https://www.aimspress.com/journal/aci

Applied Computing and Intelligence
2(1): 49–62
DOI: 10.3934/aci.2022003
Received: 11 February 2022
Revised: 30 March 2022
Accepted: 30 March 2022
Published: 31 March 2022

Research article

Novel split quality measures for stratified multilabel cross validation with application to large and sparse gene ontology datasets

Henri Tiittanen[1,*], **Liisa Holm**[1,2] **and Petri Törönen**[1]

[1] Institute of Biotechnology, Helsinki Institute of Life Sciences (HiLife), University of Helsinki, Helsinki, Finland

[2] Organismal and Evotionary Biology Research Program, Faculty of Biosciences, University of Helsinki, Helsinki, Finland

* **Correspondence:** henri.tiittanen@helsinki.fi

Academic Editor: Pasi Fränti

Abstract: Multilabel learning is an important topic in machine learning research. Evaluating models in multilabel settings requires specific cross validation methods designed for multilabel data. In this article, we show that the most widely used cross validation split quality measure does not behave adequately with multilabel data that has strong class imbalance. We present improved measures and an algorithm, OPTISPLIT, for optimizing cross validations splits. Extensive comparison of various types of cross validation methods shows that OPTISPLIT produces more even cross validation splits than the existing methods and it is among the fastest methods with good splitting performance.

Keywords: stratified cross validation; multilabel learning; multilabel cross validation; classification; gene ontology

1. Introduction

Cross validation is a central procedure in machine learning, statistics and other fields. It is used to evaluate model performance by testing models on data points excluded from training data. It is further used, besides traditional model evaluation, also when various predictive models are combined in classifier stacking [15] and when various parameters in predictive models are optimized. In standard cross validation, a dataset D is split randomly into k non-overlapping subsets (folds) $D_i, i \in k$. A model is trained for each fold i on the data $\cup_{j \in k | j \neq i} D_j$ and evaluated on D_i. So one subset, D_i, is left out of the training process and used as an evaluation data. The averaged result over all the folds represents the final performance. Cross validation is typically used when the amount of data is limited. This situation can occur also in a very large dataset when some of the studied classes are very rare. When very large

amount of data is available, one can also use standard train-test splits, where the data is divided into two non-overlapping sets, e.g. 80% for training and 20% for testing, in place of cross validation.

The typically used random split approach assumes that the positive and negative class distributions are balanced. If the class distributions are imbalanced, the resulting splits may not allow efficient learning. As an example, suppose that in binary classification settings one randomly generated fold contains all the positive data points belonging to one of the classes (see Figure 1). Then the corresponding training set consisting of the rest of the folds does not contain any positive data points of that class and the model cannot learn anything about the class.

Stratified cross validation methods are variants of cross validation that ensure that the class distributions of the folds are close to the class distributions of the whole data. With class imbalanced data, these stratified methods especially ensure the distribution of smaller classes. In this work, we focus on stratified cross validation applied to multilabel classification. Multilabel classification presents additional challenges for stratified cross validation, as each data point can belong to multiple classes simultaneously (see Figure 1). Here, each class represents a separate task for training and evaluation. The cross validation split should be formed so that the correct class distributions are maintained for all classes and all folds at the same time. This ensures that the classifier to be evaluated can be trained and tested effectively with all the created data splits. Random splitting has been historically a popular choice on multilabel data, but it has been shown to lead to poor results [9].

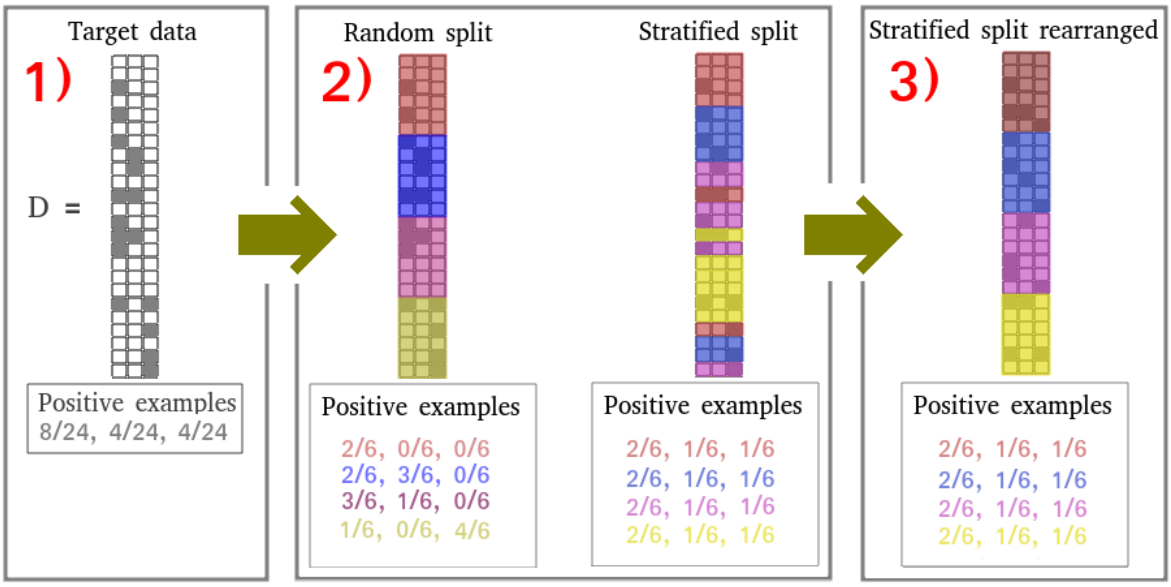

Figure 1. Examples of stratified multilabel cross validation with splits of different quality. 1) presents a target data with 24 data points and three classes. This could be a subset of a sparse, high dimensional dataset. 2) Next, the data is split into four cross validation folds in two ways, one representing a random split and the other representing a stratified split optimized over all classes. The ideal distribution of the positive data points in each fold would be 2/6 for the first class and 1/6 for the second and third class so that they would follow the class distributions of the whole data. Random split can result, like in here, in very unbalanced folds, while the stratified split follows closely the data distribution for all classes. 3) Here, the well-balanced stratified split is rearranged for increased clarity.

As dataset sizes are growing across application domains, multilabel and extreme classification are of growing interest and importance [2]. Consequently, assessing model quality on these settings is also getting increasingly important. This makes the stratified cross validation for large multilabel datasets with sparse labels a relevant research topic.

However, we show in Section 3 that there is a serious flaw in the main measure currently used to evaluate how good a given split is. In response to the issue, we present new measures that have better properties. We found current methods either too weak with respect to the new measures or impractically slow for extremely big classification datasets. Therefore, we also developed a new algorithm, OPTISPLIT, for generating multilabel cross validation datasets based on optimizing the global distribution of all classes.

The methods presented here are developed in the context of *gene ontology* (GO) data [1] for automated protein function prediction task [16]. GO represents a large set of classes that aim to describe various functions that proteins can have. It is structured as a directed acyclic graph (DAG), where members of small classes that link genes to very specific functions, are also included in larger classes that link genes to broader functional categories. As an example, genes included in the *cellular amino acid biosynthesis class* are also members of the *carboxylic acid biosynthesis class* and eventually to the *metabolic process class* [1]. This hierarchy allows the analysis of gene or protein functions with both detailed and broad functions.

The GO datasets used here are high dimensional and contain over half a million data points. The high number of classes, often in the order of thousands to tens of thousands, presents a challenge for analysis. Members of smaller, more precise classes are automatically included also in larger, broader classes. This inclusion step is repeated with larger and larger classes until one of the three root nodes of DAG is reached. Therefore, as a result, most classes contain a very small number of data points while some classes have a very large number of data points. The high number of small classes, i.e. classes that have few positive data points, makes the negative data abundant for most of the classes, resulting often in too few positive data points to train the models effectively.

This article is organized as follows. In Section 2, we review the related work. In Section 3, we show that the currently most widely used split quality measure for stratified multilabel cross validation is inadequate. We present new better measures and demonstrate the behavior of these first with synthetic data splits. In Section 4, we present an algorithm for optimizing cross validation splits with respect to any selected split quality measures. In Section 5, we present a comparison of the algorithms on a wide range of real-world datasets and conclude that the new algorithm is the best practical choice for getting balanced splits as measured by the new measures. Finally, we conclude with a discussion and present ideas for future work in Section 6.

2. Stratified multilabel cross validation

The most widely known algorithm for generating stratified multilabel cross validation splits is the *iterative stratification* (IS) [9]. It works by dividing the data points evenly into the folds, one class at a time. It always chooses the class with the fewest positive data points for the processing, and breaks ties first by the largest number of desired data points and further randomly. Smaller classes are more difficult to balance equally among the folds, so starting from them makes sure they get well distributed. Bigger classes are easier to distribute, so distributing them later is justifiable. Iterative stratification has

also been extended to consider second order relationships between labels. This method is known as *second order iterative stratification* (SOIS) [11].

The recently introduced *stratified sampling* (SS) algorithm [7] is designed to produce balanced train/test splits for extreme classification data with a high number of data points and classes. It has been shown to be faster to use than iterative stratification variants, and it often produces splits with better distributions. This method calculates the proportion of each class in training and test sets and uses differences of these proportions to calculate a score that sums the class proportion differences for each data point. The data points with the highest scores are redistributed from one to the other partition. This method needs three parameters that have to be adjusted according to the data, and it does not produce cross validation splits directly but training/test splits.

The *partitioning method based on stratified random sampling* (PMBSRS) [4] uses the similarity of the label distributions between data points to group them and then divides them into the cross validation folds [4]. A similarity score is defined as the product of the relative frequencies of the positive labels present in each data point. Then the data points are ordered by the score into a list which is cut into as many disjoint subsets S as is the desired number of cross validation folds. Each cross validation fold is then generated by randomly selecting items without replacement from each set $s \in S$ so that each fold gets an equal proportion of the samples from each s. The end result is that each fold contains elements with different scores. As a non-iterative algorithm resembling the basic random sample method, this is computationally less expensive than the iterative methods. However, this does not measure the split quality directly, but is more aimed to ensure that all folds contain an equal amount of different sized classes.

3. Split quality measures

Split quality measures are used to define how good a given cross validation split is. They allow comparison of different splits and can be used as optimization criterion in splitting algorithms. The current cross validation split quality measures either evaluate the quality of the folds directly or apply some model on the folds and compare the learning results. Here we focus on directly comparing the quality of the folds in order to make the comparisons model independent. The most commonly used measures in the literature are the *labels distribution* (LD) measure and the *examples distribution* (ED) measure [9].

Let n be the number of data points, k be the number of cross validation folds, and q be the number of classes. $D \in \{0,1\}^{n \times q}$ denotes the multilabel target set and S_j, $j \in 1 \dots k$ denotes the folds which are disjoint subsets of D. The subsets of D and S_j containing positive data points of label $i \in 1 \dots q$ are denoted as D^i and S^i_j

We define the positive and negative frequencies for fold j and label i as $p^i_j = |S^i_j|/|S_j|$ and $1 - p^i_j$, respectively. Similarly, for the whole data, positive frequency as $d^i = |D^i|/|D|$ and negative frequency as $1 - d^i$.

Then, we define

$$\mathrm{LD} = \frac{1}{q} \sum_{i=1}^{q} \left(\frac{1}{k} \sum_{j=1}^{k} \left| \frac{p^i_j}{1 - p^i_j} - \frac{d^i}{1 - d^i} \right| \right) \tag{3.1}$$

and

$$\mathrm{ED} = \frac{1}{k}\sum_{j=1}^{k}\left|\,|S_j| - \frac{|D|}{k}\right|. \tag{3.2}$$

Intuitively, LD measures how the distribution of the positive and negative data points of each label in each subset compares to the distribution in the whole data, by comparing odds between the positive and negative ratio. ED measures the deviation between the expected fold size and the actual fold size. Since the exact equality of the fold sizes is not generally important in practice, the ED score is merely useful in checking that the fold size differences are sufficiently small compared to the data size.

As noted in Section 1, it is especially important for training and evaluation that the distributions of the smallest classes are well balanced in the cross validation folds. Small classes are hardest to split well, since there are fewer ways to distribute the data points and even small differences in fold distributions give big relative differences. A good split quality measure should be able to correctly quantify the quality of the folds, even on small classes.

LD (Equation 3.1), is defined as the arithmetic mean of the differences between the positive frequencies of a cross validation fold and of the whole data over all classes and folds, using a transformation of the form $f(x) = x/(1-x)$ for both frequencies. Since the data size is fixed, the frequencies p^i_j and d^i are linear functions of the class sizes $|S^i_j|$ and $|D^i|$, respectively. Since $x < x/(1-x), \forall x \in [0,1)$ and $f'(x) = 1/(1-x^2) > 1$, the transformed quantities grow faster than the class size, resulting that the difference is greater for bigger classes than it should be. That is, the same absolute difference of p^i_j from d^i results in a larger contribution by bigger classes (see Figure 2). The contribution is equal for the special case $p^i_j = d^i$, when the difference is zero.

Hence, as an improved alternative, we propose the *relative labels distribution* (rLD) measure

$$\mathrm{rLD} = \frac{1}{q}\sum_{i=1}^{q}\left(\frac{1}{k}\sum_{j=1}^{k}\left|\frac{d^i - p^i_j}{d^i}\right|\right). \tag{3.3}$$

Intuitively, rLD measures the linear difference of the ratios of the positive frequencies between each fold and the whole data, divided with the positive frequency of the whole data. Since d^i and p^i_j depend directly on the class size, it is necessary to weight with the positive frequency of the whole data to avoid any emphasis related to class size. Now, the difference is calculated without the offending nonlinear transformation, so the measure is class size independent (see Figure 2), while still having the same main operating principle as LD of comparing the distributions (frequencies) of folds to those of the whole data. Finally, we noticed that rLD can be seen as an adaptation of a more general measure, *mean absolute percentage error* [5], as a split quality measure.

In addition, we present another measure that is insensitive to the class size, the *delta-class proportion* (DCP)

$$\mathrm{DCP} = \frac{1}{q}\sum_{i=1}^{q}\left|\frac{\max_{j\in 1\ldots k}(|S^i_j|)}{|S^i|} - \frac{1}{k}\right|, \tag{3.4}$$

where the first part of the function represents the observed result and 1/k is the positive frequency of a flat distribution. Compared to rLD, DCP does not measure the relative distributions of other folds than the largest. Rationale for DCP is that when the fold with the largest number of sparse class

members is moved to evaluation data, it will generate the worst training data for a predictive model. DCP is here a coarser measure that is used for comparison purposes and for faster optimization.

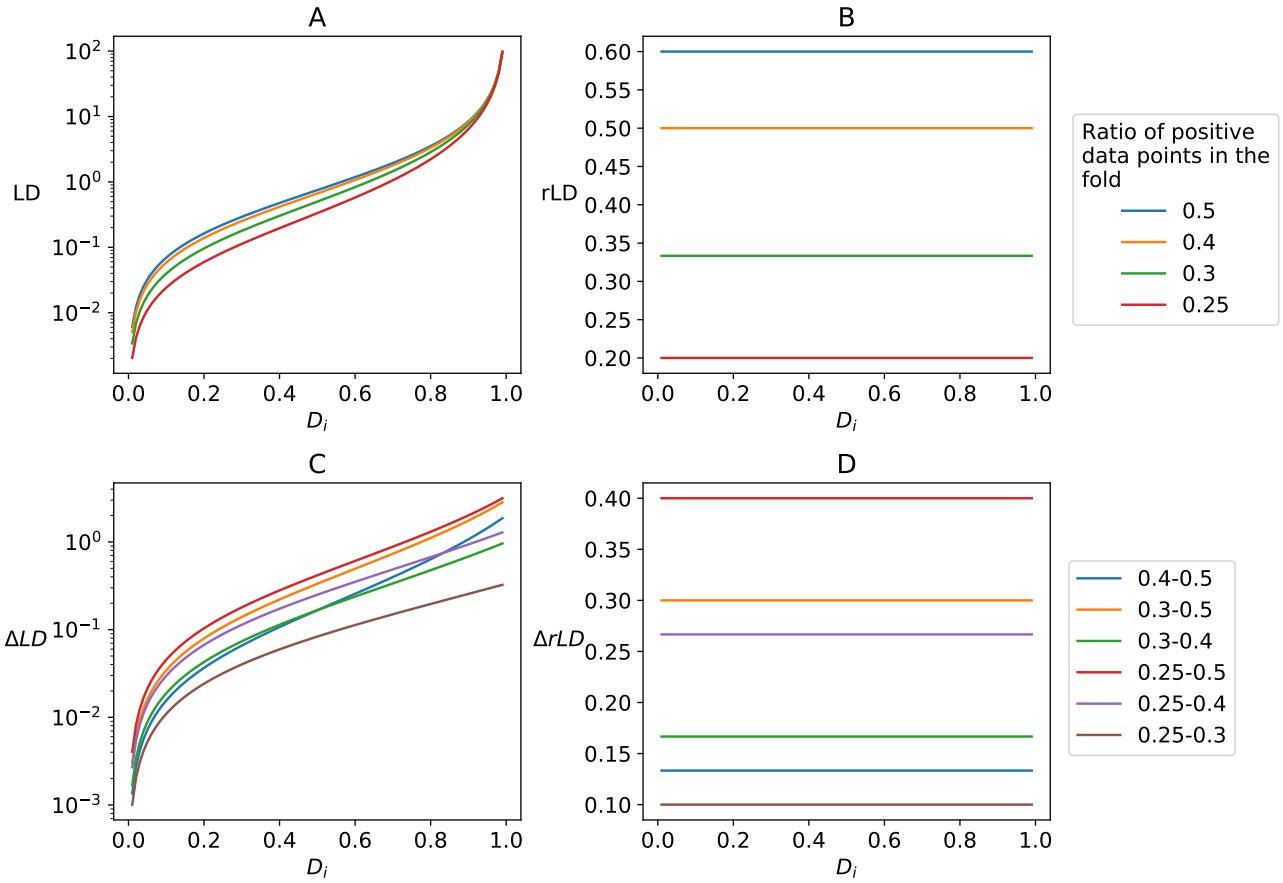

Figure 2. A graphical comparison of the LD and rLD measures within one CV-fold. The measures are evaluated against the increasing class size (D_i) on the X axis. The fold size is set to be 20% so that $S^i_j = 0.2 * D_i$ would be the perfect split. The curves in A and B show the outcomes from LD and rLD for various ratios of positive data points in the fold over different class sizes. The curves in C and D show the resulting differences in LD or rLD when the class proportion changes between two curves in the upper plots. Notice that these resulting differences have a clear trend with LD in C and stable behavior with rLD in D. So rLD is clearly balanced across the class sizes and LD is not.

4. The OPTISPLIT algorithm

In this section, we present a new general algorithm, OPTISPLIT, for optimizing cross validation splits with respect to any split quality measure that can produce class specific scores. OPTISPLIT can also be used to generate standard train-test splits by generating a cross validation split of size n with fold size of the desired test set and then forming the training set from the folds $1 \ldots n-1$ and the test set from the fold n. Unlike some of the competing methods, OPTISPLIT does not need any data specific parameters that have to be adjusted for different datasets. The details of OPTISPLIT are presented formally

in Algorithm 1.

Algorithm 1: Generating stratified cross validation splits

Input : target data $D \in \{0,1\}^{n \times m}$,
split quality measure $M : (D, \text{split}) \to \mathbb{R}^m$,
number of cross validation folds k,
max number of epochs max_epochs
Output: List of cross validation splits
Function `stratified_split`(*D, M, k, max_epochs*)

```
split_0 ← make_random_split(D, k)                    // Initial random split
epoch ← 0
repeat
    max_offset ← 0
    L_0 ← M(D, split_0)                              // Calculate score vector
    for i ∈ [m] do                                   // Optimise each class
        A ← index_of_nth_max_element(L_0, max_offset)
        split_1 ← balance(split_0, A)
        L_1 ← M(D, split_1)
        if ∑ L_0 ≤ ∑ L_1 then                        // Found improvement
            max_offset ← max_offset + 1
        else
            L_0 ← L_1
            split_0 ← split_1
        end
    end
until ∑ L_0 = ∑ L_1 or epoch = max_epochs
return split_0
```

Intuitively, we start by randomly generating the initial k subsets, i.e. cross validation folds. Let M be a measure that computes the quality of the split with respect to a single class, for example, rLD or DCP. Using M, we calculate the initial score vector L_0 for all classes. Let A be the index of the class with the highest value of L_0. We balance the folds with respect to class A by moving data points from folds with excess data points to folds without enough data points so that the class distributions are balanced for the class A (Function `balance` in Algorithm 1). After processing the class A, we recalculate the score vector.

Let L_1 be the recalculated score vector. If the sum of the recalculated scores is higher than the sum of the original scores, undo the changes and move to the next worse class. Otherwise, keep the modification and continue to the next worse class. Continue this process for all classes for many epochs until either there is no improvement any more or a desired max iterations limit is reached. We only allow balancing operations that lead to direct improvement of the sum of the scores. Since balancing a class can change the distribution of other classes, it may be possible to balance later a class that is skipped in the first epoch.

The time complexity of the Algorithm 1 consists of processing m classes, for each calculating the loss with complexity $O(nm)$, finding the class with the highest loss, $O(m)$, and redistributing the

elements $O(n)$. Therefore, the total time complexity is $O\left(nm^2\right)$. In practice, OPTISPLIT could be also easily used on top of another possibly faster method to fine tune the results.

Note, that in the accompanying practical implementation, the classes that have more positive than negative data points are balanced with respect to the negative distribution. This is not important for GO data, but could be useful in some other applications.

5. Cross validation experiments

5.1. Split quality measures

We start by presenting an experimental evaluation of the behavior of LD, rLD and DCP on class imbalanced synthetic data. We verify that LD depends on the class size while rLD and DCP do not. An instance of the synthetic data consists of a binary target data matrix and a cross validation split. The synthetic data is designed so that all the classes of the target matrix have a different amount of positive data points, and all the cross validation folds have the same class distributions for each class. Since the fold distributions are the same and the only variation is in the class sizes, a measure that depends on the class size can be clearly identified with this data. In order to ensure that the results are consistent, we repeat the experiments with multiple cross validation splits that are of varying quality: 1) perfect (all class distributions are equal among folds), 2) with varying levels of error in the distributions between folds.

5.1.1. Synthetic data

The data was constructed as follows: we generated a binary target matrix $D \in \{0,1\}^{n\times q}$ with $n = 100000, q = 100$, that is, 100000 data points and 100 classes, where, in order to make the data class imbalanced, the positive class sizes of the data were set to vary from $2k$ to $n/2$, so that $|S_j| = 2k + \frac{j(n/2-2k)}{q}$, $j \in 0 \ldots q-1$. Details of the target data are presented in Table 2 under the name SYNTHETIC.

We constructed three different 10-fold cross validations splits for SYNTHETIC, namely

- *Equal*
- *Difference*
- *One missing*

In Equal all folds contained equal amount of positive data points, in Difference there was a 20% increase in one fold, 20% decrease in one fold, and the rest were equal and in One missing one fold was missing all positive data points while the other folds were equally sized. These three alternatives represent different levels of error in data splitting, and measures should be able to separate them from each other. Furthermore, all the classes of SYNTHETIC have the same fold distributions, while the absolute class sizes are different. Therefore, a good split quality measure should give a similar score for all classes, irrespective of their size. In practical settings it is especially important to correctly divide the smaller classes since the bigger classes are naturally better distributed.

5.1.2. Result

We evaluated these three splits on the synthetic data using the measures LD, rLD and DCP. The results for each measure are presented in the subplots of Figure 3. In each plot, class size is plotted

against the class specific score given by the measure (smaller is better) for the different folds. The results confirm that the widely used LD depends on the class size. LD gives smaller values to smaller classes and higher values to bigger classes, even though their fold distributions are identical. The behavior of rLD and DCP on the synthetic data show that both rLD and DCP are not affected by the class size. Therefore, it is recommendable to use one of them instead of LD for measuring cross validation split quality.

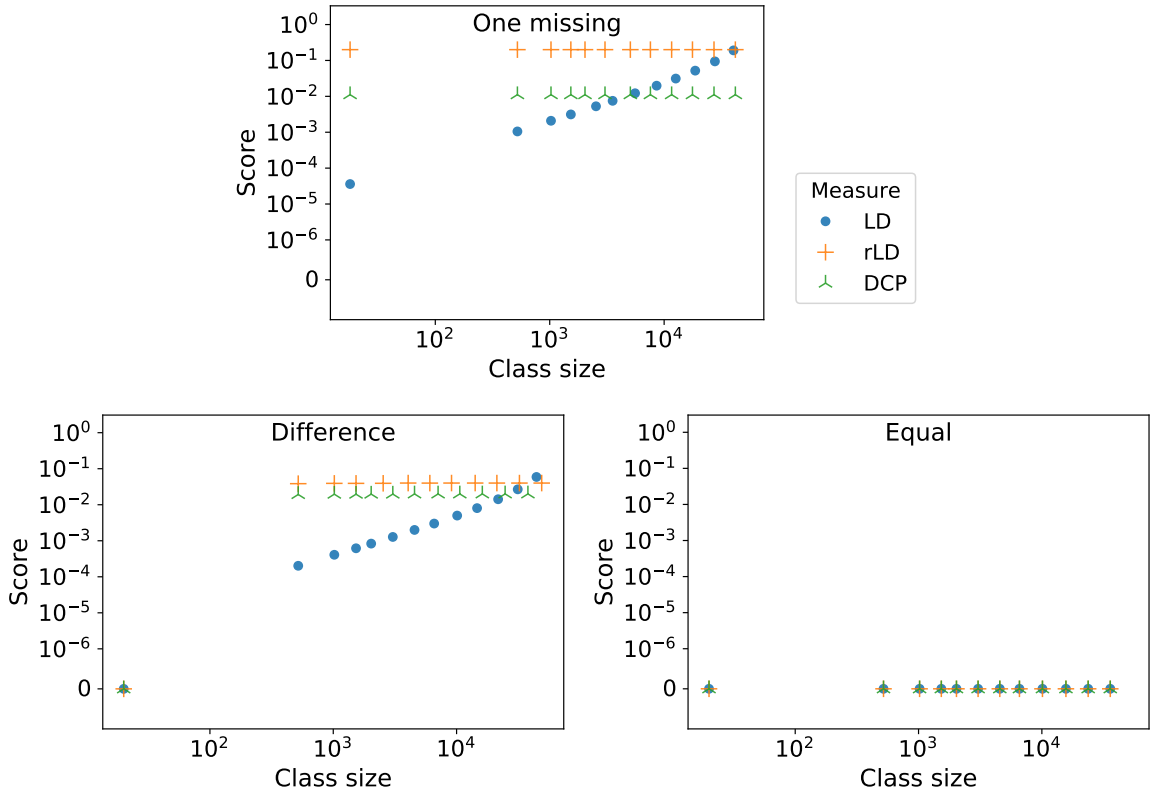

Figure 3. Behavior of cross validation split quality measures on synthetic data with cross validation splits of varying quality. The cross validation folds have the same distribution for each class, so the scores should be equal. However, LD score depends on the class size if the folds are not perfectly distributed, while rLD and DCP correctly give the same weight for all classes. This is in agreement with the results shown in Figure 2.

5.2. Comparison of algorithms

In this Section, we will examine the performance of the existing algorithms, namely, SOIS, IS, SS and PMBSRS (see Table 1) as well as our own OPTISPLIT with respect to the proposed new measures rLD and DCP.

The algorithms compared here can be divided into three categories. IS and SOIS are iterative stratification based methods, SS and OPTISPLIT are optimization based methods and PMBSRS is a random split based method. Note that the SS implementation produces train test splits, not cross validation splits. In order to compare it to the rest of the methods, we have split the data by recursively splitting it to approximately 1/k sized test sets. Thus, the method is run 5 times.

Table 1. Overview of the multilabel stratification algorithms used in the experiments. More detailed method descriptions are in main text. Cross Validation is shortened here as CV.

Name	Acronym	Description	Reference	Implementation
Iterative stratification	IS	Split data, one class at the time, starting from the smallest	[9]	[10]
Second order iterative stratification	SOIS	IS extended to second order relationships	[11]	[10]
Stratified sampling	SS	Redistribute data points based on a score	[7]	[8]
Partition method based on stratified random sampling	PMBSRS	Sample CV folds from groups of similar data points	[4]	[12]
OPTISPLIT	OPTISPLIT	Minimize the class specific global loss	New	[12]
RANDOM	RANDOM	Fully random CV split	-	[12]

We will show that if one wants to get cross validation splits that are good with respect to rLD and DCP, OPTISPLIT is the best option available since it can be used to directly optimize them. We optimized OPTISPLIT with respect to LD, rLD and DCP to compare the effect of the cost function to the outcome.

In the following experiments, we set $k = 5$. All the results presented are averages over 10 runs with different random initialization. The experiments were run using Python 3.6.8 on a machine with AMD opteron-6736 1.4GHz. Implementations of OPTISPLIT and all the experiments presented in this article are available at `https://github.com/xtixtixt/optisplit`.

5.3. Datasets

We used a wide range of diverse datasets: BIBTEX, DELICIOUS and MEDIAMILL are from the MULAN dataset collection [14] that have been used to evaluate earlier similar methods. Datasets CC (cellular component), MF (molecular function) and BP (biological process) are our own GO subset datasets used in protein function prediction (see [13, 16] for more info). These are considerably bigger and sparser than the MULAN datasets used here. The dataset WIKI10-31K [3] is a large and very sparse extreme classification dataset. Here, classes without any positive or negative data points are excluded. Detailed properties of the datasets are presented in Table 2.

5.4. Results

Scores of the split quality measures for all datasets and methods are presented in Table 3 with the following exceptions: IS and SOIS results are not presented for the biggest datasets BP and WIKI10-31K because their runtime was prohibitively high. WIKI10-31K results are not presented for SS because the implementation used produced an error when run on that particular data. For comparison purposes, we have also presented scores for random split (RANDOM).

The results show that OPTISPLIT performs better than previous methods with respect to rLD and DCP scores when optimizing with either of those. The runtimes of OPTISPLIT are also competitive when

Table 2. Properties of the datasets used in evaluations. CC, MF and BP are GO subsets. All the datasets are highly class imbalanced. Min and Max represent the minimum and maximum class sizes of the datasets. Columns 25%, 50% and 75% are the corresponding percentiles of the class sizes. Density represents the proportion of positive data points.

Data	Data size	Labels	Density	Min	25%	50%	75%	Max
BIBTEX	7395	159	0.0151	51	61	82	130	1042
DELICIOUS	16015	983	0.0193	21	58	105	258	6495
MEDIAMILL	43907	101	0.0433	31	93	312	1263	33869
CC	577424	1688	0.0077	5	66	225	891	577410
MF	637552	3452	0.0028	11	61	150	498	637533
BP	666349	11288	0.0028	4	41	123	493	666338
WIKI10-31K	20762	30938	0.0006	2	2	3	6	16756
SYNTHETIC	100000	100	0.2500	20	12512	25005	37497	50000

compared to other top performing methods. Generally, iterative stratification based methods perform quite well, but are unusable slow on bigger datasets. Random split based methods are fast but produce poor quality folds compared to more advanced methods. Optimization based methods (OPTISPLIT and SS) usually give best results and their runtimes are in the middle of iterative stratification based and random split based methods. Note that OPTISPLIT does not attempt to produce exactly equally sized splits. This results in quite high ED scores compared to some other methods. This should not be a problem in practical machine learning, especially since the relative sizes of the differences are still very small.

We can see that DCP and rLD are very correlated, the ordering of the methods is similar with respect to both measures and optimizing OPTISPLIT with respect to DCP produces nearly as good rLD results as optimizing rLD directly. However, since rLD measures the folds more thoroughly i.e. it does not just concentrate on the biggest fold it seems to be a better practical choice than DCP in most cases. For completeness, we have included LD evaluations in Table 3. As is to be expected from the results presented in Section 3, we can see that the method ordering is often considerably different with respect to LD scores. In smaller and less imbalanced datasets LD gives results more in line with rLD and DCP. For bigger and more imbalanced datasets, when LD weakness gets more pronounced, the results differ more significantly. There, LD favors iterative stratification based methods and gives random split based PMBSRS noticeably better score than to RANDOM, in contrast to rLD or DCP.

The LD scores of OPTISPLIT optimised with respect to LD are relatively good but usually not the best, especially on the GO datasets. That may be explained by noticing that the corresponding ED scores are also considerably high. It suggests that OPTISPLIT may concentrate on balancing mainly the largest classes, because LD gives those extremely high scores, and OPTISPLIT always selects the class with the maximum score to be balanced. In conclusion, OPTISPLIT seems to excel at optimising measures that are more stable and do not have extremely strong preference to particular type of classes, like the LD does.

Table 3. Performances of the algorithms evaluated on diverse datasets. The scores shown are means over 10 runs. Bold font highlights the best results for each dataset. Error marks cases where the method failed to run. Note that ED monitors only the sizes of Cross Validation splits, not their class distributions. Notice that we ran slower methods (SOIS and IS) only with the smaller datasets (absence is marked with n/a).

Dataset	Method	ED	LD	DCP	rLD	Runtime	
						Seconds	Hours
BIBTEX	OPTISPLIT$_{rLD}$	27	0.0004	0.0073	**0.0234**	5	0
	OPTISPLIT$_{DCP}$	38	0.0005	**0.0068**	0.0315	5	0
	OPTISPLIT$_{LD}$	24	**0.0003**	0.0082	0.0279	6	0
	SOIS	16	0.0005	0.0143	0.0425	5	0
	IS	17	0.0007	0.0206	0.0604	1	0
	SS	57	0.0007	0.0173	0.0465	4	0
	RANDOM	0	0.0022	0.0564	0.1693	1	0
	PMBSRS	2	0.0022	0.0558	0.1660	1	0
MEDIAMILL	OPTISPLIT$_{rLD}$	53	0.0005	0.0053	0.0187	10	0
	OPTISPLIT$_{DCP}$	7	0.0005	**0.0047**	**0.0176**	11	0
	OPTISPLIT$_{LD}$	76	0.0015	0.0227	0.0743	12	0
	SOIS	1	**0.0003**	0.0205	0.0610	77	0
	IS	1	0.0008	0.0280	0.0854	50	0
	SS	36	0.0009	0.0068	0.0231	31	0
	RANDOM	0	0.0019	0.0379	0.1142	1	0
	PMBSRS	2	0.0019	0.0386	0.1150	1	0
DELICIOUS	OPTISPLIT$_{rLD}$	35	0.0010	0.0223	0.0666	75	0
	OPTISPLIT$_{DCP}$	28	0.0010	**0.0215**	0.0772	76	0
	OPTISPLIT$_{LD}$	23	0.0008	0.0306	0.0924	80	0
	SOIS	16	0.0012	0.0458	0.1357	381	0
	IS	13	0.0013	0.0489	0.1461	11	0
	SS	75	**0.0007**	0.0221	**0.0625**	40	0
	RANDOM	0	0.0015	0.0507	0.1515	1	0
	PMBSRS	2	0.0015	0.0512	0.1525	1	0
CC	OPTISPLIT$_{rLD}$	193	8.1053	0.0065	**0.0230**	2762	0.77
	OPTISPLIT$_{DCP}$	78	8.1195	**0.0062**	0.0248	2872	0.80
	OPTISPLIT$_{LD}$	2783	7.5304	0.0317	0.0923	3006	0.84
	SOIS	5	5.7623	0.0305	0.0894	204823	56.90
	IS	1	**5.6075**	0.0448	0.1320	97120	26.98
	SS	182	8.8831	0.0133	0.0416	1118	0.31
	RANDOM	0	10.2802	0.0455	0.1342	1	0
	PMBSRS	1	6.3606	0.0448	0.1332	17	0
MF	OPTISPLIT$_{rLD}$	607	3.2147	0.0064	**0.0229**	4980	1.38
	OPTISPLIT$_{DCP}$	89	3.1782	**0.0063**	0.0252	5209	1.45
	OPTISPLIT$_{LD}$	3807	2.8137	0.0337	0.1025	5452	1.51
	SOIS	84	3.0536	0.0490	0.1450	53646	14.90
	IS	1	**2.7503**	0.0490	0.1451	20442	5.68
	SS	656	4.7935	0.0129	0.0400	1018	0.28
	RANDOM	0	6.2255	0.0493	0.1465	1	0
	PMBSRS	1	4.8213	0.0498	0.1480	17	0
BP	OPTISPLIT$_{rLD}$	612	2.1053	0.0161	**0.0516**	59436	16.51
	OPTISPLIT$_{DCP}$	173	2.0951	**0.0156**	0.0576	52412	14.56
	OPTISPLIT$_{LD}$	5845	**1.8147**	0.0521	0.1401	56636	15.73
	SOIS	n/a	n/a	n/a	n/a	n/a	n/a
	IS	n/a	n/a	n/a	n/a	n/a	n/a
	SS	279	2.9140	0.0282	0.0857	2734	0.76
	RANDOM	0	2.896	0.0568	0.1664	1	0
	PMBSRS	0	2.4250	0.0567	0.1662	21	0
WIKI10-31K	OPTISPLIT$_{rLD}$	1678	**0.0002**	0.2579	**0.7563**	3065	0.85
	OPTISPLIT$_{DCP}$	367	**0.0002**	**0.2068**	0.8033	3053	0.85
	OPTISPLIT$_{LD}$	712	**0.0002**	0.2995	0.9306	3932	1.09
	SOIS	n/a	n/a	n/a	n/a	n/a	n/a
	IS	n/a	n/a	n/a	n/a	n/a	n/a
	SS	error	error	error	error	error	error
	RANDOM	0	**0.0002**	0.3008	0.9316	1	0
	PMBSRS	2	**0.0002**	0.3013	0.9323	1	0

6. Discussion and future work

In this article, we have shown that the most widely used multilabel cross validation split quality measure, LD, does not measure split quality correctly when used on class imbalanced data. In response, we have presented new measures with better properties and have presented a new general method, OPTISPLIT, for generating and optimizing multilabel stratified cross validation splits. We have compared OPTISPLIT to existing methods and found that it produces better quality cross validation folds with respect to the new measures than the previous methods and scales well for GO sized datasets. We note for future work that OPTISPLIT could be made faster by calculating the loss only for the classes that have been modified in the previous balancing operation. In case of sparse data, that should allow it to be used even on considerably larger datasets.

During review process we became aware of a recently introduced genetic algorithm based multilabel stratified split algorithm [6] which claims good LD performance, but no open source implementation is available as of 18.3.2022. In future work it could be compared with OPTISPLIT, and in general, future work should investigate optimization based methods for optimizing rLD. For example, OPTISPLIT now uses a greedy hill-climbing approach for optimizing the target function. However, a monte carlo or simulated annealing based version could achieve even better performance.

Acknowledgments

This work was funded by NNF20OC0065157 of the Novo Nordisk Foundation. Computations were partly done using resources in Biocenter Finland's Bioinformatics platform. We thank the reviewers for their comments which improved the manuscript.

Conflict of interest

All authors declare no conflicts of interest in this paper.

References

1. M. Ashburner, C. A. Ball, J. A. Blake, D. Botstein, H. Butler, J. M. Cherry, A. P. Davis, K. Dolinski, et al., Gene ontology: tool for the unification of biology, *Nature genetics*, **25** (2000), 25–29. https://doi.org/10.1038/75556

2. S. Bengio, K. Dembczynski, T. Joachims, M. Kloft, M. Varma, Extreme Classification (Dagstuhl Seminar 18291), *Dagstuhl Reports*, **8** (2019), 62–80.

3. K. Bhatia, K. Dahiya, H. Jain, P. Kar, A. Mittal, Y. Prabhu, M. Varma, The extreme classification repository: Multi-label datasets and code, 2016.

4. F. Charte, A. Rivera, M. J. del Jesus, F. Herrera, A. Troncoso, H. Quintián, E. Corchado, On the impact of dataset complexity and sampling strategy in multilabel classifiers performance, *Hybrid Artificial Intelligent Systems*, (2016), 500–511. Springer International Publishing. https://doi.org/10.1007/978-3-319-32034-2_42

5. A. De Myttenaere, B. Golden, B. Le Grand, F. Rossi, Mean absolute percentage error for regression models, *Neurocomputing*, **192** (2016), 38–48. https://doi.org/10.1016/j.neucom.2015.12.114
6. F. Florez-Revuelta, Evosplit: An evolutionary approach to split a multi-label data set into disjoint subsets, *Applied Sciences*, **11** (2021), 2823. https://doi.org/10.3390/app11062823
7. M Merrillees, L Du, Stratified Sampling for Extreme Multi-Label Data, *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, (2021), 334–345. https://doi.org/10.1007/978-3-030-75765-6_27
8. M Merrillees, L Du, Stratified sampling for xml, 2021. Available from: `https://github.com/maxitron93/stratified_sampling_for_XML`.
9. K. Sechidis, G. Tsoumakas, I. Vlahavas, On the stratification of multi-label data, *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, (2011), 145–158. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-23808-6_10
10. P. Szymański, T. Kajdanowicz, A scikit-based Python environment for performing multi-label classification, *ArXiv e-prints*, 2017.
11. P. Szymański, T. Kajdanowicz, A network perspective on stratification of multi-label data, *Proceedings of the First International Workshop on Learning with Imbalanced Domains: Theory and Applications*, volume 74 of *Proceedings of Machine Learning Research*, (2017), 22–35.
12. H. Tiittanen, L. Holm, P. Törönen, Optisplit. Available from: `https://github.com/xtixtixt/optisplit`.
13. P. Törönen, A. Medlar, L. Holm, Pannzer2: a rapid functional annotation web server, *Nucleic acids res.*, **46** (2018), W84–W88. https://doi.org/10.1093/nar/gky350
14. G. Tsoumakas, E. Spyromitros-Xioufis, J. Vilcek, I. Vlahavas, Mulan: A java library for multi-label learning, *J. Mach. Learn. Res.*, **12** (2011), 2411–2414.
15. D. H. Wolpert, Stacked generalization, *Neural Networks*, **5** (1992), 241–259. https://doi.org/10.1016/S0893-6080(05)80023-1
16. N. Zhou, Y. Jiang, T. R. Bergquist, A. J. Lee, B. Z. Kacsoh, A. W. Crocker, K. A. Lewis, G. Georghiou, et al., The cafa challenge reports improved protein function prediction and new functional annotations for hundreds of genes through experimental screens, *Genome biol.*, **20** (2019), 1–23.

AIMS Press