Trainable Undersampling for Class-Imbalance Learning

Minlong Peng,[1] Qi Zhang,[1] Xiaoyu Xing,[1] Tao Gui,[1] Xuanjing Huang,[1]
Yu-Gang Jiang,[1] Keyu Ding,[2] Zhigang Chen[2]
School of Computer Science, Shanghai Key Laboratory of Intelligent Information Processing,
Fudan University, Shanghai Insitute of Intelligent Electroics & Systems, Shanghai, China
[1]{mlpeng16, qz, xyxing14, tgui16, xjhuang, ygj}@fudan.edu.cn
[2]{kyding, zgcheng}@iflytek.com

Abstract

Undersampling has been widely used in the class-imbalance learning area. The main deficiency of most existing undersampling methods is that their data sampling strategies are heuristic-based and independent of the used classifier and evaluation metric. Thus, they may discard informative instances for the classifier during the data sampling. In this work, we propose a meta-learning method built on the undersampling to address this issue. The key idea of this method is to parametrize the data sampler and train it to optimize the classification performance over the evaluation metric. We solve the non-differentiable optimization problem for training the data sampler via reinforcement learning. By incorporating evaluation metric optimization into the data sampling process, the proposed method can learn which instance should be discarded for the given classifier and evaluation metric. In addition, as a data level operation, this method can be easily applied to arbitrary evaluation metric and classifier, including non-parametric ones (e.g., C4.5 and KNN). Experimental results on both synthetic and realistic datasets demonstrate the effectiveness of the proposed method.

Introduction

In many application areas of data mining and machine learning, the problem of class-imbalance is ubiquitous and tasks in these areas are commonly to distinguish the minority classes or achieve a balanced classification performance (Van Hulse, Khoshgoftaar, and Napolitano 2007). In this situation, conventional accuracy-based measurements are usually misleading because they are highly dependent on the classification accuracy of the majority classes. Therefore, many more appropriate and domain interest measurements such as the F-measures, area under the curve (AUC) (Hanley and McNeil 1982), and geometric mean (GM), were developed. In general, it is assumed that a classifier works well for the class-imbalanced task if it can achieve a good performance over the given evaluation metric. However, most of the existing learning algorithms were designed to improve the accuracy (Ganganwar 2012), instead of the given evaluation metric, by minimizing the training loss. Thus there is actually a gap between the training object of the supervised classifier and the task object revealed by the evaluation metric.

Undersampling has been widely used to narrow this gap in the class-imbalance learning area. The prevailing undersampling strategies undersample instances of majority classes using different heuristics (Cieslak and Chawla 2008; Wilson 1972; Mani and Zhang 2003; Tomek 1976b; 1976a), with the hope of arriving at a more robust and fair decision boundary for the evaluation metric. The sampling probability of each example is usually decided by the global or local imbalance ratio (Cieslak and Chawla 2008) and the hyper-parameters, which are adjusted to obtain better performance over the evaluation metric. However, these undersampling strategies are usually heuristic-based. They do not take into account the form of the used classifier and evaluation metric. Thus, even using fine-tuned hyper-parameters, these strategies do not guarantee to obtain an appropriate subset matching the task object (Batista, Prati, and Monard 2004; He and Garcia 2009). A typical problem of these strategies is that they may throw away potentially useful data (Liu, Wu, and Zhou 2009).

In this work, we propose a meta-learning method built on the undersampling to address the above issues. We parametrize the data sampler and train it to optimize the classification performance over the evaluation metric. Therefore, different from previous undersampling strategies that sample instances heuristically, the parametrized data sampler is trained to distinguish which instances should be discard and which instances should be preserved. We approach the non-differentiable optimization problem for training the data sampler via reinforcement learning. Specifically, we formulate the data sampling procedure as a Markov decision process (MDP), which takes the sampling operation of each example as the action, the chosen subset as the state, and the performance of the classifier trained using the chosen subset over the evaluation metric as the reward. We show that the convergence of this algorithm is guaranteed by that of the policy search algorithm (Williams 1992). For evaluating the proposed method, we performed experiments on both synthetic and realistic imbalanced datasets. The experimental results show that the proposed method can consistently outperform different heuristic-based data sampling methods, including undersampling and oversampling, and it can also achieve comparable

performance with the specifically designed state-of-the-art cost-sensitive learning methods.

The contributions of this work can be summarized as follows: **1)** We propose a meta-learning method to incorporate the evaluation metric optimization into the undersamling process. It can be easily applied to arbitrary classifier and evaluation metric, and makes the data sampler trainable. **2)** We approach the non-differentiable optimization problem for training the data sampler via reinforcement learning and propose a practical implementation of this approach. **3)** The proposed model consistently outperforms different heuristic-based data sampling methods including undersampling and oversampling, and achieve comparable results with the specifically designed class-imbalance learning methods, which usually achieve state-of-the-art performance.

Related Work

The extensive development of undersampling in recent dacades has resulted in various strategies. A representative is the random majority undersampling (RUS). In RUS, instances of majority classes are randomly discarded from the dataset. Some other strategies have attempted to improve upon RUS by utilizing the distribution of data (Wilson 1972). For example, Near Miss (Mani and Zhang 2003) selected the examples that were the nearest to minority instances, and Cluster Centroid (Lemaître, Nogueira, and Aridas 2017) undersampled the majority class by replacing a cluster of majority samples with the cluster centroid of the KMeans algorithm. However, these strategies are all heuristic-based and commonly suffer from the problem that discarding potentially useful data.

Some undersampling methods have used the ensemble technique to overcome this problem (Błaszczyński and Stefanowski 2015; Kang and Cho 2006; Liu, Wu, and Zhou 2009). Two representatives of these methods are *EasyEnsemble* and *BalanceCascade* (Liu, Wu, and Zhou 2009). In short, *EasyEnsemble* independently samples with replacement several subsets from majority instances and builds a classifier for each subset. All the generated classifiers form a single ensemble for the final decision. *BalanceCascade* is similar to *EasyEnsemble* in structure. The main difference is that *BalanceCascade* iteratively removes the majority examples that were wrongly classified by the classifiers.

Evaluation metric optimization has been gaining in popularity in recent years (Parambath, Usunier, and Grandvalet 2014; Eban et al. 2017; Norouzi et al. 2016), but few researcher have tackled the imbalanced data classification problem. A popular solution is to approximate the discrete evaluation metric with continuous loss (Eban et al. 2017; Herschtal and Raskutti 2004), on which gradient-based updating methods can be used. The problem is that it is usually hard for many evaluation metrics to find appropriate approximations. In addition, this solution is not applicable to non-parametric classifiers such as the decision tree (DT), k-nearest neighbor (KNN), and other rule-based models. Another popular solution borrows ideas from the reinforcement learning literature. It samples from the model during training and directly optimizes the reward over the model parameters with policy gradient ascent methods (Norouzi et al. 2016; Ranzato et al. 2015). In theory, this class of methods can be applied to any evaluation metric. However, it also suffers from the problem of not being applicable to non-parametric models. The last but not the least solution, Evolutionary Undersampling (EUS) (García and Herrera 2009), applies the evolutionary algorithm to achieve this purpose. In EUS, each chromosome is a binary vector representing the presence or absence of instances in the data-set. Its time complexity is $O(TNC)$, where T is the iterated generations, N is the population size, and C is the complexity for evaluating a sample (including training and testing a classifier). The drawback of this algorithm is that, it can only incorporate with quite simple classifiers (such as 1NN), otherwise its time-complexity will be quite high.

Method

The proposed method is to train the data sampler to sample a subset of the training dataset and the goal in data selection is to make the classifier achieve the optimum performance over the evaluation metric. It is NP-hard to compute the optimum solution, thus we must resort to an approximation. In the following, we first precisely formulate this problem, and then show how to approximate it via reinforcement learning.

Formulation

Let $\mathfrak{S}(\mathrm{A})$ denote the subset of A. Then, our approach contains a training dataset $\{\mathbf{X}, \mathbf{Y}\}$, a data sampler $\boldsymbol{w}$: $\{\mathbf{X}, \mathbf{Y}\} \rightarrow \mathfrak{S}(\{\mathbf{X}, \mathbf{Y}\})$, a supervised classifier $\boldsymbol{f}$: $\boldsymbol{x} \rightarrow \hat{\boldsymbol{y}}$ that is to train on $\mathfrak{S}(\{\mathbf{X}, \mathbf{Y}\})$, and a specially defined evaluation metric $\mathcal{G} : \{\mathbf{Y}, \hat{\mathbf{Y}}\} \rightarrow \mathbb{R}$.

The problem can then be specified as finding the best possible $\boldsymbol{w}$ that we are able. Ideally, we would take $\boldsymbol{w}^*$ defined by

$$\boldsymbol{w}^*(\{\mathbf{X}, \mathbf{Y}\}) := \underset{\mathfrak{S}(\{\mathbf{X},\mathbf{Y}\})}{\arg\max}\, \mathcal{G}\left(\{\mathbf{Y}, \boldsymbol{f}(\mathbf{X}; \mathfrak{S}(\{\mathbf{X}, \mathbf{Y}\}))\}\right), \tag{1}$$

where $f(\mathbf{X}; \mathfrak{S}(\{\mathbf{X}, \mathbf{Y}\}))$ denotes the predicted labels $\hat{\mathbf{Y}}$ of $\mathbf{X}$ by the classifier $\boldsymbol{f}$ trained on $\mathfrak{S}(\{\mathbf{X}, \mathbf{Y}\})$. But in general we do not expect this to be achievable. Instead, we aim for a good approximation of $\boldsymbol{w}^*$ with

$$\boldsymbol{w}(\{\mathbf{X}, \mathbf{Y}\}) \approx \underset{\mathfrak{S}(\{\mathbf{X},\mathbf{Y}\})}{\arg\max}\, \mathcal{G}\left(\{\mathbf{Y}, f(\mathbf{X}; \mathfrak{S}(\boldsymbol{\{\mathbf{X}, \mathbf{Y}\}}))\}\right), \tag{2}$$

rather than the best one.

Characterize as a Markov Decision Process

We approach the task of approximating (1) via reinforcement learning. Specifically, we characterise the problem as a Markov decision process (MDP), as defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \mathcal{I})$, where

- $\mathcal{S}$ is the state space.
- $\mathcal{A}$ maps a state $s \in \mathcal{S}$ to a set of possible actions $\mathcal{A}(s)$ when in s.
- $\mathcal{R}$ maps a state $s \in \mathcal{S}$ and an action $a \in \mathcal{A}$ to the reward $\mathcal{R}(s, a) \in \mathbb{R}$.

- $\mathcal{T}$ characterises the transitions made by MDP $\mathcal{T}$: $\mathcal{S}\times\mathcal{A}\rightarrow\mathcal{S}$.
- $\mathcal{I}$ is the distribution of the initial state $s_0\in\mathcal{S}$.

A policy $\pi(a|\boldsymbol{s};\boldsymbol{\theta})=p(a|\boldsymbol{s};\boldsymbol{\theta})$ defines the probability of performing action a given that we are in state $\boldsymbol{s}$. Here we write $\boldsymbol{\theta}$ inside the probability to denote that the probability is determined by the parameter $\boldsymbol{\theta}$. Given a policy π, the MDP starts from sampling an initial state $\boldsymbol{s}_0$ according to $\mathcal{I}$, and then evolves according to:

$$\boldsymbol{s}_{t+1}:=\mathcal{T}(\boldsymbol{s}_t,a_t\sim\pi(a|\boldsymbol{s}_t;\boldsymbol{\theta})),$$

at each step $t\geq 1$. For reasons that will be made clear below, we only consider deterministic $\mathcal{T}$ and impose a finite horizon of T steps on our MDP, so that we do not consider states beyond T. We are now looking to find a good set of parameters $\boldsymbol{\theta}$ such that if we follow the policy $\pi(a|\boldsymbol{s};\boldsymbol{\theta})$ we will obtain a high expected reward $E_\tau[R_\tau|\pi;\boldsymbol{\theta}]$:

$$\boldsymbol{\theta}^*=\arg\max_{\boldsymbol{\theta}}E_\tau[R_\tau|\pi;\boldsymbol{\theta}]. \tag{3}$$

Here τ denotes a trajectory of the MDP. That is a sequence $\boldsymbol{s}_0,a_0,r_1,\boldsymbol{s}_1,a_1,r_2,\cdots,\boldsymbol{s}_{T-1},a_{T-1},r_T$, where r_t is the reward for having been in state $\boldsymbol{s}_t$ and taken action a_t. And $R_\tau=\sum_{t=1}^T r_t$.

In this work, we use the policy gradient method to solve the optimization problem. Weights are updated by stochastic gradient ascent in the direction that maximizes the expected reward:

$$\Delta\boldsymbol{\theta}\propto\frac{\partial\log\pi(\tau;\boldsymbol{\theta})}{\partial\boldsymbol{\theta}}R_\tau, \tag{4}$$

where $\pi(\tau;\boldsymbol{\theta})=\pi(a_0|s_0;\boldsymbol{\theta})\cdots\pi(a_{T-1}|s_{T-1};\boldsymbol{\theta})$ denotes the trajectory probability of τ.

We now show how our problem stated in the "Formulation" section can be formulated within this framework. We assume that the labeled dataset contains T examples with order fixed and $(\mathbf{x}_t,\mathbf{y}_t)$ denotes the t^{th} example. And we will refer $\mathbf{X}_{<t}$ to $\{\boldsymbol{x}_1,\cdots,\boldsymbol{x}_{t-1}\}$ for $t>1$ and $\mathbf{X}_{<t}\equiv\emptyset$ for $t\leq 1$. Then the MDP is evolved sample by sample in the index order and the state space is defined by:

$$\mathcal{S}:=\{V|V\subseteq\mathfrak{S}(\{\mathbf{X},\mathbf{Y}\})\}. \tag{5}$$

In particular, at step $t\geq 1$, the state space is defined by:

$$\mathcal{S}(t):=\{(V,(\boldsymbol{x}_t,\boldsymbol{y}_t))|V\in\mathfrak{S}(\{\mathbf{X}_{<t},\mathbf{Y}_{<t}\})\},$$

and $s_0=\mathcal{S}(0)\equiv\emptyset$. Intuitively, V gives the current subset that we have chosen as a candidate for our maximizer so far. For notational convenience, we denote the chosen set of a given state $\boldsymbol{s}$ by $V(\boldsymbol{s})$ with $V(\boldsymbol{s}_0)\equiv\emptyset$.

The action a_t at step t is to decide whether adding or not $(\mathbf{x}_t,\mathbf{y}_t)$ into $V(\boldsymbol{s}_t)$, as defined by:

$$\mathcal{A}(\boldsymbol{s}_t):=\{(\boldsymbol{x}_t,\boldsymbol{y}_t),\emptyset\}, \tag{6}$$

and the transition function is defined by:

$$\mathcal{T}(\boldsymbol{s}_t,a_t):=V(\boldsymbol{s}_t)\cup\{a_t\}. \tag{7}$$

Once the transition terminated at step T, we train the supervised classifier $\boldsymbol{f}$ on the chosen subset $V(\boldsymbol{s}_T)\cup\{a_T\}$ and treat the model performance over the given evaluation metric as the reward r_T, i.e.,

$$r_T=\mathcal{G}(\{\mathbf{Y},\boldsymbol{f}(\mathbf{X};V(\boldsymbol{s}_T)\cup\{a_t\})\}).$$

And we set the reward $r_t=0$ for all non-terminal time steps $t<T$. Thus, the episodic reward $R_\tau=\sum_{t=1}^T r_t=r_T$ is exactly the model performance when trained on the chosen subset over the given evaluation metric, thus the optimization direction is defined by:

$$\Delta\boldsymbol{\theta}\propto\frac{\partial\log\pi(\tau;\boldsymbol{\theta})}{\partial\boldsymbol{\theta}}r_T. \tag{8}$$

Once the policy has converged, we estimate $\boldsymbol{w}$ with the deterministic policy:

$$\boldsymbol{w}(\{\mathbf{X},\mathbf{Y}\})=\{a_0^*,\cdots,a_{T-1}^*\}, \tag{9}$$

where $a_t^*=\arg\max_{a\in\mathcal{A}(\boldsymbol{s}_t)}\pi(a|\boldsymbol{s}_t)$.

The general process of the training is as follows: we start sampling a subset $V=V(\mathbf{s}_T)\cup\{a_T\}$ of the training dataset following the policy π. Then we train the classifier $\boldsymbol{f}$ on V, resulting in a function $\boldsymbol{f}(;V)$ used to predict class labels for all examples and obtain the task reward $R_\tau=r_T$. After that we update the policy π and start a new episodic until π converges. The above steps are summarized in Alg. 1.

Algorithm 1 Trainable Undersampling

Input: training dataset $\{\mathbf{X},\mathbf{Y}\}$, classification procedure $\boldsymbol{f}$, initial policy $\pi(\boldsymbol{\theta}_0)$, maximum number of iteration N
Initialize: $\pi(\boldsymbol{\theta})\leftarrow\pi(\boldsymbol{\theta}_0)$; $T\leftarrow$ dataset size $|\{\mathbf{X},\mathbf{Y}\}|$
repeat
  $V(\boldsymbol{s})\leftarrow\emptyset$
  for $t=1$ to T **do**
    $\boldsymbol{s}_t\leftarrow V(\boldsymbol{s})\cup\{(\mathbf{x},\mathbf{y})\}$
    choose action $a_t\in\{(\mathbf{x}_t,\mathbf{y}_t),\emptyset\}$ in probability $a_t\sim\pi(a|\boldsymbol{s}_t;\boldsymbol{\theta})$.
    $V(\boldsymbol{s})\leftarrow V(\boldsymbol{s})\cup\{a_t\}$
  train the classifier $f(\cdot|V(\boldsymbol{s}))$ on $V(\boldsymbol{s})$
  obtain the reward $R_\tau\leftarrow\mathcal{G}(\{\mathbf{Y},\boldsymbol{f}(\mathbf{X};V(\boldsymbol{s}))\})$
  update $\boldsymbol{\theta}$ in the direction that maximizes the reward $\Delta\boldsymbol{\theta}\propto\sum_{t=1}^T\frac{\partial\log\pi(a_t|\boldsymbol{s}_t;\boldsymbol{\theta})}{\partial\boldsymbol{\theta}}R_\tau$.
until $\pi(\boldsymbol{\theta})$ converges or maximum number of iterations N exceeds.
generate $\boldsymbol{w}(\{\mathbf{X},\mathbf{Y}\})$ according to Eq. 9.
train $\boldsymbol{f}$ on $\boldsymbol{w}(\{\mathbf{X},\mathbf{Y}\})$
Return: $\boldsymbol{f}$

Convergence and Complexity

The convergence of this algorithm is inherited from the convergence of the policy search algorithm (Williams 1992). This is because we fix the training procedure of the supervised classifier, including its architecture, parameter (if it has) initialization, and hyper-parameters. Thus, for each sampled dataset, the classification performance, i.e., the reward, is fixed. The computational complexity of this algorithm is $\mathcal{O}(N(TD+C))$, where N is the episodic

number for policy updating, T is the dataset size, D is the computational cost for one-step state updating ($\boldsymbol{s}_t \rightarrow \boldsymbol{s}_{t+1}$), and C is the cost for classifier training.

Practical Implementation

We argue that the decision on whether to select an example is based on both the example itself and the distribution of the already chosen subset. To this end, we fix the order of the training dataset, forming a data sequence:

$$\{\mathbf{X}, \mathbf{Y}\} = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), \cdots, (\boldsymbol{x}_T, \boldsymbol{y}_T)\}$$

Then the state at the t step $\boldsymbol{s}_t$ is represented as a concatenation of the sequence of chosen examples before t step and the t^{th} example. We apply the gated recurrent unit (GRU) (Bahdanau, Cho, and Bengio 2014) to encode this data sequence, generating a dense vector representation $\mathbf{h}_t$ of $\boldsymbol{s}_t$. Note that the gated network takes both $\mathbf{x}$ and $\mathbf{y}$ as inputs. In addition, if the t^{th} example $(\mathbf{x}_t, \mathbf{y}_t)$ is not selected, the state presentation at the $t+1$ step transits from $\mathbf{h}_{t-1}$, namely, $\mathbf{h}_{t+1} = \text{GRU}(\mathbf{h}_{t-1}, \mathbf{x}_{t+1} \oplus \mathbf{y}_{t+1})$, otherwise it transits from $\mathbf{h}_t$ with $\mathbf{h}_{t+1} = \text{GRU}(\mathbf{h}_t, \mathbf{x}_{t+1} \oplus \mathbf{y}_{t+1})$, where $\oplus$ denotes the concatenation operation. To obtain the action probability at the t step, we feed the state representation $\mathbf{h}_t$ into a fully connected multiple-layer-perceptron (MLP):

$$p(c|\boldsymbol{s}_t) = \text{MLP}(\mathbf{h}_t) \tag{10}$$

which performs a binary classification with class 1 indicating to choose the t^{th} example, otherwise not. Examples are then sampled in the probability of $p(c = 1|\boldsymbol{s}_t)$.

To reduce the size of the policy network and achieve faster convergence, we applied the following tricks for training the data sampler: **1)** We pre-trained the policy $\pi(\boldsymbol{\theta}_0)$ with false labels. The chosen probability of minority examples was initialized as 0.9, and that of majority examples was initialized as δ, with $\delta \times$ (number of majority examples) $=$ $0.9 \times$ (number of minority examples). In addition, for tasks with a large dataset, we first pre-trained the policy on a smaller training dataset, and then incrementally increased the dataset size. This is because the state space increases exponentially with the size of the training dataset, and the time complexity for training the classifier is often super-linear of the training dataset size. Pre-training the policy on a smaller dataset can quickly obtain a good initialization for the policy and consequently results in faster convergence, thus reducing the N value of the model complexity. **2)** For tasks with high-dimensional input, we first reduced the input dimension using Principle Component Analysis (PCA), and then fed it as an input into the policy network (the supervised classifier is still trained on the original representation). This is to reduce the D value of the model complexity. **3)** We initialize the classifier using the model trained on the non-sampled dataset. This is to reduce the C value of the model complexity.

Experiments

This section presents the results of our experimental study on two synthetic and five real-world class-imbalanced datasets. On the synthetic datasets, we tested the applicability of the proposed algorithm to incorporate both parametric and non-parametric classifiers. On the real-world datasets, we evaluated the effectiveness of our proposed algorithm compared to the prevailing heuristic-based data sampling methods and some state-of-the-art methods.

Because the experiments were designed to study the effectiveness of the data sampling strategies, we assumed that the training dataset could reveal the general data distribution, and that the chosen classifier was suitable for the tested class-imbalance tasks. Based on these assumptions, we first chose the supervised classifier and its corresponding hyper-parameters for each tested task with 5-fold cross-validation on the original training dataset. Every tested method shared the architecture of the obtained classifier. As for the hyper-parameters of the sampling strategies themselves, such as the sampling probability of each class, we chose the values that maximized the best performance over 20 random runs. The performance was reported by averaging the top 5 best results obtained with the chosen hyper-parameters.

Synthetic Data

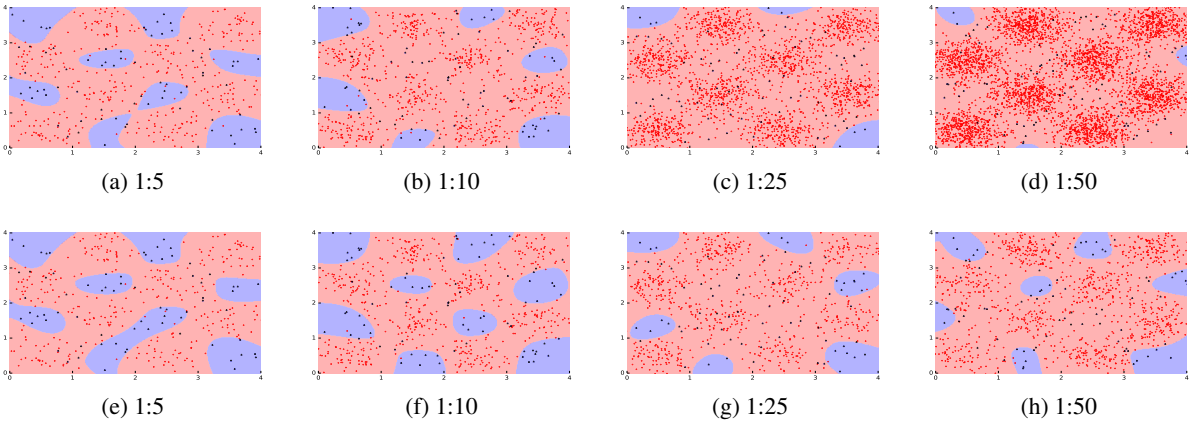
Two-Gaussian-Clouds: We created a dataset with 50,000 data points generated from a multivariate normal Gaussian distribution whose $\boldsymbol{u} = [0, 0], \Sigma = \boldsymbol{I} \in R^2$, and 1000 data points generated from a multivariate normal Gaussian distribution whose $\boldsymbol{u} = [2, 0], \Sigma = \boldsymbol{I} \in R^2$. Because this dataset was easy to obtain, we also generated a testing dataset to validate the generality. On this task, we tested the following parametric and non-parametric classifiers: *Logistic Regression (LR)*, *Support Vector Machine (SVM)*, *k-nearest neighbours (KNN)*, and *Decision Tree (DT)*. The performances were evaluated using the F1 of the minority class.

Checker Board: Five 4×4 checker board datasets with different imbalanced ratio were generated. We used the SVM with rbf kernel as the supervised classifier and evaluated the performance with the macro-F1.

Setup and Results. We implemented the GRU network with 25 hidden units and the MLP with one-layer-perceptron using Pytorch, and we used the RmsProp (Tieleman and Hinton 2012) step rule for parameter optimization with its initial learning rate set to 0.001. As for the implementation of the supervised classifiers, we used the sklearn package (Pedregosa et al. 2011).

Table 1: F1 of the minority class on the Two-Gaussian-Clouds task. ORG means training the classifier on the original dataset, and TU refers to the proposed data sampling method. *Inf* refers to the optimum value we can obtained, when the dataset size is approximately infinite.

Model	Train		Test		*Inf*	Hyper
	ORG	TU	ORG	TU		
LR	0.275	0.406	0.290	0.399	0.399	C = 10
SVM	0.106	0.409	0.084	0.396		C = 1000
KNN	0.356	0.404	0.268	0.370		neighbour = 7
DT	0.250	0.409	0.239	0.397		depth = 3



(a) 1:5 (b) 1:10 (c) 1:25 (d) 1:50

(e) 1:5 (f) 1:10 (g) 1:25 (h) 1:50

Figure 1: Class boundaries determined by SVMs (rbf kernel) on 4×4 checker board datasets. **Top:** Trained on the original dataset with different imbalance ratios. **Bottom:** Trained on the chosen subsets by our proposed data sampler. Best viewed in color. As the imbalance ratio increases, the classifier trained on the original dataset was overwhelmed by the majority class.

Table 1 lists the comparison results on the Two-Gaussian-Clouds dataset. We list the hyper-parameters used for each of the classifier, and those not explicitly mentioned apply the default setting of sklearn. In addition, we reported the best performance we can obtained in theory when the dataset size was approximately infinite, which is referred to Inf in the table. From the table, we can first observe that the tested classifiers perform poorly using the original training dataset over the F1 measurement due to the class-imbalance problem. Second, the proposed data sampling method can consistently improve the model performance for different classifiers. We argue that this is because the data sampler is optimized over the evaluation metric. For different classifiers, it can adjust its sampling strategy and accordingly the sampled dataset distribution to achieve similar and approximate optimal performances.

Figure 1 shows the class boundaries determined by the SVMs when they were trained on the original checker board dataset and on the corresponding chosen subset by our data sampler. The performance by macro-F1 trained on the original datasets are 0.831, 0.777, 0.622, and 0.564 corresponding to the imbalance ratios of 1:5, 1:10, 1:25, and 1:50, respectively. The corresponding performance trained on the chosen subsets are 0.869, 0.832, 0.706, and 0.708, respectively. In the figure, we can see that as the imbalance ratio increases, the classifier was overwhelmed by the majority class. In particular, when the ratio reaches 1:50, almost all of the examples are classified as the majority class. However, this problem is alleviated after applying our proposed undersampling strategy.

Real Data

We next assessed the effectiveness of the proposed algorithm on realistic tasks. Five real-world imbalanced datasets were selected from different domains, with various imbalanced ratios. Table 2 lists the detail of each dataset and the corresponding supervised classifier we used.

Vehicle is an imbalanced version of the Vehicle Silhouettes dataset, where the positive examples belong to class 1 (Saab) and the negative examples belong to the rest (Fernández et al. 2008). Following the work of (Kang and Cho 2006), we applied the Geometric Mean (GM) to evaluate the performance. **Page Blocks** is an imbalanced version of the Page Blocks dataset, where the negative examples belong to the page layout of class text and the positive examples belong to the rest (Fernández et al. 2008). For performance measurement, it recommends the Matthews correlation coefficient (MCC) (Matthews 1975) of the positive examples. **Credit Fraud** contains transactions made by credit cards in September 2013 by european cardholders (Dal Pozzolo et al. 2015). It is highly imbalanced, with only 492 frauds out of 284,807 transactions. For performance measurement, it recommends the AUCPRC of the Fraud class. **SMS Spam** is a set of SMS tagged messages that have been collected for SMS Spam research. It contains 5,574 SMS messages in English, tagged according being ham (legitimate) or spam. For performance measurement, it recommends the $\boldsymbol{F}_{0.5}$ of the spam class. **Diabetic Retinopathy (DR)** is an imbalanced version of the Diabetic Retinopathy Detection [1], where the negative examples belong to class 0 (No DR) and the positive examples belong to the rest. Following the work of (Leibig et al. 2017), we used the AUCROC to measure the performance.

Setup and Results. For the Credit Fraud task, we first trained the proposed data sampler on a smaller training dataset, containing all (denoted by n) of the positive examples and $10n$ negative examples. Then, for every 200 iterations, we added additional $10n$ more negative examples to the subset until all of the data were used. We implemented the GRU network with 50 hidden units for this task, while

[1] You can get more information about this dataset in the Kaggle Challenge. https://www.kaggle.com/c/diabetic-retinopathy-detection

Table 2: Description of the tested real-world datasets.

Dataset	#Attribute	#Example	Feature Format	Minority Ratio	Evaluation Metric	Used Classifier
Vehicle	18	846	Numeric	25.65%	GM	SVM (rbf)
Page blocks	10	5,472	Numeric	10.21%	MCC	MLP
Credit Fraud	28	284,807	Numeric	0.17%	AUCPRC	DT
SMS Spam	8,749	5,574	Text	13.41%	$\boldsymbol{F}_{0.5}$	LR
DR	262,144	17,563	Image	26.52%	AUCROC	CNN

Table 3: Performance of the proposed method TU and prevailing data sampling methods on the tested real-world datasets. Here, RUS refers to the random undersampling method. The second group of methods are all undersampling-based and the third group of methods are all oversampling-based.

Task	ORG	RUS	NearMiss	Cluster	TomekLink	ALLKNN	SMOTE	ADASYN	TU
Vehicle	0.935	0.949	0.877	0.937	0.938	0.858	0.935	**0.964**	**0.964**
Page-blocks	0.897	0.903	0.878	0.877	0.895	0.867	0.897	0.902	**0.915**
Credit Fraud	0.849	0.860	0.817	0.584	0.840	0.809	0.849	0.848	**0.880**
SMS Spam	0.936	0.938	0.931	0.932	0.935	0.933	0.936	0.936	**0.967**
DR	0.930	0.942	0.921	0.933	0.934	0.927	0.930	0.944	**0.958**

with 25 units for the other tasks. For the SMS Spam task, we reduced the input dimension to 100 with PCA for the policy network. For the DR task, we used the publicly available network architecture and weights provided by a participant who scored very well in the Kaggle DR competition, which we will call JFnet (Fauw 2015), as the classifier. And we retrained its last two fully connected layers on each sampled dataset.

We first compared the proposed method against prevailing data sampling methods. These methods include six undersampling methods, i.e., *Random Majority Undersampling* (RUS), *Near Miss* (NearMiss) (Mani and Zhang 2003), *Cluster Centroid* (Cluster) (Lemaître, Nogueira, and Aridas 2017), *TomekLink* (Tomek 1976b), and *ALLKNN* (Tomek 1976a), and two oversampling methods, i.e., *SMOTE* (Chawla et al. 2002) and *ADASYN* (He et al. 2008). All of them were implemented using imbalanced-learn (Lemaître, Nogueira, and Aridas 2017).

Table 3 lists the comparison results on the five real-world datasets. From the table, We can obtain the following observations. **1)** The performance of the prevailing heuristic-based data sampling methods varies considerably by dataset. None of them can consistently outperform other heuristic-based data sampling methods. This shows the drawbacks of these method that their applicabilities are limited to specific dataset. **2)** Our proposed data sampling method TU can consistently outperforms the heuristic-based data sampling methods, showing its robustness and effectiveness.

We further compared the proposed method with some state-of-the-art methods, though they may be inapplicable to some tasks. These methods include *EasyEnsemble* (Liu, Wu, and Zhou 2009), *BalanceCascade* (Liu, Wu, and Zhou 2009), *EUS* (García and Herrera 2009), and cost-sensitive learning (Parambath, Usunier, and Grandvalet 2014). Note that we replace the 1NN model of EUS with the corresponding used classifier for each dataset. In addition, we compare with the model-level evaluation metric optimization method using reinforcement learning (RL) (Wu et al. 2016), which we refer to *RL*.

Table 4 lists the results of these baselines compared to our proposed method. Results of the cost-sensitive method on the Vehicle and Credit Fraud tasks are missing because there are no published methods for the implementation on the used evaluation metrics. Result of the BalanceCascade method on the DR task is missing because the computation cost is too large, and results of the RL method on the Vehicle and Credit Fraud tasks are missing because it is not applicable to the used classifiers. From the table we can see that our proposed method can achieve comparable performance with state-of-the-art methods. Note that, though the cost-sensitive methods perform slightly better than our propose method on some tasks, they need specific designation for the given evaluation metric and cannot generally transform to other measurements. In the meanwhile, the RL method suffers the problem that cannot apply to non-parametric classifiers. In contrast, our proposed method can easily apply to arbitrary evaluation metric and classifier.

In addition, according to our aforementioned discussion in the Related Work section, the proposed method, as a meta-learning approach, can also collaborate with other data-level operations. Here, we study the applicability of the proposed to incorporate with the oversampling methods. This can also assess if the oversampling methods can create new informative instances instead of just changing the data distribution.

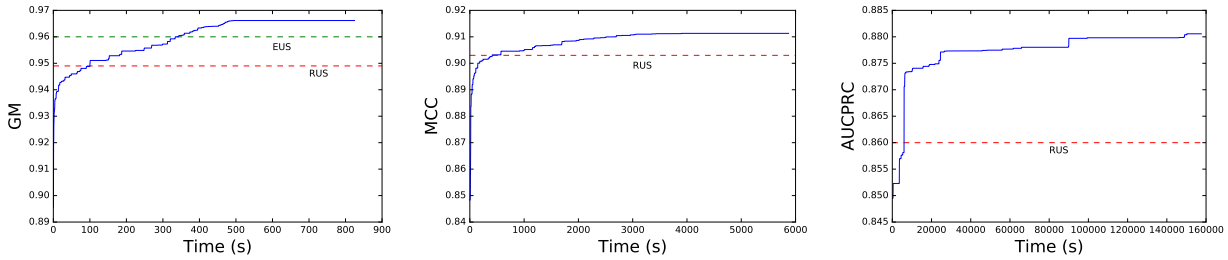
Table 5 shows the performance of our proposed method incorporating with SMOTE and ADASYN. From the table, we can observe that, though the two oversampling methods can improve the classification performance, they offer negligible improvement to our proposed method on all of the tested datasets. This means that they do not create many new informative instances but instead only change the data

Table 4: Comparison of our proposed method with state-of-the-art class-imbalance learning methods on the tested real-world datasets. Some results of these methods are missing because there are no published methods for the implementation or the computation cost is too large, or the method is not applicable to the used classifier. Cost-sensitive methods are implemented according to their referenced paper, respectively.

Task	CostSensitive	EasyEnsemble	BalanceCascade	EUS	RL	TU
Vehicle	– – –	0.953	0.955	0.960	– – –	**0.964**
Page-blocks	**0.916** (2007)	0.904	0.901	0.907	**0.916**	0.915
Credit Fraud	– – –	0.859	0.865	– – –	– – –	**0.880**
SMS Spam	0.964 (2014)	0.939	0.935	0.965	0.962	**0.967**
DR	0.950 (2007)	0.945	– – –	– – –	**0.958**	**0.958**

Table 5: Results of the proposed method incorporating with the oversampling technique. SMOTE+TU is to oversample the dataset using SMOTE and then apply the proposed method to the oversampled dataset.

Task	ORG	SMOTE	ADASYN	TU	SMOTE+TU	ADASYN+TU
Vehicle	0.935	0.964	0.964	0.964	0.964	**0.965**
Page-blocks	0.897	0.902	0.898	0.915	**0.917**	0.915
Credit Fraud	0.849	0.848	0.849	0.880	**0.881**	0.880
SMS Spam	0.936	0.936	0.936	**0.967**	0.965	**0.967**
DR	0.930	0.944	0.943	**0.958**	0.957	0.956



(a) Vehicle (b) Page blocks (c) Credit Fraud

Figure 2: Time complexity of the proposed method on three tested datasets. The red dot line denotes the performance of the random undersampling method.

distribution. Therefore, the optimum subset for the given classier chosen from the oversampled dataset is similar to that chosen from the original training dataset.

Finally, we empirically study the time complexity of the proposed method on the tested datasets. Figure 2 depicts the training process of the proposed method by time (second) on three tested datasets using a single GPU. From the figure, we can see that the proposed method can quickly outperform the random undersampling method and achieve further improvement.

Conclusion

In this work, we propose a trainable undersampling method. It incorporates the evaluation metric optimization into the data sampling procedure thus can learn which instances should be discarded and which instances should be preserved. Moreover, as a data level operation, it can easily apply to arbitrary evaluation metric and classifier, including the non-parametric ones. Empirical studies on several synthetic and realistic datasets show that this method can consistently outperform prevailing heuristic-based data sampling methods and achieve better results than the state-of-the-art methods in most of cases.

Acknowledgments

The authors wish to thank the anonymous reviewers for their helpful comments. This work was partially funded by China National Key R&D Program (No. 2017YFB1002104, 2018YFC0831105), National Natural Science Foundation of China (No. 61532011, 61751201, 61473092, and 61472088), and STCSM (No.16JC1420401,17JC1420200).

References

Bahdanau, D.; Cho, K.; and Bengio, Y. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Batista, G. E.; Prati, R. C.; and Monard, M. C. 2004. A study of the behavior of several methods for balancing machine learning

training data. *ACM SIGKDD explorations newsletter* 6(1):20–29.

Błaszczyński, J., and Stefanowski, J. 2015. Neighbourhood sampling in bagging for imbalanced data. *Neurocomputing* 150:529–542.

Calders, T., and Jaroszewicz, S. 2007. Efficient auc optimization for classification. In *European Conference on Principles of Data Mining and Knowledge Discovery*, 42–53. Springer.

Chawla, N. V.; Bowyer, K. W.; Hall, L. O.; and Kegelmeyer, W. P. 2002. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research* 16:321–357.

Cieslak, D. A., and Chawla, N. V. 2008. Start globally, optimize locally, predict globally: Improving performance on imbalanced data. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, 143–152. IEEE.

Dal Pozzolo, A.; Caelen, O.; Johnson, R. A.; and Bontempi, G. 2015. Calibrating probability with undersampling for unbalanced classification. In *Computational Intelligence, 2015 IEEE Symposium Series on*, 159–166. IEEE.

Eban, E.; Schain, M.; Mackey, A.; Gordon, A.; Rifkin, R.; and Elidan, G. 2017. Scalable learning of non-decomposable objectives. In *Artificial Intelligence and Statistics*, 832–840.

Fauw, D. 2015. J. 5th place solution of the kaggle diabetic retinopathy competition.

Fernández, A.; García, S.; del Jesus, M. J.; and Herrera, F. 2008. A study of the behaviour of linguistic fuzzy rule based classification systems in the framework of imbalanced data-sets. *Fuzzy Sets and Systems* 159(18):2378–2398.

Ganganwar, V. 2012. An overview of classification algorithms for imbalanced datasets. *International Journal of Emerging Technology and Advanced Engineering* 2(4):42–47.

García, S., and Herrera, F. 2009. Evolutionary undersampling for classification with imbalanced datasets: Proposals and taxonomy. *Evolutionary computation* 17(3):275–306.

Hanley, J. A., and McNeil, B. J. 1982. The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology* 143(1):29–36.

He, H., and Garcia, E. A. 2009. Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering* 21(9):1263–1284.

He, H.; Bai, Y.; Garcia, E. A.; and Li, S. 2008. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, 1322–1328. IEEE.

Herschtal, A., and Raskutti, B. 2004. Optimising area under the roc curve using gradient descent. In *Proceedings of the twenty-first international conference on Machine learning*, 49. ACM.

Kang, P., and Cho, S. 2006. Eus svms: Ensemble of under-sampled svms for data imbalance problems. In *International Conference on Neural Information Processing*, 837–846. Springer.

Leibig, C.; Allken, V.; Ayhan, M. S.; Berens, P.; and Wahl, S. 2017. Leveraging uncertainty information from deep neural networks for disease detection. *Scientific reports* 7(1):17816.

Lemaître, G.; Nogueira, F.; and Aridas, C. K. 2017. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research* 18(17):1–5.

Liu, X.-Y.; Wu, J.; and Zhou, Z.-H. 2009. Exploratory undersampling for class-imbalance learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 39(2):539–550.

Mani, I., and Zhang, I. 2003. knn approach to unbalanced data distributions: a case study involving information extraction. In *Proceedings of workshop on learning from imbalanced datasets*, volume 126.

Matthews, B. W. 1975. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure* 405(2):442–451.

Norouzi, M.; Bengio, S.; Jaitly, N.; Schuster, M.; Wu, Y.; Schuurmans, D.; et al. 2016. Reward augmented maximum likelihood for neural structured prediction. In *Advances In Neural Information Processing Systems*, 1723–1731.

Parambath, S. P.; Usunier, N.; and Grandvalet, Y. 2014. Optimizing f-measures by cost-sensitive classification. In *Advances in Neural Information Processing Systems*, 2123–2131.

Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; and Duchesnay, E. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12:2825–2830.

Ranzato, M.; Chopra, S.; Auli, M.; and Zaremba, W. 2015. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*.

Tieleman, T., and Hinton, G. 2012. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning* 4(2):26–31.

Tomek, I. 1976a. An experiment with the edited nearest-neighbor rule. *IEEE Transactions on systems, Man, and Cybernetics* (6):448–452.

Tomek, I. 1976b. Two modifications of cnn. *IEEE Trans. Systems, Man and Cybernetics* 6:769–772.

Van Hulse, J.; Khoshgoftaar, T. M.; and Napolitano, A. 2007. Experimental perspectives on learning from imbalanced data. In *Proceedings of the 24th international conference on Machine learning*, 935–942. ACM.

Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Reinforcement Learning*. Springer. 5–32.

Wilson, D. L. 1972. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man, and Cybernetics* (3):408–421.

Wu, Y.; Schuster, M.; Chen, Z.; Le, Q. V.; Norouzi, M.; Macherey, W.; Krikun, M.; Cao, Y.; Gao, Q.; Macherey, K.; et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.