Dynamic Self-paced Sampling Ensemble for Highly Imbalanced and Class-overlapped Data Classification

Fang Zhou · Suting Gao · Lyu Ni · Martin Pavlovski · Qiwen Dong · Zoran Obradovic · Weining Qian

Abstract Datasets with imbalanced class distribution are available in various real-world applications. A great number of approaches has been proposed to address the class imbalance challenge, but most of these models perform poorly when datasets are characterized with high class imbalance, class overlap and low data quality. In this study, we propose an effective meta-framework for high imbalance overlapped classification, called *DAPS* (*DynAmic self-Paced sampling enSemble*), which (1) leverages reasonable and effective sampling to maximize the utilization of informative instances and to avoid serious information loss and (2) assigns proper instance weights to address the issues of noisy data. Furthermore, most of the existing canonical classifiers (e.g. Decision Tree, Random Forest) can be integrated in *DAPS*. The comprehensive experimental results on both synthetic and three real-world datasets show that the *DAPS* model could obtain considerable improvements in F1-score when compared to a broad range of published models.

Keywords Dynamic self-paced sampling · Highly class imbalance · Class-overlapped data

1 Introduction

Imbalanced classification is the problem of classifying datasets with skewed class distributions and has gained increasing attention from researchers. The

This research was supported in part by NSFC grant 61902127 and Natural Science Foundation of Shanghai 19ZR1415700.

F. Zhou and S. Gao contributed equally to this work.

Fang Zhou, Suting Gao, Lyu Ni, Qiwen Dong, Weining Qian
School of Data Science and Engineering, East China Normal University, Shanghai, China.

Martin Pavlovski, Zoran Obradovic
Center for Data Analytics and Biomedical Informatics, Temple University, PA, USA
Corresponding author: Fang Zhou, E-mail: fzhou@dase.ecnu.edu.cn

distribution of instances across classes may involve a slight class bias up to a severe class imbalance in various real-world applications, such as fraud detection [27], credit risk prediction [19] and rare disease prediction [29]. Imbalanced classification poses a challenge to the existing predictive models as most of them were designed under the assumption that classes are uniformly distributed. Besides an imbalanced class ratio [8,14,28], overlap between classes [21] and presence of noise are also crucial factors affecting the predictive capability of classification models.

A plethora of approaches have been proposed for the imbalanced binary classification problem and are focused on diminishing the effect of imbalanced class ratio. For example, both data-level approaches [3,9,11,21] and ensemble-based models [14,16,18] modify the class distribution by generating (or removing) instances. However, they may be inapplicable to highly imbalanced datasets, as they are prone to overfitting or suffer from enormous computational cost, or may even lead to losing useful information. Algorithm-level methods [13,15] assign a heavier weight on all minority class instances that are misclassified. While a straightforward solution is to determine the weight value based on the imbalance ratio calculated from the available data, it might not be representative of the true underlying class distribution.

Recently, a few studies have shown that the performance of classifiers is affected more by the "amount" of class overlap rather than by the imbalanced class ratio [21,22,23]. Such studies address the overlap problem by removing negative instances from the overlap region without re-balancing the class distribution. This may cause a loss of informative instances from the majority class, especially when the overlap region is large. Furthermore, in real-world applications, datasets often contain noise or outliers, which affects a classifier' performance. Therefore, many existing approaches are limited in their modeling capacity to handle data characterized by *high class imbalance*, *class overlap* and a certain degree of *noise*.

In this work, we propose an effective meta-framework called **DAPS** (*DynAmic self-Paced sampling enSemble*) for the high imbalance overlapped binary classification problem. The framework contains two major steps, *dynamic self-paced sampling* and *instance weighting*. The goal of *dynamic self-paced sampling* is to address issues including information loss and high computational complexity caused by generating balanced datasets. The benefits of this mechanism are twofold. First, it maximizes the utilization of informative instances and prevents information loss by progressively transforming class distributions from imbalanced to balanced, acting as a more effective alternative to previous sampling techniques for coping with class imbalance. Second, a self-paced procedure is able to expand the model's generalization capability, by focusing initially on easy-to-learn instances and gradually incorporating hard-to-classify instances. The goal of *instance weighting* is to exploit valuable instances in the class overlap region in order to enhance the model's predictive capability while minimizing the effect of noisy data. The proposed instance weighting mechanism does not require any prior knowledge, which allows for integrating any standard classifier in the framework.

The main contributions of this work are summarized as follows:
(1) We propose an effective framework, *DAPS*[1], for classification of highly-imbalanced, class-overlapped and noisy data. Any canonical classifiers that provide soft or probabilistic outcomes, such as Decision Tree, Random Forest and GBDT [7], can be integrated into *DAPS*. Experimental results on both synthetic and real-world datasets validate the effectiveness of *DAPS*. Compared with the existing methods, *DAPS* is much more accurate and robust.
(2) We design a *dynamic self-paced sampling* mechanism to gradually transform a class distribution from imbalanced to balanced and sample instances based on their classification difficulty. This reduces the impact of the imbalance ratio effectively.
(3) We selectively assign weights to instances to minimize the impact of noise and leverage valuable instances lying near the decision boundary.

2 Related work

The existing approaches on imbalanced data classification can be categorized into three groups: data-level, algorithm-level and ensemble-based.

Data-level: This type of approaches aims to obtain a balanced class distribution through a preprocessing step known as resampling. The basic techniques include either removing instances from the majority class [21] or generating new instances for the minority class [3, 11].

Algorithm-level: This method class modifies the objective functions of conventional classifiers by incorporating users' preferences [13, 15]. Among this group of methods, the most popular subgroup is cost-sensitive learning [13], which assigns large and small costs to minority and majority classes, respectively, to avoid bias.

Ensemble-based: This group of methods leverages data-level or algorithm-level techniques into an ensemble scheme to build a strong classifier [14, 16, 18, 24]. Since it makes the class distribution balanced, any traditional classifiers can be subsequently applied. Due to their outstanding performance, ensemble methods became popular in real-world applications.

The existing approaches still have limitations on extremely imbalanced and class-overlapped datasets. Under-sampling may lead to severe information loss, as it only considers a small portion of instances from the majority class. Furthermore, it cannot guarantee that the selected instances are informative. Over-sampling may increase the computational cost and may lead to overfitting, as it replicates a large number of instances from the minority class. The cost matrix in cost-sensitive learning is designed by domain experts or learned by other approaches [13], thus the algorithm-level methods cannot be generalized to other tasks.

Another line of research relevant to this work is self-paced learning. Self-paced learning [10] is focused on finding informative instances and is not designed for the imbalanced classification problem. The core idea is to learn

[1] The code is available at https://github.com/ZhouF-ECNU/DAPS.

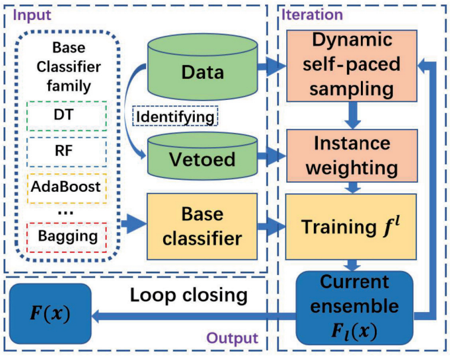


Fig. 1: Dynamic Self-Paced Ensemble Process.

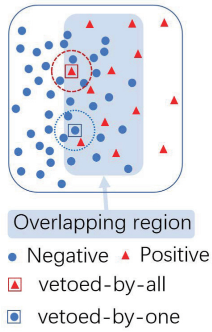


Fig. 2: Overlap region approximation.

easier aspects of a given task or easier subtasks first, and then gradually increase the difficulty level. One recent work [14] adopted self-paced learning for the imbalanced classification problem. However, since their model Self-paced Ensemble (SPE) utilizes under-sampling, it still has limitations concerning highly imbalanced and class-overlapped datasets.

In comparison, the proposed model $DAPS$ combines dynamic sampling with the idea of self-paced learning to choose informative instances. Utilizing the concept of dynamic sampling helps prevent information loss. In addition, $DAPS$ selectively assigns weights to informative instances in the class overlap region, which minimizes the effect of noisy data. This mechanism is different from boosting methods [6,7], as they over-emphasize the misclassified data points, making them sensitive to outliers and noisy data.

3 Method

In this section, we present the proposed framework $DAPS$ (described in Section 3.3). Fig. 1 shows the pipeline of the framework. $DAPS$ is an ensemble framework and contains two important steps: *dynamic self-paced sampling* and *instance weighting* (described in Sections 3.1 and 3.2, respectively). The *dynamic self-paced sampling* step generates a sequence of datasets with different class distributions by sampling instances based on how well they were predicted, rather than through a blind selection. The *instance weighting* dynamically assigns weights to the instances in the class overlap region (shown in Fig. 2). The weights of instances in the non-overlap region are maintained same in the process. Initially, $DAPS$ chooses a large number of easy-to-learn instances to build a skeleton and ensure that the labels of the majority of instances can be correctly predicted. Then it puts more focus on the hard-to-classify instances by selectively assigning larger weights to valuable instances in the class overlap region and much smaller weights to noisy data.

We now introduce the notation used throughout the paper. Let $\mathcal{D}$ denote the original training dataset and $\mathcal{D}^l$ denote the set of the selected instances in the l^{th} iteration of training. Each instance in $\mathcal{D}$ is represented as $(\mathbf{x}, y)$, where y denotes the binary class label of an instance $\mathbf{x}$. In general, people are more interested in the minority class. We let the minority class be a positive class $\mathcal{P}$ and the majority class be a negative class $\mathcal{N}$, where $\mathcal{P}$ and $\mathcal{N}$ are often defined as: $\mathcal{P} = \{(\mathbf{x}, y)|y = 1\}; \quad \mathcal{N} = \{(\mathbf{x}, y)|y = 0\}$.

Let $|\mathcal{P}|$ and $|\mathcal{N}|$ be the number of instances in the positive class $\mathcal{P}$ and negative class $\mathcal{N}$, respectively. Here it is assumed that $|\mathcal{P}| \ll |\mathcal{N}|$. We define the imbalanced ratio IR as the number of instances in $\mathcal{N}$ divided by the number of instances in $\mathcal{P}$, i.e. $IR = \frac{|\mathcal{N}|}{|\mathcal{P}|}$. The value of IR is much larger than 1 for highly imbalanced datasets.

3.1 Dynamic self-paced sampling

3.1.1 How many instances will be sampled in each iteration?

In order to address the challenges posed by over-sampling and under-sampling methods, we generate a series of datasets with various class distributions, from imbalanced to balanced.

The imbalanced ratio represents the skewness of a class distribution. Thus, we can generate datasets with varied class distributions by adjusting this ratio, from the original imbalanced ratio to a balanced ratio, through a *scheduler function* SF [26]. The imbalanced ratio IR at the l^{th} iteration is computed as $IR(l) = {IR_{ori}}^{SF(l)}$, where IR_{ori} represents the imbalanced ratio of the original dataset and $SF(l)$ is a function that returns a real value from 1 to 0, monotonically decreasing with the input l. The scheduler function SF can be defined in various ways [26]. For example, a convex function, which controls the learning process speed from slow to fast, or, a concave function, which controls the process speed from fast to slow. In this work, we consider a linear function with a constant learning speed. Thus, $SF(l)$ is defined as $SF(l) = 1 - (l/L)$, where L is the total number of iterations. At the beginning of the first iteration, when $l = 0$, $SF(0) = 1$, and the model is trained on the original training dataset ($\mathcal{D}^0 = \mathcal{D}$). When the process reaches the final iteration, $SF(L)$ is 0, so $IR(L)$ is equal to 1 and $\mathcal{D}^L$ is a balanced dataset.

Considering that the positive instances are relatively limited, the number of positive instances is not changed in each iteration. That is, $|\mathcal{P}^l| = |\mathcal{P}|$. The number of negative instances at the l^{th} iteration is then determined by the *scheduler function* SF as

$$|\mathcal{N}^l| = \lfloor IR(l) \times |\mathcal{P}| \rfloor = \lfloor |\mathcal{N}|^{SF(l)} \times |\mathcal{P}|^{(1-SF(l))} \rfloor. \tag{1}$$

3.1.2 Which instances will be sampled?

Inspired by the idea of self-paced learning [10], we design a sampling mechanism which initially focuses on the easy-to-learn instances and then progressively includes more complex ones. The purpose is to adaptively adjust the sampling strategy to improve the model's generalization capability. For a classification problem, if an instance has a high probability to be classified correctly, it is regarded as an easy-to-learn instance. Suppose that we have a trained classifier F. The hardness of an instance $\mathbf{x}$ is computed as the probability of misclassification:

$$H(\mathbf{x}, y, F) = \begin{cases} 1-p, & y=1 \\ p, & y=0, \end{cases} \tag{2}$$

where $p \in [0,1]$ is the estimated probability for the label $y=1$. The value of $H(\mathbf{x}, y, F)$ ranges from 0 to 1. A higher value of $H(\mathbf{x}, y, F)$ implies that the instance $\mathbf{x}$ is more difficult to be classified correctly.

We first split positive instances into B bins of equal width according to their hardness, where B is a hyper-parameter and each bin reflects a level of hardness. Then, we repeat the same binning procedure for negative instances. Let b_+ and b_- represent the b^{th} bin of positive and negative instances, respectively. Since the binning procedures are the same for positive and negative instances, for simplicity, we use b instead of b_+ and b_-. Let Bin_b^l denote the b^{th} bin at the l^{th} iteration, α^l be the range of misclassification probability values for all instances from $\mathcal{D}^0$, calculated at the l-th iteration, and β^l represents the minimum of those misclassification probability values. The range α^l is then split into B equal-width bins. If the hardness of an instance $\mathbf{x}$ is within $\left[\frac{\alpha^l*(b-1)}{B}+\beta^l, \frac{\alpha^l*b}{B}+\beta^l\right)$, then the instance falls into Bin_b^l, that is,

$$Bin_b^l = \left\{ (\mathbf{x}, y) \,\middle|\, \frac{\alpha^l*(b-1)}{B} + \beta^l \leq H(\mathbf{x}, y, F_l) < \frac{\alpha^l*b}{B} + \beta^l \right\}, \tag{3}$$

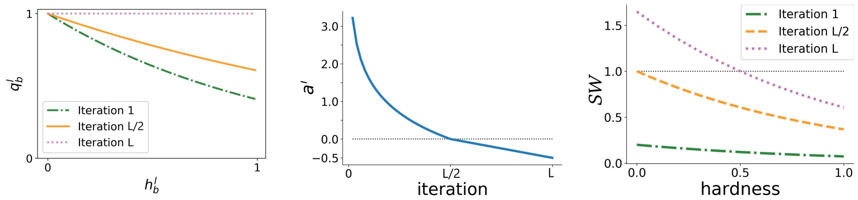
where $\alpha^l = \max_{(\mathbf{x},y)\in\mathcal{D}} H(\mathbf{x}, y, F_l) - \min_{(\mathbf{x},y)\in\mathcal{D}} H(\mathbf{x}, y, F_l)$. The average hardness of the b^{th} bin at the l^{th} iteration is calculated as

$$h_b^l = \frac{\sum_{(\mathbf{x}_i, y_i)\in Bin_b^l} H(\mathbf{x}_i, y_i, F_l)}{|Bin_b^l|}, \tag{4}$$

where $|Bin_b^l|$ is the number of instances in Bin_b^l.

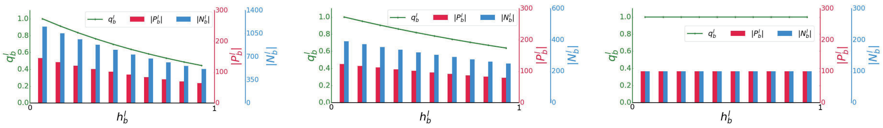
Up to this point, positive and negative instances are split separately into two sets, each containing B bins. We do not simply sample the same number of instances from each bin. Instead, we prefer to select instances with small hardness values in the early iterations, and then progressively choose more instances with large hardness values. The sampling procedure thus depends on (1) the hardness values and (2) the current iteration l. A modified exponential function $q_b^l = e^{h_b^l(l/L-1)}$ is designed to specify the portion of instances chosen from the b^{th} bin at the l^{th} iteration. The range of q_b^l is $(0,1]$. The counts of positive and negative instances selected from their respective b^{th} bins at the l-th iteration are calculated as

$$|\mathcal{P}_b^l| = \left\lfloor \frac{q_{b_+}^l}{\sum_{b_+} q_{b_+}^l} \times |\mathcal{P}^l| \right\rfloor; \quad |\mathcal{N}_b^l| = \left\lfloor \frac{q_{b_-}^l}{\sum_{b_-} q_{b_-}^l} \times |\mathcal{N}^l| \right\rfloor. \tag{5}$$



(a) q_b^l (b) a^l (c) Instance weight

Fig. 3: (a) q_b^l as a function of h_b^l; (b) a^l as a function of the iteration number; (c) vetoed instances' weights as functions of hardness.



(a) l=0 (b) $l = L/2$ (c) $l = L$

Fig. 4: Number of instances that are selected from each bin based on their average hardness at different iterations l. ($|\mathcal{P}| = 1,000$ and $|\mathcal{N}| = 10{,}000$).

A graphic illustration is shown in Fig. 3(a). During the early iterations, more instances from low-hardness bins are selected (dash-dotted green line in Fig. 3(a)). As the process continues, the fraction of instances selected from the low-hardness bins decreases. Finally, the same number of instances are selected from all bins in the last iteration (dotted pink line in Fig. 3(a)). The number of instances selected from each bin based on their average hardness, at the beginning, halfway and the end of the training process, is shown in Fig. 4.

Once $|\mathcal{P}_b^l|$ and $|\mathcal{N}_b^l|$ are determined, we randomly sample the required number of instances from the corresponding bins. Note that when $|\mathcal{N}_b^l| > |Bin_{b_-}^l|$ (or $|\mathcal{P}_b^l| > |Bin_{b_+}^l|$), some instances may be chosen multiple times so as to achieve the desired number of instances. If Bin_b^l is empty, then no instance is selected from the b^{th} bin.

3.2 Instance weighting

Once the training instances are sampled, they are assigned weights accordingly. The goal is to improve the model's predictive capability by amplifying (or diminishing) the losses of non-trivial (or noisy) instances.

We divide the entire dataset into two groups: *vetoed* and *non-vetoed* instances. A *vetoed* instance is one whose label is not in agreement with its k nearest neighbors' labels. A *vetoed* instance can be decided by two criteria: either vetoed by one neighbor or vetoed by all neighbors. A *vetoed-by-one* instance refers to an instance whose label differs from at least one of its k nearest neighbors' labels (see Eq. 6). A *vetoed-by-all* instance refers to an instance whose label differs from all of its k nearest neighbors' labels (see Eq. 7).

A *vetoed-by-one* instance and a *vetoed-by-all* instance are illustrated in Fig. 2.

$$\mathcal{D}_{vetoed-by-one} = \{(\mathbf{x}_i, y_i) | \exists \mathbf{x}_j \in KNN(\mathbf{x}_i), y_i \neq y_j\}, \tag{6}$$

$$\mathcal{D}_{vetoed-by-all} = \{(\mathbf{x}_i, y_i) | \forall \mathbf{x}_j \in KNN(\mathbf{x}_i), y_i \neq y_j\}, \tag{7}$$

where $KNN(\mathbf{x}_i)$ represents the set of an instance $\mathbf{x}_i$'s k nearest neighbors. Both *vetoed-by-one* and *vetoed-by-all* instances are *vetoed* instances. The rest are *non-vetoed* instances whose weights are set to 1 throughout the whole procedure.

Vetoed instances are probably noisy or informative but difficult to classify, typically located in the overlapped region of positive and negative data, as they are not consistent with their k nearest neighbors' labels. Thus, we design a function that puts more emphasis on non-trivial instances while decreasing the effect of noisy instances. The weight of a *vetoed* instance $\mathbf{x}$ selected from the b^{th} bin at the l^{th} iteration is calculated as

$$SW(h_b^l, a^l) = e^{-(h_b^l + a^l)}, \quad a^l = \begin{cases} -\ln\left(\frac{2l}{L}\right), & l \leq \frac{L}{2} \\ \frac{1}{2} - \frac{l}{L}, & \frac{L}{2} < l \leq L. \end{cases} \tag{8}$$

$SW(h_b^l, a^l)$ is an exponential function with inputs h_b^l and a^l, where a^l is an adjusted parameter to control the range of $SW(h_b^l, a^l)$ values. The function a^l can be any arbitrary function if it is monotonically decreasing and passes through $(\frac{L}{2}, 0)$ and $(L, -0.5)$. The value of a^l as a function of the iteration l is plotted in Fig. 3(b) and the weights of *vetoed* instances as functions of hardness are plotted in Fig. 3(c). The function $SW(h_b^l, a^l)$ slowly increases the weight of a *vetoed* instance from 0 to a higher value during the process. The weights of *vetoed* instances decrease with the increased hardness values in each iteration.

The instance weighting procedure is divided into *two phases*. Phase I: at the beginning, a^l has a large value, so the weights of *vetoed* instances are close to 0 (dash-dotted green line in Fig. 3(c)). At this time, the model is focused more on the *non-vetoed* instances (recall that the weights of the *non-vetoed* instances are 1). During the first half iterations, the value of a^l decreases steadily until it reaches zero at iteration $\frac{L}{2}$ and the weights of *vetoed* instances have a slight increase but are still below 1. When the process reaches iteration $\frac{L}{2}$, the weights of *vetoed* instances with zero hardness values are 1, and the weights of the remaining *vetoed* instances decrease with an increase in the hardness value (dashed orange line in Fig. 3(c)).

Phase II: in the second half iterations, the model focuses on the *vetoed* instances with hardness values less than 0.5. The hypothesis is that these *vetoed* instances are informative, as they are correctly predicted with a high probability. As the process continues, the value of a^l decreases slowly from 0 to -0.5, then the weights of the correctly classified *vetoed* instances are subsequently increased. At the last iteration, the weights of the *vetoed* instances with hardness values less than 0.5 are much larger than 1. For the *vetoed* instances with large hardness values, which are probably noisy data, the weights are still below 1 (dotted pink line in Fig. 3(c)).

3.3 DynAmic self-Paced sampling enSemble (DAPS)

We now formally describe the proposed model *DAPS* and its two variants $DAPS_{one}$ and $DAPS_{all}$. We first describe the procedure for the *DAPS* model and then clarify the differences among the three variants.

The pseudocode is outlined in Algorithm 1. The model begins by identifying the set of vetoed instances (Line 2). The weights of *non-vetoed* instances are set to 1, and the weights of *vetoed* instances are set to 0. In each iteration, the model first samples the required number of instances (Lines 8 - 13). This stage consists of four steps: (1) Applying the current ensemble F_l to calculate the hardness values of all instances in $\mathcal{D}$ (Line 8); (2) Splitting the original dataset $\mathcal{D}$ to B bins according to their hardness values (Line 10); (3) Computing the average hardness of each bin (Line 11); and (4) Randomly sampling instances from each bin (Lines 12 - 13). Then the model assigns the weights to the selected *vetoed* instances based on Eq. (8) and trains a base classifier f_i using the selected instances in $\mathcal{D}^l$ with weights (Line 15). This process is repeated in a fixed number of iterations (Lines 7 - 15).

The procedure is the same for *DAPS*, $DAPS_{one}$ and $DAPS_{all}$. The only difference among the three variants is the set of the *vetoed* instances. In $DAPS_{one}$, the *vetoed-by-one* instances are taken as *vetoed* instances in both positive and negative classes. Similarly, in the model $DAPS_{all}$, for both positive and negative instances, the *vetoed-by-all* instances are taken as *vetoed* instances. The size of the *vetoed* instance set in $DAPS_{one}$ is larger than that in $DAPS_{all}$. In some cases when the number of positive instances is limited, the vetoed-by-one scheme may treat all positive objects as *vetoed* instances. This will affect the model's predictive capability. Therefore, the *DAPS* model chooses the vetoed-by-one scheme for the negative instances, and the vetoed-by-all scheme for the positive instances.

4 Experiments

4.1 Experimental setup

We compared the proposed model to various approaches, including data-level sampling approaches, an algorithm-level method, the frequently used ensemble-level approaches and the state-of-the-art methods. Here, we briefly introduce these methods:
1) Data-level sampling methods:

- **RUS** (Random Under-Sampling) randomly removes $|\mathcal{N}| - |\mathcal{P}|$ instances from the majority class.
- **ROS** (Random Over-Sampling) involves supplementing the training data with multiple copies of the minority class to balance the class distribution.
- **SMOTE** (Synthetic Minority Over-sampling TEchnique) [3] generates $|\mathcal{N}| - |\mathcal{P}|$ positive instances.

Algorithm 1: DAPS: DynAmic self-Paced sampling enSemble

Input: Training set $\mathcal{D}$, base classifier f, # of nearest neighbors k, # of iterations L, # of bins B
Output: Ensemble model $F(\mathbf{x})$
Initialize: $\mathcal{P} \Leftarrow$ minority instances in $\mathcal{D}$, $\mathcal{N} \Leftarrow$ majority instances in $\mathcal{D}$;
Identify the vetoed instances;
Set the weights of vetoed instance to 0;
Set the weights of non-vetoed instances to 1;
Train f_0 using $\mathcal{D}^0$ with instance weights;
for $l = 1$ **to** L **do**
  Build an ensemble $F_l(\mathbf{x}) = \sum_{j=0}^{l-1} f_j(\mathbf{x})$;
  Calculate the hardness of all instances in $\mathcal{D}$ using $F_l(\mathbf{x})$ based on Eq. (2);
  Update $|\mathcal{P}^l|$, $|\mathcal{N}^l|$ according to Eq. (1) ;
  Split $\mathcal{P}$ and $\mathcal{N}$ into B bins each, following Eq. (3) ;
  Calculate the average hardness of each bin h_b^l ;
  Select $|\mathcal{P}_b^l|$ positive instances from the b_+^{th} bin and add them into $\mathcal{D}^l$ (b_+ ranges from 1 to B) ;
  Select $|\mathcal{N}_b^l|$ negative instances from the b_-^{th} bin and add them into $\mathcal{D}^l$ (b_- ranges from 1 to B) ;
  Assign weights to the vetoed instances in $\mathcal{D}^l$ (Eq. (8));
  Train f_l using $\mathcal{D}^l$ with instance weights ;
end
return $F(\mathbf{x}) = \mathbb{I}\left(\sum_{l=0}^{L} f_l(\mathbf{x}) > \lfloor L/2 \rfloor\right)$;

2) Algorithm-level method:

- **LR**$_{cs}$ (Logistic Regression$_{\text{cost sensitive}}$) takes the cost matrix into consideration during training.

3) Ensemble-level methods:

- **AdaBoost** [6](Adaptive Boosting) assigns larger weighs to instances that were misclassified and adds new weak learners sequentially to focus on more difficult instances.
- **GBDT** [7](Gradient Boosting Decision Tree) uses gradient descent to update a boosting ensemble model.
- **XGBoost** [5](Extreme Gradient Boosting) uses the second order derivative of the loss function as an approximation to update a boosting ensemble model.
- **Boosting**$_{smote}$ [4] applies the *SMOTE* method at each boosting iteration.
- **Bagging**$_{smote}$ [25] applies *SMOTE* method in each iteration to get each bag for bagging.
- **Boosting**$_{rus}$ [20] applies the *RUS* method at each boosting iteration.
- **Bagging**$_{rus}$ [25] uses *RUS* in each iteration as one bag in the bagging process.
- **Cascade** (BalanceCascade) [12] utilizes *RUS* to train an *AdaBoost* model at each iteration.

4) State-of-the-art methods:

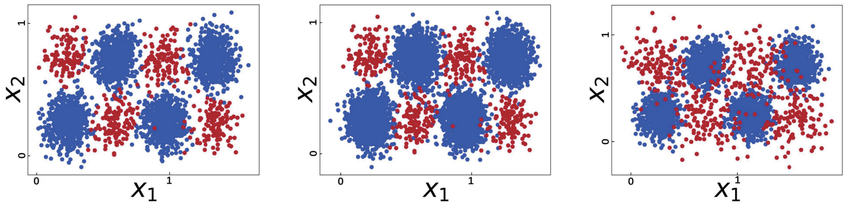
- **NB-Based** (Neighborhood-Based under-sampling) [21] removes some negative instances based on their k nearest neighbors.
  - **NB-Basic** (Basic neighborhood search) removes negative instances that have positive neighbors.
  - **NB-Tomek** (Modified Tomek link search) removes negative instances if the neighborhood between a negative instance and a positive instance is established in both directions.
  - **NB-Comm** (Common nearest neighbors search) removes the negative instances that are common nearest neighbors of any two positive instances.
  - **NB-Rec**(Recursive search) is an extension of *NB-Comm* that removes the negative instances that are common nearest neighbors of any two removed instances in the output of *NB-Comm*.
- **SPEnsemble** (Self-Paced Ensemble) [14] generates an ensemble by self-paced harmonizing data hardness via under-sampling.
- **SwitchingNED** (Class Switching according to Nearest Enemy Distance) [8] flips the labels of negative instances based on the nearest enemy distance and then trains a decision tree on the switched instances.
- **LDAM-DRW** [2] optimizes a label-distribution-aware loss function to encourage larger margins for minority classes.

To demonstrate the robustness and effectiveness of the approaches, we considered four base classifiers: DT (Decision Tree), RF (Random Forest), SVM (Support Vector Machine), and GBDT (Gradient Boosting Decision Tree) to train the models. We used the implementations of the above base classifiers from the *scikit-learn* [17] package. The max depth of the tree-based classifiers and the number of iterations in the ensemble methods were set to 10. The rest of the hyper-parameters were set to their default values in scikit-learn. The experimental settings for the other methods were the same as in their corresponding papers. As for *DAPS*, we set the number of bins B to 10, the number of iterations L to 10, and the number of nearest neighbors k was fine-tuned between 1 and 5. We chose Euclidean distance as a measure for finding the nearest neighbors of an instance. We used $F1$-score, precision and recall as evaluation metrics[2] to compare the models' performances. All experiments were conducted on a machine with Intel Xeon E5 2.1 GHz and 256 GB RAM.

4.2 Results on synthetic datasets

To better understand the proposed model, we generate a 2×4 checkerboard dataset (as shown in Fig. 5(a)) so that some characteristics, such as the imbalanced ratio, can be controlled (*SPEnsemble* [14] used a similar generation process). The dataset contains instances from eight bivariate gaussian distributions. The variance-covariance matrix of the instances in the negative class

[2] Due to space limitation, we only report precision and recall results on real-world datasets. AUPRC (i.e., the area under the precision-recall curve) does not properly reflect the performance of our model, as *DAPS* chooses 0.5 as threshold to optimize predictions.



(a) $\sigma_p^2 = 3, IR = 5$ (b) $\sigma_p^2 = 3, IR = 10$ (c) $\sigma_p^2 = 6, IR = 5$

Fig. 5: Illustration of the synthetic datasets. The blue dots are from the negative class and the red dots are from the positive class.

is $3 \cdot \mathbf{I}_2$, while that of the instances in the positive class is $\sigma_p^2 \cdot \mathbf{I}_2$, where $\mathbf{I}_2$ is an 2×2 identity matrix. We fix the number of positive instances $|\mathcal{P}|$=500 in all experiments, and determine the number of negative instances $|\mathcal{N}|$ by controlling the imbalanced ratio (see Figs. 5(a) and 5(b)). Furthermore, we vary the value of σ_p^2 to control the degree of class overlap (see Figs. 5(a) and 5(c)). Using the aforedescribed data generation procedure, a training set and an independent test set were generated. The ratio of training-to-testing size was $4:1$. Each model was run on the train/test set five times independently, and the mean and standard deviation of the resulting F1-scores were reported.

4.2.1 Effectiveness of DAPS

We first compared the effectiveness of *DAPS* and its variants with baseline models on a synthetic dataset where the covariance matrix of the instances in the positive class was $3 \cdot \mathbf{I}_2$ and the imbalanced ratio was 30. Table 1 shows the mean and standard deviation of F1-scores obtained by all models (including the one without applying any sampling technique) using 4 different base classifiers. *DAPS* and its two variants outperformed all baseline models by a notable margin. In general, when *DT* was the base classifier, *DAPS* produced 11.3%-26.3% higher F1-scores when compared to the data-level sampling methods, 3%-22% higher F1-scores when compared to the ensemble-level methods, and 1%-30.2% higher F1-scores when compared to the state-of-the-art methods. The performance is consistent when applying other base classifiers, thus verifying the effectiveness of *DAPS*.

4.2.2 Effectiveness with respect to imbalanced ratio

To verify the effectiveness of the model with respect to imbalanced ratio (IR), we changed the class distribution by varying IR between 10 and 90 in steps of 40. Due to space limitation, we only show the results of all models that applied DT as the base classifier in Table 2. Compared with all baseline models, *DAPS* and its two variants $DAPS_{one}$ and $DAPS_{all}$ produced higher F1-scores. When the imbalanced ratio was 10, *DAPS* produced 0.5%-66.6% higher average F1-scores than the other baseline models. As the imbalanced ratio increased, almost all sampling methods deteriorated rapidly. For example, when

Table 1: Mean and standard deviation of F1-scores obtained by the models using different base classifiers. Note that LR_{cs} and LDAM-DRW are left out from this experiment as they do not support different base classifiers.

Methods \ Classifiers	DT	RF	SVM	GBDT
No sampling	0.664 ± 0.000	0.486 ± 0.000	0.397 ± 0.000	0.644 ± 0.000
RUS	0.411 ± 0.015	0.484 ± 0.011	0.373 ± 0.062	0.472 ± 0.014
ROS	0.561 ± 0.010	0.597 ± 0.023	0.505 ± 0.004	0.568 ± 0.020
SMOTE	0.544 ± 0.005	0.586 ± 0.004	0.532 ± 0.004	0.564 ± 0.010
AdaBoost	0.644 ± 0.020	/[1]	/	/
GBDT	0.639 ± 0.004	/	/	/
XGBoost	0.594 ± 0.000	/	/	/
$Boosting_{smote}$	0.630 ± 0.018	0.664 ± 0.012	0.000 ± 0.000	0.656 ± 0.013
$Bagging_{smote}$	0.635 ± 0.011	0.643 ± 0.011	0.647 ± 0.017	0.649 ± 0.012
$Boosting_{rus}$	0.454 ± 0.013	0.503 ± 0.012	0.307 ± 0.164	0.539 ± 0.009
$Bagging_{rus}$	0.458 ± 0.009	0.462 ± 0.004	0.459 ± 0.012	0.460 ± 0.012
Cascade	0.523 ± 0.019	0.570 ± 0.018	0.506 ± 0.017	0.595 ± 0.017
SPEnsemble	0.627 ± 0.016	0.676 ± 0.015	0.641 ± 0.011	0.679 ± 0.006
SwitchingNED	0.372 ± 0.005	/	/	/
NB-Basic	0.621 ± 0.000	0.644 ± 0.000	0.453 ± 0.000	0.678 ± 0.000
NB-Tomek	0.416 ± 0.000	0.475 ± 0.000	0.451 ± 0.000	0.467 ± 0.000
NB-Comm	0.664 ± 0.000	0.616 ± 0.000	0.397 ± 0.000	0.539 ± 0.000
NB-Rec	0.664 ± 0.000	0.598 ± 0.000	0.397 ± 0.000	0.558 ± 0.000
$DAPS_{one}$	0.660 ± 0.008	0.694 ± 0.007	0.448 ± 0.034	0.591 ± 0.015
$DAPS_{all}$	0.659 ± 0.013	0.695 ± 0.004	0.463 ± 0.019	0.586 ± 0.011
DAPS	**0.674** ± 0.008	**0.698** ± 0.003	**0.696** ± 0.001	**0.698** ± 0.011

[1] The original implementations of *XGBoost*, *AdaBoost*, *GBDT* and *SwitchingNED* have DT fixed as their base classifier.

Table 2: Mean and standard deviation of F1-scores obtained by the models with respect to imbalance ratio (IR).

Methods	$IR = 10$	$IR = 50$	$IR = 90$
RUS	0.705 ± 0.020	0.344 ± 0.013	0.221 ± 0.037
ROS	0.742 ± 0.021	0.504 ± 0.025	0.312 ± 0.022
SMOTE	0.755 ± 0.004	0.432 ± 0.028	0.298 ± 0.029
LR_{cs}	0.146 ± 0.000	0.037 ± 0.000	0.024 ± 0.000
AdaBoost	0.777 ± 0.013	0.627 ± 0.018	0.508 ± 0.009
GBDT	0.791 ± 0.003	0.578 ± 0.004	0.456 ± 0.016
XGBoost	0.807 ± 0.000	0.633 ± 0.000	0.368 ± 0.000
$Boosting_{smote}$	0.769 ± 0.010	0.624 ± 0.021	0.474 ± 0.023
$Bagging_{smote}$	0.797 ± 0.007	0.642 ± 0.022	0.498 ± 0.007
$Boosting_{RUS}$	0.703 ± 0.017	0.366 ± 0.013	0.230 ± 0.008
$Bagging_{RUS}$	0.706 ± 0.012	0.395 ± 0.006	0.236 ± 0.008
Cascade	0.742 ± 0.010	0.419 ± 0.016	0.258 ± 0.018
SPEnsemble	0.775 ± 0.023	0.638 ± 0.020	0.446 ± 0.019
SwitchingNED	0.613 ± 0.008	0.281 ± 0.003	0.176 ± 0.002
NB-Basic	0.596 ± 0.000	0.433 ± 0.000	0.243 ± 0.000
NB-Tomek	0.679 ± 0.000	0.337 ± 0.000	0.179 ± 0.000
NB-Comm	0.787 ± 0.000	0.643 ± 0.000	0.378 ± 0.000
NB-Rec	0.782 ± 0.000	0.643 ± 0.000	0.378 ± 0.000
LDAM-DRW	0.638 ± 0.092	0.416 ± 0.088	0.346 ± 0.035
$DAPS_{one}$	0.809 ± 0.005	0.636 ± 0.007	0.526 ± 0.014
$DAPS_{all}$	0.805 ± 0.005	0.648 ± 0.014	0.521 ± 0.025
DAPS	**0.812** ± 0.005	**0.705** ± 0.009	**0.565** ± 0.011

the imbalanced ratio increased to 50, the average F1-score gap between the baselines and *DAPS* increased to 23.1%. When the imbalanced ratio increased to 90 (which implies that the dataset is highly imbalanced), the difference between *DAPS* and the alternatives is evident, 5.7%-54.1%. In particular, LR_{cs} obtained the worst result, that is 0.024.

Compared with the boosting methods, when the dataset was relatively imbalanced ($IR = 10$), *DAPS* obtained 0.5%-3.5% higher average F1-scores (p-value $\leq$ 0.069). When IR increased to 90, *DAPS* achieved statistically significantly better results (p-value $\leq$ 2e-05). The results demonstrate the robustness of *DAPS* with respect to the imbalanced ratio.

4.2.3 Effectiveness with respect to class overlap

In order to evaluate the effectiveness of the model with respect to class distribution overlap, we altered the value of σ_p^2 from 4 to 8. Table 3 shows the results of all methods that applied DT as the base classifier. Compared with all baseline models, *DAPS* and its two variants produced the highest F1-scores. When the covariance factor in the covariance matrix of the positive class was set to $4{\cdot}\mathbf{I}_2$, the class distribution was less overlapped, thus making the data relatively easy to classify. *DAPS* obtained 1.5% (p-value$\leq$0.004) higher average F1-score when compared to *XGBoost*, and 6.4%-63.1% higher average F1-scores among the remaining approaches. With the increased σ_p^2 value, the positive instances become more scattered and the overlapped region of the two classes becomes larger (see Fig. 5(c)). The performances of the most approaches clearly deteriorated. For example, when the σ_p^2 value increased from 4 to 6, the average F1-scores of the baselines (except *GBDT*, *NB-Comm* and *NB-Rec*) dropped by around 7.03%, however, the performance of *DAPS* decreased only by 6.5% and still produced 6.1%-8.5% higher F1-scores than *GBDT*, *NB-Comm* and *NB-Rec*.

As the class distribution became more overlapped, for example when σ_p^2 was set to 8, *DAPS* produced at least 18.7% higher average F1-scores than the alternatives. These results imply the advantage of applying *DAPS* on datasets with highly overlapped class distributions.

4.2.4 Comparison among the three variants of the model

We proceed by comparing the performance among the three variants of the proposed model. The last three rows in Tables 1, 2 and 3 show the results of the three approaches in various experimental settings. From the results, it can be observed that *DAPS* performs better than $DAPS_{one}$ and $DAPS_{all}$, particularly on extremely imbalanced data.

Compared to $DAPS_{one}$, *DAPS* always produces higher F1-scores. When data becomes more imbalanced or the class distribution is more overlapped, the advantage of *DAPS* is more obvious. In Table 2, with an increased imbalanced ratio from 10 to 90, *DAPS* improved the average F1-scores by 0.3%-6.9%. A similar trend can be observed in Table 3. When σ_p^2 increased from 4 to 8, *DAPS* obtained an average lift of 3.1%. Since $DAPS_{one}$ chose the vetoed-by-one criteria to determine the vetoed instances, compared with *DAPS*, more positive instances will belong to the group of *vetoed* instances. For example, when IR increases from 10 to 90, the fraction of positive instances identified by *DAPS* as *vetoed* instances increased from 3.6% to 13.3%, whereas the fraction

Table 3: Mean and standard deviation of F1-scores obtained by the models with respect to class overlapping.

Methods	$\sigma_p^2 = 4$	$\sigma_p^2 = 6$	$\sigma_p^2 = 8$
RUS	0.354 ± 0.026	0.288 ± 0.022	0.242 ± 0.004
ROS	0.482 ± 0.021	0.433 ± 0.018	0.352 ± 0.019
SMOTE	0.460 ± 0.011	0.408 ± 0.017	0.320 ± 0.017
LR_{cs}	0.063 ± 0.000	0.060 ± 0.000	0.059 ± 0.000
AdaBoost	0.611 ± 0.011	0.525 ± 0.010	0.457 ± 0.006
GBDT	0.524 ± 0.000	0.568 ± 0.005	0.447 ± 0.000
XGBoost	0.679 ± 0.000	0.612 ± 0.000	0.464 ± 0.000
$Boosting_{smote}$	0.620 ± 0.031	0.510 ± 0.019	0.429 ± 0.002
$Bagging_{smote}$	0.630 ± 0.012	0.539 ± 0.023	0.449 ± 0.013
$Boosting_{rus}$	0.381 ± 0.005	0.295 ± 0.015	0.259 ± 0.006
$Bagging_{rus}$	0.397 ± 0.011	0.320 ± 0.008	0.276 ± 0.006
Cascade	0.462 ± 0.023	0.375 ± 0.027	0.321 ± 0.015
SPEnsemble	0.606 ± 0.022	0.501 ± 0.026	0.424 ± 0.011
SwitchingNED	0.328 ± 0.002	0.272 ± 0.005	0.241 ± 0.004
NB-Basic	0.347 ± 0.000	0.246 ± 0.000	0.224 ± 0.000
NB-Tomek	0.366 ± 0.000	0.277 ± 0.000	0.254 ± 0.000
NB-Comm	0.492 ± 0.000	0.544 ± 0.000	0.444 ± 0.000
NB-Rec	0.492 ± 0.000	0.544 ± 0.000	0.444 ± 0.000
LDAM-DRW	0.527 ± 0.046	0.437 ± 0.045	0.314 ± 0.046
$DAPS_{one}$	0.677 ± 0.014	0.613 ± 0.014	0.464 ± 0.009
$DAPS_{all}$	0.674 ± 0.005	0.603 ± 0.015	0.463 ± 0.014
DAPS	**0.694** ± 0.008	**0.629** ± 0.015	**0.525** ± 0.016

increased from 15.6% to 44% when choosing $DAPS_{one}$. Since the weights of *vetoed* instances are set to be much smaller than 1 in the first few iterations, which may lead to insufficient learning of positive instances.

Compared to $DAPS_{all}$, $DAPS$ always produces higher F1-scores. With an increased imbalanced ratio, $DAPS$ improved the average F1-scores by 0.7%-5.7%. As the class distribution becomes more overlapped by altering σ_p^2 from 4 to 8, the average improvement made by $DAPS$ was 10.8%. Since $DAPS_{all}$ chose the vetoed-by-all criteria, only a small number of negative instances are treated as vetoed, which may overemphasize the effect of some trivial or noisy instances. For instance, when σ_p^2 changes from 4 to 8, the fraction of negative instances identified by $DAPS$ as *vetoed* instances increased from 4.55% to 7%, however, the fraction dropped from 0.11% to 0.04% when choosing $DAPS_{all}$.

4.2.5 Effect of dynamic sampling and instance weighting

We assessed the benefit of the dynamic sampling and instance weighting mechanisms w.r.t. classification accuracy by designing the following variants:

1) $DAPS_{-dynamic}$: A variant of $DAPS$ that utilizes under-sampling instead of dynamic sampling.

2) $DAPS_{-weight}$: A $DAPS$ variant that omits the weight mechanism by setting all instance weights to 1.

We applied DT as the base classifier and summarized the results in Table 4 and Table 5. In Table 4, we fixed σ_p^2 and altered the imbalanced ratio. When $\sigma_p^2 = 3$ and $IR = 10$, $DAPS$ produced ∼2.1% higher average F1-scores than the two variants. With the increased IR value, the superiority of $DAPS$ was more clear. Specifically, when $IR = 90$, $DAPS$ produced ∼8.75% higher F1-score than the other two variants. Next, when we fixed the imbalanced ratio

Table 4: Effect of dynamic sampling and instance weighting mechanisms with respect to imbalanced ratio(IR).

Methods	$\sigma_p^2 = 3, IR = 10$	$\sigma_p^2 = 3, IR = 50$	$\sigma_p^2 = 3, IR = 90$
$DAPS_{-dynamic}$	0.807 ± 0.008	0.676 ± 0.007	0.509 ± 0.011
$DAPS_{-weight}$	0.775 ± 0.015	0.613 ± 0.009	0.446 ± 0.033
$\boldsymbol{DAPS}$	**0.812** ± 0.005	**0.705** ± 0.009	**0.565** ± 0.011

Table 5: Effect of dynamic sampling and instance weighting mechanisms with respect to class overlap.

Methods	$\sigma_p^2 = 4, IR = 30$	$\sigma_p^2 = 6, IR = 30$	$\sigma_p^2 = 8, IR = 30$
$DAPS_{-dynamic}$	0.671 ± 0.010	0.566 ± 0.005	0.489 ± 0.009
$DAPS_{-weight}$	0.602 ± 0.030	0.595 ± 0.016	0.465 ± 0.006
$\boldsymbol{DAPS}$	**0.694** ± 0.008	**0.629** ± 0.015	**0.525** ± 0.016

and altered σ_p^2 to change the degree of class overlap in Table 5, the superiority of *DAPS* still held. For example, when $\sigma_p^2 = 4$, *DAPS* achieved 2.3% higher F1-score compared to the variant without the dynamic sampling mechanism and 9.2% higher F1-score than the one that did not utilize instance weights.

4.2.6 Sensitivity of hyper-parameter L

To test the sensitivity of the iteration number L, we changed the value L from 2 to 100 and conducted the experiment on a synthetic dataset ($\sigma_p^2 = 8$, $IR = 30$). The average F1-scores of 5 independent runs of *DAPS* and the ensemble methods are plotted in Fig. 6.

It is clear that *DAPS* obtained an F1-score of 0.508 when L was only 5 and maintained the highest accuracy with the increased L. Compared with under-sampling based ensemble approach $Boosting_{rus}$ and over-sampling based ensemble approach $Boosting_{smote}$, *DAPS* demonstrates competitive performance. The boosting methods took more number of iterations to reach their highest F1-scores. For example, the optimal number of iterations for *XGBoost* and *AdaBoost* are 20 and 50, respectively, while *GBDT* needs more than 100 iterations. However, the highest F1-scores obtained by *XGBoost* and *AdaBoost* were still smaller than the one produced by *DAPS*. In addition, after *Xgoost* and *AdaBoost* reached their highest accuracy, a decrease in the F1-score is observed for both methods, which indicates the overfitting of the models. Furthermore, *Cascade* and *SPEnsemble* need more than 100 iterations to obtain better results. The results demonstrate that *DAPS* is robust to the different selection of L and can converge quickly.

4.3 Real-world public datasets

The effectiveness of *DAPS* was further assessed on three real-world public datasets whose characteristics are listed in Table 6.

Credit contains European cardholders' transactions from September 2013 [19]. Given a transaction, the goal is to predict whether it is fraudulent or not. The

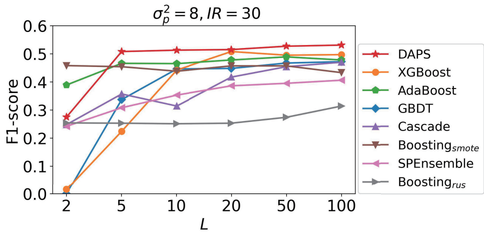


Fig. 6: F1-score as a function of the number of iterations L.

Table 6: Basic characteristics of the real-world datasets.

Datesets	#Instances	#Attributes	IR
Credit	284,807	31	579.9
Yeast4	1,484	8	28.4
Yeast6	1,484	8	39.2
PPD	1,000,000	35	7.5

Table 7: Mean and standard deviation of F1-score, precision and recall values obtained on real-world datasets.

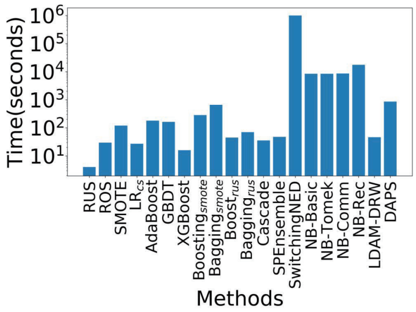
Methods	Credit			Yeast4			Yeast6			PPD		
	F1	precision	recall	F1	precision	recall	F1	precision	recall	F1	precision	recall
RUS	0.03 ± 0.01	0.01 ± 0.01	0.89 ± 0.05	0.26 ± 0.03	0.15 ± 0.02	0.88 ± 0.08	0.16 ± 0.02	0.08 ± 0.01	0.80 ± 0.16	0.31 ± 0.01	0.21 ± 0.02	0.57 ± 0.01
ROS	0.28 ± 0.06	0.17 ± 0.05	0.79 ± 0.05	0.30 ± 0.10	0.25 ± 0.06	0.39 ± 0.19	0.47 ± 0.09	0.44 ± 0.32	0.57 ± 0.24	0.31 ± 0.00	0.21 ± 0.00	0.55 ± 0.01
SMOTE	0.17 ± 0.06	0.09 ± 0.03	0.81 ± 0.08	0.32 ± 0.06	0.23 ± 0.04	0.57 ± 0.23	0.29 ± 0.09	0.20 ± 0.05	0.51 ± 0.22	0.26 ± 0.01	0.27 ± 0.01	0.25 ± 0.01
LR$_{cs}$	0.11 ± 0.04	0.06 ± 0.02	0.88 ± 0.03	0.29 ± 0.03	0.17 ± 0.03	0.83 ± 0.17	0.26 ± 0.06	0.15 ± 0.04	**0.85** ± 0.17	0.29 ± 0.00	0.19 ± 0.00	0.58 ± 0.00
AdaBoost	0.73 ± 0.08	0.80 ± 0.11	0.67 ± 0.11	0.32 ± 0.13	0.41 ± 0.16	0.29 ± 0.16	0.45 ± 0.08	0.57 ± 0.11	0.37 ± 0.07	0.12 ± 0.01	0.35 ± 0.01	0.07 ± 0.01
GBDT	0.72 ± 0.06	0.79 ± 0.13	0.69 ± 0.13	0.13 ± 0.13	0.50 ± 0.50	0.07 ± 0.08	0.45 ± 0.05	0.60 ± 0.06	0.37 ± 0.07	0.02 ± 0.00	**0.58** ± 0.01	0.01 ± 0.00
XGBoost	0.80 ± 0.07	0.86 ± 0.12	0.76 ± 0.10	0.28 ± 0.16	**0.58** ± 0.34	0.19 ± 0.12	0.46 ± 0.09	0.55 ± 0.05	0.40 ± 0.12	0.09 ± 0.00	0.53 ± 0.00	0.05 ± 0.00
Boosting$_{smote}$	0.80 ± 0.05	0.86 ± 0.13	0.76 ± 0.08	0.29 ± 0.14	0.37 ± 0.21	0.27 ± 0.16	0.49 ± 0.13	0.48 ± 0.18	0.54 ± 0.18	0.12 ± 0.01	0.36 ± 0.02	0.07 ± 0.01
Bagging$_{smote}$	0.75 ± 0.08	**0.87** ± 0.12	0.67 ± 0.12	0.18 ± 0.21	0.25 ± 0.24	0.16 ± 0.20	0.56 ± 0.11	0.65 ± 0.02	0.51 ± 0.16	0.12 ± 0.01	0.33 ± 0.01	0.07 ± 0.01
Boosting$_{rus}$	0.05 ± 0.01	0.02 ± 0.01	0.91 ± 0.05	0.23 ± 0.04	0.13 ± 0.03	0.90 ± 0.15	0.19 ± 0.02	0.10 ± 0.01	**0.85** ± 0.17	0.30 ± 0.01	0.21 ± 0.01	0.53 ± 0.01
Bagging$_{rus}$	0.07 ± 0.01	0.03 ± 0.01	0.89 ± 0.05	0.24 ± 0.04	0.13 ± 0.02	0.82 ± 0.16	0.21 ± 0.02	0.12 ± 0.02	**0.85** ± 0.17	0.27 ± 0.01	0.18 ± 0.01	0.52 ± 0.01
Cascade	0.35 ± 0.16	0.23 ± 0.12	**0.96** ± 0.04	0.23 ± 0.04	0.13 ± 0.02	0.77 ± 0.19	0.22 ± 0.04	0.13 ± 0.03	0.82 ± 0.12	0.23 ± 0.04	0.15 ± 0.04	0.50 ± 0.11
SPEnsemble	0.44 ± 0.11	0.30 ± 0.10	0.85 ± 0.05	0.30 ± 0.04	0.20 ± 0.06	0.77 ± 0.18	0.27 ± 0.01	0.16 ± 0.01	0.77 ± 0.16	0.32 ± 0.01	0.26 ± 0.01	0.40 ± 0.02
SwitchingNED	0.06 ± 0.02	0.03 ± 0.01	0.91 ± 0.04	0.23 ± 0.03	0.13 ± 0.03	**0.90** ± 0.15	0.19 ± 0.04	0.10 ± 0.02	**0.85** ± 0.17	0.28 ± 0.01	0.19 ± 0.01	0.55 ± 0.01
NB-Basic	0.76 ± 0.08	0.72 ± 0.13	0.81 ± 0.07	0.28 ± 0.06	0.17 ± 0.04	0.76 ± 0.18	0.33 ± 0.09	0.20 ± 0.06	0.80 ± 0.16	0.20 ± 0.00	0.11 ± 0.00	**1.00** ± 0.00
NB-Tomek	0.20 ± 0.03	0.11 ± 0.02	0.86 ± 0.05	0.24 ± 0.07	0.13 ± 0.04	0.84 ± 0.11	0.15 ± 0.04	0.08 ± 0.03	0.82 ± 0.18	0.29 ± 0.00	0.19 ± 0.00	0.60 ± 0.00
NB-Comm	0.73 ± 0.09	0.76 ± 0.19	0.72 ± 0.07	0.31 ± 0.15	0.30 ± 0.17	0.33 ± 0.17	0.33 ± 0.09	0.26 ± 0.08	0.42 ± 0.10	0.10 ± 0.00	0.43 ± 0.00	0.05 ± 0.00
NB-Rec	0.73 ± 0.09	0.76 ± 0.19	0.72 ± 0.07	0.26 ± 0.15	0.29 ± 0.21	0.25 ± 0.15	0.37 ± 0.06	0.29 ± 0.06	0.48 ± 0.07	0.10 ± 0.00	0.43 ± 0.00	0.05 ± 0.00
LDAM-DRW	0.82 ± 0.02	0.83 ± 0.02	0.81 ± 0.02	0.18 ± 0.25	0.21 ± 0.28	0.16 ± 0.23	0.33 ± 0.14	0.44 ± 0.19	0.29 ± 0.13	0.11 ± 0.02	0.54 ± 0.03	0.06 ± 0.01
DAPS	**0.82** ± 0.06	0.84 ± 0.11	0.80 ± 0.08	**0.39** ± 0.19	0.55 ± 0.31	0.35 ± 0.23	**0.56** ± 0.13	**0.66** ± 0.19	0.51 ± 0.16	**0.32** ± 0.01	0.27 ± 0.01	0.38 ± 0.01

dataset contains 492 frauds out of 284,807 transactions. It is a highly imbalanced dataset with an IR of 578.9 : 1. We used 5-fold cross-validation to evaluate the performance of the models.

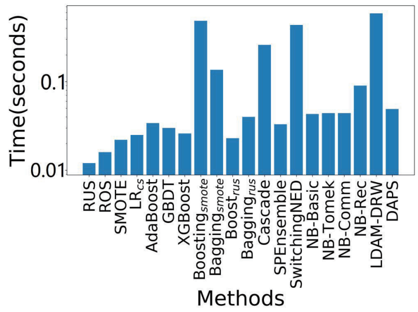
Yeast is used to predict the Cellular localization sites of proteins [1] and contains 1,484 instances of multiple classes. We selected one of the classes to be positive class and the remaining classes constituted the negative class. **Yeast4** and **Yeast6** denote the datasets with *ME2* and *EXC* as the positive class, respectively. **Yeast4** includes 51 positive instances and its IR is 28.1:1, while **Yeast6** contains 35 positives with an IR of 39.2:1. Five-fold cross-validation was used to assess the performance of the models.

PPD[3] is the "Mirror cup" competition dataset, containing 1,000,000 repayment records. It provides relevant attribute information, such as basic information of borrowers, portrait labels and behavior logs. The task is to predict whether a given user will repay on time or not. We used 35 features for the problem, nine of which are basic information features, such as age, and the others are derived features, e.g., total amount of the loan due 30 days before repayment. We used the data from January 1, 2018 to December 31, 2018 for training and the data from February 1, 2019 to March 31, 2019 for testing. There were 700,000 loan records for training, out of which 616,057 instances were good loans and 83,943 instances were default loans, resulting in an IR of 7.3 : 1. The testing set contains 300,000 records, out of which 266,751 were good and 33,249 were default loans, with the IR being 8 : 1.

[3] https://ai.ppdai.com/



(a) PPD



(b) Yeast4

Fig. 7: Efficiencies of all models on the PPD and Yeast4 datasets.

4.3.1 Classification performance

Table 7 shows the F1-score, precision and recall values obtained by *DAPS* and all baseline models on the three datasets. The base classifier used was *DT*. From the results, it is evident that *DAPS* considerably outperformed its alternative approaches in terms of F1-score. On the *Credit* dataset, an extremely imbalanced dataset, *DAPS* and *LDAW-DRW* produced compareable results. However, *DAPS* obtained 53.9%-78.8% higher F1-scores compared with the data-level and the algorithm-level approaches, and 28.6% higher F1-scores compared with the rest baselines, on average. The approaches that utilize an under-sampling scheme, such as *RUS*, $Boosting_{rus}$ and $Bagging_{rus}$ performed quite poorly. The reason behind this most likely lies in the under-sampling scheme which may cause considerable information loss on extremely imbalanced datasets. On the *Yeast4* and *Yeast6* datasets whose both imbalanced ratios are moderate, *DAPS* achieved similar improvements. For instance, *DAPS* produced 7.2%-40.6%, 10.5%-30.4%, 1.1%-37.8%, and 8.5%-41.5% higher F1-scores compared to the data-level, algorithm-level, ensemble-level and state-of-the-art methods, respectively. On the *PPD* dataset whose imbalanced ratio is relatively small, *DAPS* still produced the best result. Namely, *DAPS* produced 0.4%-30% higher F1-scores (p-value $\leq$ 0.037) than all of the baselines, thus showing considerable advantages over its competitors.

From Table 7 we observe that *DAPS* is capable of providing the best trade-off between precision and recall, although it could not achieve the best precision or recall alone. However, some alternatives obtained higher recall values at the expense of precision, and vise versa.

4.3.2 Run-time analysis

We demonstrate the run-time of all methods on different sizes of real datasets in Fig. 7, and combine it with Table 7 to analyze the efficiency and effectiveness of the models.

On the *PPD* dataset, a large-scale dataset, *DAPS* took 858 seconds to finish the training process. The most time-consuming step is the identification of vetoed instances through the k-nearest neighbors technique, as it requires calculating pairwise distances, which took 740 seconds. *RUS* was the fastest approach (see Fig. 7(a)), but obtained 1.5% lower F-score (p-value = 5.8e-09) than *DAPS*. *AdaBoost* and *GBDT* were ∼4.7 times faster than *DAPS*, but obtained much worse results. For example, the F1-score of *GBDT* was 0.02. *XGBoost* took 16 seconds to finish the training process with a F1-score of 0.095. $Boosting_{rus}$, $Bagging_{rus}$, *Cascade* and *SPEnsemble*, which utilize an under-sampling scheme to remove a lot of majority instances, were more efficient than *DAPS*, but they obtained 0.4%-10.9% lower F1-scores (p-value $\leq$ 0.037). Compared with *NB*-based approaches, *DAPS* was approximately 10 times faster and achieved statistically significantly better results, ∼15.2% higher F1-scores (p-values $\leq$ 7e-13). *LDAM-DRW* was 19 times faster than *DAPS* but obtained a much lower F1-score. Thus, *DAPS* provides the best trade-off between accuracy and efficiency on the large-scale datasets.

On the *Yeast4* dataset, which is a medium size dataset, *DAPS* took similar running time as most methods and took much less time than the ensemble-level methods which utilize an over-sampling scheme (such as $Boosting_{smote}$, see Fig. 7(b)). The superiority of *DAPS* in terms of effectiveness and efficiency is evident on the medium-sized datasets.

5 Conclusions

In this paper, we proposed *DAPS*, a novel meta-framework for classification of highly imbalanced, class overlapped and low-quality data. To effectively prevent from vital information loss and noise disturbance, we designed two mechanisms: (1) dynamic self-paced sampling to identify informative instances; and (2) assigning weights to vetoed instances to better handle both non-trivial and noisy data in regions of class overlap. The comprehensive experimental results verify the effectiveness and robustness of *DAPS*.

References

1. Asuncion, A., Newman, D.: UCI machine learning repository (2007)
2. Cao, K., Wei, C., Gaidon, A., Arechiga, N., Ma, T.: Learning imbalanced datasets with label-distribution-aware margin loss. In: Proceedings of the 33rd International Conference on Neural Information Processing Systems, pp. 1567–1578 (2019)
3. Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P.: SMOTE: synthetic minority over-sampling technique. Journal of Artificial Intelligence Research pp. 321–357 (2002)
4. Chawla, N.V., Lazarevic, A., Hall, L.O., Bowyer, K.W.: SMOTEBoost: Improving prediction of the minority class in boosting. In: European conference on principles of data mining and knowledge discovery, pp. 107–119. Springer (2003)
5. Chen, T., T. He, T., Benesty, M., Khotilovich, V., Tang, Y.: Xgboost: extreme gradient boosting. R package version 0.4-2 pp. 1–4 (2015)
6. Freund, Y., Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences pp. 119–139 (1997)

7. Friedman, J.H.: Stochastic gradient boosting. Computational Statistics pp. 367–378 (2002)
8. Gónzalez, S., García, S., Lázaro, M., Figueiras-Vidal, A.R., Herrera, F.: Class switching according to nearest enemy distance for learning from highly imbalanced data-sets. Pattern Recognition pp. 12–24 (2017)
9. He, H., Bai, Y., Garcia, E.A., Li, S.: ADASYN: Adaptive synthetic sampling approach for imbalanced learning. In: IEEE International Joint Conference on Neural Networks, pp. 1322–1328 (2008)
10. Kumar, M.P., Packer, B., Koller, D.: Self-paced learning for latent variable models. In: NIPS, pp. 1189–1197 (2010)
11. Last, F., Douzas, G., Bacao, F.: Oversampling for imbalanced learning based on k-means and smote. arXiv preprint arXiv:1711.00837 (2017)
12. Liu, X.Y., Wu, J., Zhou, Z.H.: Exploratory undersampling for class-imbalance learning. IEEE Transactions on Systems, Man, and Cybernetics pp. 539–550 (2008)
13. Liu, X.Y., Zhou, Z.H.: The influence of class imbalance on cost-sensitive learning: An empirical study. In: International Conference on Data Mining, pp. 970–974. IEEE (2006)
14. Liu, Z., Cao, W., Gao, Z., Bian, J., Chen, H., Chang, Y., Liu, T.: Self-paced ensemble for highly imbalanced massive data classification. In: IEEE 36th International Conference on Data Engineering, pp. 841–852 (2020)
15. Lu, C., Ke, H., Zhang, G., Mei, Y., Xu, H.: An improved weighted extreme learning machine for imbalanced data classification. Memetic Computing pp. 27–34 (2019)
16. O'Brien, R., Ishwaran, H.: A random forests quantile classifier for class imbalanced data. Pattern Recognition pp. 232–249 (2019)
17. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al.: Scikit-learn: Machine learning in python. Journal of Machine Learning Research pp. 2825–2830 (2011)
18. Peng, M., Zhang, Q., Xing, X., Gui, T., Huang, X., Jiang, Y.G., Ding, K., Chen, Z.: Trainable undersampling for class-imbalance learning. In: Proceedings of the AAAI Conference on Artificial Intelligence, pp. 4707–4714 (2019)
19. Pozzolo, A.D., Boracchi, G., Caelen, O., Alippi, C., Bontempi, G.: Credit card fraud detection: a realistic modeling and a novel learning strategy. IEEE Transactions on Neural Networks and Learning Systems pp. 3784–3797 (2017)
20. Seiffert, C., Khoshgoftaar, T.M., Van, H.J., Napolitano, A.: RUSBoost: A hybrid approach to alleviating class imbalance. IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans pp. 185–197 (2009)
21. Vuttipittayamongkol, P., Elyan, E.: Neighbourhood-based undersampling approach for handling imbalanced and overlapped data. Information Sciences pp. 47–70 (2020)
22. Vuttipittayamongkol, P., Elyan, E.: Overlap-based undersampling method for classification of imbalanced medical datasets. In: IFIP International Conference on Artificial Intelligence Applications and Innovations, pp. 358–369. Springer (2020)
23. Vuttipittayamongkol, P., Elyan, E., Petrovski, A., Jayne, C.: Overlap-based undersampling for improving imbalanced data classification. In: International Conference on Intelligent Data Engineering and Automated Learning, pp. 689–697. Springer (2018)
24. Wallace, B.C., Small, K., Brodley, C.E., Trikalinos, T.A.: Class imbalance, redux. In: 2011 IEEE 11th International Conference on Data Mining, pp. 754–763. IEEE (2011)
25. Wang, S., Yao, X.: Diversity analysis on imbalanced data sets by using ensemble models. In: 2009 IEEE Symposium on Computational Intelligence and Data Mining, pp. 324–331. IEEE (2009)
26. Wang, Y., Gan, W., Yang, J., Wu, W., Yan, J.: Dynamic curriculum learning for imbalanced data classification. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 5017–5026 (2019)
27. Wei, W., Li, J., Cao, L., Ou, Y., Chen, J.: Effective detection of sophisticated online banking fraud on extremely imbalanced data. WWW pp. 449–475 (2013)
28. Wu, F., Jing, X.Y., Shan, S., Zuo, W., Yang, J.Y.: Multiset feature learning for highly imbalanced data classification. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 31 (2017)
29. Yuan, X., Xie, L., Abouelenien, M.: A regularized ensemble framework of deep learning for cancer detection from multi-class, imbalanced training data. Pattern Recognition pp. 160–172 (2018)