Towards Automated Imbalanced Learning with Deep Hierarchical Reinforcement Learning

Daochen Zha
Kwei-Herng Lai
daochen.zha@rice.edu
Rice University
Houston, TX, USA

Qiaoyu Tan
Sirui Ding
Na Zou
Texas A&M University
College Station, TX, USA

Xia Hu
xia.hu@rice.edu
Rice University
Houston, TX, USA

ABSTRACT

Imbalanced learning is a fundamental challenge in data mining, where there is a disproportionate ratio of training samples in each class. Over-sampling is an effective technique to tackle imbalanced learning through generating synthetic samples for the minority class. While numerous over-sampling algorithms have been proposed, they heavily rely on heuristics, which could be sub-optimal since we may need different sampling strategies for different datasets and base classifiers, and they cannot directly optimize the performance metric. Motivated by this, we investigate developing a learning-based over-sampling algorithm to optimize the classification performance, which is a challenging task because of the huge and hierarchical decision space. At the high level, we need to decide how many synthetic samples to generate. At the low level, we need to determine where the synthetic samples should be located, which depends on the high-level decision since the optimal locations of the samples may differ for different numbers of samples. To address the challenges, we propose AutoSMOTE, an automated over-sampling algorithm that can jointly optimize different levels of decisions. Motivated by the success of SMOTE [4] and its extensions, we formulate the generation process as a Markov decision process (MDP) consisting of three levels of policies to generate synthetic samples within the SMOTE search space. Then we leverage deep hierarchical reinforcement learning to optimize the performance metric on the validation data. Extensive experiments on six real-world datasets demonstrate that AutoSMOTE significantly outperforms the state-of-the-art resampling algorithms. The code is at https://github.com/daochenzha/autosmote

CCS CONCEPTS

• **Computing methodologies** → **Reinforcement learning**; *Machine learning approaches.*

KEYWORDS

Imbalanced Learning; Reinforcement Learning; Automated Machine Learning; Classification

CIKM '22, October 17–21, 2022, Atlanta, GA, USA

ACM ISBN 978-1-4503-9236-5/22/10…$15.00
https://doi.org/10.1145/3511808.3557474

ACM Reference Format:
Daochen Zha, Kwei-Herng Lai, Qiaoyu Tan, Sirui Ding, Na Zou, and Xia Hu. 2022. Towards Automated Imbalanced Learning with Deep Hierarchical Reinforcement Learning. In *Proceedings of the 31st ACM International Conference on Information and Knowledge Management (CIKM '22), October 17–21, 2022, Atlanta, GA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3511808.3557474

1 INTRODUCTION

Imbalanced learning is a fundamental challenge in many real-world applications, such as fraud detection, fake news detection, and medical diagnosis [12, 31, 33], where there is a disproportionate ratio of training samples in each class. This phenomenon will negatively affect the classification performance since the standard classifiers will tend to be dominated by the majority class and perform poorly on the minority class [5]. A common strategy to tackle imbalanced learning is resampling, which focuses on modifying the training data to balance the data distribution. In contrast to the algorithm-level solutions that modify the classifier [16], resampling is argued to be more flexible as it does not make any assumption on the classifier so that it is generally applicable to various classifiers [32].

Over-sampling is an effective resampling technique through generating new synthetic samples for the minority class [16]. One of the most popular over-sampling methods in the literature is SMOTE [4], which generates synthetic samples by performing linear interpolation between minority instances and their neighbors, illustrated at the top of Figure 1. In contrast to the random over-sampling approach that randomly duplicates the minority instances [12], SMOTE can make the decision regions larger and less specific, which could help alleviate the over-fitting issue [8]. Despite its success, SMOTE can easily generate noisy samples since all the decisions are randomly made. For example, in Figure 1, one of the synthetic samples generated by SMOTE interleaves with the majorities, which could degrade the performance.

Numerous extensions have been proposed to improve SMOTE with better sampling strategies (there are at least 85 SMOTE variants as of the year of 2019 [15]). To name a few, ADASYN [10] generates more synthetic samples for the instances that are harder to learn, which is quantified by the ratio of the majority instances in the nearest neighbors. BorderlineSMOTE [8] and SVMSMOTE [29] only over-sample the minority instances in the borderline, where the former identifies the borderline based on the nearest neighbors and the latter trains an SVM to achieve this. To avoid generating noisy samples, ANS [34] proposes to adapt the number of neighbors needed for each instance based on a 1-nearest neighbor model.

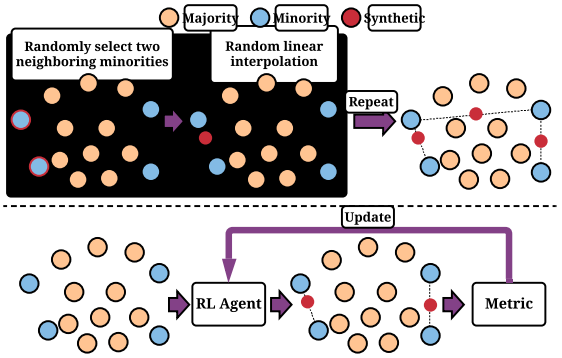

Figure 1: The decisions are randomly made in SMOTE (top) while the decisions in AutoSMOTE are made with an RL agent to optimize the performance on the validation set (bottom).

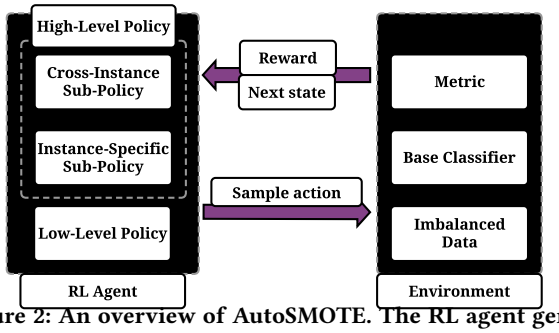

Figure 2: An overview of AutoSMOTE. The RL agent generates synthetic samples (actions) based on the current data distribution (state) with a high-level policy for deciding sampling ratios, and a low-level policy for performing actual sampling, where the high-level policy consists of two sub-policies that collaboratively make decisions. The environment takes as input the action and transits to the next state. The performance metric of the base classifier on the validation data serves as the reward to update the RL agent.

However, the existing SMOTE variants heavily rely on the heuristics to perform over-sampling, which could be sub-optimal. On the one hand, the heuristic sampling strategies are often designed based on some assumptions, such as samples in the borderline are more important [8, 29]. However, the assumptions may not well hold for all the datasets and all the base classifiers, since we may need different sampling strategies in different scenarios. On the other hand, the heuristic sampling strategies cannot directly optimize the performance metric and may not deliver a desirable generalization performance. Motivated by this, we investigate the possibility of developing a learning-based over-sampling algorithm to directly optimize the performance metric. Specifically, we aim to study the following research question: *Given a dataset and a base classifier, how can we optimize the over-sampling strategy such that the trained classifier can achieve the best generalization performance?*

It is non-trivial to achieve the above goal for the following challenges. **First**, it is hard to directly optimize the performance metric. The sampling is independent of the classifier so that it can only indirectly impact the performance. We need an effective mechanism to fill this gap so that the sampling strategy can be learned. **Second**, the over-sampling problem has a huge decision space since the number of generated samples can be arbitrarily large, and each synthetic sample can be anywhere in the feature space. **Third**, over-sampling is a very complex decision that requires hierarchical reasoning. At the high level, we need to decide the over-sampling ratio, i.e., how many synthetic samples should be generated. At the low level, we need to decide where the synthetic samples should be located. The low-level decision depends on the high-level decision in that the optimal locations of the samples may differ for different numbers of samples. The existing algorithms designed for similar problems, such as automated hyperparameter tuning [45] and neural architecture search [6], often focus on a flat and much simpler search space so that they cannot be directly applied to model the potential interaction effect of the different levels of decisions in the over-sampling problem. We need a tailored algorithm to jointly optimize the hierarchical decisions to achieve the best performance.

To tackle the above challenges, we propose AutoSMOTE, an automated over-sampling algorithm that defines the search space based on SMOTE and leverages deep reinforcement learning (RL) to optimize the generalization performance, illustrated at the bottom of Figure 1. Motivated by the success of SMOTE and its extensions, we define a hierarchical search space based on the generation process of SMOTE to reduce the decision space. At the high level, we go through the instances one by one and decide how many samples will be generated around the current instance. At the low level, we decide which neighbors to perform linear interpolation and the interpolation weight to generate a new sample. The high-level policy is further decomposed into a cross-instance sub-policy for predicting an overall over-sampling ratio for all the instances and an instance-specific sub-policy for making personalized decisions for each instance. Then we formulate this hierarchical search space as a Markov decision process (MDP), where the three levels of the policies collaboratively make decisions. We leverage deep hierarchical RL to solve the MDP and jointly train the three levels of the policies to optimize the reward, which is obtained by the performance metric on the validation data. Extensive experiments on six real-world datasets demonstrate the superiority of AutoSMOTE. To summarize, we make the following contributions.

- Formally define the problem of automated over-sampling for imbalanced classification.
- Define a hierarchical search space for this problem based on the generation process of SMOTE. The designed search space can cover all the possible generated samples by SMOTE and most of the SMOTE extensions.
- Propose AutoSMOTE for automated over-sampling. We formulate the decision process as an MDP and leverage deep hierarchical RL to directly optimize the generalization performance. We also present an implementation that runs the sampling and RL training in parallel on CPU and GPU.
- Conduct extensive experiments to evaluate AutoSMOTE. We show that AutoSMOTE outperforms the state-of-the-art samplers under different configurations of imbalanced ratios and base classifiers. In addition, we present comprehensive hyperparameter and ablation studies.

2 PROBLEM STATEMENT

We focus on binary imbalanced classification problems. Let $\mathcal{X} = \{\mathbf{X}^{\text{maj}}, \mathbf{X}^{\text{min}}\}$ be an imbalanced dataset, where $\mathbf{X}^{\text{maj}} \in \mathbb{R}^{N^{\text{maj}} \times D}$ denotes the majority instances, $\mathbf{X}^{\text{min}} \in \mathbb{R}^{N^{\text{min}} \times D}$ denotes the minority instances, N^{maj} is the number of majority instances, N^{min} is the number of minority instances, and D is the feature dimension. We define $\text{IR} = \frac{N^{\text{maj}}}{N^{\text{min}}}$ as the imbalanced ratio, where $\text{IR} > 1$. In a typical classification task, we aim to train a classifier on the training set $\mathcal{X}^{\text{train}}$, tune the performance on the validation set $\mathcal{X}^{\text{val}}$, and evaluate the trained classifier on the testing set $\mathcal{X}^{\text{test}}$. However, when IR is large, the classifier may have poor performance, particularly on the minorities. Over-sampling techniques tackle this problem by augmenting $\mathbf{X}^{\text{min}}$ with some synthetic samples such that the classifier can perform well on both majority and minority classes.

Based on the above notations and intuitions, we formally define the problem of automated over-sampling for imbalanced classification as follows. Given $\mathcal{X}^{\text{train}}$, $\mathcal{X}^{\text{val}}$, and a classifier C, we aim to generate some synthetic samples based on $\mathcal{X}^{\text{train}}$ to improve the generalization performance of C. Formally, the objective is to identify the best synthetic samples $\mathbf{X}^{\text{syn}} \in \mathbb{R}^{N^{\text{syn}} \times D}$, where N^{syn} is the number of synthetic samples, such that the performance of C on $\mathcal{X}^{\text{val}}$ can be maximally improved when training C on $\mathcal{X}^{\text{train}} \cup \{\mathbf{X}^{\text{syn}}\}$.

3 METHODOLOGY

Figure 2 shows an overview of AutoSMOTE. AutoSMOTE on the highest level includes an environment, which trains the base classifier on the over-sampled data and evaluates it on the validation set, and an RL agent, which learns an over-sampling strategy to optimize the performance on the validation set in a trial-and-error fashion. The RL agent consists of a high-level policy, which decides how many synthetic samples will be generated around each original training instance, and a low-level policy, which decides how the instances interpolate with the neighboring instances. The high-level policy is further decomposed into a cross-instance sub-policy for predicting an overall over-sampling ratio for all the instances and an instance-specific sub-policy for making personalized decisions of how many synthetic samples to generate for each instance. The three (sub-)policies work collaboratively to generate synthetic samples, interact with the environment, and are updated based on the reward signal. We first elaborate on the proposed hierarchical synthetic sample generation process in Section 3.1. Then we formulate this process as a Markov decision process (MDP) with hierarchical policies in Section 3.2. Finally, in Section 3.3, we introduce how to optimize the MDP with hierarchical RL and present a practical implementation accelerated by multi-processing.

3.1 Hierarchical Synthetic Sample Generation

Given the training set $\mathcal{X}^{\text{train}}$, the goal is to generate synthetic samples $\mathbf{X}^{\text{syn}} \in \mathbb{R}^{N^{\text{syn}} \times D}$ based on $\mathcal{X}^{\text{train}}$, which is a complex decision because 1) we need to determine the number of samples to generate, i.e., the value of N^{syn}, and 2) we need to decide where each sample is, i.e., the values of each row of $\mathbf{X}^{\text{syn}}$. This leads to an extremely large search space in that there can be an arbitrary number of samples, and each sample can be anywhere in the feature space. This subsection introduces how we reduce the search space and formulate it as a hierarchical decision process based on SMOTE [4].

SMOTE is one of the most popular over-sampling techniques with many extensions built upon it [2, 15]. The key idea is to generate a new sample by performing linear interpolation between a minority instance and one of its nearest neighbors. Specifically, SMOTE augments the minority instances by repeatedly executing the following steps until the desired number of synthetic samples is reached: 1) randomly pick a minority instance, 2) find the nearest minority neighbors of this instance and randomly pick a neighbor, 3) perform linear interpolation between the selected instance and the neighbor to generate a new sample, where the interpolation weight is uniformly sampled in the range of $[0, 1]$; that is, the new sample is a "mixup" of the two original instances and lies between them. We note that all the decisions in SMOTE are randomly made. As such, many heuristics have been proposed to make the sampling process more effective, such as only over-sampling the borderline samples [8], and generating more samples for the instances that are harder to learn [10]. The generated samples of the existing extensions often still fall within the generation space defined by SMOTE.

Motivated by the effectiveness of SMOTE and its extensions, we propose to formulate the search space based on SMOTE with a hierarchical decision process. Figure 3 provides an illustrative example of how we augment four minority instances with a two-level decision. At the high level, we go through the instances one by one and decide how many samples will be generated around the current instance, which leads to a 4-step decision. For each high-level step, we will make a g-step low-level decision to perform linear interpolation, where g is the output of the high-level step. In each low-level step, we decide which neighbors to perform linear interpolation and the interpolation weight to generate a new sample. After going through all the instances, we generate 8 synthetic samples in total.

The proposed generation process has several desirable properties. First, it can significantly reduce the search space because we only need to decide how the current instance interpolates with its neighbors rather than blindly sampling a point in the feature space. Second, it can make personalized decisions. For example, we can generate more samples for some instances and less for the other instances. Third, it can cover all the possible generated samples by SMOTE. Moreover, since the existing SMOTE extensions often follow the generation space of SMOTE, we can essentially cover the majority of the SMOTE extensions as well.

3.2 Formulating the Generation as MDP

This subsection formulates the above hierarchical decision process as MDP. A naive way to achieve this is to use a flat policy. Specifically, in each step, the agent will either make a high-level decision or a low-level decision. Taking the sampling process in Figure 3 as an example, a flat policy will make the following decisions sequentially. In step 1, the agent makes a high-level decision and outputs 1. In step 2, the agent switches to low-level decision and outputs neighbor 2 and $\lambda = 0.25$. In step 3, the agent comes back to the high-level decision and outputs 4. In steps 4 to 7, the agent again makes low-level decisions. Eventually, the agent will take 12 steps in total to complete the generation process.

However, this flat process will make the agent hard to train for two reasons. First, the number of steps of the MDP can be extremely

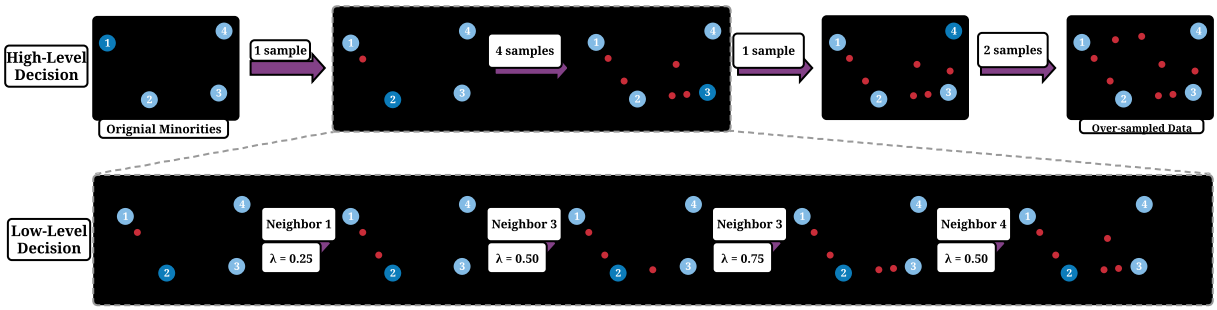

Figure 3: An illustration of the hierarchical decision process. We go through the minority instances one by one, where the darker blue instance is the current instance to be augmented. At the high level, we decide how many synthetic instances will be sampled around the current instance. At the low level, we decide which neighboring instances to perform linear interpolation and the interpolation weights λ. The low-level decision depends on the high-level decision since the length of the low-level sampling is determined by the decisions made in the high-level.

large for the real-world dataset. Solving a long MDP is notoriously difficult [35]. Second, the high-level decision and low-level decision have very different action spaces. Specifically, the high-level actions are the numbers of samples to generate, while the low-level actions are neighbors and interpolation weights. It is difficult to model the two very different action spaces under a unified policy.

To address these issues, we propose to view the MDP from a hierarchical perspective. We decompose the MDP by corresponding the high-level decision with a high-level policy π_h and the low-level decision with a low-level policy π_l, where π_h and π_l use the same state features and rewards with different action spaces. We define the state, high-level/low-level actions, and reward below.

- **State** s: a vector of features describing the data distribution. We empirically use the following features: the original features of the current instance, and the data distribution features of the current instance. We will provide more details of these features after defining the state and reward.
- **High-level action** g: an integer describing the number of samples to generate for the current instance. This essentially defines the goal of the low-level policy.
- **Low-level action** a: an action that describes which neighbor to perform interpolation and the corresponding interpolation weight. We reduce the action space by only considering the top K neighbors. We then discretize the interpolation weight λ and choose it from the set $\{0, 0.25, 0.5, 0.75, 1.0\}$, which leads to $5K$ possible actions in each step. It is possible that formulating it as a continuous action will lead to better performance, which we will study in our future work.
- **Reward** r: the reward is obtained by the performance metric on the validation data. Both π_h and π_l will receive a zero reward in the intermediate steps and receive a reward indicating the performance metric in the final step.

The state features consist of two parts as follows.

- **Original features:** We use the original features of the current instance as the first part of the state features.
- **Data distribution features:** These features describe how many synthetic instances have already been generated around the current instance so that the policies can make decisions based on the previously generated synthetic samples in the MDP. Specifically, we use a 10-dimensional one-hot encoding to achieve this. Whenever an instance is used for linear interpolation (i.e., the instance is either used as the starting point or selected as a neighbor), we add one to the count of this instance. Then we use 10 bins to compress the length of the feature. For example, if the count is in the range of [0, 9], the count is put into the first bin such that the first element of the feature vector is 1 with the others being 0.

The two policies generate samples as follows. Let $N^{\min}$ be the total number of minority instances. The high-level policy will generate one episode with $N^{\min}$ steps. In each high-level step $t_h \in \{1, 2, ..., N^{\min}\}$, π_h takes as input the current state s_{t_h} and proposes the goal for the low-level policy g_{t_h}, i.e., how many samples to generate. Then π_l takes as input g_{t_h} and tries to accomplish this goal by taking g_{t_h} steps, where in each low-level step $t_l \in \{1, 2, ..., g_{t_h}\}$, π_l takes as input the current state s_{t_l} and outputs the sampling action a_{t_l}. The overall generation process will result in one high-level episode with length of $N^{\min}$ and $N^{\min}$ low-level episodes, whose lengths are determined by the outputs of π_h. Then, we obtain a reward with generated samples and set the final steps of all the high-level and low-level episodes to be the obtained reward, while all the intermediate steps receive zero reward.

In our preliminary experiments, we find that when IR is large, we often need to generate more samples. This will significantly enlarge the action space of π_h, which makes the policy harder to train. To reduce the action space, we further decompose π_h into a cross-instance sub-policy $\pi_h^{(1)}$ and an instance-specific sub-policy $\pi_h^{(2)}$, where $\pi_h^{(1)}$ and $\pi_h^{(2)}$ use the same state features and rewarding scheme but differs in the action space. Specifically, $\pi_h^{(1)}$ only takes one step in the generation process and outputs a cross-instance scaling factor $g^{(1)} \in \{0, 1, ..., G^{(1)}\}$, where $G^{(1)}$ is a hyperparameter. $\pi_h^{(2)}$ performs $N^{\min}$ steps and outputs an instance-specific scaling factor $g^{(2)} \in \{0, 1, ..., G^{(2)}\}$ for each instance, where $G^{(2)}$ is a hyperparameter. Then the high-level action is obtained by $g = g^{(1)} \times g^{(2)}$. Algorithm 1 summarizes how $\pi_h^{(1)}$, $\pi_h^{(2)}$ and π_l collaboratively interact with the environment to generate samples.

The above design of three-level hierarchical policies enjoys several advantages. First, it can significantly reduce the length of the episodes. Specifically, the episode length of the $\pi_h^{(1)}$ is only 1, the

Algorithm 1 Generation process of AutoSMOTE

Input: cross-instance sub-policy $\pi_h^{(1)}$, instance-specific sub-policy $\pi_h^{(2)}$, low-level policy π_l, minority instances $\mathbf{X}^{\min} \in \mathbb{R}^{N^{\min} \times D}$, max cross-instance scaling factor G_1, max instance-specific scaling factor G_2, max number of neighbors K
Sample $g^{(1)}$ in the set of $\{0, 1, ..., G^{(1)}\}$ with $\pi_h^{(1)}$
for instance ID = 1, 2, ..., $N^{\min}$ **do**
  Sample $g^{(2)}$ in the set of $\{0, 1, ..., G^{(2)}\}$ with $\pi_h^{(2)}$
  $g \leftarrow g^{(1)} \times g^{(2)}$
  for iteration = 1, 2, ..., g **do**
    Sample a top-K neighbor of the current instance and an interpolation weight with π_l, and generate a new sample
  end for
end for

episode length of $\pi_h^{(2)}$ is $N^{\min}$, and the length of each low-level episode is determined by g, all of which are much smaller than that of the flat counterpart. Second, by further decomposing the high-level policy, we can significantly reduce the action space from $G^{(1)} \times G^{(2)}$ to $G^{(1)}$ and $G^{(2)}$ for the two sub-policies. Third, each level of the hierarchy plays a different role in the decision, where $\pi_h^{(1)}$ makes the dataset-level decisions of the desired over-sampling ratio, $\pi_h^{(2)}$ makes personalized decisions to allow generating more samples for some instances, and π_l performs actual sampling based on the specified goals. As such, we can naturally model them with three separate policies to learn these three very different skills.

3.3 Optimizing the Generation Process with Deep Hierarchical RL

This subsection introduces how we optimize the three-level hierarchical policies with deep hierarchical RL. We first describe the training objectives for the three policies. Then we present the overall training procedure accelerated by multi-processing.

To enable the training of the policies with gradient descent, we parameterize the three policies. Following the idea of actor-critic [35], we associate each of the policies with a policy-value network. For $\pi_h^{(1)}$, we first use an MLP to process the state and produce a state representation. Then the state representation is processed by a policy head and a value head, where the policy head is a fully-connected layer followed by a Softmax layer to produce action probabilities $\pi_h^{(1)}(g^{(1)}|s)$, and the value head $V_h^{(1)}(s)$ is a fully-connected layer with output dimension of 1. We use the same procedure to process $\pi_h^{(2)}$ to obtain $\pi_h^{(2)}(g^{(2)}|s)$ and $V_h^{(2)}(s)$. For π_l, the value head $V_l(s)$ is obtained in the same way as above. For the policy head of π_l, we extract action features, which include data distribution and interpolation weight. The data distribution features are extracted in the same way as the state features, and the interpolation weight features are obtained by one-hot encoding. The action features are then concatenated with the state representation, followed by an MLP to produce confidence scores for state-action pairs. Finally, a Softmax layer is applied to all the state-action pairs to produce action probabilities $\pi_l(a|s)$.

Algorithm 2 Training of AutoSMOTE

1: **Input:** $\pi_h^{(1)}, \pi_h^{(2)}, \pi_l, \mathbf{X}^{\min}, G_1, G_2, K$, total number of iterations I, three buffer sizes $B_h^{(1)}$, $B_h^{(2)}$, and B_l
2: Initialize three queue buffers $\mathcal{B}_h^{(1)}$, $\mathcal{B}_h^{(2)}$, $\mathcal{B}_l$
3: **for** iteration = 1, 2, ..., I **do**
4: Generate samples following Algorithm 1 and store the generated episodes to $\mathcal{B}_h^{(1)}$, $\mathcal{B}_h^{(2)}$ and $\mathcal{B}_l$
5: Train on the augmented training data, get reward on validation data, and set the final steps of all the episodes to be the obtained reward with all the intermediate rewards as 0
6: **if** size($\mathcal{B}_h^{(1)}$) $\geq B_h^{(1)}$ **then**
7: Pop out $B_h^{(1)}$ steps of data and update $\pi_h^{(1)}$ with Eq. 2
8: **end if**
9: **if** size($\mathcal{B}_h^{(2)}$) $\geq B_h^{(2)}$ **then**
10: Pop out $B_h^{(2)}$ steps of data and update $\pi_h^{(2)}$ with Eq. 2
11: **end if**
12: **if** size($\mathcal{B}_l$) $\geq B_l$ **then**
13: Pop out B_l steps of data and update π_l with Eq. 2
14: **end if**
15: **end for**

We adopt the feudal hierarchy approaches [30] to train the policies, where each level of the policies observes the environment in different granularity, and the three policies are updated simultaneously. To train the policies, we adopt IMPALA [7], a modern distributed deep RL algorithm. Here, we mainly introduce the high-level procedure since RL itself is not our focus; for readers who are interested in how IMPALA works, please refer to [7]. Let s_t, a_t, and r_t be the state, action, and reward at step t, respectively. For brevity, here we abuse the notation of a_t, which will actually be $g_t^{(1)}$ and $g_t^{(2)}$ in the context of $\pi_h^{(1)}$ and $\pi_h^{(2)}$, respectively. We consider an n-step trajectory $(s_t, a_t, r_t)_{t=t'}^{t=t'+n}$. IMPALA uses a V-trace target for $s_{t'}$ for tackling the delayed model update:

$$v_{t'} = V(s_{t'}) + \sum_{t=t'}^{t'+n-1} \gamma^{t-t'} \Big(\prod_{i=t'}^{t-1} c_i\Big) \delta_t V, \tag{1}$$

where $V(s_{t'})$ is the value head for $s_{t'}$, $\delta_t V = \rho_t(r_t + \gamma V(s_{t+1}) - V(x_t))$ is the temporal difference, and c_i and ρ_t are truncated importance sampling weights. The loss at step t is defined as

$$L_t = \rho_t \log \pi(a_t|s_t)(r_t + \gamma v_{t+1} - V(s_t)) + \frac{1}{2}(v_t - V(s_t))^2, \tag{2}$$

where $\pi(a_t|s_t)$ is the policy head, and $V(s_t)$is the value head. Batch training will be further used to update the model for multiple steps at a time. The V-trace correction is helpful because training and evaluating a classifier may result in substantial delays. The three policies are updated simultaneously based on Eq. 2.

Algorithm 2 summarizes the training procedure. We first initialize three buffers to temporally store the generated data from the environment in line 2, i.e., tuples of ⟨state, action, reward⟩. In each iteration, we generate new samples (line 4) and obtain rewards on the validation data (line 5). The three policies are updated periodically with the RL objectives (lines 6 to 14). The generated samples

Table 1: Dataset statistics with imbalanced ratios of 20/50/100.

	# Majorities	# Minorities	# Features	Domain
Phoneme	3818	190/76/38	5	Audio
PhishingWebsites	6157	307/123/61	68	Security
EEGEyeState	8257	412/165/82	14	EEG
Mozilla4	10437	521/208/104	5	Product defect
MagicTelescope	12332	616/246/123	10	Telescope
Electricity	26075	1303/521/260	14	Electricity

with the highest validation performance will be used to re-train the classifier, which will be further evaluated on a hold-out testing set.

Multi-Processing Implementation. To maximally exploit the computational resources, we run the sampling (lines 4 and 5) with multiple processes in the CPU and do the learning (lines 6 to 14) in the main process with GPU. We find that the sampling part is CPU-intensive because the base classifier is trained with CPU. In contrast, the learning part can be accelerated with GPU since updating the model can be accelerated by parallelism. Modern GPU servers often have multiple CPU cores. This motivates us to accelerate the training with multi-processing. Specifically, we run multiple actors, where each actor runs a separate clasffier to sample data. Then we run a single learner on GPU to update the model weights with the data collected from the actors. The actors and the learner communicate with shared memory.

4 EXPERIMENTS

The experiments aim to answer the following research questions. **RQ1:** How does AutoSMOTE compare with the state-of-the-art techniques for imbalanced classification (Sections 4.2)? **RQ2:** Can AutoSMOTE outperform the numerous variants of SMOTE (Sections 4.3)? **RQ3:** How does each component contribute to the performance of AutoSMOTE and how does AutoSMOTE compare with simple random search (Sections 4.4)? **RQ4:** How do the hyperparameters impact the performance of AutoSMOTE (Sections 4.5)? **RQ5:** How does the learned balancing strategy of AutoSMOTE compare with the existing over-sampling strategies (Sections 4.6)?

4.1 Experimental Settings

Datasets. The experiments are conducted on six binary classification datasets: **Phoneme**, **PhishingWebsites**, **EEGEyeState**, **Mozilla4**, **MagicTelescope**, and **Electricity**. All the datasets are publicly available at OpenML[1][36]. We perform the following preprocessing steps. First, we identify the numerical features and the categorical features from the datasets. Second, we scale the numerical features with `StardardScaler` in sklearn, which subtracts each value with the mean value and scales the resulting value with the standard deviation. Third, we impute the missing values with 0 for all the features. Fourth, for the categorical features, we use a one-hot encoding. Since these datasets are relatively balanced, following the previous work [2, 38, 54], we artificially create the imbalanced datasets by randomly under-sampling the minority class with different imbalanced ratios. Then we randomly split 60%/20%/20% of the data as the training/validation/testing sets, respectively. Table 1 summarizes the statistics of the imbalanced datasets.

[1]https://www.openml.org/

Baselines. First, we include all the **under-sampling**, **over-sampling**, and **combined over- and under-sampling** methods provided in Imbalanced-learn[2]. Second, we consider two state-of-the-art **generative models** designed for generating realistic samples based on variational autoencoder (TVAE) and generative adversarial networks (CTGAN) [41]. We leverage TVAE and CTGAN to augment the minority class. Third, we involve MESA [26], a state-of-the-art **meta-learning** algorithm, which similarly maximizes the validation performance with ensemble learning of under-sampling strategies. Finally, we compare AutoSMOTE with 85 SMOTE variants provided in Smote-variants package[3] [15] in Section 4.3.

Evaluation Metric. Following the previous work of imbalanced classification [1, 12, 31], we use Macro-F1 and Matthews Correlation Coefficient (MCC) to evaluate the performance. Macro-F1 calculates the F-measure separately for each class and averages them. MCC takes into account true and false positives and negatives. Both of them can well reflect the performance of the minority class.

Base Classifiers and Imbalanced Ratios. The performance of a resampling algorithm is highly sensitive to the adopted base classifier and the imbalanced ratio of the dataset [14, 26]. An algorithm that performs well under one configuration may not necessarily perform well under another configuration. For a comprehensive evaluation, we rank the samplers under 12 configurations with four representative classifiers, including SVM, KNN, DecisionTree and AdaBoost, and with three imbalanced ratios of 20, 50, and 100. We report the average ranks of Macro-F1 or MCC of the samplers across the 12 configurations for each dataset and the overall ranks across all the datasets and configurations. In this way, a higher ranked algorithm tends to perform well under different base classifiers and imbalanced ratios. We run all the experiments five times and report the average results. We further employ Wilcoxon signed rank test [40] to rigorously compare the samplers.

Guidance of Validation Set. For a fair comparison, *all the samplers (including all the baselines) leverage the validation set to search the sampling strategy or tune hyperparameters based on the performance on the validation set*: **1)** for AutoSMOTE, we store the over-sampled data with the best validation performance in search. Then we use the classifier trained on the stored data for evaluation. **2)** for MESA, we similarly use the performance on the validation data to train the ensemble strategy. **3)** for the other baselines, we grid-search the desired ratio of the number of minority samples over the number of majority samples after resampling (which is often a dominant hyperparameter and is suggested to be tuned by [14]) in the set of $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$ on the validation set. For all the samplers, we use the best configuration discovered on the validation set and report the results on the testing set, which is unseen in training or tuning the samplers.

Implementation Details. For AutoSMOTE, we set the max instance-specific scaling factor $G_2 = 10$, the max instance-specific scaling factor $G_1 = 4 \times \text{IR}/G_1$ such that $G_1 \times G_2$ is 4 times of the imbalanced ratio, the max number of neighbors $K = 30$, the total number of iterations $I = 1000$, the buffer sizes $B_h^{(1)} = 2$, $B_h^{(1)} = 300$, $B_l = 300$. We adopt Adam optimizer with a learning rate of 0.005. All the three policies use 128-128 MLP. We run 40

[2]https://imbalanced-learn.org/
[3]https://github.com/analyticalmindsltd/smote_variants

Table 2: Average ranks (the lower the better) of AutoSMOTE and baselines in terms of Macro-F1/MCC. We use ▲ to denote the cases where AutoSMOTE is significantly better than baselines w.r.t. the Wilcoxon signed rank test (p < 0.05).

Category	Method	Dataset						Overall
		Phoneme	PhishingWebsites	EEGEyeState	Mozilla4	MagicTelescope	Electricity	
No-resampling	-	16.50▲/16.75▲	9.75 /9.42	16.08▲/17.08▲	12.75▲/12.83▲	15.92▲/13.00▲	18.17▲/15.67▲	14.86▲/14.12▲
Under-sampling	ClusterCentroids	13.25▲/14.58▲	19.50▲/19.58▲	13.67▲/14.33▲	15.67▲/15.42▲	16.42▲/19.58▲	15.25▲/18.25▲	15.62▲/14.58▲
	CondensedNearestNeighbour	16.62▲/17.46▲	16.92▲/16.96▲	17.75▲/19.17▲	19.88▲/20.46▲	14.33▲/14.92▲	13.58▲/14.58▲	16.51▲/17.19▲
	EditedNearestNeighbours	14.17▲/15.42▲	11.83 /12.42	15.71▲/16.88▲	11.96▲/12.46▲	13.04▲/10.71	15.25▲/15.50▲	13.66▲/13.90▲
	RepeatedEditedNearestNeighbours	14.71▲/17.04▲	15.29▲/15.96▲	15.33▲/17.17▲	13.88▲/14.12▲	10.38▲/8.88	14.62▲/14.88▲	14.03▲/14.67▲
	AllKNN	14.04▲/15.46▲	13.46▲/13.46	15.50▲/16.75▲	13.88▲/14.29▲	10.71▲/8.38	16.08▲/17.17▲	13.94▲/14.25▲
	InstanceHardnessThreshold	14.21▲/13.38▲	20.67▲/20.67▲	14.29▲/14.46▲	17.79▲/18.04▲	10.25 /10.25	12.58▲/12.83▲	14.97▲/14.94▲
	NearMiss	24.42▲/24.75▲	24.17▲/24.42▲	22.58▲/20.92▲	23.25▲/23.33▲	25.00▲/25.00▲	19.58▲/21.08▲	23.17▲/23.25▲
	NeighbourhoodCleaningRule	16.33▲/17.83▲	13.08 /13.25▲	15.12▲/15.88	12.83▲/13.25▲	10.79 /9.12 ▲	12.54▲/11.71▲	13.45▲/13.51▲
	OneSidedSelection	17.21▲/18.21▲	11.08 /10.67▲	15.92▲/16.17▲	13.83▲/14.17▲	15.62▲/13.21▲	17.12▲/15.71▲	15.13▲/14.69▲
	RandomUnderSampler	12.50▲/10.00▲	17.42▲/17.58▲	11.00▲/9.25 ▲	12.83▲/12.92▲	10.17 /11.67	10.83▲/12.25▲	12.46▲/12.28▲
	TomekLinks	16.88▲/17.46▲	10.04 /9.38	14.12▲/14.04▲	14.12▲/14.29▲	15.62▲/12.96▲	17.38▲/16.21▲	14.69▲/14.06▲
Over-sampling	RandomOverSampler	6.75 /8.17	12.33▲/12.75▲	5.00 /5.58	8.00 ▲/8.42 ▲	9.92 ▲/13.33	8.58 ▲/10.83▲	8.43 ▲/9.85 ▲
	SMOTE	7.25 /8.67 ▲	10.42▲/10.67▲	7.00 ▲/6.67	12.00▲/12.00▲	11.42▲/14.17▲	7.17 ▲/7.42 ▲	9.21 ▲/9.93 ▲
	SMOTEN	16.83▲/18.25▲	10.71 /10.54	12.42▲/15.58▲	9.58 ▲/10.08	18.17▲/17.33▲	17.83▲/18.67▲	14.26▲/15.08▲
	ADASYN	7.33 /8.00	9.75 /9.58	7.50 ▲/8.17 ▲	12.58▲/12.25▲	10.17 /12.08▲	8.00 ▲/8.50 ▲	9.22 ▲/9.76 ▲
	BorderlineSMOTE	6.92 /8.67	9.42 ▲/9.25	9.67 ▲/10.92▲	9.17 ▲/9.33 ▲	7.50 /9.75	4.67 /5.08	7.89 ▲/8.83 ▲
	KMeansSMOTE	15.92▲/16.67▲	10.00 /9.83	16.08▲/16.92▲	12.83▲/12.79▲	14.92▲/12.17	17.92▲/15.42▲	14.61▲/13.97▲
	SVMSMOTE	6.25 /9.08 ▲	10.17▲/10.00	7.25 /7.75	8.25 ▲/9.33 ▲	6.67 /8.58	4.50 /4.83	7.18 ▲/8.26 ▲
Combined over- and under-sampling	SMOTEENN	6.25 /6.50	14.67 /14.50	7.17 /6.92	10.75▲/10.08	8.00 /7.42	9.75 ▲/9.67 ▲	9.43 ▲/9.18 ▲
	SMOTETomek	8.67 /9.25 ▲	9.58 /9.75	6.67 ▲/6.42	11.42▲/11.08▲	8.58 /10.25	7.75 ▲/7.92 ▲	8.78 ▲/9.11 ▲
Generative models	CTGAN	12.08 /9.33 ▲	11.75▲/11.42▲	11.50▲/12.58	10.42▲/9.25 ▲	15.17 /15.42▲	12.75 /11.83▲	12.28▲/11.64▲
	TVAE	14.25▲/12.17▲	9.42 /9.17	23.50▲/18.50▲	16.17▲/16.58▲	20.92▲/20.92▲	19.58▲/17.25▲	17.31▲/15.76▲
Meta-learning	MESA	19.92▲/7.33	17.08▲/16.50▲	20.17▲/12.17▲	17.50▲/13.25▲	19.58▲/18.92▲	20.83▲/18.50▲	19.18▲/14.44▲
Auto-sampling	AutoSMOTE	**5.75 /4.58**	**6.50 /7.67**	**4.00 /4.75**	**3.67 /4.96**	**5.75 /7.00**	**2.67 /3.25**	**4.72 /5.37**

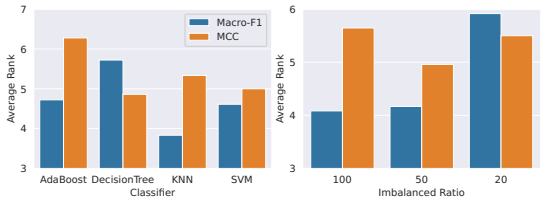

Figure 4: Performance of AutoSMOTE with different base classifiers (left) and imbalanced ratios (right).

Table 3: AutoSMOTE versus 85 SMOTE variants. We only list the average ranks of the top 5 algorithms due to space limitation. ▲ suggests AutoSMOTE is significantly better.

Sampler	Macro-F1	Sampler	MCC
AutoSMOTE	**8.83**	AutoSMOTE	**15.08**
SupervisedSMOTE	15.22▲	SupervisedSMOTE	16.56
GASMOTE	15.25▲	ClusterSMOTE	27.29▲
ClusterSMOTE	18.58▲	ANDSMOTE	28.39▲
SMOTEPSO	22.11▲	BorderlineSMOTE1	28.94▲

actors in parallel to train the base classifiers. For the baselines, we use authors' implementation[4] with the default hyperparameters except that we change the metric to Macro-F1 and MCC. For TVAE and CTGAN, we use the authors' implementation[5] with the default hyperparameters. We use NVIDIA GeForce RTX 2080 Ti GPUs.

4.2 Comparison with the Baselines

To answer **RQ1**, we compare AutoSMOTE against the state-of-the-art resampling methods for imbalanced classification. Table 2 reports the overall ranks of Macro-F1 and MCC of the samplers on each of the datasets across all the configurations of base classifier and imbalanced ratio. We further report the overall ranks of AutoSMOTE under different classifiers and imbalanced ratios in Figure 4 to show insights into how different configurations impact the performance. We make the following observations.

First, AutoSMOTE significantly and consistently outperforms all the baselines across all the datasets. Specifically, AutoSMOTE is ranked the top among the 25 samplers for all the datasets, and the overall ranks across the datasets are significantly better than all the baselines w.r.t. Wilcoxon signed rank test. This demonstrates the superiority of the RL-based sampling strategy. An interesting observation is that while MESA also performs searching on the validation set, it cannot deliver competitive performance as AutoSMOTE. A possible explanation is that MESA searches the ensemble strategies of under-sampling, which could lose information. This phenomenon can also be verified from the observation that the over-sampling methods outperform the under-sampling methods in general for all the datasets. AutoSMOTE performs better than the two generative models, which is because AutoSMOTE can optimize the generalization performance with RL while the generative models can only model the data distribution. This also suggests that the SMOTE search space is effective since we can identify very strong synthetic strategies within the search space.

Second, AutoSMOTE delivers consistent performance across different classifiers and imbalanced ratios. The average rank of AutoSMOTE is better than 6.3 across all the classifiers and better than 6 across all the imbalanced ratios. The results suggest that AutoSMOTE can well accommodate different configurations. A possible reason is that AutoSMOTE can identify personalized synthetic strategies for different configurations with RL.

[4]https://github.com/ZhiningLiu1998/mesa
[5]https://github.com/sdv-dev/CTGAN

Table 4: Average ranks of AutoSMOTE and the ablations on Mozilla4. ▲ suggests full AutoSMOTE is significantly better.

	Macro-F1	MCC
w/o cross-instance sub-policy	3.79▲	4.12▲
w/o instance-specific sub-policy	2.75	3.75▲
w/o low-level policy	4.58▲	4.21▲
Flat policy	4.67▲	5.25▲
Random search	6.92▲	5.75▲
Merging training and validation sets	2.83	2.83
Full AutoSMOTE	**2.46**	**2.08**

4.3 Comparison with SMOTE Variants

Since AutoSMOTE builds upon the SMOTE search space, we compare AutoSMOTE with the numerous SMOTE variants to investigate **RQ2**. We report the overall ranks of AutoSMOTE against 85 SMOTE variants across all the datasets in Table 3. We observe that AutoSMOTE significantly outperforms the 85 SMOTE variants. It is ranked 8.85 and 15.08 among all the samplers in terms of Macro-F1 and MCC, respectively. We note that AutoSMOTE shares the same search space as the majority of the SMOTE variants. Thus, the improvement can be mainly attributed to searching with RL.

4.4 Ablation Study

To study **RQ3**, we compare AutoSMOTE with several ablations. First, we remove each of the three policies in the hierarchy to show that each level of the policies contributes to the performance. Specifically, for removing the cross-instance sub-policy, we make $g_1 = G_1/2$ (which is the mean value of g_1 when we use cross-instance sub-policy). For removing the instance-specific sub-policy, we assume $g_2 = G_2/2$. For removing the low-level policy, we assume the neighbors and the interpolation weights are randomly selected. Second, we consider a flat policy baseline, which directly makes the low-level decisions. Specifically, the flat policy directly predicts the neighbors, interpolation weights, and a boolean indicating whether to go to the next instance. The flat policy is also trained with RL. Third, we consider a random search baseline[6], which randomly makes the high-level and low-level decisions within the same decision space as AutoSMOTE. The generated synthetic samples that lead to the best validation performance are used for evaluation. Fourth, we consider a variant that trains the classifier on both the training set and validation set. Specifically, we merge the original training and validation sets to form a new training set. Then the classifier and AutoSMOTE are both trained on the new training set. This ablation is designed to study whether it is necessary to separate a validation set. For a fair comparison, we set $I = 1000$ for AutoSMOTE and all the ablations.

Table 4 shows the ranks of AutoSMOTE and the ablations on the Mozilla4 dataset. We observe that AutoSMOTE outperforms all the ablations. First, removing either of the three policies will degrade the performance, which demonstrates the necessity of modeling each level of the decision. Second, the flat policy is worse than AutoSMOTE, which is expected because the flat policy suffers from very long MDP, making RL harder to train. Third, random search shows very poor performance. This is because it is hard to identify a strong synthetic strategy within the massive search space of SMOTE, which demonstrates the effectiveness of RL. Finally, although merging training and validation sets can provide more data to the classifier, it may negatively affect the performance, which could be explained by over-fitting. We observe that it can achieve near-perfect performance on the training set while the testing performance remains low. Thus, it is necessary to separate a validation set so that we can optimize the generalization performance.

[6]We have tried applying the existing techniques designed for hyperparameter tuning or neural architecture search to our problem. However, we find they are often not applicable because they cannot deal with the hierarchical search space, where the low-level search space has variable sizes since it depends on the high-level decisions.

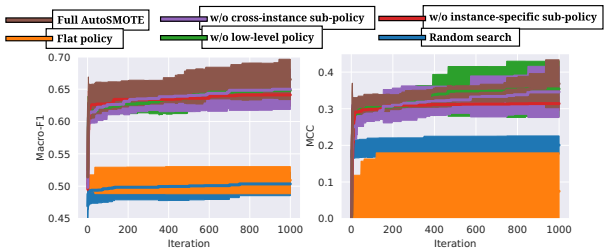

Figure 5: Performance of AutoSMOTE and the ablations w.r.t. the number of searching iterations on Mozilla4 with imbalanced ratio of 50 and base classifier of SVM.

To better understand the searching efficiency, we visualize in Figure 5 the best validation performance w.r.t. the number of search iterations on Mozilla4 with an imbalanced ratio of 50 and SVM as the base classifier. Note that we have excluded the ablation that merges the validation set because it tends to over-fit the validation data so that its validation performance is meaningless. We can observe that AutoSMOTE achieves better results in terms of both final performance and sample efficiency, i.e., achieving good performance with less number of iterations. We also observe that the flat policy and random search get stuck during the search, which again verifies the superiority of hierarchical RL.

4.5 Analysis of the Hyperparameters

For **RQ4**, we study the impact of G_1, G_2, and K on Mozilla4 dataset. First, we fix $G_2 = 10$ and $K = 30$, and vary G_1 such that $G_1 \times G_2/\text{IR}$ ranges from 1 to 8 (left-hand side of Figure 6). The best performance is achieved when $G_1 \times G_2/\text{IR} = 4$. A possible explanation is that a too low ratio will make the search space too restricted to discover good synthetic strategies, while a too-large ratio will make searching more difficult. Second, we fix $G_1 \times G_2/\text{IR} = 4$ and $K = 30$, and vary G_2 (middle of Figure 6). We find a too small G_2 will worsen the performance. Recall that the instance-specific sub-policy makes personalized decisions. A small G_2 will restrict the personalization, which could explain why it causes unsatisfactory performance. Similarly, we observe a performance drop when G_2 is large, which could also be explained by the difficulty brought by the larger search space. Third, we fix G_1 and G_2, and vary K. We observe a significant performance drop when K is very small, which verifies the effectiveness of performing interpolation.

Overall, we observe that G_1, G_2, and K will control the trade-off between performance and searching efficiency in each level of the decisions. If the value is too small, it may restrict the search space and lead to worse performance. In contrast, a too-large value tends to make the searching more difficult and may also negatively affect the performance given a limited searching budget.

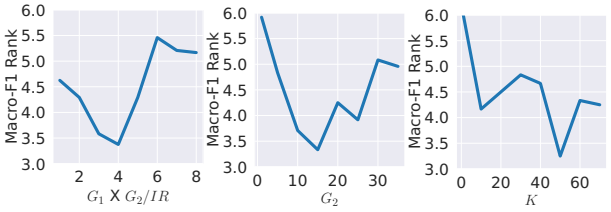

Figure 6: Hyperparameter study on Mozilla4.

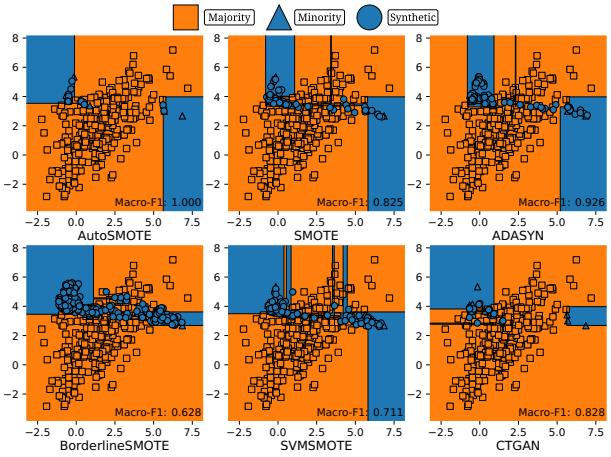

Figure 7: Visualization of the generated synthetic samples and the decision boundary of DecisionTree on a toy data using AutoSMOTE and other over-sampling techniques.

4.6 Case Study

To answer **RQ5**, we apply AutoSMOTE, several SMOTE variants, and CTGAN to a 2-dimensional toy dataset. This dataset is publicly available[7] and is originally synthesized for anomaly detection. We under-sample the minority instances with an imbalanced ratio of 30, which results in 450 majority instances and 35 minority instances. Then we split 60% of the data for training, 20% of the data for validation, and 20% for testing. This toy dataset is challenging because the minorities form two clusters in the two sides of the majority instances so that the sampler could easily generate noisy samples. We use Macro-F1 as the performance metric. Figure 7 illustrates the generated synthetic samples and the decision boundary obtained by training a DecisionTree classifier on the over-sampled data. We observe that all the samplers except AutoSMOTE tend to generate some noisy samples that interleave with the majorities between the two clusters, which degrades the performance. In contrast, AutoSMOTE can identify a tailored synthetic strategy that achieves 1.0 Macro-F1 through directly optimizing the performance metric, which demonstrates the effectiveness of the learning-based sampler.

5 RELATED WORK

Imbalanced Learning. Existing methods can be mainly grouped into three categories: data-level methods that aim to balance the data distributions by modifying the training data [3, 12, 31, 37, 38, 54], algorithm-level approaches that try to modify the classifier such as the loss function [19, 25], and hybrid solutions that combine the above two [16]. Our work falls into data-level methods. Data-level methods can be further divided into three subcategories, including generating new samples for the minority class (over-sampling) [4, 10], removing instances from the majority class (under-sampling) [44], and hybrid methods that combine the above two [28]. One advantage of over-sampling is that it will not lose information. However, it may be prone to over-fitting. AutoSMOTE tackles this problem by optimizing the generalization performance, thereby alleviating the over-fitting issue. A recent work [26] proposes a meta-learning algorithm, which similarly aims to optimize the performance metric on the validation data. However, they focus on learning ensemble strategies for under-sampling, which may lose information. In contrast, AutoSMOTE is an over-sampling method so that the resampled data can preserve all the information. Another line of imbalanced learning work is outlier detection [9, 18], which is often an unsupervised task.

Automated Machine Learning (AutoML). AutoML techniques have recently show promise in various data mining tasks [13, 17, 20–24, 39, 42, 43, 46, 48, 52, 55, 56]. The key idea is to leverage machine learning to optimize data mining solutions, such as neural architecture search [6], hyperparameter tuning [45], and pipeline search [11]. AutoSMOTE also falls into this line of research. In contrast to the previous work, over-sampling exhibits unique challenges with a huge and complex decision space that requires hierarchical reasoning. The existing AutoML approaches often can only deal with simple and flat search spaces. To this end, we develop a tailored search space based on SMOTE and leverage deep hierarchical RL to jointly optimize different levels of the decisions.

Deep RL. Deep RL has achieved remarkable success in games [7, 27, 47, 49–51, 53]. Deep RL is designed for goal-oriented tasks, where the agent is trained based on the reward signal. Recently, deep hierarchical RL has shown promise in tackling tasks with long horizons [30]. The key idea is decompose the MDP into decisions in different granularities such that each level of the decisions can be more effectively leaned. However, the successes of RL are often only demonstrated in simulated games. Our work suggests that deep hierarchical RL can help tackle the data imbalance problem, which is a real-world data mining challenge.

6 CONCLUSIONS AND FUTURE WORK

This work investigates learning-based sampling algorithms for tackling the imbalanced learning problem. To this end, we first formulate a hierarchical search space based on SMOTE. Then we leverage deep hierarchical RL to jointly optimize different levels of decisions to achieve the best generalization performance on the validation set. The proposed over-sampling algorithm, namely AutoSMOTE, is evaluated against the state-of-the-art samplers and also numerous SMOTE variants. Extensive experiments demonstrate the superiority of AutoSMOTE over heuristics. In the future, we will extend AutoSMOTE to perform combined over- and under-sampling and deal with other data types such as images, graphs, and time series.

ACKNOWLEDGEMENTS

The work is, in part, supported by NSF (#IIS-1849085, #IIS-1900990, #IIS-1939716). The views and conclusions in this paper should not be interpreted as representing any funding agencies.

[7] https://github.com/shubhomoydas/ad_examples/tree/master/ad_examples/datasets/anomaly/toy2/fullsamples

REFERENCES

[1] Sabri Boughorbel, Fethi Jarray, and Mohammed El-Anbari. 2017. Optimal classifier for imbalanced data using Matthews Correlation Coefficient metric. *PloS one* 12, 6 (2017), e0177678.

[2] Mateusz Buda, Atsuto Maki, and Maciej A Mazurowski. 2018. A systematic study of the class imbalance problem in convolutional neural networks. *Neural Networks* 106 (2018), 249–259.

[3] Zixin Cai, Xinyue Wang, Mingjie Zhou, Jian Xu, and Liping Jing. 2019. Supervised class distribution learning for GANs-based imbalanced classification. In *ICDM*.

[4] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. 2002. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research* 16 (2002), 321–357.

[5] Nitesh V Chawla, Nathalie Japkowicz, and Aleksander Kotcz. 2004. Special issue on learning from imbalanced data sets. *ACM SIGKDD explorations newsletter* 6, 1 (2004), 1–6.

[6] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2019. Neural architecture search: A survey. *The Journal of Machine Learning Research* 20, 1 (2019), 1997–2017.

[7] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. 2018. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *ICML*.

[8] Hui Han, Wen-Yuan Wang, and Bing-Huan Mao. 2005. Borderline-SMOTE: a new over-sampling method in imbalanced data sets learning. In *ICIC*.

[9] Songqiao Han, Xiyang Hu, Hailiang Huang, Mingqi Jiang, and Yue Zhao. 2022. ADBench: Anomaly Detection Benchmark. *arXiv preprint arXiv:2206.09426* (2022).

[10] Haibo He, Yang Bai, Edwardo A Garcia, and Shutao Li. 2008. ADASYN: Adaptive synthetic sampling approach for imbalanced learning. In *IJCNN*.

[11] Yuval Heffetz, Roman Vainshtein, Gilad Katz, and Lior Rokach. 2020. Deepline: Automl tool for pipelines generation using deep reinforcement learning and hierarchical actions filtering. In *KDD*.

[12] Justin M Johnson and Taghi M Khoshgoftaar. 2019. Survey on deep learning with class imbalance. *Journal of Big Data* 6, 1 (2019), 1–54.

[13] Patrick Koch, Oleg Golovidov, Steven Gardner, Brett Wujek, Joshua Griffin, and Yan Xu. 2018. Autotune: A derivative-free optimization framework for hyperparameter tuning. In *KDD*.

[14] György Kovács. 2019. An empirical comparison and evaluation of minority oversampling techniques on a large number of imbalanced datasets. *Applied Soft Computing* 83 (2019), 105662.

[15] György Kovács. 2019. Smote-variants: A python implementation of 85 minority oversampling techniques. *Neurocomputing* 366 (2019), 352–354.

[16] Bartosz Krawczyk. 2016. Learning from imbalanced data: open challenges and future directions. *Progress in Artificial Intelligence* 5, 4 (2016), 221–232.

[17] Kwei-Herng Lai, Daochen Zha, Guanchu Wang, Junjie Xu, Yue Zhao, Devesh Kumar, Yile Chen, Purav Zumkhawaka, Minyang Wan, Diego Martinez, et al. 2021. TODS: An Automated Time Series Outlier Detection System. In *AAAI*.

[18] Kwei-Herng Lai, Daochen Zha, Junjie Xu, Yue Zhao, Guanchu Wang, and Xia Hu. 2021. Revisiting time series outlier detection: Definitions and benchmarks. In *NeurIPS*.

[19] Mingchen Li, Xuechen Zhang, Christos Thrampoulidis, Jiasi Chen, and Samet Oymak. 2021. AutoBalance: Optimized Loss Functions for Imbalanced Data. *NeurIPS* (2021).

[20] Ting Li, Junbo Zhang, Kainan Bao, Yuxuan Liang, Yexin Li, and Yu Zheng. 2020. Autost: Efficient neural architecture search for spatio-temporal prediction. In *KDD*.

[21] Yuening Li, Zhengzhang Chen, Daochen Zha, Kaixiong Zhou, Haifeng Jin, Haifeng Chen, and Xia Hu. 2021. Automated Anomaly Detection via Curiosity-Guided Search and Self-Imitation Learning. *IEEE Transactions on Neural Networks and Learning Systems* (2021).

[22] Yuening Li, Zhengzhang Chen, Daochen Zha, Kaixiong Zhou, Haifeng Jin, Haifeng Chen, and Xia Hu. 2021. Autood: Neural architecture search for outlier detection. In *ICDE*.

[23] Yuening Li, Daochen Zha, Praveen Venugopal, Na Zou, and Xia Hu. 2020. Pyodds: An end-to-end outlier detection system with automated machine learning. In *WWW*.

[24] Kunpeng Liu, Yanjie Fu, Pengfei Wang, Le Wu, Rui Bo, and Xiaolin Li. 2019. Automating feature subspace exploration via multi-agent reinforcement learning. In *KDD*.

[25] Xu-Ying Liu and Zhi-Hua Zhou. 2006. The influence of class imbalance on cost-sensitive learning: An empirical study. In *ICDM*.

[26] Zhining Liu, Pengfei Wei, Jing Jiang, Wei Cao, Jiang Bian, and Yi Chang. 2020. MESA: Boost Ensemble Imbalanced Learning with MEta-SAmpler. In *NeurIPS*.

[27] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[28] Ajinkya More. 2016. Survey of resampling techniques for improving classification performance in unbalanced datasets. *arXiv preprint arXiv:1608.06048* (2016).

[29] Hien M Nguyen, Eric W Cooper, and Katsuari Kamei. 2011. Borderline oversampling for imbalanced data classification. *International Journal of Knowledge Engineering and Soft Data Paradigms* 3, 1 (2011), 4–21.

[30] Shubham Pateria, Budhitama Subagdja, Ah-hwee Tan, and Chai Quek. 2021. Hierarchical Reinforcement Learning: A Comprehensive Survey. *ACM Computing Surveys (CSUR)* 54, 5 (2021), 1–35.

[31] Neelam Rout, Debahuti Mishra, and Manas Kumar Mallick. 2018. Handling imbalanced data: a survey. In *ASISA*.

[32] B Santoso, H Wijayanto, KA Notodiputro, and B Sartono. 2017. Synthetic over sampling methods for handling class imbalanced problems: A review. In *IOP conference series: earth and environmental science*, Vol. 58. IOP Publishing, 012031.

[33] Kai Shu, Amy Sliva, Suhang Wang, Jiliang Tang, and Huan Liu. 2017. Fake news detection on social media: A data mining perspective. *ACM SIGKDD explorations newsletter* 19, 1 (2017), 22–36.

[34] Wacharasak Siriseriwan and Krung Sinapiromsaran. 2017. Adaptive neighbor synthetic minority oversampling technique under 1NN outcast handling. *Songklanakarin J. Sci. Technol* 39, 5 (2017), 565–576.

[35] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[36] Joaquin Vanschoren, Jan N Van Rijn, Bernd Bischl, and Luis Torgo. 2014. OpenML: networked science in machine learning. *ACM SIGKDD Explorations Newsletter* 15, 2 (2014), 49–60.

[37] Jing Wang and Min-Ling Zhang. 2018. Towards mitigating the class-imbalance problem for partial label learning. In *KDD*.

[38] Wentao Wang, Suhang Wang, Wenqi Fan, Zitao Liu, and Jiliang Tang. 2020. Global-and-local aware data generation for the class imbalance problem. In *SDM*.

[39] Yicheng Wang, Xiaotian Han, Chia-Yuan Chang, Daochen Zha, Ulisses Braga-Neto, and Xia Hu. 2022. Auto-PINN: Understanding and Optimizing Physics-Informed Neural Architecture. *arXiv preprint arXiv:2205.13748* (2022).

[40] Wikipedia. 2022. Wilcoxon signed-rank test — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Wilcoxon%20signed-rank%20test&oldid=1084875027. [Online; accessed 19-May-2022].

[41] Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. 2019. Modeling Tabular data using Conditional GAN. In *NeurIPS*.

[42] Chengrun Yang, Yuji Akimoto, Dae Won Kim, and Madeleine Udell. 2019. OBOE: Collaborative filtering for AutoML model selection. In *KDD*.

[43] Chengrun Yang, Jicong Fan, Ziyang Wu, and Madeleine Udell. 2020. Automl pipeline selection: Efficiently navigating the combinatorial space. In *KDD*.

[44] Show-Jane Yen and Yue-Shi Lee. 2006. Under-sampling approaches for improving prediction of the minority class in an imbalanced dataset. In *Intelligent Control and Automation*. Springer, 731–740.

[45] Tong Yu and Hong Zhu. 2020. Hyper-parameter optimization: A review of algorithms and applications. *arXiv preprint arXiv:2003.05689* (2020).

[46] Daochen Zha, Louis Feng, Bhargav Bhushanam, Dhruv Choudhary, Jade Nie, Yuandong Tian, Jay Chae, Yinbin Ma, Arun Kejariwal, and Xia Hu. 2022. AutoShard: Automated Embedding Table Sharding for Recommender Systems. In *KDD*.

[47] Daochen Zha, Kwei-Herng Lai, Songyi Huang, Yuanpu Cao, Keerthana Reddy, Juan Vargas, Alex Nguyen, Ruzhe Wei, Junyu Guo, and Xia Hu. 2021. RLCard: a platform for reinforcement learning in card games. In *IJCAI*.

[48] Daochen Zha, Kwei-Herng Lai, Mingyang Wan, and Xia Hu. 2020. Meta-AAD: Active anomaly detection with deep reinforcement learning. In *ICDM*.

[49] Daochen Zha, Kwei-Herng Lai, Kaixiong Zhou, and Xia Hu. 2019. Experience Replay Optimization. In *IJCAI*.

[50] Daochen Zha, Kwei-Herng Lai, Kaixiong Zhou, and Xia Hu. 2021. Simplifying deep reinforcement learning via self-supervision. *arXiv preprint arXiv:2106.05526* (2021).

[51] Daochen Zha, Wenye Ma, Lei Yuan, Xia Hu, and Ji Liu. 2021. Rank the Episodes: A Simple Approach for Exploration in Procedurally-Generated Environments. In *ICLR*.

[52] Daochen Zha, Zaid Pervaiz Bhat, Yi-Wei Chen, Yicheng Wang, Sirui Ding, Anmoll Kumar Jain, Mohammad Qazim Bhat, Kwei-Herng Lai, Jiaben Chen, et al. 2022. AutoVideo: An Automated Video Action Recognition System. In *IJCAI*.

[53] Daochen Zha, Jingru Xie, Wenye Ma, Sheng Zhang, Xiangru Lian, Xia Hu, and Ji Liu. 2021. DouZero: Mastering DouDizhu with Self-Play Deep Reinforcement Learning. In *ICML*.

[54] Tianxiang Zhao, Xiang Zhang, and Suhang Wang. 2021. Graphsmote: Imbalanced node classification on graphs with graph neural networks. In *WSDM*.

[55] Xiangyu Zhao, Haochen Liu, Wenqi Fan, Hui Liu, Jiliang Tang, Chong Wang, Ming Chen, Xudong Zheng, Xiaobing Liu, and Xiwang Yang. 2021. Autoemb: Automated embedding dimensionality search in streaming recommendations. In *ICDM*.

[56] Yue Zhao, Ryan Rossi, and Leman Akoglu. 2021. Automatic unsupervised outlier model selection. *NeurIPS* (2021).