked machine learning platform OpenML. *ArXiv e-prints*, 2017. [p11]

F. Charte and D. Charte. Working with multilabel datasets in R: The mldr package. *The R Journal*, 7(2): 149–162, 2015. [p1, 9]

M. Culp, K. Johnson, and G. Michailidis. *ada: an R package for stochastic boosting*, 2012. URL https://cran.r-project.org/package=ada. [p12]

P. N. da Silva, E. C. Gonçalves, A. Plastino, and A. A. Freitas. Distinct chains for different instances: an effective strategy for multi-label classifier chains. In *European Conference, ECML PKDD 2014*, pages 453–468, 2014. doi: 10.1007/978-3-662-44851-9_29. [p5]

T. G. Dietterich. Ensemble methods in machine learning. *Lecture Notes in Computer Science*, 1857:1–15, 2000. doi: 10.1007/3-540-45014-9_1. [p6]

S. Diplaris, G. Tsoumakas, P. A. Mitkas, and I. Vlahavas. Protein classification with multiple algorithms. In *Advances in Informatics*, pages 448–456. Springer, 2005. doi: 10.1007/11573036_42. [p11, 12]

A. Elisseeff and J. Weston. A kernel method for multi-labelled classification. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 681–687. MIT Press, 2002. [p1, 11, 12]

S. Godbole and S. Sarawagi. Discriminative methods for multi-labeled classification. In *Advances in Knowledge Discovery and Data*, volume LNCS3056, pages 22–30, 2004. doi: 10.1007/978-3-540-24775-3_5. [p7]

H. Ishwaran and U. B. Kogalur. *Random Forests for Survival, Regression and Classification (RF-SRC)*, 2016. URL http://cran.r-project.org/package=randomForestSRC. [p2]

B. Klimt and Y. Yang. The enron corpus: A new dataset for email classification research. *Machine Learning: ECML 2004*, pages 217–226, 2004. doi: 10.1007/978-3-540-30115-8_22. [p11, 12]

M. B. Kursa and A. A. Wieczorkowska. Multi-label ferns for efficient recognition of musical instruments in recordings. In *International Symposium on Methodologies for Intelligent Systems*, pages 214–223. Springer, 2014. doi: 10.1007/978-3-319-08326-1_22. [p1, 2]

M. Lang, H. Kotthaus, P. Marwedel, C. Weihs, J. Rahnenführer, and B. Bischl. Automatic model selection for high-dimensional survival analysis. *Journal of Statistical Computation and Simulation*, 85 (1):62–76, 2015. doi: 10.1080/00949655.2014.929131. [p1]

A. Liaw and M. Wiener. Classification and regression by randomForest. *R News: The Newsletter of the R Project*, 2(3):18–22, 2002. [p12]

A. McCallum. Multi-label text classification with a mixture model trained by EM. *AAAI'99 Workshop on Text Learning*, pages 1–7, 1999. [p1]

E. Montañés, R. Senge, J. Barranquero, J. R. Quevedo, J. J. del Coz, and E. Hüllermeier. Dependent binary relevance models for multi-label classification. *Pattern Recognition*, 47(3):1494–1508, 2014. doi: 10.1016/j.patcog.2013.09.029. [p2, 3, 4, 6, 7, 12]

P. Probst. Multilabel classification with R package mlr. figshare. 2017. doi: 10.6084/m9.figshare.3384802.v5. URL https://dx.doi.org/10.6084/m9.figshare.3384802.v5. [p12]

J. Read. Scalable multi-label classification. *Hamilton, New Zealand: University of Waikato*, 2010. [p11, 12]

J. Read and P. Reutemann. Meka: A multi-label extension to WEKA, 2012. URL http://meka.sourceforge.net/. [p1]

J. Read, B. Pfahringer, G. Holmes, and E. Frank. Classifier chains for multi-label classification. *Machine Learning*, 85:333–359, 2011. doi: 10.1007/s10994-011-5256-5. [p4, 5, 11, 12]

J. Read, L. Martino, and D. Luengo. Efficient Monte Carlo methods for multi-dimensional learning with classifier chains. *Pattern Recognition*, (Mdc):1–36, 2013. doi: 10.1016/j.patcog.2013.10.006. [p5]

C. Sanden and J. Z. Zhang. Enhancing multi-label music genre classification through ensemble techniques. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 705–714, 2011. doi: 10.1145/2009916.2010011. [p1]

R. E. Schapire and Y. Singer. BoosTexter: A boosting-based system for text categorization. *Machine Learning*, 39:135–168, 2000. [p1]

R. Senge, J. J. d. Coz Velasco, and E. Hüllermeier. Rectifying classifier chains for multi-label classification. *Space, 2 (8)*, 2013. [p3, 5, 8]

J. Sill, G. Takacs, L. Mackey, and D. Lin. Feature-Weighted Linear Stacking. *ArXiv e-prints*, 2009. [p7]

R. Simon. Resampling strategies for model assessment and selection. In W. Dubitzky, M. Granzow, and D. Berrar, editors, *Fundamentals of Data Mining in Genomics and Proteomics SE - 8*, pages 173–186. Springer US, 2007. doi: 10.1007/978-0-387-47509-7_8. [p4]

K. Trohidis, G. Tsoumakas, G. Kalliris, and I. P. Vlahavas. Multi-label classification of music into emotions. *ISMIR*, 8:325–330, 2008. doi: 10.1186/1687-4722-2011-426793. [p1, 11, 12]

G. Tsoumakas and I. Katakis. Multi label classification: An overview. *International Journal of Data Warehousing and Mining*, 3(3):1–13, 2007. doi: 10.4018/jdwm.2007070101. [p1]

G. Tsoumakas, E. S. Xioufis, J. Vilcek, and I. P. Vlahavas. Mulan: A java library for multi-label learning. *Journal of Machine Learning Research*, 12:2411–2414, 2011. [p1]

J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo. OpenML: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013. doi: 10.1145/2641190.2641198. [p11]

H. Wang, X. Liu, B. Lv, F. Yang, and Y. Hong. Reliable multi-label learning via conformal predictor and random forest for syndrome differentiation of chronic fatigue in traditional chinese medicine. *PLoS ONE*, 9(6), 2014. doi: 10.1371/journal.pone.0099565. [p11]

D. H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992. doi: 10.1016/S0893-6080(05)80023-1. [p7]

M. L. Zhang and Z. H. Zhou. M3MIML: A maximum margin method for multi-instance multi-label learning. In *Proceedings - IEEE International Conference on Data Mining, ICDM*, pages 688–697, 2008. doi: 10.1109/ICDM.2008.27. [p1, 11, 12]

Z.-H. Zhou and M.-l. Zhang. Multi-instance multilabel learning with application to scene classification. *Neural Information Processing Systems*, 40(7):2038–2048, 2007. [p11, 12]

Philipp Probst
Department of Medical Informatics, Biometry and Epidemiology
LMU Munich
81377 Munich
Germany
probst@ibe.med.uni-muenchen.de

Quay Au
Department of Statistics
LMU Munich
80539 Munich
Germany
quay.au@stat.uni-muenchen.de

Giuseppe Casalicchio
Department of Statistics
LMU Munich
80539 Munich
Germany
giuseppe.casalicchio@stat.uni-muenchen.de

Clemens Stachl
Department of Psychology
LMU Munich
80802 Munich
Germany
clemens.stachl@psy.lmu.de

Bernd Bischl
Department of Statistics
LMU Munich
80539 Munich
Germany
bernd.bischl@stat.uni-muenchen.de

Appendices

	BR(rf)	CC(rf)	NST(rf)	DBR(rf)	STA(rf)	BR(ad)	CC(ad)	NST(ad)	DBR(ad)	STA(ad)	RFERN	RFSRC	BR(fl)
birds*	0.4481	0.4481	0.4466	0.4497	0.4451	**0.4156**	0.4218	0.4171	0.4233	0.4202	0.9830	0.4777	0.5226
emotions	0.6846	0.6575	0.6728	**0.6457**	0.6626	0.6777	0.6643	0.7031	0.6845	0.6828	0.7992	0.6829	1.0000
genbase*	0.0333	0.0363	0.0393	0.0423	0.0363	**0.0045**	**0.0045**	**0.0045**	0.0060	**0.0045**	0.2115	0.0091	1.0000
langLog*	0.6836	0.6829	0.6884	0.6842	0.6856	0.6521	**0.6349**	0.6418	0.6438	0.6466	0.8589	0.6856	0.7021
enron*	0.8531	0.8413	0.8560	0.8408	0.8484	0.8496	**0.819**	0.8484	0.8320	0.8408	1.0000	0.8619	0.9982
reuters	0.3620	0.3405	0.3575	0.311	0.3515	0.349	**0.2945**	0.338	0.3495	0.3385	0.5830	0.3695	1.0000
image	0.6635	0.6150	0.6505	0.575	0.6445	0.63	**0.539**	0.6275	0.6225	0.619	0.8365	0.6955	1.0000
scene	0.4225	0.3926	0.4217	0.3835	0.4046	0.3913	**0.3095**	0.3805	0.3610	0.3648	0.7540	0.4570	1.0000
yeast*	0.8316	0.7600	0.8201	0.8167	0.8155	0.8304	**0.7563**	0.8134	0.8217	0.806	0.9338	0.8337	0.9855
slashdot*	0.6140	0.5994	0.6116	**0.5859**	0.6052	0.6489	0.5923	0.6449	0.6658	0.6396	0.9966	0.6142	0.9675

Table 5: Subset 0/1 loss

	BR(rf)	CC(rf)	NST(rf)	DBR(rf)	STA(rf)	BR(ad)	CC(ad)	NST(ad)	DBR(ad)	STA(ad)	RFERN	RFSRC	BR(fl)
birds*	0.6153	0.6126	0.6197	0.6169	0.6232	0.6589	0.657	0.6604	0.6581	**0.6621**	0.0999	0.5753	0.4774
emotions	0.5453	0.5649	0.5464	**0.5849**	0.5609	0.5519	0.5676	0.5408	0.5727	0.5427	0.5503	0.5332	0.0000
genbase*	0.9834	0.9806	0.9796	0.9773	0.9806	**0.9972**	**0.9972**	**0.9972**	0.9957	**0.9972**	0.8884	0.9950	0.0000
langLog*	0.3185	0.3188	0.3140	0.3188	0.3161	0.3553	**0.3741**	0.366	0.363	0.3615	0.1953	0.3161	0.2979
enron*	0.4693	0.4757	0.4694	0.4804	0.4742	0.483	**0.4987**	0.4824	0.4919	0.4847	0.1859	0.4394	0.2241
reuters	0.6625	0.6856	0.6682	0.7199	0.6754	0.6873	**0.7414**	0.6912	0.7197	0.6964	0.5620	0.6482	0.0000
image	0.4068	0.4585	0.4142	0.5225	0.4228	0.4446	**0.5508**	0.4458	0.5366	0.4564	0.4467	0.3578	0.0000
scene	0.6064	0.6333	0.6067	0.6463	0.6233	0.646	0.7201	0.6505	**0.7313**	0.6725	0.5513	0.5654	0.0000
yeast*	0.5091	0.5320	0.5138	0.514	0.5205	0.5182	**0.5345**	0.522	0.5068	0.5239	0.3674	0.4945	0.3361
slashdot*	0.4274	0.4421	0.4285	0.4569	0.4385	0.3883	0.4507	0.3925	**0.4613**	0.3982	0.1651	0.4202	0.0325

Table 6: Accuracy

	BR(rf)	CC(rf)	NST(rf)	DBR(rf)	STA(rf)	BR(ad)	CC(ad)	NST(ad)	DBR(ad)	STA(ad)	RFERN	RFSRC	BR(fl)
birds*	0.2763	0.2752	0.2897	0.2859	0.2936	0.3755	0.3865	0.3772	0.3687	0.3784	**0.8352**	0.1949	0.0000
emotions	0.6197	0.6474	0.6187	0.6847	0.6358	0.6335	0.6708	0.6293	0.7189	0.6237	**0.8276**	0.6001	0.0000
genbase*	0.9846	0.9819	0.9809	0.9786	0.9819	**0.9977**	**0.9977**	**0.9977**	0.9962	**0.9977**	0.9962	0.9955	0.0000
langLog*	0.0334	0.0330	0.0270	0.0331	0.0308	0.0971	0.1191	0.1056	0.0995	0.102	**0.9264**	0.0301	0.0000
enron*	0.5426	0.5487	0.5421	0.5580	0.5466	0.5611	0.5902	0.5619	0.6314	0.5633	**0.771**	0.4959	0.2613
reuters	0.6733	0.6959	0.6801	0.7338	0.6875	0.7038	0.754	0.7046	**0.9032**	0.7123	0.8598	0.6559	0.0000
image	0.4192	0.4696	0.4228	0.5562	0.4335	0.4581	0.5691	0.4603	**0.7787**	0.4724	0.7374	0.3598	0.0000
scene	0.6148	0.6373	0.6134	0.6555	0.6306	0.6614	0.7243	0.6613	0.8174	0.6879	**0.9173**	0.5662	0.0000
yeast*	0.5722	0.6097	0.5788	0.6035	0.5874	0.5951	0.6229	0.5978	0.6104	0.6013	**0.6296**	0.5442	0.3365
slashdot*	0.4267	0.4412	0.4270	0.4574	0.4391	0.3834	0.4526	0.3868	0.6984	0.3931	**0.8065**	0.4094	0.0000

Table 7: Recall

	BR(rf)	CC(rf)	NST(rf)	DBR(rf)	STA(rf)	BR(ad)	CC(ad)	NST(ad)	DBR(ad)	STA(ad)	RFERN	RFSRC	BR(fl)
birds*	0.8812	0.8889	0.8764	**0.9056**	0.8874	0.8461	0.8349	0.8401	0.8648	0.8605	0.0859	0.8996	
emotions	0.7627	0.7242	0.7499	0.7265	**0.7644**	0.7537	0.7014	0.739	0.6783	0.7347	0.5869	0.7577	
genbase*	0.9987	0.9987	0.9987	0.9987	0.9987	**0.9995**	**0.9995**	**0.9995**	**0.9995**	**0.9995**	0.8917	**0.9995**	
langLog*	0.7267	**0.7356**	0.7058	0.7207	0.6882	0.6874	0.7228	0.7133	0.7014	0.6965	0.0632	0.7233	
enron*	0.7283	0.7188	0.7305	0.7092	0.7331	0.7235	0.6807	0.7198	0.6371	0.7233	0.1973	**0.7448**	0.5135
reuters	0.9411	0.9168	0.9346	0.8995	0.9298	0.9014	0.7689	0.8983	0.7465	0.8931	0.5715	**0.9562**	
image	0.7899	0.7333	0.8029	0.7086	0.7865	0.7841	0.6281	0.7814	0.6036	0.7737	0.4813	**0.83**	
scene	0.9071	0.8956	0.9112	0.8917	0.9143	0.8936	0.81	0.8856	0.7879	0.8872	0.5662	**0.9233**	
yeast*	0.7372	0.7218	0.7351	0.7055	0.7389	0.7225	0.6947	0.7233	0.6827	0.7159	0.4361	**0.7508**	0.7478
slashdot*	0.8365	0.8127	0.8298	0.7927	0.8277	0.8119	0.6804	0.8161	0.5025	0.8196	0.1679	**0.8366**	

Table 8: Precision (For the featureless learner we have no precision results for several datasets. The reason is that the featureless learner does not predict any value in all observations in these datasets. Hence, the denominator in the precision formula is always zero. *mlr* predicts NA in this case.)

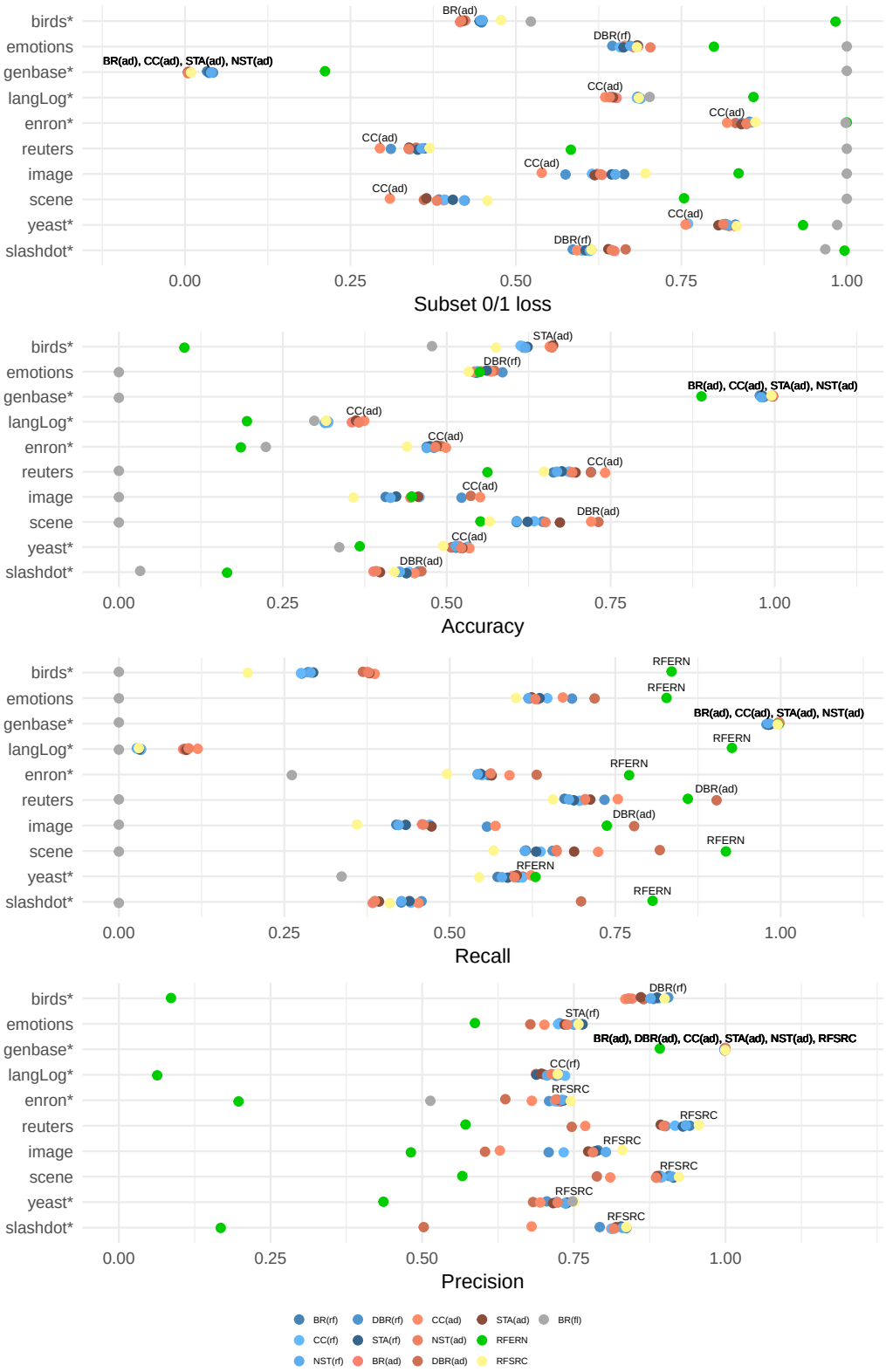


Figure 2: Results for the remaining measures.