# Online Rare Category Detection for Edge Computing

Yufei Cui[§], Qiao Li[§], Sarana Nutanong[‡], Chun Jason Xue[§]
[§]Department of Computer Science, City University of Hong Kong, Hong Kong, China
[‡]School of Information Science and Technology, VISTEC, Thailand

***Abstract*—Identifying rare categories is an important data management problem in many application fields including video surveillance, ecological environment monitoring and precision medicine. Previous solutions in literature require all data instances to be first delivered to the server. Then, the rare categories identification algorithms are executed on the pool of data to find informative instances for human annotators to label. This incurs large bandwidth consumption and high latency. To deal with the problems, we propose a light-weight rare categories identification framework. At the sensor side, the designed online algorithm filters less informative data instances from the data stream and only sends the informative ones to human annotators. After labeling, the server only sends labels of the corresponding data instances in response. The sensor-side algorithm is extended to enable cooperation between embedded devices for the cases that data is collected in a distributed manner. Experiments are conducted to show our framework dramatically outperforms the baseline. The network traffic is reduced by 75% on average.**

***Index Terms*—rare category detection, machine learning, real-time system, edge computing**

## I. Introduction

Rare category identification (RCI) problem has been studied over the last decade [4] [6]. Algorithms that are able to characterize and identify rare classes from unlabeled data instances are important in many application domains including automated video surveillance, ecological environmental monitoring and precision medicine. In these areas, automatically finding data instances tagged with rare labels is beneficial in detecting abnormal human activities and suspicious events [9] [10] such that: 1) the oracle (often human experts) can immediately judge and decide whether to inform service providers to take actions 2) the identified rare data instances can be further analyzed and stored [6]. For example, suppose there are video cameras and body sensors monitoring the daily life of a patient, if the patient falls down by accident, the RCI algorithm will detect this category through the abnormal data pattern. Consequently, the doctor will be notified to judge if the patient needs help from care workers and if medical treatment is required. Then the data pattern will be stored in the database for further analysis and updating the RCI algorithm.

The previous solutions of RCI problem require all the data being delivered to a centralized server before the RCI algorithm can be executed. [3] [2] [5] This causes two major drawbacks: high network bandwidth consumption and high latency. In particular, delivering a large amount of raw data instances (most of which are useless) through the public network to server causes unnecessarily high network bandwidth consumption. In addition, the delivery process brings high transmission delay. Waiting for the completion of gathering a compact dataset brings high latency.

Motivated by the above two aspects, we propose *Near-sensor Online Rare Category identification* (NORC) framework. The framework extends the traditional RCI framework to a client-server model. The key idea is to run a light-weight RCI algorithm on the near-sensor embedded devices to filter the less valuable data instances, which take a majority part. The embedded device makes decisions on data instances coming in a streaming way rather than on a pool of data. Only a small proportion of raw data instances will be delivered through the public network to the server side. The server only acts as a proxy to re-direct data instances to the oracle for labeling, afterwards re-directs labels back to the near-sensor side. In this way, both network bandwidth consumption and latency can be reduced.

To accomplish this framework design, we overcome these challenges:

**Preprocessing.** Efficient preprocessing of raw data is required on the embedded devices. An incremental principle component analysis (PCA) based scheme is designed to efficiently process the raw data.

**Distributed data.** For data collected in a distributed manner on several embedded devices, the framework should enable collaboration between devices such that the performance of RCI is guaranteed. The proposed NORC framework is extended to deal with data instances distributed on multiple devices, dubbed as *DNORC*.

Experiments are conducted on two Raspberry Pi's to show the performance of NORC framework. The experiment shows that, without sacrificing the effectiveness of finding rare category, NORC saves more than 75% of network traffic compared with the baseline method. *DNORC* improves the precision of RCI by around 10% over NORC.

The remainder of this paper is as follows. Section II presents the background and motivation. The proposed solution is presented in Section III. Section IV shows the experiments and result analysis. Finally, Section V concludes this work.

## II. Background and Motivation

Rare category identification (RCI) [4] [6] adopts the idea of Active Learning (AL) [8] [1] in the sense that each selected instance is labeled by human annotators and is added back to the labeled dataset. The overall process of AL and RCI is shown in Figure 1. In the pool-based RCI scenarios, there is a small set of labeled training data $\mathcal{L} = \{\mathbf{x}_n, y_n\}_{n=1}^{N}$ and a large

Corresponding author: Qiao Li, qiaoli045@gmail.com

978-3-9819263-2-3/DATE19/©2019 EDAA

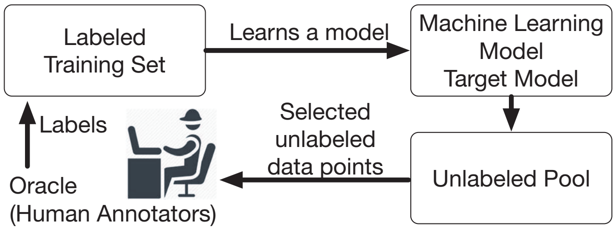

Labeled Training Set
Learns a model
Machine Learning Model Target Model
Labels
Selected unlabeled data points
Unlabeled Pool
Oracle (Human Annotators)

Fig. 1. Overview of active learning and rare category identification.

pool of unlabeled data $\mathcal{U} = \{\mathbf{x}_m, y_m\}_{m=1}^{M}$. The pool-based RCI proceeds by iteratively: 1) training a classifier model $f$ on $\mathcal{L}$, and 2) using a query function $\mathcal{Q}(f, \mathcal{L}, \mathcal{U})$ to select an unlabeled instance $i^*$ to be labeled, removing the $i^*$-th instance from $\mathcal{U}$ and adding the $(\mathbf{x}^*, y^*)$ pair to $\mathcal{L}$. The goal is to discover find all possible labels in the dataset using smallest number of queries.

Previous AL-based solutions require all the data instances to first be delivered from the sensors and aggregated at the server. Then, the server runs a pool-based RCI algorithm in the coordination with human annotators. This incurs problems in the following two aspects: high bandwidth consumption and high latency.

The raw data should be delivered through the network to the server, which causes dramatic network congestion when the data volume is large (e.g., video stream). However, with running the RCI algorithm, only a small proportion of data will be selected out of the whole dataset. For example, for finding all the rare categories, the state-of-the-art algorithm LOFRCD [6] only queries few data points ranging from 1.23% to 2.08% out of the four test datasets. The detailed statistics are shown in Table I.

TABLE I
TOTAL NUMBER, NUMBER OF SELECTED INSTANCES AND SELECTED PROPORTION IN LOFRCD

| Dataset | Synthetic | Page block | Shuttle | KDD cup |
|---|---|---|---|---|
| Total amount | 1203 | 5473 | 14500 | 26289 |
| Selected | 25 | 99 | 179 | 360 |
| Percentage | 2.08% | 1.81% | 1.23% | 1.37% |

On the other hand, waiting for the completion of gathering a compact dataset brings high latency. A solution should be proposed to filter less useful data instances before data being delivered to the public network.

## III. Proposed Solution

### A. NORC Framework

NORC framework extends the RCI framework to a client-server model. The overall process is illustrated in Figure 2. We use *NORC processor* to denote the software module running the above algorithms.

The execution of RCI algorithms is moved from the server to the embedded devices. Specifically, a sensor collects raw data and delivers each data instance to the NORC processor in a streaming way. NORC processor then decides which instance is likely to belong to a rare category. The selected unlabeled data points are delivered to oracle through public network. The oracle then tags each data instance with a label and updates the server side labeled set with new $(\mathbf{x}, y)$ pairs. Labels are delivered through the public network to near-sensor side labeled set on embedded devices, constituting new $(\mathbf{x}, y)$ pairs. The near-sensor side labeled set is then used to update the learning model in NORC processor.

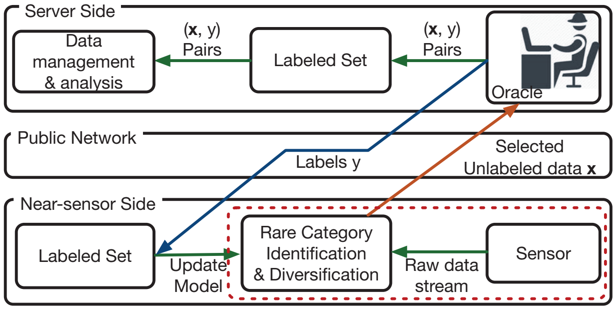


Fig. 2. Overview of near-sensor online rare category identification framework.

### B. NORC Algorithm

In NORC processor, a likelihood-based algorithm is for RCI. The idea of this method is to build a statistic model for all existing data instances with known labels. Then the algorithm decides if a new data instance is worth labeling according to the probability of the new instance under the current statistic model, i.e., likelihood. For efficiently building the statistic model, we use Kernel Density Estimation (KDE) with Gaussian kernel. $\hat{f}(\mathbf{x}) = \frac{1}{|\mathcal{L}|}\sum_{\mathbf{x}_j \in \mathcal{L}} \prod_i \frac{1}{h_i} K(\frac{\mathbf{x}_i - \mathbf{x}_j}{h_i})$, where $K(\cdot)$ is a Gaussian kernel and $h_i$ is the bandwidth of each data point $\mathbf{x}_i$. KDE is used because it is a non-parametric model, which is suitable for computation-limited and storage-limited embedded devices.

The basic NORC algorithm is illustrated as follows. Initially, there is a small set $\mathcal{L}$ with all data divided into $L$ subsets $D_l$, i.e., $\mathcal{L} = \{D_l = \{\mathbf{x}_n, y_n\}_{y_n = l}\}_{l=1}^{L}$, where $L$ is the number of found labels. The set of few unlabeled instances, $\mathcal{U} = \{\mathbf{x}_m\}_{m=1}^{M}$, is used to cache at most $M$ unlabeled data instances. For each known label $l$, the algorithm estimates the density of $\mathbf{x}$ who has label $l$ and gets a model $f_l(\cdot)$. For a new data instance $\mathbf{x}^*$ in the data stream, the algorithm estimates maximum probability of the $\mathbf{x}^*$ over all the models as the score $s^*$, i.e., $s^* = \max_l(f_l(\mathbf{x}^*))$. The algorithm only selects instances whose scores are less than a certain threshold $\mathcal{T}$ to transfer to the server. The value of threshold $\mathcal{T}$ is determined by the average likelihood estimated over $\mathcal{U}$. When the NORC processor receives a label $y^*$ for the queried $\mathbf{x}^*$, it builds a new KDE model on $(\mathbf{x}^*, y^*)$ if $y^*$ is new to $\mathcal{L}$. Otherwise, the NORC processor adds $\mathbf{x}^*$ to the existing subset $D_{y^*}$ of $\mathcal{L}$ and updates the model $f_{y^*}$. The algorithm keeps running until all the possible labels are found.

For efficiently preprocessing training data, incremental principle component analysis (PCA) [7] is adopted, which trains PCA by mini-batches of data with batch-size $b$. The idea is to cache the unlabeled data points for incrementally training PCA rather than drop them. Initially, the incremental PCA algorithm is trained on the union set of $\mathcal{L}$ and $\mathcal{U}$, after which $\mathcal{U}$ is cleared. The algorithm caches a point in $\mathcal{U}$ who has a score $s^*$ larger

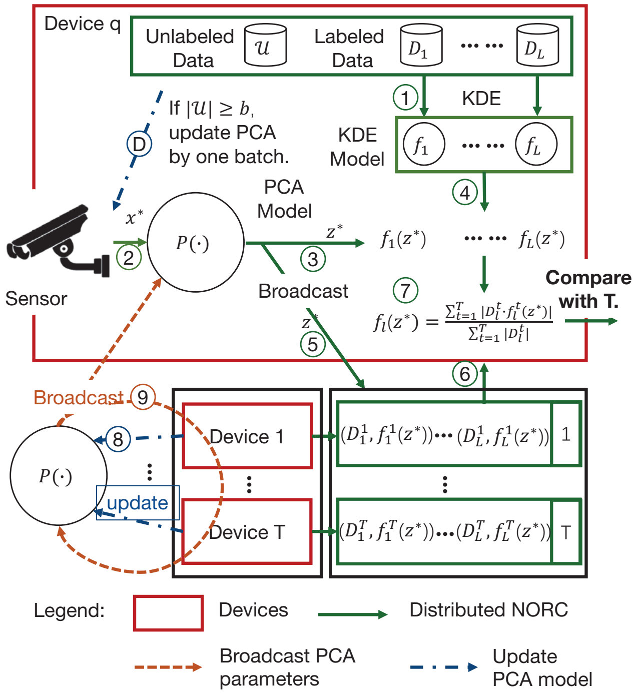


Fig. 3. Overview of NORC on distributed data

than the threshold $\mathcal{T}$ rather than drop it. When the size of $\mathcal{U}$ reaches batch-size $b$, the algorithm updates incremental PCA model by one batch and clears the cache.

### C. NORC with Distributed Data (DNORC)

In a more realistic case, sensors that are close to each other might collect similar objects belonging to one category. If the close embedded devices run NORC algorithm independently, they are likely to send duplicated objects back to the server. This causes unnecessary bandwidth and labeling cost.

On the other hand, the close sensors might collect objects from the same category but dissimilar in the input space. In this case, two models $f_l^1(\cdot)$ and $f_l^2(\cdot)$ for this category are trained by partial set of data instances, which causes inaccurate probability density estimation. A decent model should take advantage of all the nearby data instances from one category.

The overview of DNORC is shown in Figure 3. Compared to the centralized scheme, the proposed distributed method has a lower latency and does not require a local centralized embedded device. Each step is marked with $(1, \cdots, 9)$ for an easier illustration. DNORC is elaborated as follows:

- Each embedded device $t$ trains the set of model $F^t = \{f_l^t(\cdot)\}_{l=1}^L$ with its own set of labeled data instances $\mathcal{L}^t = \{D_l^t = \{\mathbf{x}_n, y_n\}_{y_n=l}^t\}_{l=1}^L$, as shown in step 1.
- Take device $q$ as an example. When a new $\mathbf{x}^*$ is collected by a device $q$ (step 2), $q$ applies PCA to $\mathbf{x}^*$ (step 3) and broadcasts $\mathbf{z}^* = P(\mathbf{x}^*)$ to all its neighboring embedded devices (step 4). $q$ then calculates the probability $\{P_l^q(\mathbf{z}^*)\}_{l=1}^L$ under all models $\{f_l^q(\cdot)\}_{l=1}^L$ (step 5).
- All neighboring devices then proceed this step. Take device $e$ as an example. It calculates the set of probabilities $\{f_l^e(\mathbf{z}^*)\}_{l=1}^L$ with the local models. Then, $e$ transmits the number of labeled examples in each category and the probabilities, i.e., $\{(|D_l|, f_l^e(\mathbf{z}^*))\}_{l=1}^L$, back to device $q$ (step 6).
- Device $q$ receives the (number, probability) pairs from all the neighboring devices and calculates the global probability by $f_l(\mathbf{z}^*) = \frac{\sum_{t=1}^T |D_l^t| \cdot f_l^t(\mathbf{z}^*)}{\sum_{t=1}^T |D_l^t|}$, where $t$ denotes the index of devices including $q$ itself. $T$ is the total number of devices (step 7). The global probability is used to compare with threshold $\mathcal{T}$ for making a decision.

The subsequent procedures follow NORC framework.

For globally consistent preprocessing of raw data, all the embedded devices share one group of PCA parameters. When number of unlabeled instances in $\mathcal{U}$ on a device reaches limit, the device updates PCA parameters by one batch and clears the buffer $\mathcal{U}$ (step 8). The device then broadcasts new parameters to all embedded devices for an update (step 9).

## IV. Experiments

### A. Experiment Setup

Experiments are conducted on two Raspberry Pi's (Raspberry Pi 3 model B+) running the Raspbian system. Each device has an ARM Cortex-A53 1.4GHz processor and 1GB SRAM. The Raspberry Pi's are connected through Ethernet port which has 300 Mbps transmission rate. The NORC framework is implemented based on Python 3.7 and Scipy library. We use Wireshark 2.6.3 to capture the network traffic.

Four methods are evaluated: 1) *NORC*. The basic NORC algorithm runs on two devices independently. 2) *DNORC*. The two devices run the DNORC algorithm collaboratively. 3) *Full Pass*. All data instances are directly delivered to the server without selection. 4) *Random*. Data instances are randomly selected on each device with the frequency of quires calculated according to the result of NORC algorithm. In particular, the frequency is calculated by the fraction of number of data instances selected by NORC and number of observed data instances in data streams.

We test NORC framework on three real-world datasets: MNIST, Page blocks and Shuttle. MNIST is a handwritten digits dataset with 70000 784-dimensional images and 10 categories and the patterns are nearly uniformly distributed across all categories. For the RCI case, 5 categories are randomly selected to be the majorities (5000 data instances each) and the rests are rare categories (100 data instances each). Page blocks (5,473 instances) and Shuttle (14,500 instances) are highly skewed dataset which means the distributions of instances over categories are highly unbalanced. The majority classes of Page blocks contains approximately 90% of data while that of Shuttle contains 80% of data.

Each dataset is partitioned into two subsets, and each subset forms a data stream which will be collected by one embedded device. Two settings are used for the partition ratio: (90%, 10%) and (98%, 2%). The second setting is only applied to MNIST because the other two datasets have limited data instances. In the partition, the data instances in each category will be divided into two subsets based on the partition ratios.

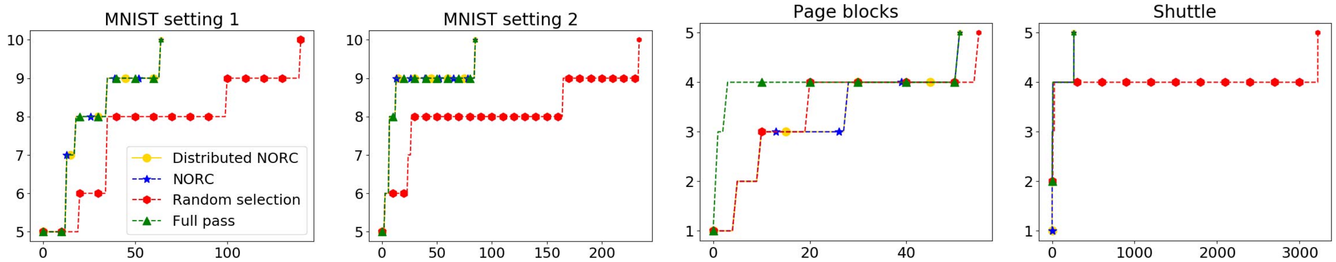
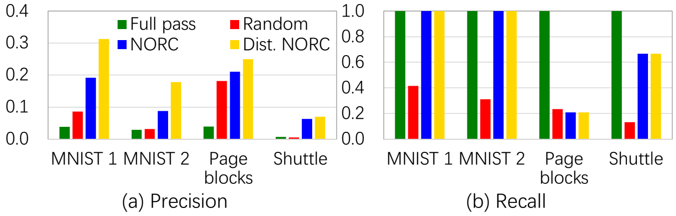


Fig. 4. RC identification curves on real-world datasets. X axis: Number of streaming data instances. Y axis: Number of discovered labels.



(a) Precision

(b) Recall

Fig. 5. Precision and recall

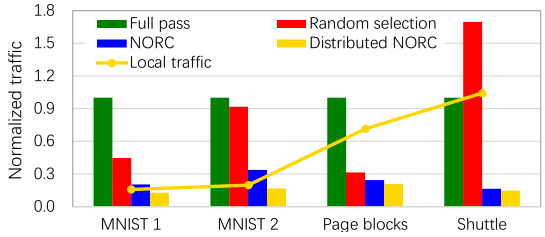


Fig. 6. Client-server network traffic and local network traffic

The reason an unbalanced partitioning is adopted is that we want the two embedded devices to simulate the case that data are highly distributed among many devices (more than 10 devices). In this way, device that are given less data instances represents a single device in a large device cluster. The other device represents the rest devices in a this cluster.

### B. Results

**RCI curve.** Figure 4 shows the curves of found categories. Full pass is supposed to be the best because it queries every instance. NORC and DNORC have the same performance as Full pass in three datasets, outperforming Random. On Page blocks dataset, NORC and DNORC miss 2 rare instances at the beginning. They still outperform Random and find the last rare instances at the same time with Full Pass in the end.

**Precision and recall.** In RCI problem, precision denote how many selected instances are rare and recall denotes how many rare instances are selected. The results in Figure 5 show that DNORC outperforms rest methods in precision and NORC also outperforms Full pass and Random. This shows that our DNORC improves the precision by utilizing neighboring information. It's shown that on MNIST, NORC and DNORC both have a maximum recall value 1.0. On Page blocks, one might consider use a larger threshold value at the beginning to increase the sensitivity.

**Network traffic.** The total traffic between two embedded devices and the server is shown in Figure 6. The data is amount of network traffic of different methods normalized to Full pass network traffic. NORC saves at least 65% of total bandwidth while DNORC saves over 75% of total bandwidth.

Less network bandwidth consumption relieves network congestion, which gives a lower transmission delay. In this way, **latency** for delivering data instances to server is reduced. On the other hand, the proposed NORC algorithm is able to process data stream, which further reduces latency.

Figure 6 also shows the local network traffic which is captured between embedded devices running DNORC. The amount of local traffic under MNIST experiments is small while it is relatively large under Page blocks and Shuttle. The reason is that only compressed data instances are transmitted between embedded devices. The dimension of MNIST data is compressed from 784 to 40 (19.6:1), which is much larger than the ratios of Page blocks and Shuttle that are 1.7:1 and 1.5:1 respectively.

**Time consumption.** The time consumption of NORC algorithm is tested on MNIST dataset. On average, NORC requires 165ms to process one instance. It costs totally 79.5 seconds for NORC to find all the rare categories on average.

## V. Conclusion

In this work, a near-sensor online rare category identification (NORC) framework is proposed. Experiments are conducted on Raspberry Pi's with real-world datasets. The results show that the proposed framework can significantly reduce network bandwidth consumption without sacrificing performance.

## VI. Acknowledgment

This work is supported by NSFC 61572411.

## References

[1] Y. Guo and D. Schuurmans. Discriminative batch mode active learning. In *NIPS*, 2008.
[2] J. He and J. Carbonell. Prior-free rare category detection. In *SIAM*, 2009.
[3] J. He, Y. Liu, and R. Lawrence. Graph-based rare category detection. In *ICDM*, 2008.
[4] T. M. Hospedales, S. Gong, and T. Xiang. Finding rare classes: Active learning with generative and discriminative models. *TKDE*, 2013.
[5] H. Huang, Q. He, J. He, and L. Ma. Radar: Rare category detection via computation of boundary degree. In *PAKDD*, 2011.
[6] H. Lin, S. Gao, D. Gotz, F. Du, J. He, and N. Cao. Rclens: Interactive rare category exploration and identification. *tvcg*, 2018.
[7] D. A. Ross, J. Lim, R.-S. Lin, and M.-H. Yang. Incremental learning for robust visual tracking. *IJCV*, 2008.
[8] B. Settles. Active learning literature survey. *Science*, 2010.
[9] W. Sultani, C. Chen, and M. Shah. Real-world anomaly detection in surveillance videos. In *CVPR*, 2018.
[10] Y.-K. Wang, C.-T. Fan, K.-Y. Cheng, and P. S. Deng. Real-time camera anomaly detection for real-world video surveillance. In *ICMLC*, 2011.