The imprecise Dirichlet model as a basis for a new boosting classification algorithm

Lev V. Utkin
Department of Control, Automation and System Analysis
Saint Petersburg State Forest Technical University
Institutski per. 5, 194021, Saint Petersburg, Russia
Tel.: +7-812-6709262 email: lev.utkin@gmail.com

Abstract

A new algorithm for ensemble construction based on adapted restricting a set of weights of examples in training data to avoid overfitting and to reduce a number of iterations is proposed in the paper. The algorithm called IDMBoost (Imprecise Dirichlet Model Boost) applies Walley's imprecise Dirichlet model for modifying the restricted sets of weights depending on the number and location of classification errors. Updating of weights within the restricted set (simplex) is carried out by using its extreme points. The proposed algorithm has a double adaptation procedure. The first adaptation is carried out within every restricted simplex like the AdaBoost. The second adaptation reduces and changes the restricted sets of possible weights of examples. Various numerical experiments with real data illustrate the proposed algorithm.

Keywords: Machine learning; classification; boosting; AdaBoost; imprecise Dirichlet model; extreme points; training data.

1 Introduction

The binary classification problem can be formally written as follows. Given n training data (examples, instances, patterns) $S = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_n, y_n)\}$, in which $\mathbf{x}_i \in \mathbb{R}^m$ represents a feature vector involving m features and $y_i \in \{-1, 1\}$ represents the class of the associated examples, the task of classification is to construct an accurate classifier $c : \mathbb{R}^m \to \{-1, 1\}$ that maximizes the probability that $c(\mathbf{x}_i) = y_i$ for $i = 1, ..., n$. Generally $\mathbf{x}_i$ may belong to an arbitrary set $\mathcal{X}$, but we consider a special case for simplicity when $\mathcal{X} = \mathbb{R}^m$.

One of the very popular approaches to classification is the ensemble methodology. A basic idea of the classifier ensemble learning is to construct multiple classifiers from the

original data and then to aggregate their predictions when classifying unknown samples. It is carried out by means of weighing several weak or base classifiers and by combining them in order to obtain a strong classifier that outperforms every one of them. The formal definitions of weak and strong classifiers were introduced by Schapire [27]. According to these definitions, the weak classifier is a learning algorithm whose classification error strictly less than that of random guessing, for example, 0.5 in the binary case. The strong classifier can be arbitrarily accurate and its classification error can have arbitrarily small probability.

A lot of ensemble-based methods have been developed in last decades. The improvement in performance arising from ensemble combinations is usually the result of a reduction in variance of the classification error [7]. This occurs because the usual effect of ensemble averaging is to reduce the variance of a set of classifiers. According to Zhou [35], it is interesting to consider the following three main techniques combining many methods and their modifications: bagging, stacking and boosting. Bagging [3] aims to improve accuracy by combining multiple classifiers. One of the most powerful bagging methods is random forests [4], which uses a large number of individual decision trees. Another technique for achieving the highest generalization accuracy in the framework of ensemble-based methods is stacking [33]. This technique is used to combine different classifiers by means of a meta-learner that takes into account which classifiers are reliable and which are not.

The most well known ensemble-based technique is boosting which improves the performance of weak classifiers by means of their combining into a single strong classifier. Both boosting and bagging techniques use voting for combining the weak classifiers. However, the voting mechanism is differently implemented. In particular, examples in bagging are chosen with equal probabilities. Boosting supposes to choose the examples with probabilities that are proportional to their weights [25].

A detailed analysis of many ensemble-based methods can be found in a review proposed by Ferreira and Figueiredo [11]. The authors compare a huge number of modifications of boosting algorithms. One of the first books studying combination rules for improving classification performance was written by Kuncheva [18]. An interesting review of ensemble-based methods is proposed by Polikar [23]. A comprehensive analysis of combination algorithms and their application to machine learning approaches such as classification, regression, clusterization can be also found in a review paper written by Rokach [25]. It should be noted that the above works are only a small part of books and review papers devoted to this methodology.

The first practical and efficient boosting algorithm is AdaBoost (Adaptive Boosting) proposed by Freund and Schapire [14]. AdaBoost is a general purpose boosting algorithm that can be used in conjunction with many other learning algorithms to improve their performance via an iterative process. However, as point out by Ferreira and Figueiredo [11], AdaBoost has two fundamental differences from boosting. First, examples from a training set are drawn for classification from an iteratively updated sample distribution defined on elements of the training set. Second, weak classifiers are combined through weighted majority voting, where voting weights are based on the number of misclassified examples and their weights in accordance with the sample distribution. The sample distribution

ensures that examples misclassified by the previous classifier (by the weak classifier at the previous iteration) are more likely to be included in the training data of the next classifier.

In a more formal form, the Adaboost can be explained as follows. Initially, identical weights $h = (1/n, ..., 1/n)$ are assigned to all examples. In each iteration, the weights of all misclassified examples are increased while the weights of correctly classified examples are decreased (see Algorithm 1). As a consequence, the weak classifier is forced to focus on the difficult examples of the training set by performing additional iterations and creating more classifiers. Furthermore, a weight α_t is assigned to every individual classifier. This weight measures the overall accuracy of the classifier and is a function of the total weight $e(t)$ of misclassified examples calculated with respect to the distribution $h(t)$. The weight $\alpha_t \geq 0$ measures also the importance that is assigned to the classifier c_t. Thus, higher weights are given to more accurate classifiers. These weights are used for classification of new examples. The distribution $h(t)$ is updated using the rule shown in Algorithm 1. It is simply to see that if both $y_i \in \{-1, 1\}$ and $c_t(x_i) \in \{-1, 1\}$ take identical values (the i-th example is correctly classified), then we have $y_i c_t(x_i) = 1$. Hence, the updated weight is $h_i(t+1) = h_i(t) \cdot \exp(-\alpha_t) \leq h_i(t)$. When the i-th example is misclassified, then y_i and $c_t(x_i)$ take different values, and we have $y_i c_t(x_i) = -1$. The updated weight in this case is $h_i(t+1) = h_i(t) \cdot \exp(\alpha_t) \geq h_i(t)$. So, the effect of this updating rule is to increase weights of examples misclassified by $h(t)$ and to decrease weights of correctly classified examples. Thus, the weight tends to concentrate on "hard" examples. The final classifier c is a weighted majority vote of T weak classifiers.

Algorithm 1 The AdaBoost algorithm

Require: T (number of iterations), S (training set)
Ensure: $c_t, \alpha_t,\ t = 1, ..., T$
  $t \leftarrow 1$; $h_i(1) \leftarrow 1/n$; $i = 1, ..., n$
  repeat
    Build classifier c_t using distribution $h(t)$
    $e(t) \leftarrow \sum_{i:c_t(x_i) \neq y_i} h_i(t)$
    if $e(t) > 0.5$ **then**
      $T \leftarrow t - 1$
      exit Loop
    end if
    $\alpha_t \leftarrow \frac{1}{2} \ln\left(\frac{1-e(t)}{e(t)}\right)$
    $h_i(t+1) \leftarrow h_i(t) \cdot \exp(-\alpha_t y_i c_t(x_i))$
    Normalize $h(t+1)$ to be a proper distribution
    $t++$
  until $t > T$
  $c(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{T} \alpha_t c_t(\mathbf{x})\right)$

The main advantage of AdaBoost over other boosting techniques is that it is adaptive.

This is a very important feature of the AdaBoost, which is realized by modifying the weights of misclassified and correctly classified examples from the training set according to the rule given above.

In fact, the uniform probability distribution (equal weights assigned to all examples) is replaced by another probability distribution in each iteration of the algorithm in order to improve the performance accuracy in the next iteration. The weights of examples in boosting methods can be regarded as a discrete probability distribution (probability mass function) over a training set. The algorithm searches for a suitable probability distribution starting from the uniform distribution. The updated probability distribution can be arbitrary, i.e. the set of possible probability distributions $\mathcal{P}$ is not restricted and is the unit simplex. On the one hand, this arbitrariness can be viewed as a way for searching for an optimal solution, and this is a positive feature of the AdaBoost algorithm. On the other hand, this arbitrariness is one of the shortcomings of this approach. It has been reported by many authors that the main reason for poor classification results of AdaBoost in the high-noise regime is that the algorithm produces a skewed data distribution, by assigning too large weights to a few hard-to-learn examples. This leads to overfitting.

Many modifications of AdaBoost have been proposed to overcome the problem of overfitting. Roughly, they can be divided into two large groups. Methods from the first group use early stopping rules (see, for example, [20, 34]). However, Mease and Wyner [20] illustrate by means of simple and visual examples that conventional early stopping rules may lead to incorrect classification results.

The second group consists of the methods which restrict the set $\mathcal{P}$ of weights of examples in training data (see, for example, [10, 21, 28, 29]). There are other modifications of the AdaBoost, which do not belong to the above groups (see, for example, [5, 17, 19]).

We dwell on the algorithm proposed by Utkin [29]. Before discussing ideas of this algorith, we consider the probabilistic interpretation of weights of examples. Freund and Schapire [14] in their pioneering work used the so-called on-line allocation model to introduce the AdaBoost algorithm. They regarded every element of a training set as an allocation agent having a strategy to choose with some probability h_i and some loss. This probability is defined by Freund and Schapire as the normalized weight in the boosting algorithm. The probabilities of all agents (elements of the training set) form the corresponding probability distribution $h = (h_1, ..., h_n)$ which is defined on the finite sample space. Many authors adhere to this interpretation of weights.

The main idea of the algorithm proposed by Utkin [29] is to restrict the possible area $\mathcal{P}$ where the probability distribution can move by means of the imprecise statistical models [31], including the well-known ε-contaminated (robust) model [16], the pari-mutuel model [22], the constant odds-ratio model [31], bounded vacuous and bounded ε-contaminated robust models. The imprecise models produce compact convex sets of probability distributions $\mathcal{P}^* \subseteq \mathcal{P}$. The properties of compactness and convexity imply that every probability distribution from the set $\mathcal{P}^*$ can be obtained as the linear combination of its extreme points with certain weights λ. Hence, the procedure of modification of h accepted in the standard AdaBoost is replaced by modification of λ. The ideas of using the extreme points and the

imprecise statistical models in [29] give the corresponding name of the algorithm EPIBoost (Extreme Points Imprecise Boost). Similar models restricting the set of weights of examples by means of the imprecise statistical models for regression problems have been proposed by Utkin and Wiencierz [30]. It should be noted that two interesting ensemble-based classification methods using the imprecise probability models called by credal ensembles of classifiers have been proposed by Corani and Antonucci [6]. According to the methods [6], the credal ensemble recognizes the prior-dependent examples in a training set, namely the instances whose most probable class varies when different priors over the models are considered.

It turns out that an interesting modification of the AdaBoost using the EPIBoost [29] can be obtained by applying the Walley's imprecise Dirichlet model (IDM) [32]. The IDM can be regarded as a set of all Dirichlet distributions restricted by some set of its parameters which are defined by means of the common Bayesian updating procedure. The IDM has a number of advantages and its choice allows us to model the reduction of the initial largest set of weights in the form of the unit simplex accepted in the AdaBoost. The proposed algorithm differs from the EPIBoost by the possibility to modify not only probability distribution h in accordance with a certain rule within $\mathcal{P}$, but to modify also the set of weights $\mathcal{P}$ depending on the number of classification errors and their location at a current iteration. The procedure of the modification can be carried out by means of the IDM in a special way. This procedure and its analysis are proposed in the given work. We call the proposed boosting model as IDMBoost in order to indicate that it is constructed by using the IDM.

The proposed IDMBoost is characterized by two adaptation procedures. One of the procedures is the iterative updating of the subset of weights $\mathcal{P}$ within the unit simplex of weights. It is similar to the adaptation procedure implemented in the AdaBoost, but with another updating rule. In contrast to the AdaBoost adaptation procedure where weights of examples tend to concentrate on "hard" examples, the proposed procedure "smooths" this tendency due to moving subsets of weight vectors within the unit simplex, but not vectors directly. Another adaptation procedure is implemented for every subset $\mathcal{P}$. It adapts vectors of weights within $\mathcal{P}$. This is similar to adaptation procedure implemented in the EPIBoost.

In contrast to the EPIBoost, the proposed modification has an important property inherited from the IDM. By using the IDMBoost, we do not need to choose one specific prior for the set of weights $\mathcal{P}$. We take the unit simplex before updating or iterations. This is a common point with the AdaBoost.

It is shown in the paper that the IDMBoost is invariant to the number of examples in a training set. Moreover, the IDMBoost is invariant to the number of classes in a classification problem. This allows us to extend the binary classification IDMBoost on the case of the multiclass classification.

The paper is organized as follows. Section 2 presents the EPIBoost algorithm proposed in [29]. A short introduction into the IDM is provided in Section 3. The proposed IDMBoost algorithm is considered in detail in Section 4. Numerical experiments with real

data illustrating accuracy of the proposed algorithm are given in Section 5. In Section 6, concluding remarks are made.

2 Extreme Points Imprecise Boost (EPIBoost)

By accepting the probabilistic interpretation of weights, we assume that there exists a set $\mathcal{P}$ of probability distributions defined by some information about the training set or by some other reasons. There are various kinds of the information which could be exploited. By using the imprecise statistical models, we can use the information about confidence probabilities for the set $\mathcal{P}$, about the number of examples in the training set. Another kind of the additional information arises when we have imbalanced data sets. Class imbalance occurs when there are significantly fewer training instances of one class compared to another class. In this case, the weights of examples from the small class increase at a higher rate compared to those of the prevalent class. In many applied problems, there is some information about the most important examples or about typical examples for a given class. This prior information can also change the set $\mathcal{P}$.

At the same time, we do not need to have additional information for determining the restricted set $\mathcal{P}$. We can apply the imprecise statistical models [31]. In fact, this case will be mainly considered below. Two extreme special cases have to be pointed out here. When we do not have the additional information or do not want to exploit imprecise statistical models, the set $\mathcal{P}$ is the largest one, and it coincides with the unit simplex having n vertices. Another extreme special case is a precise probability distribution. In this case, the set $\mathcal{P}$ is reduced to a point in the unit simplex.

We assume that $\mathcal{P}$ is a finite-dimensional compact convex set, i.e., it is generated by finitely many linear constraints. This implies that it is totally defined by its extreme points or vertices denoted $\mathcal{E}(\mathcal{P})$. Suppose that we have a set of r extreme points $q_k = (q_1^{(k)}, ..., q_n^{(k)})$ of the set $\mathcal{P}$, i.e., $q_k \in \mathcal{E}(\mathcal{P})$, $k = 1, ..., r$. Then every probability distribution $h = (h_1, ..., h_n)$ from $\mathcal{P}$ can be represented as the linear combination of the extreme points

$$h = \sum_{k=1}^{r} \lambda_k \cdot q_k. \tag{1}$$

Here $\lambda = (\lambda_1, ..., \lambda_r)$ is a vector of weights such that $\lambda_1 + ... + \lambda_r = 1$ and $\lambda_k \geq 0$, $k = 1, ..., r$.

The main idea of the proposed adaptation procedure is to change the weights λ instead of the probabilities h for decreasing the misclassification measure. It does not mean that the probability distribution h is not changed. It is changed inside the set $\mathcal{P}$ due to changes of the weights λ. However, the weights λ do not have additional restrictions and can be changed in the unit simplex having r vertices. In other words, we cannot change h in the unit simplex, but we can change the weights λ within the unit simplex without restrictions. The change of λ has to be directed to minimize the misclassification measure.

So, the first step is to assign the value $1/r$ to every element of the initial λ. The next step is to compute the error measure of the classifier c_t. It can be done by introducing the k-th extreme point misclassification rate denoted by ε_k as follows:

$$\varepsilon_k = \sum_{i:c_t(x_i)\neq y_i} q_i^{(k)}.$$

The measure ε_k means the contribution of the k-th extreme point into the classification errors. Then the probability of all incorrectly classified examples can be determined as

$$e(t) = \sum_{k=1}^{r} \lambda_k \varepsilon_k.$$

It is shown in the work [29] that the error $e(t)$ coincides with the same measure in the AdaBoost algorithm defined by Freund and Schapire [14].

Now we can write the updating rule as follows:

$$\lambda_k(t+1) = \lambda_k(t) \cdot \exp\left(-\alpha_t \cdot (1 - 2\varepsilon_k)\right)$$

where

$$\alpha_t = \frac{1}{2} \ln\left(\frac{1-e(t)}{e(t)}\right).$$

According to the proposed rule, if $\varepsilon_k > 0.5$, then λ_k is increased. If $\varepsilon_k \leq 0.5$, then λ_k is decreased. Finally, we write the algorithm of the EPIBoost which is represented as Algorithm 2.

3 The imprecise Dirichlet model

So far, we have considered the classification model with using sets of probability distributions without indication how these sets could be constructed. One of the possible ways for that is to use the IDM.

Let $U = \{u_1, ..., u_n\}$ be a set of possible outcomes u_j. Assume the standard multinomial model: N observations are independently chosen from U with an identical probability distribution $\Pr\{u_j\} = p_j$ for $j = 1, ..., n$, where each $p_j \geq 0$ and $\sum_{j=1}^{n} p_j = 1$. Denote $p = (p_1, ..., p_n)$. Let l_j denote the number of observations of u_j in n trials, so that $l_j \geq 0$ and $\sum_{j=1}^{n} l_j = n$. Under the above assumptions the random variables $l_1, ..., l_n$ have a multinomial distribution.

The Dirichlet (s, t) prior distribution for p, where $\mathbf{t} = (t_1, ..., t_n)$, has the following probability density function [8]:

$$\pi(\mathbf{p}) = \Gamma(s) \left(\prod_{j=1}^{n} \Gamma(st_j)\right)^{-1} \cdot \prod_{j=1}^{n} p_j^{st_j - 1}.$$

Algorithm 2 The EPIBoost algorithm

Require: T (number of iterations), S (training set), $q_1, ..., q_r$ (extreme points of $\mathcal{P}$)
Ensure: $c_t, \alpha_t,\ t = 1, ..., T$

$t \leftarrow 1$
$\lambda_k(1) \leftarrow 1/r$; $k = 1, ..., r$, or it can be randomly selected from the unit simplex
repeat
  Compute the vector $h \leftarrow \sum_{k=1}^{r} \lambda_k(t) \cdot q_k$
  Train classifier c_t using distribution h
  Compute error rate $\varepsilon_k \leftarrow \sum_{i:c_t(x_i) \neq y_i} q_i^{(k)}$
  Compute error $e(t) \leftarrow \sum_{k=1}^{r} \lambda_k(t) \varepsilon_k$
  if $e(t) > 0.5$ **then**
    $T \leftarrow t - 1$
    exit Loop.
  end if
  $\alpha_t \leftarrow \frac{1}{2} \ln \left(\frac{1-e(t)}{e(t)} \right)$
  $\lambda_k(t+1) \leftarrow \lambda_k(t) \cdot \exp\left(-\alpha_t \cdot (1 - 2\varepsilon_k)\right)$
  Normalize $\lambda(t+1)$ to be a proper distribution
  $t++$
until $t > T$
$c(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{T} \alpha_t c_t(\mathbf{x}) \right)$

Here the parameter $t_i \in (0,1)$ is the mean of p_i under the Dirichlet prior; the hyperparameter $s > 0$ determines the influence of the prior distribution on posterior probabilities; the vector $\mathbf{t}$ belongs to the interior of the n-dimensional unit simplex denoted by $S(1,n)$; $\Gamma(\cdot)$ is the Gamma-function which satisfies $\Gamma(x+1) = x\Gamma(x)$ and $\Gamma(1) = 1$.

Given the data $\mathbf{n} = (l_1, ..., l_n)$ the Dirichlet $(s, \mathbf{t})$ prior density generates a posterior density function

$$\pi(\mathbf{p}|\mathbf{n}) \propto \prod_{j=1}^{n} p_j^{l_j + st_j - 1},$$

which is seen to be the probability density function of a Dirichlet $(N+s, \mathbf{t}^*)$ distribution, where $t_j^* = (l_j + st_j)/(N+s)$.

One o the important properties of the Dirichlet distribution is that its marginal distributions are also Dirichlet distributions.

Walley [32] pointed out several reasons for using a set of Dirichlet distributions to model prior ignorance about probabilities p:

1. Dirichlet prior distributions are mathematically tractable because they generate Dirichlet posterior distributions;

2. sets of Dirichlet distributions are very rich because they produce the same inferences as their convex hulls and any prior distribution on the simplex can be approximated by a finite mixture of Dirichlet distributions;

3. the most common Bayesian models for prior ignorance about probabilities p are Dirichlet distributions.

The imprecise Dirichlet model is defined by Walley [32] as the set of all Dirichlet $(s, \mathbf{t})$ distributions such that $\mathbf{t} \in S(1,n)$. We will see below that the choice of this model allows us to model prior ignorance about probabilities of examples in the training set.

For the IDM, the hyperparameter s determines how quickly upper and lower probabilities of events converge as statistical data accumulate. Walley [32] defined s as a number of observations needed to reduce the imprecision (difference between upper and lower probabilities) to half its initial value. Smaller values of s produce faster convergence and stronger conclusions, whereas large values of s produce more cautious inferences. At the same time, the value of s must not depend on n or a number of observations. The detailed discussion concerning the parameter s and the IDM can be found in [1, 32].

Let A be any non-trivial subset of a sample space $\{u_1, ..., u_n\}$, i.e., A is not empty and $A \neq U$, and let $l(A)$ denote the observed number of occurrences of A in N trials, $l(A) = \sum_{u_j \in A} l_j$. Then, according to [32], the predictive probability $P(A|\mathbf{n}, \mathbf{t}, s)$ under the Dirichlet posterior distribution is

$$P(A|\mathbf{n}, \mathbf{t}, s) = \left(l(A) + st(A)\right) / \left(N + s\right),$$

where $t(A) = \sum_{u_j \in A} t_j$.

It should be noted that $P(A|\mathbf{n},\mathbf{t},s)=0$ if A is empty and $P(A|\mathbf{n},\mathbf{t},s)=1$ if $A=U$.

By maximizing and minimizing $P(A|\mathbf{n},\mathbf{t},s)$ over $\mathbf{t}\in S(1,n)$, we obtain the posterior upper and lower probabilities of A:

$$\underline{P}(A|\mathbf{n},s)=l(A)/\left(N+s\right),\ \overline{P}(A|\mathbf{n},s)=\left(l(A)+s\right)/\left(N+s\right). \tag{2}$$

Before making any observations, $l(A)=N=0$, so that $\underline{P}(A|\mathbf{n},s)=0$ and $\overline{P}(A|\mathbf{n},s)=1$ for all non-trivial events A. This is the vacuous probability model. Therefore, by using the IDM, we do not need to choose one specific prior. In contrast, the objective standard Bayesian approach [2] aims at modeling prior ignorance about the chances p by characterizing prior uncertainty by a single prior probability distribution.

4 The imprecise Dirichlet model and the AdaBoost algorithm

Another idea for improving the AdaBoost algorithm is to change the set $\mathcal{P}$ in each iteration, i.e., we construct T sets $\mathcal{P}(t)$, $t=1,...,T$, produced in accordance with a certain rule. This implies that we get different sets of extreme points for each iteration $q_k(t)=(q_1^{(k)}(t),...,q_n^{(k)}(t))$, i.e., $q_k(t)\in\mathcal{E}(\mathcal{P}(t))$, $k=1,...,r(t)$. Moreover, the number of extreme points may also depend on t. Then every probability distribution $h=(h_1,...,h_n)$ from $\mathcal{P}(t)$ can be represented as the linear combination of the extreme points

$$h=\sum_{k=1}^{r(t)}\lambda_k\cdot q_k(t).$$

The next task is to define a justified rule for modifying the sets $\mathcal{P}(t)$. Let us consider how sets $\mathcal{P}(t)$ could be changed by intuition. First of all, the set $\mathcal{P}(t)$ should be reduced with the number of iterations in order to avoid the problem of overfitting. Second, the set $\mathcal{P}(t)$ should move around within the unit simplex of weights by taking into account incorrectly classified examples in order to apply the important procedure of adaptation used in the AdaBoost. Third, its size should depend on the number of classification errors. Fourth, the set $\mathcal{P}(t)$ should be simply determined in order to avoid the hard computation.

All the above properties of sets $\mathcal{P}(t)$ can be satisfied by using the IDM.

Let indicator functions $I_i(t)=I(c_t(x_i)\neq y_i)$, $i=1,...,n$, be possible outcomes for the t-th iteration, $I(t)=\{I_1(t),...,I_n(t)\}$ be a set of possible outcomes $I_i(t)$. The indicator function I_i takes the value 1 when the i-th example is misclassified in the t-th iteration.

We also assume that every example, say the j-th example, has an unknown probability (weight) denoted $\Pr\{I_j\}=p_j$, which can be viewed as the probability of the error. Larger numbers of errors produce their larger probabilities. In turn, larger probabilities of errors produce larger weights of the corresponding examples. The t-th classifier can be regarded as a generator of errors with unknown probabilities $p=(p_1,...,p_n)$. It is not important

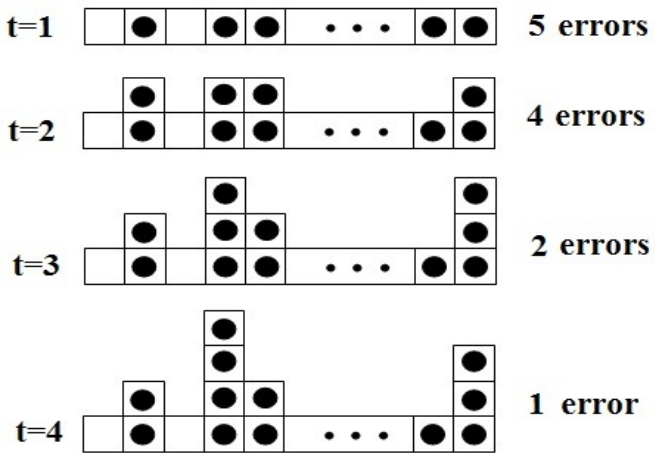

Figure 1: A schematic accumulation process of errors

that every classifier uses different weights of examples for classification. The difference of the weights is a factor of randomness and probabilistic nature of errors.

Let $l(t)$ be the accumulated total number of errors (non-zero indicator functions) after t iterations, $l_j(t)$ denote the accumulated number of misclassifications of the j-th example after t iterations, i.e., $l_i(t) = \sum_{j=1}^{t} I_i(j)$, $\sum_{j=1}^{n} l_j(t) = l(t)$.

A schematic accumulation process of errors after each iteration ($t = 1, ..., 4$) is shown in Fig. 1. The misclassified examples are marked with cells containing black balls. There are 5 misclassified examples after the first iteration, 4 misclassified examples after the second iteration, 2 misclassified examples after the third iteration, 1 misclassified example after the fourth iteration. The corresponding numbers of accumulated errors are $l(1) = 5$, $l(2) = 9$, $l(3) = 11$, $l(4) = 12$.

The above assumptions imply that we can accumulate statistical data about errors with each iteration. Moreover, the random variables $l_1(t), ..., l_n(t)$ have the multinomial distribution. Hence, we can apply the imprecise Dirichlet model to determine the set of probabilities $\mathcal{P}(t)$.

According to the IDM, the probability p_i of the i-th misclassified example after t iterations lies in the interval

$$\frac{l_i(t)}{l(t)+s} \leq p_i \leq \frac{l_i(t)+s}{l(t)+s}. \tag{3}$$

These intervals for all $i = 1, ..., n$ produce the set $\mathcal{P}(t)$. Therefore, the next task is to determine extreme points of the set $\mathcal{P}(t)$.

The result proved in Proposition 1 is well known. However, it is provided for the detailed explanation of an important property of the IDM.

Proposition 1 *The set of probability distributions $\mathcal{P}(t)$ produced by n inequalities (3) has*

Table 1: Extreme points of sets of weights

t	l_1	l_2	l_2	$q_1^{(k)}$	$q_2^{(k)}$	$q_3^{(k)}$
				1	0	0
1	1	0	0	1/2	1/2	0
				1/2	0	1/2
				2/3	1/3	0
2	1	1	0	1/3	2/3	0
				1/3	1/3	1/3
				3/4	1/4	0
3	2	1	0	2/4	2/4	0
				2/4	1/4	1/4
				3/5	1/5	1/5
4	2	1	1	2/5	2/5	1/5
				2/5	1/5	2/5

n extreme points such that the k-th extreme point, $k = 1, ..., n$, is of the form:

$$q_k^{(k)}(t) = \frac{l_k(t) + s}{l(t) + s},\ q_i^{(k)}(t) = \frac{l_i(t)}{l(t) + s},\ i = 1, ..., n,\ i \neq k.$$

Proof. Denote the lower and upper bounds for p_i from (3) as $L_i(t)$ and $U_i(t)$, respectively. Constraints producing the set $\mathcal{P}(t)$ have n variables. Therefore, every extreme point is produced by n equalities, including the equality $p_1 + ... + p_n = 1$. Without loss of generality, we suppose that $n-1$ equalities with numbers $1, ..., n-1$ are of the form: $p_i = L_i(t)$. Then the n-th inequality is

$$1 - \frac{\sum_{i=1}^{n-1} l_i(t)}{l(t) + s} = \frac{l(t) - \sum_{i=1}^{n-1} l_i(t) + s}{l(t) + s} = \frac{l_n(t) + s}{l(t) + s} = U_n(t).$$

We get $p_n = U_n(t)$. This implies that the k-th extreme point contains one element $p_k = U_k(t)$ and $n-1$ elements $p_i = L_i(t)$, $i \neq k$.

Let us prove that every extreme point can not contain larger than one element $U_k(t)$ being the upper bound. Suppose now that two elements of an extreme point with numbers $n-1$ and n are $U_{n-1}(t)$ and $U_n(t)$, respectively, and $n-3$ elements are $L_i(t)$, $i = 2, ..., n$. Then the first element is

$$p_1 = \frac{l(t) + s - s - s - \sum_{i=2}^{n} l_i(t)}{l(t) + s} = \frac{l_1(t) - s}{l(t) + s} < L_1(t).$$

We get the contradiction. The same contradiction takes place when we consider an arbitrary number of equalities $p_k = U_k(t)$. This implies that every extreme point contains only one element $U_k(t)$, as was to be proved. ■

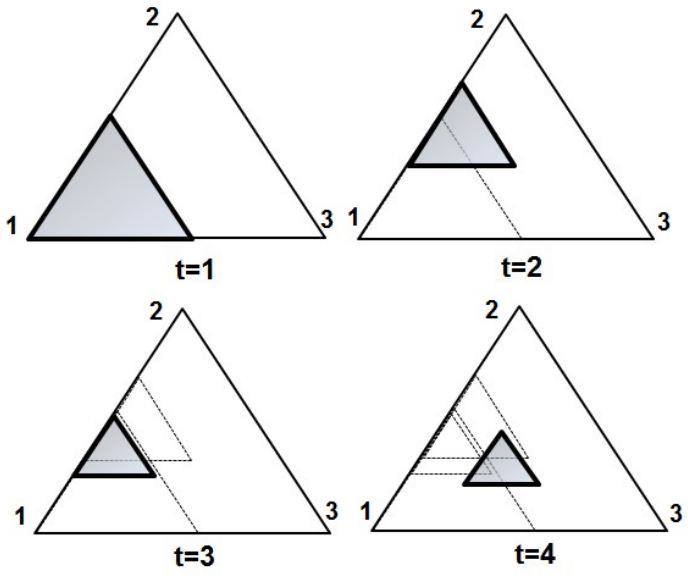

Figure 2: An illustration of set modifications with iterations

A visual illustrative example of modifying the set $\mathcal{P}(t)$ can be provided by using the training set consisting of three examples, i.e., $n = 3$. Numbers of accumulated errors $l_1(t)$, $l_2(t)$, $l_3(t)$ and extreme points $\mathcal{E}(\mathcal{P}(t))$ are shown in Table 1. The hyperparameter is taken $s = 1$. It is shown in Fig. 2 how the shaded triangle moves around within the unit simplex $S(1, n)$ and how it changes its size with respect to the iterations of the boosting algorithm or to the number of accumulated errors. Places of the former sets $\mathcal{P}(t_0)$, $t_0 < t$, are shown in Fig. 2 by means of triangles with dashed thin sides. It can be seen from Fig. 2 and Table 1 that the sets $\mathcal{P}(t)$ obtained by means of the IDM satisfy conditions stated at the beginning of this section.

Finally, we can write the algorithm called the IDMBoost which is represented as Algorithm 3.

One of the important properties of the IDM is that we do not need to choose one specific prior. This implies that we have prior ignorance about probabilities of events, which can be represented by the unit simplex. In other words, we start from the unit simplex accepted in the standard AdaBoost before iteration procedures and before getting the classification errors. This is a common point with the AdaBoost.

We get the algorithm having a double adaptation procedure. The first adaptation is carried out within every simplex. It is similar to the same procedure in the AdaBoost and differs only by modifying the weights $\lambda_k(t)$ instead of $h_i(t)$ in the AdaBoost. The second adaptation procedure is due to the IDM. This procedure adapts the set of possible weights of examples to the classification errors.

It turns out that the iterative updating of the set of weights $\mathcal{P}(t)$ is an adaptation

Algorithm 3 The IDMBoost algorithm

Require: T (number of iterations), S (training set), s (hyperparameter of the IDM)

Require: $q_1(0) = (1, 0, ..., 0)$, $q_2(0) = (0, 1, ..., 0), ..., q_n(0) = (0, 0, ..., 1)$ (prior extreme points of $\mathcal{P}$)

Ensure: $c_t, \alpha_t,\ t = 1, ..., T$

$t \leftarrow 1$

$\lambda_k(1) \leftarrow 1/r$; $k = 1, ..., r$, or it can be randomly selected from the unit simplex

repeat

Compute the vector $h \leftarrow \sum_{k=1}^{r} \lambda_k(t) \cdot q_k(t)$

Train classifier c_t using distribution h

Compute indicator functions $I_i(t) = I(c_t(x_i) \neq y_i)$, $i = 1, ..., n$.

Compute error rate $\varepsilon_k \leftarrow \sum_{i=1}^{n} q_i^{(k)}(t) I_i(t)$

Normalize ε_k to be a proper distribution

Compute error $e(t) \leftarrow \sum_{k=1}^{r} \lambda_k \varepsilon_k$

if $e(t) > 0.5$ **then**

$T \leftarrow t - 1$

exit Loop

end if

$\alpha_t \leftarrow \frac{1}{2} \ln\left(\frac{1-e(t)}{e(t)}\right)$

$\lambda_k(t+1) \leftarrow \lambda_k(t) \cdot \exp\left(-\alpha_t \cdot (1 - 2\varepsilon_k)\right)$

Normalize $\lambda(t+1)$ to be a proper distribution

Compute accumulated numbers of misclassifications $l_i(t) = \sum_{j=1}^{t} I_i(j)$, $i = 1, ..., n$, $\sum_{j=1}^{n} l_j(t) = l(t)$

Compute n extreme points $q_k(t) = (q_1^{(k)}(t), ..., q_n^{(k)}(t))$, $k = 1, ..., n$, where $q_k^{(k)}(t) = \frac{l_k(t)+s}{l(t)+s}$, $q_i^{(k)}(t) = \frac{l_i(t)}{l(t)+s}$, $i = 1, ..., n$, $i \neq k$

until $t > T$

procedure itself because it moves around within the unit simplex of weights by taking into account incorrectly classified examples. If we consider a point in the $\mathcal{P}(t)$, for example, with equal weights $\lambda(t)$, then its weight $h_i(t)$ is increased when the i-th example is misclassified. If the i-th example is correctly classified, then the weight $h_i(t)$ is decreased. The subset $\mathcal{P}(t)$ on the whole moves towards extreme points corresponding to misclassified examples. This adaptation procedure of $\mathcal{P}(t)$ is similar to the adaptation procedure implemented in the AdaBoost. However, in contrast to the AdaBoost adaptation procedure where weights of examples tend to concentrate on "hard" examples, the proposed procedure based on the IDM "smooths" this tendency due to moving subsets of weight vectors within the unit simplex, but not vectors directly. At the same time, we also adapt vectors of weights as in the AdaBoost, but this adaptation is carried out within small subsets $\mathcal{P}(t)$. This again leads to the avoiding the overfitting as it has been shown in the EPIBoost.

The choice of misclassified examples after every iteration for constructing the IDM model gives us an important *invariance property* of the proposed IDMBoost which is formulated as follows. The IDMBoost is invariant to the number of examples in a training set. The set of weights $\mathcal{P}(t)$ depends only on the accumulated total number of errors $l(t)$ after t iterations and the hyperparameter s of the IDM.

Remark 1 *It should be noted that the standard AdaBoost is a special case of the IDMBoost when the parameter s is very large, i.e., $s \to \infty$. In this case, the set $\mathcal{P}(t)$ is not changed. It is the unit simplex. The case $s = 0$ is also interesting. The set $\mathcal{P}(t)$ is reduced to a single probability distribution with the k-th probability $l_k(t)/l(t)$, $k = 1, ..., n$. The point in the unit simplex of weights corresponding to the probability distribution moves around within the simplex only by taking into account numbers of errors. The case $s \to 0$ can be also regarded as a special case of the AdaBoost, but the rule for updating weights in this case is written as follows:*

$$h_i(t+1) = h_i(t)\frac{l(t)}{l(t+1)} + \begin{cases} 0, & c_t(x_i) = y_i, \\ 1/l(t+1), & c_t(x_i) \neq y_i. \end{cases}$$

It is simply to prove that there holds $h_1(t+1) + ... + h_n(t+1) = 1$. Suppose that we have r misclassified examples with numbers from the index set $J \subset \{1, ..., n\}$ after the t-th iteration. Then all probabilities $h_i(t+1)$, $i \notin J$, are decreased because there holds $l(t+1) = l(t) + r < l(t)$. This implies that all probabilities $h_i(t+1)$, $i \in J$, are increased due to condition of the probability sum. So, the weight of the i-th example is decreased when this example is correctly classified. At the same time, the weight of the i-th example is increased when the example is misclassified. The same tendency is observed in the standard AdaBoost.

Remark 2 *The sets of probabilities obtained by means of the IDM can also be considered in the framework of an ε-contaminated model [16]. As pointed out by Seidenfeld and Wasserman in the discussion part of Walley's paper [32], the IDM has the same lower and upper probabilities as the ε-contaminated model (a class of probabilities which for fixed*

$\varepsilon \in (0,1)$ and p_i is the set $\{(1-\varepsilon)p_i + \varepsilon q_i\}$). Here $p_i = c_i/N$, $q_1, ..., q_n$ is an arbitrary distribution or an arbitrary point in the unit simplex, c_i is the number of occurences of the i-th event. There is also the connection between parameters of the models which is of the form: $\varepsilon = s/(N+s)$.

It is important to note that the IDMBoost deals with accumulated errors and it does not depend on the number of classes. In fact, every example in the IDMBoost is regarded as misclassified or correctly classified. This implies that the IDMBoost can be simply extended on some multiclass variants. We consider one of the possible multiclass variants, which is based on the well-known AdaBoost.M1 algorithm proposed by Freund and Schapire [13] as a multiclass extension of the AdaBoost. The AdaBoost.M1 formally differs from AdaBoost by the rule for updating weights $h(t)$ and by the final decision rule which are of the form:

$$h_i(t+1) = h_i(t) \cdot \begin{cases} \exp(-\alpha_t), & c_t(x_i) = y_i, \\ 1, & c_t(x_i) \neq y_i. \end{cases},$$

$$c(\mathbf{x}) = \arg \max_{y \in \{1,...,K\}} \left(\sum_{i=1}^{T} \alpha_t I(c_t(\mathbf{x}) = y \right),$$

respectively. Here we suppose that $y_i \in \{1, ..., K\}$.

It can be seen from the above that the number of classes is not important for the main part of the AdaBoost.M1 algorithm except for the final decision rule. The above updating rule takes into account only misclassified or correctly classified examples and does not consider types of errors, i.e., their membership to classes. Therefore, the EPIBoost and the IDMBoost can be extended on the multiclass versions by changing the rule for updating weights λ. It follows from the EPIBoost [29] that comparison of the k-th extreme point misclassification rate ε_k and the k-th extreme point correct classification rate $1 - \varepsilon_k$ plays the same role as the condition $c_t(x_i) = y_i$. Hence, the rule for updating weights λ can be written as follows:

$$\lambda_k(t+1) = \lambda_k(t) \cdot \begin{cases} \exp(-\alpha_t), & \varepsilon_k \leq 1 - \varepsilon_k, \\ 1, & \varepsilon_k > 1 - \varepsilon_k. \end{cases}$$

By substituting the above rule into Algorithm 3, we get a multiclass version of the IDMBoost corresponding to the AdaBoost.M1.

5 Numerical experiments

We illustrate the method proposed in this paper via several examples, all computations have been performed using the statistical software R [26]. We investigate the performance of the proposed method and compare it with the standard AdaBoost by considering the accuracy measure (ACC), which is the proportion of correctly classified cases on a sample of data, i.e., ACC is an estimate of a classifier's probability of a correct response. This

measure is often used to quantify the predictive performance of classification methods. It is an important statistical measure of the performance of a binary classification test. It can formally be written as $ACC = N_T/N$. Here N_T is the number of test data for which the predicted class for an example coincides with its true class, and N is the total number of test data.

We will denote the accuracy measure for the proposed IDMBoost algorithm as ACC_{IDM}, for the standard AdaBoost as ACC_{st}, for the EPIBoost as ACC_{EPI}.

The proposed method has been evaluated and investigated by publicly available data sets from the UCI Machine Learning Repository [12]. A brief introduction about these data sets are given in Table 2, while more detailed information can be found from, respectively, the data resources. Table 2 shows the number of features m for the corresponding set and numbers of examples in negative n_0 and positive n_1 classes, respectively.

It should be noted that the first two classes (carcinoma, fibro-adenoma) in the Breast Tissue data set are united and regarded as the negative class. Two classes (disk hernia, spondylolisthesis) in the Vertebral Column 2C data set are united and viewed as the negative class. The class CYT (cytosolic or cytoskeletal) in Yeast data set is regarded as negative. Other classes are united as the positive class. The first class in the Seeds data set is viewed as negative. Rosa and Canadian varieties of wheat in the Seeds data set are united. The largest class "cytoplasm" (143 examples) in the Ecoli data set is regarded as the negative classes, other classes are united. The second and the third classes in the Wine data set are united. In the Image Segmentation data set, we unite all classes except for the first class "brickface" into the positive class. The class "brickface" is accepted as negative.

The number of instances for training will be denoted as n. Moreover, we take $n/2$ examples from every class for training. These are randomly selected from the classes. The remaining instances in every data set are used for validation. It should be noted that the accuracy measures are computed as average values by means of the random selection of n training examples from the data sets many times.

In all these examples, the logistic regression models as weak classifiers are used, i.e., we solve the following optimization problem:

$$\min_{(\beta_0,\beta)\in\mathbb{R}^{m+1}} - \left[\frac{1}{n}\sum_{i=1}^{n} y_i(\beta_0 + \mathbf{x}_i^{\mathrm{T}}\beta) - \log\left(1 + e^{(\beta_0+\mathbf{x}_i^{\mathrm{T}}\beta)}\right)\right] + \lambda \left\|\beta\right\|_2^2/2.$$

Here β_0, β are parameters of the regression, λ is the tuning parameter. The logistic regression can be regarded as a special case of a general problem solved by means of the R-package "glmnet" developed by Friedman et al. [15]. The tuning parameter λ is computed by using the function cv.glmnet() with 5-fold cross validation.

The IDMBoost algorithm evaluation aims to investigate the performance of the proposed algorithm by different values of the parameter s and to compare it with the standard AdaBoost. Therefore, experiments are performed to evaluate the effectiveness of the proposed method under different values of s (mainly $s = 1, ..., 8$) and different numbers of training examples n.

Table 2: A brief introduction about data sets

Data sets	m	n_0	n_1
Pima Indian Diabetes	8	268	500
Mammographic masses	5	516	445
Parkinsons	22	147	48
Breast Tissue	9	36	70
Indian Liver Patient	10	416	167
Breast Cancer Wisconsin Original	9	458	241
Seeds	7	70	140
Ionosphere	34	225	126
Ecoli	7	143	193
Vertebral Column 2C	6	210	100
Yeast	8	463	1021
Blood transfusion	4	570	178
Steel Plates Faults	27	1783	158
Fertility Diagnosis	9	12	88
Wine	13	59	119
QSAR biodegradation	41	356	699
Seismic bumps	15	2414	170
Image Segmentation	19	90	120
Spambase	57	1813	2788

Table 3: The largest differences Δ by different n

	n			
Data sets	20	40	60	100
Pima Indian Diabetes	0.181	0.030	0.022	0.005
Mammographic masses	0.046	0.094	0.019	0.006
Indian Liver Patient	0.154	0.157	0.009	0.003
Breast Tissue	0.043	0.046	−0.025	–
Parkinsons	0.408	0.004	0.001	−0.011
Breast Cancer Wisconsin Original	0.128	0.061	0.190	0.244
Vertebral Column 2C	0.076	−0.001	−0.012	−0.024
Blood Transfusion	0.225	0.049	0.015	−0.006
Seeds	0.146	0.205	0.050	−0.037
Ionosphere	0.282	0.269	0.248	0.109
Ecoli	0.115	0.154	0.195	0.085
Steel Plates Faults	0.437	−0.049	−0.11	−0.058
Wine	0.009	0.069	0.067	0.07
Yeast	0.003	−0.019	−0.010	0.025
QSAR biodegradation	0	0	−0.013	−0.007
Seismic bumps	0.781	0.526	0.472	0.25
Banknote	−0.004	0.082	0.161	0.137
Image Segmentation	0.085	0.050	0.050	0.012
Spambase	0	0	0.116	0.162

In order to compare the IDMBoost and the standard AdaBoost, we compute the largest difference between accuracies of the IDMBoost and the AdaBoost by different s, i.e., we compute differences $\Delta = \max_s ACC_{\text{IDM}} - ACC_{\text{st}}$ for various data sets and for different values of n. The positive value of Δ indicates that the proposed algorithm outperforms the standard AdaBoost at least for one value of s from the set $s = 1, ..., 8$. Negative value says that the standard AdaBoost is better in comparison with the proposed algorithms. We analyze the IDMBoost and the AdaBoost by using $T = 10$ iterations. The computation results are given in Table 3. One can see from the table that the proposed algorithm outperforms the AdaBoost for many data sets analyzed especially by small numbers of examples in training sets. A value of Δ for the Breast Tissue data set is missing because the data set has only 36 training examples in the negative class (see Table 2). The large values of Δ for some training data, for example, for the Parkinsons, Steel Plates Faults data sets by $n = 20$ stem from the fact that the AdaBoost with the weak classifier in the form of the logistic regression provides unsatisfactory results.

Examples of dependence of the IDMBoost accuracy on values of the hyperparameter s are shown in Figs. 3-5. One can see from the figures that there are some optimal values of s for several examples providing the largest value of the accuracy measure ACC_{IDM}.

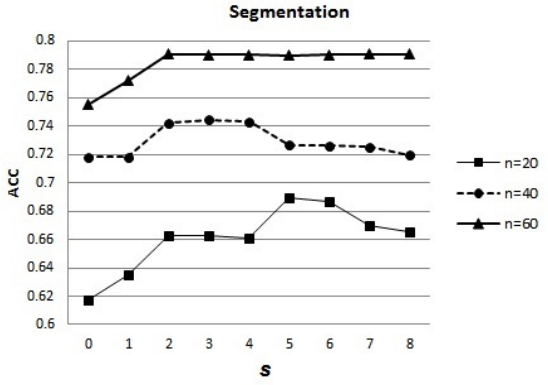

Figure 3: Accuracy by different s and n for the Image Segmentation data set

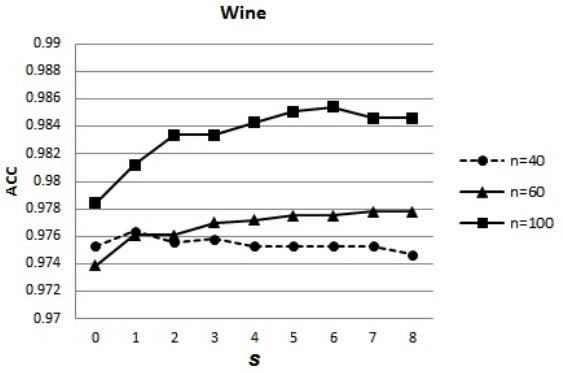

Figure 4: Accuracy by different s and n for the Wine data set

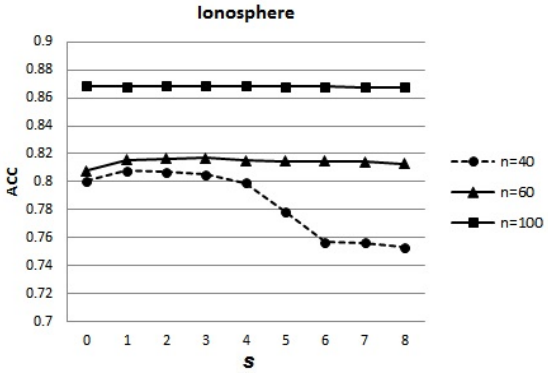

Figure 5: Accuracy by different s and n for the Ionosphere data set

Table 4: The t-test by different n for the IDMBoost

n	t	p-value	95%-CI
20	3.595	0.0021	$[0.068, 0.260]$
40	2.984	0.0079	$[0.027, 0.155]$
60	2.519	0.0214	$[0.013, 0.139]$
100	2.457	0.0251	$[0.008, 0.100]$

In order to formally show the outperforming of the proposed algorithm in comparison with the standard AdaBoost, we apply the t-test which has been proposed and described by Demsar [9] for testing whether the average difference in the performance of two classifiers is significantly different from zero. Since we use the differences Δ, then we compare them with 0 which represents the standard AdaBoost. If we denote the performance score of a classifier Δ_i on the i-th out of M data sets, then the t statistics is computed as $\overline{\Delta}\sqrt{n}/\sigma$. Here $\overline{\Delta}$ is the mean value of scores, i.e., $\overline{\Delta} = \sum_{i=1}^{M} \Delta_i/M$; σ is the standard deviation of scores. The t statistics is distributed according to the Student distribution with $M-1$ degrees of freedom. The results of computing the t statistics, p-values and the corresponding 95 percent confidence intervals (95%-CI) on the basis of Table 3 are given in Table 4.

It can be seen from Table 4 that the IDMBoost provides better results in comparison with the standard AdaBoost. At the same time, it follows from the case $n = 100$ that the advantage of the proposed algorithm may be under a question when the training set increases. This implies that the algorithm can be especially useful when small training sets are available.

Another question is how the number of iterations T impacts on the classification performance. Differences Δ by different values of $T = 10, 100, 200$ and by different values of n for Ecoli data set are shown in Table 5. It can be seen from the table that the differences mainly decrease as T increases. Similar results are given in Table 6 where the same measures are provided for the Image Segmentation data set. Moreover, one can clearly see from Table 6 that the AdaBoost outperforms the IDMBoost by $T = 200$. However, it should be pointed out here that the accuracy ACC_{IDM} of the IDMBoost is computed only by $T = 10$ and the computed measures are compared with accuracy measures ACC_{st} of the AdaBoost by $T = 10, 100, 200$. This implies that the accuracy achieved by means of the AdaBoost with 100 or 200 iterations can be obtained by using the IDMBoost with $T = 10$ iterations. The same results are shown in Table 7 where values of Δ are provided for the Banknote data set. However, we face in this example with the problem of overfitting of the AdaBoost by large values of iterations. One can see from Table 7 that values of Δ increase as T increases by $n = 20$.

It is interesting to compare the IDMBoost and the EPIBoost. We use the imprecise ε-contaminated model for implementing the EPIBoost [29]. Moreover, we compute its classification accuracy by taking the following values of ε: 0.1, 0.2, 0.3, 0.4. In order to compare the IDMBoost and the EPIBoost, we compute the difference between the largest accuracies of the IDMBoost by different s and the EPIBoost by different ε, i.e., we compute

Table 5: Values of Δ for the Ecoli data set by different n and T

T	n			
	20	40	60	100
10	0.115	0.154	0.195	0.085
100	0.069	0.097	0.160	0.103
200	0.017	0.083	0.157	0.092

Table 6: Values of Δ for the Image Segmentation data set by different n and T

T	n			
	20	40	60	100
10	0.085	0.050	0.049	0.012
100	0.007	0.075	0.013	0.007
200	-0.011	0.039	-0.009	-0.013

Table 7: Values of Δ for the Banknote data set by different n and T

T	n			
	20	40	60	100
10	-0.004	0.082	0.161	0.137
100	0.028	0.100	0.031	0.071
200	0.037	0.009	0.036	0.174

Table 8: Differences δ by different n

	n			
Data sets	20	40	60	100
Pima Indian Diabetes	0.028	0.011	−0.001	−0.001
Mammographic masses	0.013	0.009	0.005	0
Indian Liver Patient	0.081	0.038	0.029	0.001
Breast Tissue	0.140	0.056	0	–
Parkinsons	0.213	0.040	0.006	0.001
Breast Cancer Wisconsin Original	0.012	0.004	0.001	0
Vertebral Column 2C	0.084	−0.014	−0.010	−0.005
Blood Transfusion	0.151	0.017	0.021	0.004
Seeds	0.084	0.011	0.002	−0.007
Ionosphere	0.113	0.016	−0.020	0.001
Ecoli	0.051	0.063	0.02	0
Steel Plates Faults	0.234	−0.064	−0.031	0.001
Wine	0.010	0.003	0.003	0.292
Yeast	−0.002	−0.021	−0.004	0.005
QSAR biodegradation	0	0	−0.003	0
Seismic bumps	0.256	0.121	0.504	0.004
Banknote	0.075	0.005	0.001	−0.002
Image Segmentation	0.012	0.031	0.020	0.002
Spambase	0	0	−0.003	−0.005

$\delta = \max_s ACC_{\text{IDM}} - \max_\varepsilon ACC_{\text{EPI}}$ for various data sets and for different values of n. The computation results are given in Table 8. One can see from the table that the proposed algorithm outperforms the EPIBoost for many data sets analyzed especially by $n = 20$ and 40.

6 Conclusion

In this paper we have presented a new algorithm for ensemble construction called the IDMBoost and based on adaptive restricting a set of weights of examples in training data by applying the imprecise Dirichlet model. The IDMBoost algorithm aims to avoid the problem of overfitting and to use the additional adaptation taking into account the number of examples in the training set, the accumulated number and location of classification errors. One of the ideas underlying the algorithm is updating the probability distribution of examples within the modifying restricted sets. It is carried out by replacing the probability distributions of examples by weights in the linear combination of extreme points of the restricted sets, which, in turn, can move within the unit simplex.

The IDMBoost is rather simple from the computation point of view. In fact, it differs

from the standard AdaBoost in the following steps: computing the probability distributions h, computing the error rate ε_t, computing the numbers of accumulated misclassifications and computing the extreme points of restricted sets of weights. This does not mean that other steps coincide with the standard AdaBoost algorithm. They are just similar from the complexity point of view.

The various numerical experiments with real data have illustrated the outperforming of the proposed algorithm and its ability to reduce the number of iterations. On the one hand, this reduction can decrease the computation time for classification. On the other hand, it allows us to avoid the problem of overfitting. The experimental results also showed that there are certain values of the hyperparameter s of the IDM providing the largest accuracy. The above says that the IDMBoost algorithm might outperform the standard AdaBoost by a suitable choice of s. Unfortunately, the optimal parameter s can be obtained only by considering all possible values in a predefined grid.

It should be pointed out also the important difference between the EPIBoost and the IDMBoost. The EPIBoost additionally to parameters of weak classifiers requires to choose the corresponding imprecise statistical model (the ε-contaminated model, the pari-mutuel model, the constant odds-ratio model, bounded vacuous and bounded ε-contaminated robust models) and its parameters for restricting the set of weights. The choice of the imprecise model is crucial for the classification accuracy and for avoiding the overfitting. The IDMBoost depends only on the hyperparameter s of the IDM. It is self-adjusted due to the iterative updating of subsets of weights within the unit simplex of weights.

We have studied only the modification of the AdaBoost method. However, one can see from the proposed approach that it can be extended on many boosting-like models whose detailed review is given, for example, by Ferreira and Figueiredo [11]. The quality and efficiency of new modifications are a direction for future research. Another direction for future research is to use one of the imprecise statistical models applied in the EPIBoost before iterations as an initial condition restricting the weights of examples and then to modify it after every iteration by exploiting the imprecise Bayesian inference. In this case, the EPIBoost becomes to be a basis for new imprecise classification models.

Acknowledgement

I would like to express my appreciation to the anonymous referees whose very valuable comments have improved the paper. The reported study was partially supported by the Ministry of Education and Science of Russian Federation and by RFBR, research project No. 14-01-00165-a.

References

[1] J.-M. Bernard. An introduction to the imprecise Dirichlet model for multinomial data. *International Journal of Approximate Reasoning*, 39(2-5):123–150, 2005.

[2] J.M. Bernardo and A.F.M. Smith. *Bayesian Theory*. Wiley, Chichester, 1994.

[3] L. Breiman. Bagging predictors. *Machine Learning*. 24(2): 122-140, 1996.

[4] L. Breiman. Random forests. *Machine Learning*. 45: 5–32, 2001

[5] T. Bylander and L. Tate. Using validation sets to avoid overfitting in AdaBoost. In *Proceedings of the 19th International FLAIRS Conference*, pages 544–549, 2006.

[6] G. Corani and A. Antonucci. Credal ensembles of classifiers. *Computational Statistics & Data Analysis*, 71: 818-831, 2014.

[7] R.A. Dara, M.S. Kamel, and N. Wanas. Data dependency in multiple classifier systems. *Pattern Recognition*, 42(7): 1260-1273, 2009.

[8] M. DeGroot. *Optimal Statistical Decisions*. McGraw-Hill, New York, 1970.

[9] J. Demsar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*. 7: 1–30, 2006.

[10] C. Domingo and O. Watanabe. MadaBoost: A modification of AdaBoost. In *Proceedings of the Thirteenth Annual Conference on Computational Learning Theory*, pages 180–189, San Francisco, 2000. Morgan Kaufmann.

[11] A.J. Ferreira and M.A.T. Figueiredo. Boosting algorithms: A review of methods, theory, and applications. In C. Zhang and Y. Ma, editors, *Ensemble Machine Learning: Methods and Applications*, pages 35–85. Springer, New York, 2012.

[12] A. Frank, A. Asuncion. UCI machine learning repository, 2010. URL `http://archive.ics.uci.edu/ml`

[13] Y. Freund and R.E. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on Machine Learning*, vol. 96, pp. 148-156. 1996.

[14] Y. Freund and R.E. Schapire. A decision theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[15] J.H. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2010.

[16] P.J. Huber. *Robust Statistics*. Wiley, New York, 1981.

[17] R. Jin, Y. Liu, L. Si, J. Carbonell, and A. Hauptmann. A new boosting algorithm using input-dependent regularizer. In *Proceedings of Twentieth International Conference on Machine Learning (ICML 03)*, pages 1–9, Washington DC, 2003. AAAI Press.

[18] L.I. Kuncheva. *Combining Pattern Classifiers: Methods and Algorithms*. Wiley-Interscience, New Jersey, 2004.

[19] A. Lorbert, D. M. Blei, R. E. Schapire, and P. J. Ramadge. A Bayesian boosting model. *ArXiv e-prints*, 2012.

[20] D. Mease and A. Wyner. Evidence contrary to the statistical view of boosting. *Journal of Machine Learning Research*, 9:131–156, 2008.

[21] M. Nakamura, H. Nomiya, and K. Uehara. Improvement of boosting algorithm by modifying the weighting rule. *Annals of Mathematics and Artificial Intelligence*, 41:95–109, 2004.

[22] R. Pelessoni, P. Vicig, and M. Zaffalon. Inference and risk measurement with the pari-mutuel model. *International Journal of Approximate Reasoning*, 51(9):1145–1158, 2010.

[23] R. Polikar. Ensemble learning. In C. Zhang and Y. Ma, editors, *Ensemble Machine Learning: Methods and Applications*, pages 1–34. Springer, New York, 2012.

[24] G. Ridgeway. *GBM 1.5 package manual*, 2005.

[25] L. Rokach. Ensemble-based classifiers. *Artificial Intelligence Review*, 33(1-2):1–39, 2010.

[26] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2005.

[27] R.E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

[28] R.A. Servedio. Smooth boosting and learning with malicious noise. *Journal of Machine Learning Research*, 4(9):633–648, 2003.

[29] L.V. Utkin. An imprecise boosting-like approach to classification. *International Journal of Pattern Recognition and Artificial Intelligence*, 27(8):1–24, 2013.

[30] L.V. Utkin and A. Wiencierz. An imprecise boosting-like approach to regression. In Fabio Cozman, Therry Denoeux, Sébastien Destercke, and Teddy Seidfenfeld, editors, *ISIPTA '13, Proceedings of the Eighth International Symposium on Imprecise Probability: Theories and Applications*, pages 345–354. SIPTA, 2013.

[31] P. Walley. *Statistical Reasoning with Imprecise Probabilities*. Chapman and Hall, London, 1991.

[32] P. Walley. Inferences from multinomial data: Learning about a bag of marbles. *Journal of the Royal Statistical Society, Series B*, 58:3–57, 1996.

[33] D.H. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.

[34] T. Zhang and B. Yu. Boosting with early stopping: Convergence and consistency. *Annals of Statistics*, 33(4):1538–1579, 2005.

[35] Z.-H. Zhou. Ensemble learning. *Encyclopedia of Biometrics*. Springer US, 270-273, 2009.