Information Sciences 544 (2021) 197–213

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Synthetic Sample Generation for Label Distribution Learning

Manuel González [a,*], Julián Luengo [a], José-Ramón Cano [b], Salvador García [a]

[a] Department of Computer Science and Artificial Intelligence, University of Granada, Granada 18071, Spain
[b] Department of Computer Science, University of Jaén, EPS of Linares, Avenida de la Universidad S/N, Linares 23700, Jaén, Spain

ARTICLE INFO

Article history:
Received 2 November 2019
Received in revised form 23 July 2020
Accepted 25 July 2020
Available online 4 August 2020

Keywords:
Label Distribution Learning
Data pre-processing
Oversampling
Machine learning

ABSTRACT

Label Distribution Learning (LDL) is a general learning framework that assigns an instance to a distribution over a set of labels rather than a single label or multiple labels. Current LDL methods have proven their effectiveness in many machine learning applications. As of the first formulation of the LDL problem, numerous studies have been carried out that apply the LDL methodology to various real-life problem solving. Others have focused more specifically on the proposal of new algorithms. The purpose of this article is to start addressing the LDL problem as of the data pre-processing stage. The baseline hypothesis is that, due to the high dimensionality of existing LDL data sets, it is very likely that this data will be incomplete and/or that poor data quality will lead to poor performance once applied to the learning algorithms. In this paper, we propose an oversampling method, which creates a superset of the original dataset by creating new instances from existing ones. Then, we apply already existing algorithms to the pre-processed training set in order to validate the effcacy of our method. The effectiveness of the proposed SSG-LDL is verified on several LDL datasets, showing significant improvements to the state-of-the-art LDL methods.

1. Introduction

A supervised learning process essentially consists of assigning a label or set of labels to each of the samples. In existing learning paradigms, there are mainly two cases of label assignment: (1) a single label is assigned to an instance, and (2) multiple labels are assigned to each instance. Single-label learning (SLL) assumes that all instances in the training set are labeled according to the first case. Multi-Label Learning (MLL) [40,29] allows instances in the training set to be labeled using the second method. Thus, MLL can deal with ambiguity when an instance belongs to more than one class (label) [40].

However, in real-life problems, we can find cases for which MLL is not sufficient since the level of description of each label is not the same. The Label Distribution Learning (LDL) concept showed up for first time in 2013 [26] and was formally described in 2016 [23] in order to deal with label ambiguity in mapping when one instance is not necessarily mapped to one label. The aim of this paradigm is to answer the question "how much does each label describe the instance?" instead of "which label can describe the instance?" [23].

From the first formulation of the LDL problem, numerous studies have been carried out applying the LDL methodology to various real-life problem solving situations, e.g., sense of beauty recognition [34], facial age estimation [26], personality recognition in social media [44], image emotion classification [45], pre-release prediction of crowd opinion on movies

* Corresponding author.
E-mail addresses: manuel.gonzalez.es@gmail.com (M. González), julianlm@decsai.ugr.es (J. Luengo), jrcano@ujaen.es (J.-R. Cano), salvagl@decsai.ugr.es (S. García).

https://doi.org/10.1016/j.ins.2020.07.071
0020-0255/

[24], crowd counting in public video surveillance [49], ... Other studies have focused more particularly on the creation of new learners or the improvement of existing learners such as: AA-*k*NN, SA-IIS, SA-BFGS [23], LDL forests [36], Logistic boosting regression for LDL [43], Deep Label Distribution [20], LDL-SCL [50], Structured random forest for LDL [14].

The purpose of this article is to address the LDL problem from the data pre-processing stage [21,22]. Studying the existing LDL datasets, we observe that the number of samples they contain is significantly low as opposed to the high dimensionality of input characteristics as well as output labels. This leads us to the hypothesis that the datasets with sparse data will lead to poor performance once applied to the learning algorithms.

The pursued objective is to improve the performance of the underlying LDL learners without modifying them, by adding a preliminary pre-processing stage applied to the original dataset. It is an extra layer that can be added in a simple way to the machine learning process.

Over the last few years, many solutions have been proposed to deal with this problem. One of the most well-known and commonly used is the data sampling technique [4] in which the training instances are modified in such a way as to produce a more efficient class distribution that allow learners to perform in a similar manner to standard learning.

In the specialized literature, we can find several studies that show the effect of resampling the training set to treat incomplete datasets. Those studies have empirically proved that, applying a pre-processing step is usually a useful solution [12,35]. Furthermore, the main advantage of these techniques is that they are independent of the underlying learner.

Nowadays, researchers have proposed a few attempts to apply pre-processing in LDL, as the feature selection algorithm described in Ref. [42], but the oversampling approach has not yet been studied or applied to the LDL paradigm and that is why, in this paper, we have developed an oversampling method [33], which creates a superset of the original dataset by creating new instances from existing ones. The technique devised is based on one of the most renowned approaches in the oversampling area called: "Synthetic Minority Oversampling Technique" (SMOTE) [11]. Briefly, its main objective is to create new minority class examples to oversample the training set by interpolating several minority class instances that lie together.

The problem statement can thus be described as building a new optimized dataset, an extension of the original, which improves the performance of the underlying learners.

Synthetic Sample Generation for LDL, or from now on SSG-LDL, is our proposal for a pre-processing algorithm adapted to LDL restrictions. We will start from the basic structure of SMOTE, which we will extend to be able to deal with LDL datasets. In our case we have to take a step away from SMOTE because we do not want to deal with minority classes, rather we want to create synthetic samples of the most spatially distant points. To this end, we will also introduce a new technique to select remote points, inspired by the probabilistic selection used in genetic algorithms [27]. Another main challenge is how to generate synthetic samples from the ones available in the training set since, unlike SMOTE, we do not only have to deal with the dimension of the characteristics but we must also handle the dimension of the labels to get a synthetic sample that is compatible with LDL. It is noteworthy to mention that SSG-LDL is not a simple data augmentation such as those commonly used in deep learning to preprocess and to enlarge the set of training images, such as those used in Ref. [20].

In order to evaluate the proposal proficiency, we will compare three state-of-the-art LDL learners applying our SSG-LDL pre-processing step to the training set and measuring six aspects of their performance. We will repeat the experiment over 15 real-world datasets and validate the results of the empirical comparisons using Wilcoxon and Bayesian Sign tests.

The rest of the paper is organized as follows. First, a brief review and discussion of the related work on LDL and synthetic feature generation are given in Section 2. The proposed Synthetic Sample Generation LDL pre-processing method is described in Section 3. Then the details of the experiments are reported in Section 4. Finally, the results and conclusions are drawn in Section 5 and Section 6.

2. Preliminaries

In this section, the formulation of LDL (Section 2.1) and the LDL algorithms considered for our case study (Section 2.2) are presented. Furthermore, some basic concepts on synthetic sample generation are introduced (Section 2.3), providing the necessary background required to properly present the study carried out in this paper.

2.1. Formulation of Label Distribution Learning

Suppose that we are given a set of m samples $S = \{(x_1, D_1), \ldots, (x_m, D_m)\}$ to train, where $x_i = \{x_{i1}, x_{i2}, \ldots, x_{iq}\}$ is a q-dimensional vector. For each instance x_i, the label distribution is denoted by $D_i = \left\{d_{x_i}^{y_1}, d_{x_i}^{y_2}, \ldots, d_{x_i}^{y_c}\right\}$ where $y \in Y,\ Y = \{y_1, y_2, \ldots, y_c\}$ denotes the complete set of labels. The constant c is the number of possible labels and $d_{x_i}^{y_j}$ is the description degree of the particular jth label y_j to the particular ith instance x_i. According to the definition, each description degree should satisfy the constraints $d_x^y \in [0, 1]$ and $\sum_y d_x^y = 1$.

2.2. Label Distribution Learning algorithms

The resolution of an LDL problem can be approached from different points of view. Depending on the approach chosen, the algorithm to be developed will vary substantially, being either a completely new algorithm developed specifically to deal

with LDL constraints, or an adaptation of existing classification algorithms, reformulated to work with such constraints. In all cases, the algorithms chosen for our study are "state of the art" algorithms.

The first one is the AA-*k*NN adaptation described in [23], given a new instance x, its k nearest neighbors are first found in the training set. Then, the mean of the label distributions of all the k nearest neighbors is calculated as the label distribution of x. We have chosen this algorithm due to its simplicity and the ease with which it can be enhanced, it will provide a good starting point for a comparative analysis with other more complex methods.

The second is the specialized algorithm SA-BFGS that learns by optimizing an energy function based on the maximum entropy model using a quasi-Newton method. Of the six LDL algorithms proposed by Geng [23], the SA-BFGS stood out due to its great advantage in precision and efficiency.

Another state-of-the-art specialized algorithm is called Structured Random Forest (StructRF) [14]. It is a general LDL model that treats the distribution as an integral whole. In StructRF, all label distributions are mapped to a discrete space at each split node in a random forest. StructRF has proven to be able to train fast and it reaches higher accuracies and lower standard deviations among different measurements.

2.3. Motivation for using oversampling technology

Nowadays, one of the main challenges for supervised learning algorithms is still how to deal with datasets that are either incomplete or have significantly skewed class distributions. Both problems can be addressed using an initial data preprocessing that allows us to improve the quality obtained by the learning algorithm used.

The synthetic generation of samples allows us to approach this problem by generating new artificial samples from the originals, thus creating an expanded set of data that we can plug in as input to our learning algorithm.

An example of a widely used technique that achieves this task is the Synthetic Minority Oversampling Technique now widely known as SMOTE [11]. This preprocessing technique carries out an oversampling technique to rebalance the original training set. The basic concept of the SMOTE procedure is to accomplish an interpolation using neighboring minority class samples to improve the generalization capacity of the classifier. The SMOTE preprocessing technique has become a pioneer in the research community and has been widely applied in different scenarios such as face recognition, social media, software engineering, medical diagnosis or social networks. Due to its popularity and influence, SMOTE is considered to be one of the most influential data preprocessing/sampling algorithms in machine learning and data mining [22,19].

As a result, the synthetic generation of samples has been used successfully in multiple areas of study such as:

- Dynamic environments in which data arrives continuously, or data stream. An example of a pre-processing technique to deal with these data streams is Learn++.NSE-SMOTE [16].
- When a data stream is received over time and we have time information at our disposal, we refer to time series classification. The methods SPO and INOS [7] propose an integration of SMOTE in time series classification.
- In a perfect situation, we want to train classifiers using diverse labeled data that thoroughly represent all classes. However, in many real-life applications, there is a huge amount of unlabeled data and thus obtaining a representative subset is a complex process. We refer to semi-supervised classification that utilizes unlabeled data to improve the predictive performance. Several methods based on SMOTE have been developed for this learning paradigm, GS4 [32], SEG-SSC [38] and OCHS-SSC [17]. These methods generate synthetic examples to diminish the drawbacks produced by the absence of labeled examples.
- Several ideas, also based on SMOTE, have been proposed to tackle multi-instance learning, e.g. Instance-SMOTE and Bag-SMOTE [41]. The Instance-SMOTE algorithm creates synthetic minority instances in each bag, without creating new bags. Besides, Bag-SMOTE creates new synthetic minority bags with new instances.
- In multilabel classification [29] each data instance is associated with a vector of outputs, instead of only one value. MLSMOTE [10] is the most popular extension of SMOTE designed for multilabel classification. Its objective is to produce synthetic instances related to minority labels.
- Regression tasks consider the output variable as continuous and hence, the values are represented by real numbers. SMOTER is the SMOTE-based contribution of oversampling regression [37].

2.4. Principles of the proposed method

Up until now, all existing LDL studies and algorithms have dealt with raw datasets [23,36,43,20,50,14] and have not applied any kind of pre-processing that might later facilitate the learning of the algorithm. The pursued objective of this study is to improve the performance of the underlying LDL learners without modifying them. To achieve this goal we have created an oversampling method adapted to the LDL restrictions in order to extend the initial datasets by applying synthetic sample generation and use them as input for the state-of-the-art LDL algorithms. Later we will analyze the results and see if they can improve the results obtained by learners. The mechanism of the proposed method can be depicted as in Fig. 1.

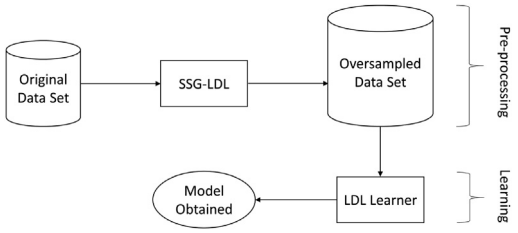


Fig. 1. Principle of the proposed method.

3. Synthetic Sample Generation for Label Distribution Learning

SSG-LDL is the proposed pre-processing algorithm that has been adapted to LDL restrictions.

The main idea of this method is to obtain an optimized dataset that will be made up of the original samples plus the artificially generated ones. We define a synthetic sample as a sample generated based on a set of real samples, on which we apply interpolation techniques between remote points that are within a defined neighborhood. The new samples obtained using this procedure are thus added to the raw dataset. Throughout the following subsections we will explain the entire process in detail, covering from how to select the candidate samples to how to obtain a new synthetic sample specially designed to meet the constraints of an LDL problem.

The formal procedure works as follows. First, the total percentage of oversampling N (an integer value) is set up, then, an iterative process is carried out, composed of several steps. In the initial step, a remote sample is selected at random from the training set (this process is described in Section 3.2). Next, its k nearest neighbors are obtained. Finally, one of these k instances is randomly chosen to compute the new sample by interpolation. This synthetic sample generation process is described in detail in Section 3.3 where we will explain how to interpolate the features vector and also how the label dimension is created using the k nearest neighbor label vectors. This process is repeated iteratively until $N \times m/100$ synthetic samples are obtained. The whole process is summarized in Algorithm 1.

Algorithm 1. SSG-LDL

Function SSG-LDL($S[]$, N, k)**:**

input : $S[] \leftarrow$ original training set ;
$N \leftarrow$ % of oversampling ;
$k \leftarrow$ considered neighbors ;
output: $S'[] \leftarrow$ the oversampled training set
$S' = S$;
$T = m * N/100 \leftarrow$ nº synthetic samples to create;
for i=1 *to* T **do**
  j = SelectSample(S) ;
  ss = CreateSyntheticSample(S, j, k) ;
  $S' \leftarrow ss$;
  $i = i + 1$;
end
End Function

3.1. Distances

As previously mentioned, the first step of the algorithm consists in selecting a random sample located far away from the others. To measure the distance between two samples we have chosen the Euclidean distance, adapted to LDL, that we will

apply not only to the dimension of the features (x) but also to the dimension of the labels (D). The two independent parameters f_x and f_y allow us to have two degrees of freedom when calculating the distance between two samples. Keeping this in mind is of interest if a wider space must be covered when creating synthetic LDL samples. Therefore, the final distance between two samples is the sum of the feature's distance multiplied by the factor f_x and the label's distance multiplied by the factor f_y. What we calculate at this step, and what will later help us to select remote samples, is the average distance between a sample and all the others as summarized in Eq. 1.

$$\overline{DIST}[x_i] = f_x \frac{\sum_{l=1}^{m} \text{euclidean}(x_i, x_l)}{m} + f_y \frac{\sum_{l=1}^{m} \text{euclidean}(D_i, D_l)}{m}. \tag{1}$$

3.2. Divergence cumulative selection

A part of the SSG-LDL algorithm is based on choosing isolated samples from which synthetic samples are generated in order to fill in the data set in less populated areas. To do so we will rely on the distances calculated in the previous section, adding some randomness to the process. The implemented method is inspired by the selection mechanism called "roulette wheel selection", widely used in genetic algorithms [27].

In divergence cumulative selection, the probability of being selected will increase as the distance increases. This has been achieved by dividing the mean distance of a sample to other samples by the total distance, thereby normalizing them to 1 (cumulative probability vector). Then a random selection is made, similarly to how a roulette wheel is spun. Therefore, the probability P that sample x_i will be selected is defined according to Eq. (2).

$$P_{x_i} = \frac{\overline{DIST}[x_i]}{\sum_{j=1}^{m} \overline{DIST}[x_j]}. \tag{2}$$

The whole process is summarized in Algorithm 2. The algorithm generates a random number between 0 and the sum of the frequencies. Then, it goes through the array of frequencies, producing a running total. At some point the running total exceeds the generated threshold. The index at that point corresponds to the selected instance.

Algorithm 2. Divergence Cumulative Selection Algorithm

```
Function SelectSample(S[]):
1   input  : S[] ← original training set
2   output: i ← the selected sample index
    #Calculate the total distance
3   total_dist = 0 ;
4   for i=1 to m do
5       total_dist = total_dist + DIST[i] ;
    end
    #Return a random position according to the cumulative
     probabilities vector
6   r = random(0, 1) * total_dist ;
7   for i=1 to m do
8       r = r − DIST[i] ;
9       if r < 0 then
10          return i ;
11
        end
12
    end
End Function
```

3.3. Synthetic features generation

Remember that in the formulation of an LDL problem we have, on the one hand, a vector of features that quantifies the sample and, on the other hand, a vector of labels that qualifies the sample. It is an important constraint to take into account

when generating a synthetic sample as we will have to create both dimensions. The algorithm we have developed here is therefore divided as follows:

To generate the synthetic vector of characteristics we randomly choose one of the closest neighbors (the definition of close is made by taking into account the distance vector calculated in Section 3.1). From these two samples a new one is interpolated that will be located in an intermediate point between the two. The calculation of this intermediate point also introduces a randomness factor in order to carry out the most uniform possible sampling of the data space. Fig. 2 could be used to help understand this process.

Regarding the label synthetic creation, the value assigned to each position of the label is the mean of the k nearest neighbors.

The formal procedure works as follows. First, the k nearest neighbors of the sample selected by the selection algorithm are calculated and their indexes are stored. The second step is to create the different elements of the feature array, and to do so we randomly choose one of the neighbors obtained in the previous step and calculate a random intermediate value between the sample and the selected neighbor. This process is repeated iteratively for each of the q features of the sample. Finally, the vector of labels is generated by taking into account all the nearest k neighbors and obtaining the value for the new sample as the arithmetic mean of each label vector. The whole process is summarized in Algorithm 3.

Algorithm 3. Create a Synthetic Sample

```
Function CreateSyntheticSample(S[], i, k):
    input  : S[] ← original training set
             i ← selected sample index
             k ← nearest neighbors to be considered
    output: ss ← the synthetic sample created
    #Step 1 - Compute k nearest neighbors for i, and save the indexes
     in the nnarray
    #Step 2 - Generate the synthetic features
    nn = random(1, k);
    for attr = 1 to q do
        Compute: dif = S[nnarray[nn]].x_attr − S[i].x_attr;
        Compute: gap = random(0, 1);
        ss.x_attr = S[i].x_attr + gap × dif;
    end
    #Step 3 - Generate the synthetic labels
    for lbl = 1 to c do
        for nn = 1 to k do
            ss.d_lbl += S[nnarray[nn]].d_lbl;
        end
        ss.d_lbl /= k;
    end
    return ss;
End Function
```

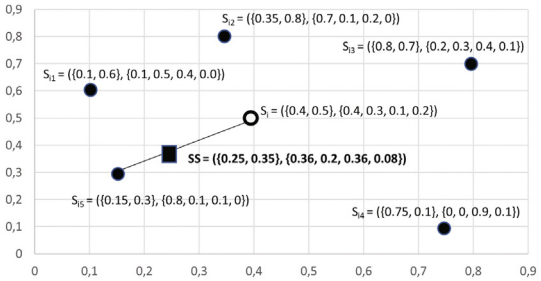


Fig. 2. An illustration of how to create the synthetic data points in the SSG-LDL algorithm.

A simple example of SSG-LDL is illustrated in Fig. 2. Here, an S_i remote sample is selected as the basis for the creation of a new synthetic data point. Based on a distance metric and using a value of $k = 5$, five nearest neighbors (samples S_{i1} to S_{i5}) are chosen from the training set. Finally, the synthetic sample generation is carried out in order to obtain a new synthetic sample *SS*. Specifically in this example the value obtained for the first characteristic of the synthetic sample is equal to $0.4 + (0.15 - 0.4) \times random(0, 1)$. For a random value of 0.6 we get a result of 0.25. The rest of the positions of the characteristic vector are calculated in the same way. As for the label vector, the synthetic value obtained corresponds to the average of the values of the 5 neighbors. In our case, the value of the first label is equal to $(0.1 + 0.7 + 0.2 + 0.0 + 0.8)/5 = 0.36$. The other positions of the label vector are calculated in the same way.

4. Experimental framework

This section is devoted to introducing the experimental framework used in the different empirical studies of the paper. In our experiments, we have included 15 datasets of a good variety of real-world problems. These are referred to in the following Section 4.1.

In order to evaluate the learners' proficiency, we have employed six measures (described in Section 4.2) for the different aspects of their performance.

For each dataset and learner, these measures were computed over a set merged from the test predictions of a 10-fold cross validation set (10-fcv). In the first pre-processing step we apply our synthetic sample generation algorithm described in Section 3 to each training set and then we use them to train the learners.

Finally, the Wilcoxon test and Bayesian Sign test [15,5] are used to validate the results of the empirical comparisons. In the Bayesian Sign test, a distribution of the differences of the results achieved using methods *L* (original learner without applying SSG-LDL) and *R* (learner using the SSG-LDL pre-processing method) is computed into a graphical space divided into 3 regions: left, rope and right. The location of most of the distribution in these sectors indicates the final decision: the superiority of algorithm *L*, statistical equivalence and the superiority of algorithm *R*, respectively. The KEEL package [39] has been used to compute the Wilcoxon test and the R package rNPBST [8] was used to extract the graphical representations of the Bayesian Sign tests analyzed in the following empirical studies. The Rope limit parameter used to represent the Bayesian Sign test is 0.0001.

4.1. Datasets

There are 15 real-world datasets employed in the experiments in total. This is a compilation of the datasets most used in the referenced LDL articles [23,24,36,43,50,14]. The statistics are shown in Table 1.

The first to the tenth datasets were collected from ten biological experiments on the budding yeast Saccharomyces cerevisiae [18]. Each dataset includes 2465 yeast genes, and an associated phylogenetic profile vector with a length of 24 is utilized to represent each gene. In a biological experiment, the gene expression level is usually disparate at each discrete time point, so the labels correspond to the time point. Data from different time courses of gene expression in yeast S. cerevisiae were combined and clustered. The data were obtained from the time courses during the following processes: the cell division cycle after synchronization by alpha factor arrest (Yeast_alpha, 18 time points); centrifugal elutriation (Yeast_elu, 14 time points), and with a temperature-sensitive cdc15 mutant (Yeast_cdc, 15 time points); sporulation (Yeast_spo/spo5/spoem, 7 time points plus four additional samples); shock by high temperature (Yeast_heat, 6 time points); reducing agents (Yeast_dtt, 4 time points) and low temperature (Yeast_cold, 4 time points); and the diauxic shift (Yeast_diau, 7 time points). All data were gathered using DNA microarrays with elements that represented almost all the ORFs (Open Reading Frame) of the fully sequenced S. cerevisiae genome; all measurements were made against a reference sample of time 0 except for the

Table 1
Datasets used in experiments.

No.	Datasets	Examples (n)	Features (q)	Labels (l)
1	Yeast_alpha	2465	24	18
2	Yeast_cdc	2465	24	15
3	Yeast_cold	2465	24	4
4	Yeast_diau	2465	24	7
5	Yeast_dtt	2465	24	4
6	Yeast_elu	2465	24	14
7	Yeast_heat	2465	24	6
8	Yeast_spo	2465	24	6
9	Yeast_spo5	2465	24	3
10	Yeast_spoem	2465	24	2
11	SJAFFE	213	243	6
12	SBU_3DFE	2500	243	6
13	Movie	7755	1869	5
14	Natural_Scene	2000	294	9
15	Human_Gene	30542	36	68

cell-cycle experiments, in which an unsynchronized sample was used. The contribution of each sample to the gene similarity score from a given process was weighted by the inverse of the square root of the number of samples analyzed from that process.

Datasets JAFFE [31] and BU_3DFE [47] are two widely used facial expression image datasets. In JAFFE, ten expressors posed 3 or 4 examples of each of the six basic facial expressions (fear, disgust, happiness, anger, sadness, surprise) and a neutral face for a total of 213 images of facial expressions. For the simplicity of the experimental design, only Japanese women participated. Each woman took pictures of herself while looking through a semi-reflective plastic sheet into the camera. Tungsten lights were placed to create a uniform illumination of the face. The images were then printed in monochrome and digitized using a flatbed scanner. Finally, images were labelled by 92 people with a 5 or 6 component vector with ratings averaged over all subjects. The similarities between these semantic vectors were computed using Euclidean distance. There were 100 subjects who participated in BU_3DFE face scans. The resulting database consists of 60% women and 40% men with a variety of ethnic/racial backgrounds. Each subject performed the same basic facial expressions as in JAFFE. With the exception of the neutral expression, each of the six prototypic expressions includes four levels of intensity. Therefore, there are 25 instant 3D expression models for each subject, resulting in a total of 2500 3D facial expression models in the database. Each of these images were subsequently scored by 23 people using the same scale as used in JAFFE. Each dataset is represented by a 243-dimensional feature vector extracted using the Local Binary Patterns method (LBP) [3]. The score for each emotion is regarded as the description degree, and the description degrees (normalized gene expression level) of all the six emotions constitute a label distribution for a particular facial expression image.

Dataset Movie includes 7755 movies. There is a total of 54,243,292 ratings from 478,656 different users on a scale from 1 to 5 integral stars from ®Netflix. The percentage of each rating level is regarded as the label distribution. There are numeric and categorical attributes in the dataset such as genre, director, country, year, budget and so on. After transforming the categorical attributes into binary vectors, the final feature vector of each movie is 1,869-dimensional.

The Natural Scene dataset [6] is collected from 2,000 natural scene images. Each image is divided into 49 blocks using a 7×7 grid, then the first and second moments (mean and variance) of each band are computed, corresponding to a low-resolution image and to computationally inexpensive texture features, respectively. The end result is a $49\times2\times3$ = 294-dimension feature vector per image. The images were later labeled by three human observers with multiple labels selected from 9 possible values, i.e., sun, sky, water, cloud, mountain, snow, desert, building, and plant. Rankers were required to rank the relevant labels in descending order of relevance and, as expected, the rankings for the same image from different rankers are highly inconsistent. So, a nonlinear programming process [25] is applied to achieve the label distribution.

The Human Gene dataset is much larger than the other datasets used in this experiment. This dataset is collected from biological research on the relationship between human genes and diseases. Each of the 30,542 human genes is represented by the 36 numerical descriptors for a gene sequence proposed in [48]. This 36-D vector was deduced using a modified method based on I-TN curve to display the specific features of protein-coding genes in Aeropyrum pernix K1 genome. There are 68 different disease labels in total, and the normalized gene expression level for each disease is considered to be the description degree of the corresponding disease label. The gene expression level of different diseases provides a measure of the description degree of the label for every human gene.

4.2. Evaluation measure selection

We propose using a set of six measures when comparing different LDL algorithms: Chebyshev Distance, Clark Distance, Canberra Metric, Kullback–Leibler Divergence, Cosine Coefficient and Intersection Similarity as shown in Table 2. Each of these measures come from a different syntactic family summarized in [9] and are relatively widely used in the related areas. Thus, they can represent the different aspects of the algorithms well. Another reason that has led us to choose these mea-

Table 2
Evaluation measure for LDL learners. ↓ *means that the lowest value is the best and* ↑ *means the opposite.*

Name	Formula
Chebyshev (Cheby)↓	$Dis\left(D,\widehat{D}\right)=max_j\lvert d_j-\hat{d}_j\rvert$
Clark↓	$Dis\left(D,\widehat{D}\right)=\sqrt{\sum_{j=1}^{c}\frac{\left(d_j-\hat{d}_j\right)^2}{\left(d_j+\hat{d}_j\right)^2}}$
Canberra (Can)↓	$Dis\left(D,\widehat{D}\right)=\sum_{j=1}^{c}\frac{\lvert d_j-\hat{d}_j\rvert^2}{d_j+\hat{d}_j}$
Kullback–Leibler (KL)↓	$Dis\left(D,\widehat{D}\right)=\sum_{j=1}^{c}d_j\ln\frac{d_j}{\hat{d}_j}$
Cosine (Cos)↑	$Sim\left(D,\widehat{D}\right)=\frac{\sum_{j=1}^{c}d_j\hat{d}_j}{\sqrt{\sum_{j=1}^{c}d_j^2}\sqrt{\sum_{j=1}^{c}\hat{d}_j^2}}$
Intersection (Inter)↑	$Sim\left(D,\widehat{D}\right)=\sum_{j=1}^{c}min\left(d_j,\hat{d}_j\right)$

sures over others is that they are the most widely used in the referenced LDL studies [23,45,24,36,43,50,14], thus allowing us to make a more accurate comparison.

4.3. Parameters

Fig. 3 shows the influence of the different values of the parameter N on the results and the percentage of oversampling applied to the initial dataset. We can observe that when the oversampling percentage is over 300%, the value obtained by the learning algorithm gets stuck. This is the value selected for our synthetic sample generation algorithm in all experiments. Increasing this parameter to over 300% would negatively affect performance without achieving any significant improvements. The figure above represents how the Kullback–Leibler divergence on the *yeast_spoem* dataset varies; these variations are similar for all other measures and datasets. Other parameter values in synthetic sample generation are: $k = 5$, where k is the number of neighbors considered. With regard to the calculation of distances between points, we have taken 50% of the distance between features and 50% of the distance between labels into account.

There were 4 neighbors selected for the k-NN algorithm. For the BFGS algorithm, the convergence criterion ϵ has been setup to 10^{-5}. Regarding the parameters used for the StructRF algorithm, we have respected the same values as those used in the original proposal papers: numbers of trees = 50, sampling ratio = 0.8, max. depth = 20, min. size of leaf = 5.

An overview of all these parameters can be found in Table 3.

The choice of parameters has been made according to the standards and recommendations given by the authors in the original proposal papers. Due to the fact that the experimental evaluation comprises a high number of datasets, having to adjust each parameter individually for each dataset would be unreasonable. In fact, our idea is just the opposite; we aim to compare the datasets in the most general scenario possible. We have not performed any tuning to adapt these parameters,

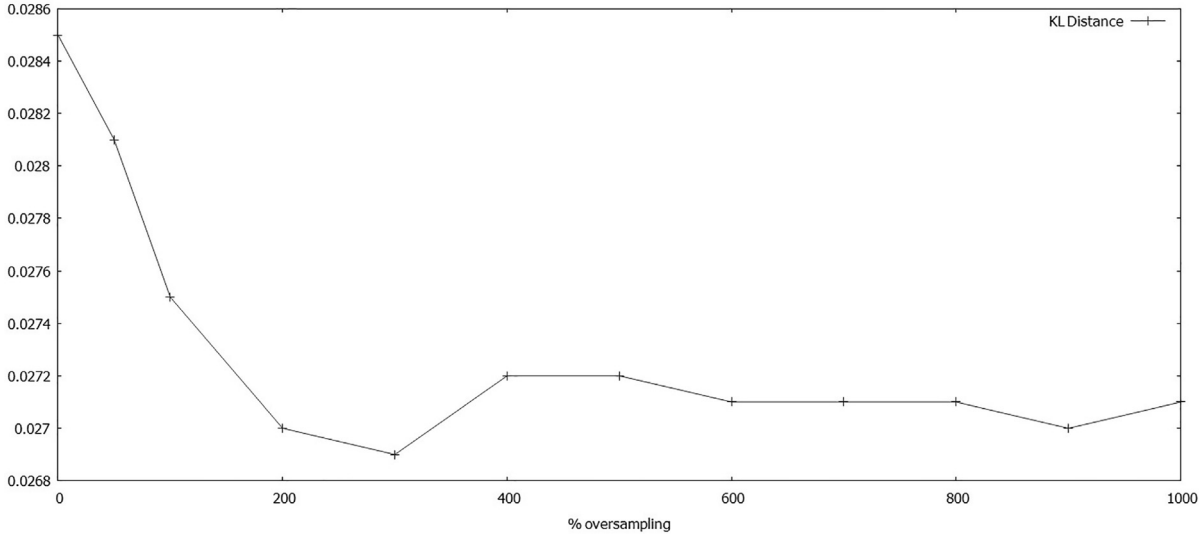


Fig. 3. Influence of experiment parameters on the results using the yeast_spoem dataset.

Table 3
Summary of the parameters.

Algorithm	Parameter	Description	Value
k-NN	k	Number of selected neighbors	4
BFGS	ϵ	Convergence criterion: must be less than ϵ before successful termination	10^{-5}
StructRF	trees	Number of trees	50
	sampling	Sampling ratio of data	0.8
	max. depth	Maximum depth of the tree	20
	min. leaf	Minimum size of the leaf	5
SSG-LDL	N	% of oversampling	300
	k	Number of selected neighbors	5
	f_x	Feature distance factor	0.5
	f_y	Label distance factor	0.5

because our objective is not to maximize the accuracy or any other performance measure, but to fairly compare the algorithms and their robustness in a common environment on different datasets.

Table 4
Experimental Results (mean ± std) measured by Chebyshev Distance ↓.

Dataset	*k*-NN	SSG-LDL+*k*-NN	BFGS	SSG-LDL+BFGS	StructRF	SSG-LDL+StructRF
Yeast_alpha	0.0148 ± 0.0007	**0.0144 ± 0.0007**	**0.0135 ± 0.0008**	**0.0135 ± 0.0008**	**0.0134 ± 0.0008**	**0.0134 ± 0.0008**
Yeast_cdc	0.0177 ± 0.001	**0.0172 ± 0.001**	0.0163 ± 0.0009	**0.0162 ± 0.0009**	0.0162 ± 0.0009	**0.0161 ± 0.0009**
Yeast_cold	0.0554 ± 0.0021	**0.0542 ± 0.002**	0.0512 ± 0.0018	**0.051 ± 0.0019**	0.0501 ± 0.0016	**0.05 ± 0.0016**
Yeast_diau	0.0393 ± 0.0009	**0.0381 ± 0.0012**	**0.037 ± 0.0014**	**0.037 ± 0.0015**	**0.0359 ± 0.0015**	0.036 ± 0.0016
Yeast_dtt	0.0393 ± 0.0016	**0.038 ± 0.0016**	0.0361 ± 0.0013	**0.0359 ± 0.0013**	0.0353 ± 0.0013	**0.035 ± 0.0012**
Yeast_elu	0.0177 ± 0.0005	**0.0172 ± 0.0005**	**0.0163 ± 0.0006**	**0.0163 ± 0.0006**	**0.0161 ± 0.0005**	**0.0161 ± 0.0006**
Yeast_heat	0.0451 ± 0.0012	**0.0438 ± 0.001**	0.0423 ± 0.0008	**0.0422 ± 0.0008**	0.0407 ± 0.0009	**0.0406 ± 0.0009**
Yeast_spo	0.0643 ± 0.0024	**0.0621 ± 0.0024**	0.0584 ± 0.0037	**0.0582 ± 0.0036**	0.0578 ± 0.003	**0.0574 ± 0.0031**
Yeast_spo5	0.0962 ± 0.0043	**0.0949 ± 0.0043**	0.0914 ± 0.005	**0.0913 ± 0.0051**	**0.0874 ± 0.0046**	0.0882 ± 0.0044
Yeast_spoem	0.0924 ± 0.0036	**0.0905 ± 0.0043**	**0.0868 ± 0.0049**	0.0869 ± 0.0052	**0.0836 ± 0.0045**	0.0841 ± 0.0037
SJAFFE	0.1155 ± 0.018	**0.1153 ± 0.0183**	0.1603 ± 0.016	**0.1142 ± 0.0147**	**0.1094 ± 0.0108**	0.1141 ± 0.0136
SBU_3DFE	0.135 ± 0.0048	**0.1327 ± 0.0052**	**0.11 ± 0.0039**	0.1231 ± 0.0048	**0.1183 ± 0.0058**	0.1225 ± 0.0053
Movie	**0.1284 ± 0.0066**	0.1317 ± 0.0081	0.1398 ± 0.0161	**0.1276 ± 0.0095**	**0.1104 ± 0.0048**	0.1195 ± 0.0071
Natural_Scene	**0.3168 ± 0.0081**	0.318 ± 0.0084	0.3549 ± 0.0159	**0.3217 ± 0.0142**	**0.2738 ± 0.0116**	0.2863 ± 0.0117
Human_Gene	0.0652 ± 0.0048	**0.0618 ± 0.0042**	**0.0533 ± 0.0036**	**0.0533 ± 0.0036**	0.0553 ± 0.0048	**0.0538 ± 0.004**
Average	0.0829 ± 0.0040	**0.0820 ± 0.0042**	0.0845 ± 0.0051	**0.0792 ± 0.0046**	**0.0736 ± 0.0038**	0.0755 ± 0.0040

Table 5
Experimental Results (mean ± std) measured by Clark Distance ↓.

Dataset	*k*-NN	SSG-LDL+*k*-NN	BFGS	SSG-LDL+BFGS	StructRF	SSG-LDL+StructRF
Yeast_alpha	0.2321 ± 0.0112	**0.2256 ± 0.0119**	0.2107 ± 0.0126	**0.2103 ± 0.0126**	0.2095 ± 0.0125	**0.2094 ± 0.0125**
Yeast_cdc	0.2375 ± 0.0134	**0.2314 ± 0.0139**	0.2165 ± 0.0129	**0.2161 ± 0.013**	0.2158 ± 0.0134	**0.2152 ± 0.0132**
Yeast_cold	0.1509 ± 0.007	**0.1475 ± 0.0066**	0.1398 ± 0.0055	**0.1394 ± 0.0058**	0.1367 ± 0.0053	**0.1366 ± 0.0054**
Yeast_diau	0.2122 ± 0.0042	**0.2064 ± 0.006**	0.2008 ± 0.0079	**0.2007 ± 0.0082**	**0.1947 ± 0.0076**	0.195 ± 0.0078
Yeast_dtt	0.1068 ± 0.0045	**0.1035 ± 0.0045**	0.0984 ± 0.0039	**0.0979 ± 0.0038**	0.0961 ± 0.0034	**0.0956 ± 0.0035**
Yeast_elu	0.2182 ± 0.0048	**0.2118 ± 0.0053**	0.1992 ± 0.0055	**0.1988 ± 0.0056**	0.1974 ± 0.0054	**0.1969 ± 0.0057**
Yeast_heat	0.1955 ± 0.0044	**0.1901 ± 0.0034**	0.1828 ± 0.0027	**0.1824 ± 0.0028**	0.177 ± 0.0028	**0.1766 ± 0.0028**
Yeast_spo	0.2715 ± 0.0112	**0.2633 ± 0.0112**	0.25 ± 0.0164	**0.2494 ± 0.0159**	0.2467 ± 0.0139	**0.2458 ± 0.014**
Yeast_spo5	0.1933 ± 0.0099	**0.1906 ± 0.0099**	0.1842 ± 0.0112	**0.1839 ± 0.0113**	**0.1767 ± 0.0104**	0.178 ± 0.0096
Yeast_spoem	0.1374 ± 0.0053	**0.1345 ± 0.0063**	**0.1293 ± 0.0072**	**0.1293 ± 0.0077**	**0.1247 ± 0.0066**	0.1254 ± 0.0054
SJAFFE	0.4137 ± 0.0489	**0.4119 ± 0.0502**	0.6466 ± 0.0462	**0.4254 ± 0.0427**	**0.39 ± 0.035**	0.405 ± 0.0421
SBU_3DFE	0.4255 ± 0.0134	**0.4166 ± 0.0139**	**0.3784 ± 0.0122**	0.3852 ± 0.0144	**0.368 ± 0.0173**	0.3786 ± 0.0167
Movie	**0.5686 ± 0.0393**	0.5849 ± 0.0466	0.6035 ± 0.056	**0.5675 ± 0.0405**	**0.5039 ± 0.0233**	0.5467 ± 0.0371
Natural_Scene	**1.8253 ± 0.0343**	1.9844 ± 0.0382	**2.3817 ± 0.0239**	2.4503 ± 0.0203	**2.3946 ± 0.0227**	2.4306 ± 0.0239
Human_Gene	2.3872 ± 0.1012	**2.3171 ± 0.0845**	2.1111 ± 0.082	**2.1109 ± 0.0794**	2.1776 ± 0.1232	**2.1291 ± 0.0905**
Average	**0.505 ± 0.0209**	0.508 ± 0.0208	0.5289 ± 0.0204	**0.5165 ± 0.0189**	**0.5073 ± 0.0202**	0.511 ± 0.0193

Table 6
Experimental Results (mean ± std) measured by Canberra Metric ↓.

Dataset	*k*-NN	SSG-LDL+*k*-NN	BFGS	SSG-LDL+BFGS	StructRF	SSG-LDL+StructRF
Yeast_alpha	0.7577 ± 0.0372	**0.7343 ± 0.0404**	0.6847 ± 0.0432	**0.6831 ± 0.043**	0.6812 ± 0.0434	**0.6801 ± 0.0428**
Yeast_cdc	0.7179 ± 0.0395	**0.6981 ± 0.04**	0.6493 ± 0.0394	**0.6479 ± 0.0401**	0.6472 ± 0.041	**0.6448 ± 0.0401**
Yeast_cold	0.2611 ± 0.0129	**0.2552 ± 0.0118**	0.2407 ± 0.0097	**0.2402 ± 0.0099**	0.2359 ± 0.0092	**0.2356 ± 0.0097**
Yeast_diau	0.456 ± 0.0109	**0.4443 ± 0.014**	**0.431 ± 0.0186**	0.4313 ± 0.0195	**0.4177 ± 0.0178**	0.4184 ± 0.0187
Yeast_dtt	0.1836 ± 0.0075	**0.1779 ± 0.007**	0.1693 ± 0.0061	**0.1685 ± 0.0058**	0.1649 ± 0.005	**0.1643 ± 0.0052**
Yeast_elu	0.6444 ± 0.0155	**0.6242 ± 0.0164**	0.5838 ± 0.0159	**0.5824 ± 0.0162**	0.578 ± 0.0156	**0.5769 ± 0.0159**
Yeast_heat	0.3924 ± 0.0096	**0.3814 ± 0.0076**	0.3647 ± 0.0058	**0.3637 ± 0.0059**	0.3541 ± 0.0063	**0.3528 ± 0.006**
Yeast_spo	0.5594 ± 0.0232	**0.543 ± 0.0234**	0.5137 ± 0.0335	**0.513 ± 0.0321**	0.5066 ± 0.0274	**0.5046 ± 0.0275**
Yeast_spo5	0.2972 ± 0.0145	**0.2934 ± 0.0143**	0.2829 ± 0.0163	**0.2826 ± 0.0165**	**0.2713 ± 0.0151**	0.2735 ± 0.0143
Yeast_spoem	0.1913 ± 0.0074	**0.1872 ± 0.0088**	**0.1798 ± 0.01**	0.1799 ± 0.0107	**0.1733 ± 0.0092**	0.1744 ± 0.0076
SJAFFE	0.8527 ± 0.0962	**0.8471 ± 0.0998**	1.3595 ± 0.11	**0.8783 ± 0.0992**	**0.8089 ± 0.0769**	0.84 ± 0.0938
SBU_3DFE	0.8746 ± 0.029	**0.8611 ± 0.0302**	**0.7831 ± 0.0246**	0.8265 ± 0.0338	**0.7817 ± 0.0354**	0.8087 ± 0.0361
Movie	**1.0946 ± 0.0766**	1.1255 ± 0.0927	1.1679 ± 0.1163	**1.0905 ± 0.0833**	**0.9595 ± 0.0441**	1.0412 ± 0.0737
Natural_Scene	**4.2609 ± 0.0866**	4.8189 ± 0.1051	**6.5546 ± 0.0914**	6.7306 ± 0.0831	**6.4559 ± 0.095**	6.6115 ± 0.1026
Human_Gene	16.2737 ± 0.7555	**15.7894 ± 0.6271**	14.4531 ± 0.6124	**14.4487 ± 0.5935**	14.8633 ± 0.8857	**14.5496 ± 0.6531**
Average	1.8545 ± 0.0815	**1.8521 ± 0.0759**	1.8945 ± 0.0769	**1.8711 ± 0.0728**	1.86 ± 0.0885	**1.8584 ± 0.0765**

Table 7
Experimental Results (mean ± std) measured by Kullback–Leibler Divergence ↓.

Dataset	*k*-NN	SSG-LDL+*k*-NN	BFGS	SSG-LDL+BFGS	StructRF	SSG-LDL+StructRF
Yeast_alpha	0.0066 ± 0.0006	**0.0063 ± 0.0006**	**0.0055 ± 0.0006**	**0.0055 ± 0.0006**	**0.0055 ± 0.0006**	**0.0055 ± 0.0006**
Yeast_cdc	0.0083 ± 0.0009	**0.0079 ± 0.0009**	**0.0070 ± 0.0008**	**0.0070 ± 0.0008**	0.0070 ± 0.0009	**0.0069 ± 0.0008**
Yeast_cold	0.0142 ± 0.0014	**0.0136 ± 0.0014**	**0.0122 ± 0.0011**	**0.0122 ± 0.0012**	**0.0118 ± 0.0011**	**0.0118 ± 0.0011**
Yeast_diau	0.0150 ± 0.0008	**0.0142 ± 0.0009**	**0.0131 ± 0.0011**	**0.0131 ± 0.0011**	**0.0125 ± 0.001**	**0.0125 ± 0.0011**
Yeast_dtt	0.0073 ± 0.0007	**0.0069 ± 0.0007**	0.0063 ± 0.0006	**0.0062 ± 0.0006**	0.0061 ± 0.0006	**0.0060 ± 0.0006**
Yeast_elu	0.0074 ± 0.0003	**0.007 ± 0.0004**	**0.0062 ± 0.0004**	**0.0062 ± 0.0004**	**0.0061 ± 0.0004**	**0.0061 ± 0.0004**
Yeast_heat	0.0146 ± 0.0007	**0.0138 ± 0.0005**	**0.0126 ± 0.0004**	**0.0126 ± 0.0004**	0.0120 ± 0.0004	**0.0119 ± 0.0004**
Yeast_spo	0.0302 ± 0.0024	**0.0283 ± 0.0024**	0.0246 ± 0.0029	**0.0245 ± 0.0028**	0.0243 ± 0.0024	**0.0241 ± 0.0025**
Yeast_spo5	0.0333 ± 0.0033	**0.0322 ± 0.0036**	**0.0293 ± 0.0028**	**0.0293 ± 0.0028**	**0.027 ± 0.0025**	0.0276 ± 0.0022
Yeast_spoem	0.0285 ± 0.0023	**0.0272 ± 0.0026**	**0.0245 ± 0.0023**	0.0246 ± 0.0024	**0.0228 ± 0.002**	0.0233 ± 0.0018
SJAFFE	0.0730 ± 0.0162	**0.0724 ± 0.0165**	0.1639 ± 0.0245	**0.0754 ± 0.0166**	**0.0603 ± 0.0089**	0.0655 ± 0.0116
SBU_3DFE	0.0907 ± 0.0048	**0.0872 ± 0.0048**	**0.0634 ± 0.0038**	0.0689 ± 0.0038	**0.0659 ± 0.0054**	0.0703 ± 0.0046
Movie	**0.1255 ± 0.0137**	0.1313 ± 0.0167	0.1543 ± 0.0405	**0.1211 ± 0.0177**	**0.0901 ± 0.0061**	0.1046 ± 0.0114
Natural_Scene	1.1924 ± 0.0765	**1.1011 ± 0.063**	1.112 ± 0.0999	**0.8061 ± 0.0292**	**0.6436 ± 0.0211**	0.696 ± 0.0259
Human_Gene	0.3006 ± 0.0248	**0.2815 ± 0.0206**	0.2365 ± 0.0184	**0.236 ± 0.018**	0.248 ± 0.0265	**0.239 ± 0.0197**
Average	0.1298 ± 0.01	**0.1221 ± 0.009**	0.1248 ± 0.0133	**0.0966 ± 0.0066**	**0.0829 ± 0.0053**	0.0874 ± 0.0056

Table 8
Experimental Results (mean ± std) measured by Cosine Coefficient ↑.

Dataset	*k*-NN	SSG-LDL+*k*-NN	BFGS	SSG-LDL+BFGS	StructRF	SSG-LDL+StructRF
Yeast_alpha	0.9935 ± 0.0006	**0.9938 ± 0.0006**	**0.9946 ± 0.0006**	**0.9946 ± 0.0006**	**0.9946 ± 0.0006**	**0.9946 ± 0.0006**
Yeast_cdc	0.992 ± 0.0008	**0.9924 ± 0.0008**	**0.9933 ± 0.0007**	**0.9933 ± 0.0007**	**0.9933 ± 0.0008**	**0.9933 ± 0.0007**
Yeast_cold	0.9866 ± 0.0012	**0.9872 ± 0.0011**	0.9885 ± 0.0009	**0.9886 ± 0.0010**	**0.9889 ± 0.0009**	**0.9889 ± 0.0009**
Yeast_diau	0.9862 ± 0.0008	**0.987 ± 0.0008**	**0.9879 ± 0.0010**	**0.9879 ± 0.0010**	**0.9885 ± 0.0010**	**0.9885 ± 0.0011**
Yeast_dtt	0.9931 ± 0.0005	**0.9934 ± 0.0005**	**0.9941 ± 0.0004**	**0.9941 ± 0.0004**	**0.9943 ± 0.0004**	**0.9943 ± 0.0004**
Yeast_elu	0.9928 ± 0.0003	**0.9932 ± 0.0003**	0.994 ± 0.0004	**0.9941 ± 0.0004**	**0.9941 ± 0.0003**	**0.9941 ± 0.0004**
Yeast_heat	0.9861 ± 0.0007	**0.9869 ± 0.0006**	**0.9880 ± 0.0004**	**0.9880 ± 0.0004**	0.9886 ± 0.0004	**0.9887 ± 0.0005**
Yeast_spo	0.9716 ± 0.002	**0.9734 ± 0.002**	0.9769 ± 0.0026	**0.977 ± 0.0025**	0.9773 ± 0.0021	**0.9775 ± 0.0021**
Yeast_spo5	0.9705 ± 0.0025	**0.9714 ± 0.0028**	**0.9741 ± 0.0023**	**0.9741 ± 0.0023**	**0.9762 ± 0.002**	0.9756 ± 0.0018
Yeast_spoem	0.9754 ± 0.0019	**0.9764 ± 0.0023**	**0.979 ± 0.0019**	0.9789 ± 0.002	**0.9804 ± 0.0018**	0.9799 ± 0.0016
SJAFFE	0.9308 ± 0.0163	**0.9313 ± 0.0166**	0.8739 ± 0.0179	**0.9293 ± 0.0154**	**0.9429 ± 0.0083**	0.9381 ± 0.0107
SBU_3DFE	0.9118 ± 0.0047	**0.915 ± 0.0047**	**0.9389 ± 0.0034**	0.9324 ± 0.0037	**0.9348 ± 0.0055**	0.9309 ± 0.0044
Movie	**0.9174 ± 0.0074**	0.9135 ± 0.0093	0.9072 ± 0.0188	**0.9209 ± 0.0103**	**0.9407 ± 0.0039**	0.9319 ± 0.0066
Natural_Scene	**0.7043 ± 0.0137**	0.704 ± 0.0108	0.6724 ± 0.0149	**0.719 ± 0.0104**	**0.7895 ± 0.0091**	0.767 ± 0.0129
Human_Gene	0.7663 ± 0.0184	**0.7847 ± 0.0136**	0.8343 ± 0.0102	**0.8347 ± 0.0099**	0.8202 ± 0.0205	**0.8306 ± 0.0126**
Average	0.9386 ± 0.0048	**0.9402 ± 0.0045**	0.9398 ± 0.0051	**0.9471 ± 0.0041**	**0.9536 ± 0.0038**	0.9516 ± 0.0038

Table 9
Experimental Results (mean ± std) measured by Intersection Similarity ↑.

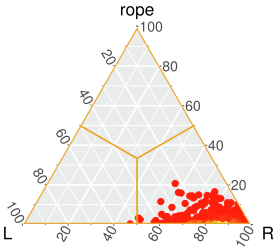
Dataset	*k*-NN	SSG-LDL+*k*-NN	BFGS	SSG-LDL+BFGS	StructRF	SSG-LDL+StructRF
Yeast_alpha	0.9581 ± 0.0020	**0.9594 ± 0.0022**	0.9622 ± 0.0024	**0.9623 ± 0.0024**	**0.9624 ± 0.0024**	**0.9624 ± 0.0023**
Yeast_cdc	0.9527 ± 0.0025	**0.954 ± 0.0026**	0.9573 ± 0.0026	**0.9574 ± 0.0026**	0.9574 ± 0.0027	**0.9576 ± 0.0026**
Yeast_cold	0.9356 ± 0.0031	**0.937 ± 0.0028**	0.9407 ± 0.0023	**0.9408 ± 0.0023**	**0.9419 ± 0.0022**	**0.9419 ± 0.0023**
Yeast_diau	0.9367 ± 0.0017	**0.9383 ± 0.002**	**0.9403 ± 0.0027**	0.9402 ± 0.0028	**0.9421 ± 0.0026**	0.942 ± 0.0027
Yeast_dtt	0.9547 ± 0.0017	**0.9561 ± 0.0016**	0.9582 ± 0.0013	**0.9584 ± 0.0013**	0.9593 ± 0.0011	**0.9595 ± 0.0011**
Yeast_elu	0.9545 ± 0.0011	**0.956 ± 0.0012**	0.9588 ± 0.0011	**0.9589 ± 0.0012**	0.9592 ± 0.0011	**0.9593 ± 0.0011**
Yeast_heat	0.9356 ± 0.0017	**0.9374 ± 0.0014**	0.9401 ± 0.001	**0.9403 ± 0.001**	0.9419 ± 0.0011	**0.9421 ± 0.001**
Yeast_spo	0.9076 ± 0.0036	**0.9104 ± 0.0037**	0.9154 ± 0.0053	**0.9156 ± 0.0051**	0.9167 ± 0.0042	**0.917 ± 0.0043**
Yeast_spo5	0.9038 ± 0.0043	**0.9051 ± 0.0043**	0.9086 ± 0.005	**0.9087 ± 0.0051**	**0.9126 ± 0.0046**	0.9118 ± 0.0044
Yeast_spoem	0.9076 ± 0.0036	**0.9095 ± 0.0043**	**0.9132 ± 0.0049**	0.9131 ± 0.0052	**0.9164 ± 0.0045**	0.9159 ± 0.0037
SJAFFE	0.8529 ± 0.0176	**0.8539 ± 0.0182**	0.7788 ± 0.0162	**0.8496 ± 0.0185**	**0.8622 ± 0.0132**	0.8568 ± 0.0164
SBU_3DFE	0.8394 ± 0.0053	**0.8425 ± 0.0053**	**0.8616 ± 0.0041**	0.8521 ± 0.0056	**0.8592 ± 0.0065**	0.8546 ± 0.0063
Movie	**0.8153 ± 0.0109**	0.8095 ± 0.014	0.804 ± 0.0199	**0.8171 ± 0.0137**	**0.8432 ± 0.006**	0.8285 ± 0.0107
Natural_Scene	**0.5634 ± 0.0098**	0.5588 ± 0.0079	**0.5248 ± 0.0145**	0.5226 ± 0.0082	**0.5919 ± 0.0097**	0.5633 ± 0.0098
Human_Gene	0.7417 ± 0.0133	**0.7525 ± 0.0104**	0.7842 ± 0.0092	**0.7845 ± 0.0089**	0.7739 ± 0.0159	**0.7814 ± 0.0105**
Average	0.8773 ± 0.0055	**0.8787 ± 0.0055**	0.8765 ± 0.0062	**0.8814 ± 0.0056**	**0.8894 ± 0.0052**	0.8863 ± 0.0053

5. Results and analysis

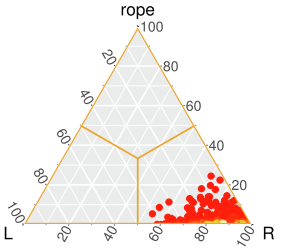
This section presents the results of the empirical studies and their analyses. For each learner we will compare the version with and without pre-processing, showing their different strengths. As those algorithms have been tested using 10-fcv, the performance is represented using "mean ± standard deviation". Each of the tables shows the experimental results obtained for each learner, comparing them with the results obtained from same learner but using SSG-LDL pre-processing. In each

Table 10
Wilcoxon Signed Ranks test: SyntheticLDL+k-NN vs. k-NN.

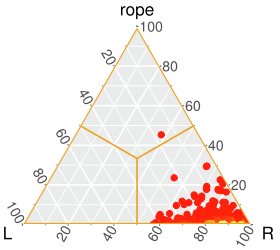
Measure	R^+	R^-	p-value
Cheby	101	19	0.018066
Clark	92	28	0.073
Can	92	28	0.073
KL	107	13	0.005372
Cos	103	17	0.012452
Inter	93	27	0.06372



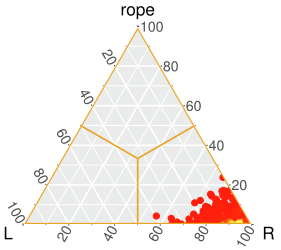
(a) Chebyshev Distance



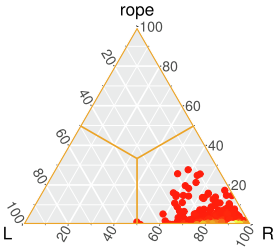
(b) Clark Distance



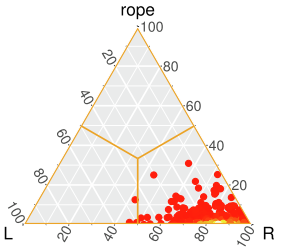
(c) Canberra Metric



(d) Kullback-Leibler Divergence



(e) Cosine Coefficient



(f) Intersection Similarity

Fig. 4. Bayesian Sign test comparing k-NN (L) vs. SSG-LDL+k-NN (R).

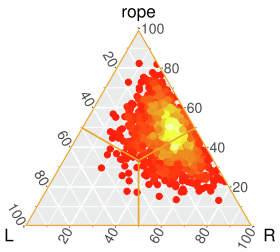
case the best result is highlighted in bold. The last row is the mean of each column. The best average is also highlighted in bold. The results of the different measures are shown in Tables 4–9.

5.1. Evaluation of SSG-LDL+k-NN vs. k-NN

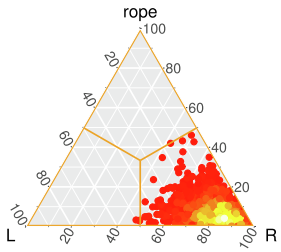
A comparison between k-NN and k-NN using a previously pre-processed dataset using SSG-LDL highlights the best ranking of the learner when using a preprocessed dataset.

Table 11
Wilcoxon Signed Ranks test: SyntheticLDL+BFGS vs. BFGS.

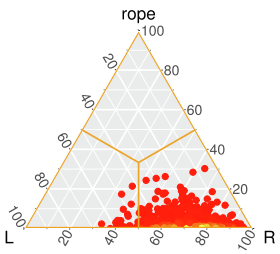
Measure	R^+	R^-	p-value
Cheby	94	26	0.05536
Clark	81	24	0.0785
Can	90	30	0.0946
KL	76	29	0.15308
Cos	89	31	0.107
Inter	87	33	0.13538



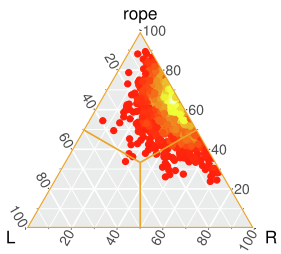
(a) Chebyshev Distance



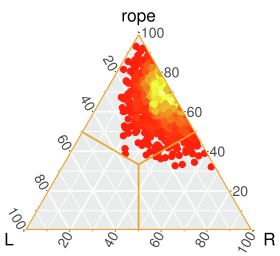
(b) Clark Distance



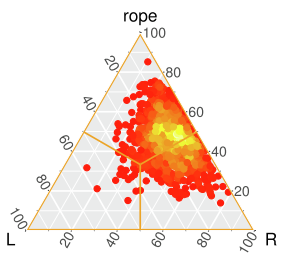
(c) Canberra Metric



(d) Kullback-Leibler Divergence



(e) Cosine Coefficient



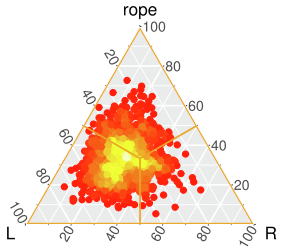
(f) Intersection Similarity

Fig. 5. Bayesian Sign test comparing BFGS (L) vs. SSG-LDL+BFGS (R).

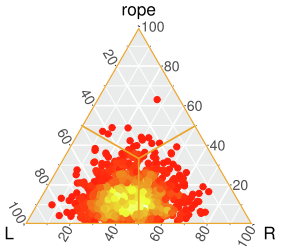
As previously mentioned, we have used the Wilcoxon Signed Ranks test and the Bayesian Sign test to corroborate the significance of the differences between our approach and the selected methods. Table 10 includes the outcome of the Wilcoxon test comparing both learners. All the hypotheses of equivalence are rejected with small p-values. Fig. 4 graphically represents the difference between using data pre-processing or not and its statistical significance in terms of accuracy. The fol-

Table 12
Wilcoxon Signed Ranks test: SyntheticLDL+StructRF vs. StructRF.

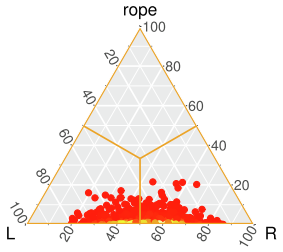
Measure	R^+	R^-	p-value
Cheby	42	78	$\geqslant 0.2$
Clark	49	71	$\geqslant 0.2$
Can	53	67	$\geqslant 0.2$
KL	44	76	$\geqslant 0.2$
Cos	40	81	$\geqslant 0.2$
Inter	44	77	$\geqslant 0.2$



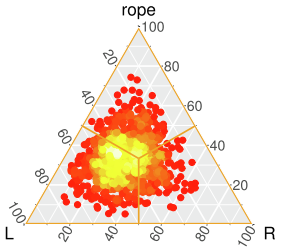
(a) Chebyshev Distance



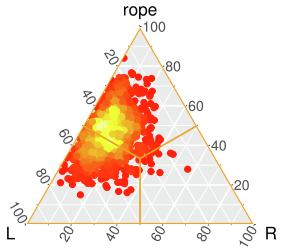
(b) Clark Distance



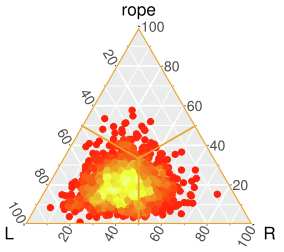
(c) Canberra Metric



(d) Kullback-Leibler Divergence



(e) Cosine Coefficient



(f) Intersection Similarity

Fig. 6. Bayesian Sign test comparing StructRF (L) vs. SSG-LDL+StructRF (R).

lowing heat-maps clearly indicate the significant superiority of SSG-LDL, as the computed distributions are always located in the right region.

5.2. Evaluation of SSG-LDL+BFGS vs. BFGS

A comparison between BFGS and BFGS using a previously pre-processed dataset with synthetic feature generation also highlights the best ranking (for all distance measures) of the learner when using a preprocessed dataset. Table 11 includes the outcome of the Wilcoxon test comparing both learners. All the hypotheses of equivalence are rejected with small p-values. Fig. 5 graphically represents the difference between using data pre-processing or not and its statistical significance in terms of accuracy. The heat-maps clearly indicate significant superiority, for almost all measures, when applying SSG-LDL, since the computed distributions are always located in the right region. In view of the results of this statistical analysis, there is a high probability that the results obtained will improve after applying this pre-processing method.

5.3. Evaluation of SSG-LDL+StructRF vs. StructRF

Let's see how the SSG-LDL algorithm behaves when it is combined with a learning method as robust as StructRF. As for the general classification, we see that applying pre-processing improves the results in 40% of the cases and ties in 15% of the cases. If we go into further detail we observe that the behavior varies according to the dataset: for the *Human_Gene* for example we notice that applying synthetic sample generation does significantly improve the original results although it does not substantially alter the results for the *Yeast* datasets and causes slightly worse outputs for the sets related to facial expression images, movies or natural scenes. Table 12 includes the outcome of the Wilcoxon test comparing both learners. Fig. 6 also corroborates these results statistically speaking, showing that applying pre-processing on the StructRF algorithm can be advantageous in a high percentage of cases. In summary, by applying a data pre-processing method such as the one presented here, SSG-LDL, significantly improves the results obtained by the learning algorithm. In the case of *k*-NN, previously applying an SSG-LDL treatment makes the learner behave more efficiently in practically all datasets. This also happens when the same type of pre-processing is carried out on a BFGS learning method, obtaining notable improvement in the results. Even when using a method as robust as StructRF, it is also advantageous to apply the SSG-LDL as it can improve results in a high percentage of experiments.

6. Conclusion

LDL datasets suffer high dispersion that can negatively affect learners' performance.

In this paper, we proposed a novel method of synthetic data generation that adapts to LDL constraints. The SSG-LDL proposal can be applied, in a previous pre-processing stage, to any training set and then can be plugged in as an input to any learning algorithm.

We have based the design of this algorithm on techniques that have already demonstrated their potential in the area of data pre-processing. We have then extended and adapted them to be compatible with a distribution of labels as the output.

In order to verify the effectiveness of the solution designed, it has been applied to three state of the art learners in the LDL scope and we have verified that the results obtained are very encouraging.

The SSG-LDL method is the first proposal of oversampling adapted to LDL restrictions. In future research we want to make improvements on the current proposal to make it even stronger. Some ideas that we can anticipate are as follows:

- To select the instances to be oversampled, SSG-LDL uses the Euclidean distance. Some recent methods of oversampling such as MDO [2] build synthetic examples that have the same Mahalanobis distance from each examined class mean as the other minority examples. Thus, an alternative procedure that we would consider integrating in future studies is to improve learning in the region of minority instances by preserving the covariance and the probability during the generation of synthetic examples. The Hellinger distance metric, based on probability distributions, is more tolerant to skewed class distributions. This metric has been applied in the context of data reduction [46] and it also be of interest to introduce it in an oversampling method such as SSG-LDL.
- ProLSFEO-LDL [28] is a novel method that simultaneously addresses the prototype selection and the label-specific feature selection pre-processing techniques, specifically developed to deal with LDL restrictions. Devising a method that intelligently combines SSG-LDL as an oversampling method and ProLSFEO-LDL as a data reduction strategy could yield a very effective pre-processing method.
- Combining SMOTE and a boosting procedure such as SMOTEBoost [13] or MDOBoost [1] usually improves the prediction performance. Extending either of these two methods to use SSG-LDL and making them compatible with LDL problems is an interesting approach to follow.
- The filtering of artificial samples is a frequent operation that supports the success of SMOTE on real data. Adding a rebalancing stage to SSG-LDL similar to AMSCO [30] could mitigate the potential drawback of generating overlapping and noisy examples.

CRediT authorship contribution statement

Manuel González: Conceptualization, Methodology, Software, Investigation, Writing - review & editing. **Julián Luengo:** Validation, Supervision. **José-Ramón Cano:** Supervision, Resources. **Salvador García:** Funding acquisition, Project administration, Supervision.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work is supported by the Spanish National Research Project TIN2017-89517-P.

References

[1] L. Abdi, S. Hashemi, To combat multi-class imbalanced problems by means of over-sampling and boosting techniques, Soft Computing 19 (12) (2015) 3369–3385.
[2] L. Abdi, S. Hashemi, To combat multi-class imbalanced problems by means of over-sampling techniques, IEEE Transactions on Knowledge and Data Engineering 28 (1) (2016) 238–251.
[3] T. Ahonen, A. Hadid, M. Pietikainen, Face description with local binary patterns: application to face recognition, IEEE Transactions on Pattern Analysis & Machine Intelligence 12 (2006) 2037–2041.
[4] R. Barandela, J.S. Sánchez, V. Garca, E. Rangel, Strategies for learning in class imbalance problems, Pattern Recognition 36 (3) (2003) 849–851.
[5] A. Benavoli, G. Corani, J. Demšar, M. Zaffalon, Time for a change: a tutorial for comparing multiple classifiers through bayesian analysis, The Journal of Machine Learning Research 18 (1) (2017) 2653–2688.
[6] M.R. Boutell, J. Luo, X. Shen, C.M. Brown, Learning multi-label scene classification, Pattern Recognition 37 (9) (2004) 1757–1771.
[7] H. Cao, X.-L. Li, D.Y. Woon, S.-K. Ng, Integrated oversampling for imbalanced time series classification, IEEE Transactions on Knowledge and Data Engineering 25 (12) (2013) 2809–2822.
[8] J. Carrasco, S. García, M. del Mar Rueda, F. Herrera, rnpbst: An r package covering non-parametric and bayesian statistical tests, in: International Conference on Hybrid Artificial Intelligence Systems, Springer, 2017, pp. 281–292.
[9] S.-H. Cha, Comprehensive survey on distance/similarity measures between probability density functions, City 1 (2) (2007) 1.
[10] F. Charte, A.J. Rivera, M.J. del Jesus, F. Herrera, Mlsmote: approaching imbalanced multilabel learning through synthetic instance generation, Knowledge-Based Systems 89 (2015) 385–397.
[11] N.V. Chawla, K.W. Bowyer, L.O. Hall, W.P. Kegelmeyer, Smote: synthetic minority over-sampling technique, Journal of Artificial Intelligence Research 16 (2002) 321–357.
[12] N.V. Chawla, D.A. Cieslak, L.O. Hall, A. Joshi, Automatically countering imbalance and its empirical relationship to cost, Data Mining and Knowledge Discovery 17 (2) (2008) 225–252.
[13] N.V. Chawla, A. Lazarevic, L.O. Hall, K.W. Bowyer, Smoteboost: improving prediction of the minority class in boosting, in: European Conference on Principles of Data Mining and Knowledge Discovery, Springer, 2003, pp. 107–119.
[14] M. Chen, X. Wang, B. Feng, W. Liu, Structured random forest for label distribution learning, Neurocomputing 320 (2018) 171–182.
[15] J. Derrac, S. García, D. Molina, F. Herrera, A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms, Swarm and Evolutionary Computation 1 (1) (2011) 3–18.
[16] G. Ditzler, R. Polikar, Incremental learning of concept drift from streaming imbalanced data, IEEE Transactions on Knowledge and Data Engineering 25 (10) (2013) 2283–2301.
[17] A. Dong, F.-L. Chung, S. Wang, Semi-supervised classification method through oversampling and common hidden space, Information Sciences 349 (2016) 216–228.
[18] M.B. Eisen, P.T. Spellman, P.O. Brown, D. Botstein, Cluster analysis and display of genome-wide expression patterns, Proceedings of the National Academy of Sciences 95 (25) (1998) 14863–14868.
[19] A. Fernández, S. Garcia, F. Herrera, N.V. Chawla, Smote for learning from imbalanced data: progress and challenges, marking the 15-year anniversary, Journal of Artificial Intelligence Research 61 (2018) 863–905.
[20] B. Gao, C. Xing, C. Xie, J. Wu, X. Geng, Deep label distribution learning with label ambiguity, IEEE Transactions on Image Processing 26 (6) (2017) 2825–2838.
[21] S. García, J. Luengo, F. Herrera, Data Preprocessing in Data Mining vol. 72 (2015).
[22] S. García, J. Luengo, F. Herrera, Tutorial on practical tips of the most influential data preprocessing algorithms in data mining, Knowledge-Based Systems 98 (2016) 1–29.
[23] X. Geng, Label distribution learning, IEEE Transactions on Knowledge and Data Engineering 28 (7) (2016) 1734–1748.
[24] X. Geng, P. Hou, Pre-release prediction of crowd opinion on movies by label distribution learning, in: Twenty-Fourth International Joint Conference on Artificial Intelligence, vol. 2015-January, Affiliation: School of Computer Science and Engineering, Southeast University, Nanjing, China, 2015, pp. 3511–3517..
[25] X. Geng, L. Luo, Multilabel ranking with inconsistent rankers, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 3742–3747.
[26] X. Geng, C. Yin, Z. Zhou, Facial age estimation by learning from label distributions, IEEE Transactions on Pattern Analysis and Machine Intelligence 35 (10) (2013) 2401–2412.
[27] D.E. Goldberg, Genetic algorithms in search, Optimization, and Machine Learning (1989).
[28] M. González, J.-R. Cano, S. García, Prolsfeo-ldl: Prototype selection and label-specific feature evolutionary optimization for label distribution learning, Applied Sciences 10 (9) (2020) 3089.
[29] F. Herrera, F. Charte, A.J. Rivera, M.J. Del Jesus, Multilabel Classification, Springer, 2016, pp. 17–31..
[30] J. Li, S. Fong, R.K. Wong, V.W. Chu, Adaptive multi-objective swarm fusion for imbalanced data classification, Information Fusion 39 (2018) 1–24.
[31] M. Lyons, S. Akamatsu, M. Kamachi, J. Gyoba, Coding facial expressions with gabor wavelets, in: Proceedings Third IEEE International Conference on Automatic Face and Gesture Recognition, IEEE, 1998, pp. 200–205.
[32] P. Moutafis, I.A. Kakadiaris, Gs4: Generating synthetic samples for semi-supervised nearest neighbor classification, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, 2014, pp. 393–403.

[33] R.C. Prati, G.E. Batista, D.F. Silva, Class imbalance revisited: a new experimental setup to assess the performance of treatment methods, Knowledge and Information Systems 45 (1) (2015) 247–270.
[34] Y. Ren, X. Geng, Sense beauty by label distribution learning, in: IJCAI. Affiliation: MOE Key Laboratory of Computer Network and Information Integration, School of Computer Science and Engineering, Southeast University, Nanjing, China, 2017, pp. 2648–2654.
[35] A. Roy, R.M. Cruz, R. Sabourin, G.D. Cavalcanti, A study on combining dynamic selection and data preprocessing for imbalance learning, Neurocomputing 286 (2018) 179–192.
[36] W. Shen, K. Zhao, Y. Guo, A. Yuille, Label distribution learning forests, in: Advances in Neural Information Processing Systems, vol. 2017-December. Affiliation: Key Laboratory of Specialty Fiber Optics and Optical Access Networks, Shanghai Institute for Advanced Communication and Data Science, School of Communication and Information Engineering, Shanghai University, China; Affiliation: Department of Computer Science, Johns Hopkins University, United States, 2017, pp. 835–844..
[37] L. Torgo, P. Branco, R.P. Ribeiro, B. Pfahringer, Resampling strategies for regression, Expert Systems 32 (3) (2015) 465–476.
[38] I. Triguero, S. García, F. Herrera, Seg-ssc: A framework based on synthetic examples generation for self-labeled semi-supervised classification, IEEE Transactions on Cybernetics 45 (4) (2015) 622–634.
[39] I. Triguero, S. Gonzalez, J.M. Moyano, S. Garcia, J. Alcala-Fdez, J. Luengo, A. Fernandez, M. Jose del Jesus, L. Sanchez, F. Herrera, Keel 3.0: An open source software for multi-stage analysis in data mining, International Journal of Computational Intelligence Systems 10 (1) (2017) 1238–1249.
[40] G. Tsoumakas, I. Katakis, Multi-label classification: an overview, International Journal of Data Warehousing and Mining (IJDWM) 3 (3) (2007) 1–13.
[41] J. Wang, B. Yun, P. Huang, Y.-A. Liu, Applying threshold smote algorithm with attribute bagging to imbalanced datasets, in: International Conference on Rough Sets and Knowledge Technology, Springer, 2013, pp. 221–228.
[42] Y. Wang, J. Dai, Label distribution feature selection based on mutual information in fuzzy rough set theory, 2019, Vol. 2019-July..
[43] C. Xing, X. Geng, H. Xue, Logistic boosting regression for label distribution learning, in: Proceedings of the IEEE conference on computer vision and pattern recognition, vol. 2016-January. Affiliation: Key Lab of Computer Network and Information Integration, Ministry of Education, School of Computer Science and Engineering, Southeast University, Nanjing, 211189, China, 2016, pp. 4489–4497..
[44] D. Xue, Z. Hong, S. Guo, L. Gao, L. Wu, J. Zheng, N. Zhao, Personality recognition on social media with label distribution learning, IEEE Access 5 (2017) 13478–13488.
[45] J. Yang, D. She, M. Sun, Joint image emotion classification and distribution learning via deep convolutional neural network, in: IJCAI. Affiliation: College of Computer and Control Engineering, Nankai University, Tianjin, China, 2017, pp. 3266–3272..
[46] L. Yin, Y. Ge, K. Xiao, X. Wang, X. Quan, Feature selection for high-dimensional imbalanced data, Neurocomputing 105 (2013) 3–11.
[47] L. Yin, X. Wei, Y. Sun, J. Wang, M.J. Rosato, A 3d facial expression database for facial behavior research, in: 7th International Conference on Automatic Face and Gesture Recognition (FGR06), IEEE, 2006, pp. 211–216.
[48] J.-F. Yu, D.-K. Jiang, K. Xiao, Y. Jin, J.-H. Wang, X. Sun, Discriminate the falsely predicted protein-coding genes in aeropyrum pernix k1 genome based on graphical representation, Match-Communications in Mathematical and Computer Chemistry 67 (3) (2012) 845.
[49] Z. Zhang, M. Wang, X. Geng, Crowd counting in public video surveillance by label distribution learning, Neurocomputing 166 (2015) 151–163.
[50] X. Zheng, X. Jia, W. Li, Label distribution learning by exploiting sample correlations locally, in: Thirty-Second AAAI Conference on Artificial Intelligence, 2018.