Learning On-the-Job to Re-rank Anomalies from Top-1 Feedback

Hemank Lamba
School of Computer Science
Carnegie Mellon University
hlamba@cs.cmu.edu

Leman Akoglu
H. John Heinz III College
Carnegie Mellon University
lakoglu@andrew.cmu.edu

Abstract

In many anomaly mining scenarios, a human expert verifies the anomaly at-the-top (as ranked by an anomaly detector) before they move on to the next. This verification produces a label—true positive (TP) or false positive (FP). In this work, we show how to leverage this label feedback for the top-1 instance to quickly re-rank the anomalies in an online fashion. In contrast to a detector that ranks once and goes offline, we propose a detector called OJRANK that works alongside the human and continues to learn (how to rank) *on-the-job*, i.e., from every feedback. The benefits OJRANK provides are two-fold; it *reduces* (i) *the false positive rate* by 'muting' the anomalies similar to FP instances; as well as (ii) *the expert effort* by elevating to the top the anomalies similar to a TP instance. We show that OJRANK achieves statistically significant improvement on both detection precision and human effort over the offline detector as well as existing state-of-the-art ranking strategies, while keeping the per feedback response time (to re-rank) well below a second.

1 Introduction

Given an anomaly mining setting in which a human expert needs to verify the anomalousness of instances as ranked by a detection algorithm, starting at the top of the ranked list, how can we leverage the labels they produce along the way to re-rank the anomalies? How can we update the ranking fast, without stalling the expert?

Anomaly detection has been mainly considered a standalone task that precedes any action-taking. In most applications, however, post-hoc human validation is either mandatory or necessary. For example, in auditing systems for insurance claims, expense invoices, tax returns, etc., the anomalies may be indicative of errors or fraud. However, one cannot automatically decline to pay-back the anomalous cases—errors must be located and fraudulent activities must be verified by human experts. For surveillance systems such as user behavior tracking or systems monitoring, anomalies may be indicative of malicious activities, however it may be undesirable to automatically shut down the anomalous user accounts or the running processes before verification. Similarly for knowledge discovery tasks, such as spotting new objects in sky images or novelties in particle physics experiments, it is also necessary for the human expert to validate the anomalies before claiming a discovery. In these types of anomaly mining settings, the human expert essentially produces a label (true or false positive) through verification of each next, yet-unverified top-1 instance.

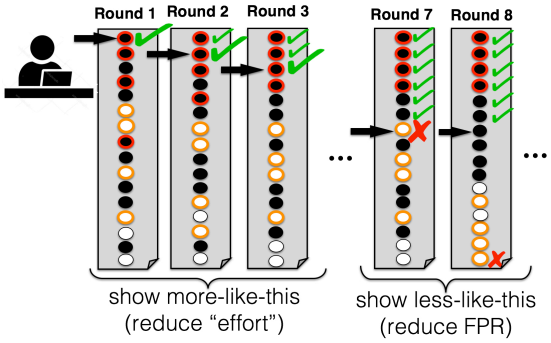
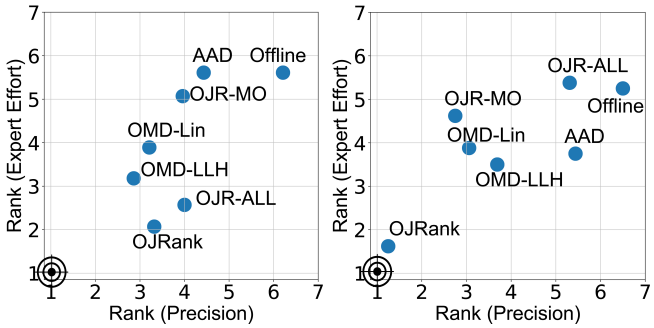


Figure 1: Illustration of OJRANK. Filled instances are true anomalies, unfilled are nominals, color depicts similarity. Upon each feedback, OJRANK re-ranks the instances, aiming to (a) push up similar True-Positives (red filled) & (b) 'mute' similar False-Positives (orange unfilled); (a) helps reduce expert effort, and both (a,b) increase true positive rate.

In this work, our aim is to leverage each label feedback to update our detection model to produce a new ranking. The idea is to interleave each feedback provided by the expert with a re-ranking by the updated model, which potentially also changes the top-1 instance the expert sees next. As such, this is a setting in which the detection model works alongside the expert, and learns to (re)rank *on the job* or in other words learns while the expert is working.

The goals of the re-ranking are two-fold. First is to improve the detection performance within the expert's verification budget. For example, if an auditor has the capacity and time to validate b invoices in a given day, the goal is to reach as high precision at b as possible. Intuitively, the more the number of detected anomalies, the higher is the return (i.e., savings from error and fraud) on investment (i.e., a day's of expert's work). Second goal is to reduce the expert's verification effort, which we define in



(a) Average rank of seven comparison methods over two sets of data; (left) BENCHMARK and (right) CLUSTERED; with respect to two metrics of interest; *precision@b* and *expert effort*.

Metric	Baselines / Datasets	AAD [1]	OMD Lin[2]	OMD LLH[2]	OJR MO	OJR ALL
prec. @b	BENCHMARK	0.015	0.5	0.5	0.005	0.008
	CLUSTERED	0.003	0.007	0.027	0.003	0.003
expert effort	BENCHMARK	0.001	0.010	0.024	$1e-4$	0.014
	CLUSTERED	0.027	0.007	0.012	0.004	0.004

(b) p-values for Wilcoxon signed ranked test between OJRANK and baseline methods for *precision@b* and *expert effort* over two sets of data. Note that except two cases (shaded in gray), performance gains are significant at 0.05.

Figure 2: **OJRANK** outperforms simple as well as state-of-the-art baselines *significantly* for two metrics - *precision@b* and *expert effort*. (See §5 for details)

terms of the similarity between consecutive instances expert gets to verify. Intuitively, the more similar instances they see in sequence, the lower would be the context switch and hence their verification effort. Besides those goals, the requirement is to update the ranking fast so as not to stall the expert waiting to be presented with the next top-1 instance. To these ends, we propose an On-the-Job (online) re-RANKing technique called OJRANK that employs a 'more-like-this' strategy upon a true positive feedback and 'less-like-this' strategy on encountering a false positive feedback, as illustrated in Figure 1 (see caption).

There exist related work on learning to rank from top-1 feedback for information retrieval tasks [3, 4]. However, due to the applications being different, their goals differ. Specifically, these work aim to learn from all the feedback to improve the performance of their *final* model. As such, they focus on metrics on the quality of the post-feedback ranked list whereas we aim to maximize precision on the instances labeled/verified by the expert. The two most related state-of-the-art work on feedback-based anomaly ranking are AAD [1, 5] and OMD [2]. They have used various loss functions and optimization algorithms for re-ranking anomalies based on top-1 feedback, toward improving precision at the budget, but *without any emphasis on expert effort*. OJRANK employs the same underlying tree-based ensemble detector as in these work and outperforms both AAD and OMD in terms of both precision and (especially) expert effort, as shown in Figure 2. We provide more details about the datasets and baselines in Section 5. To summarize, the contributions of this paper can be outlined as follows.

- **On-the-Job Learning to Re-rank Anomalies:** We address the problem of learning to re-rank anomalies on the job, i.e. while the expert is working toward verifying the top-ranked anomalies. Each verification of the top-1 instance produces a label, which our proposed OJRANK uses to update the ranking presented to the expert next. To this end, we employ a pairwise learning to rank objective coupled with a carefully-designed online gradient descent learning, where the update equation has a clear interpretation for the detection task.
- **Higher Precision, Lower Effort:** We demonstrate that OJRANK employs a 'more-like-this' update strategy upon receiving a true positive (TP) feedback, and a 'less-like-this' strategy upon a false positive (FP) feedback. Both help achieve higher detection precision as they respectively boost TPs and mute FPs. At the same time, 'more-like-this' updates enable *similar* anomalies to be pushed up in the rank order and shown consecutively, which helps reduce expert effort.
- **Time and Space Efficiency:** OJRANK updates ranking after every label feedback, during which the user stalls to be presented with the next top-1 instance to verify. We show that OJRANK's online updates take constant-time in complexity and are near-instantaneous empirically, where the re-ranking is done within one fifth of a second on average. Moreover, OJRANK requires only linear space on the number of instances.

The code, datasets used in the experiments and supplementary information is available at https://ojrank.github.io.

2 Related Work

We discuss related work in two categories: *active sampling* (which carefully selects the instances to be labeled) versus *top-1 feedback* (which simply selects the top instance—no strategy is involved). We note that active learning (AL) and rare category discovery (RCD) fall under the former category, while on-the-job learning (OJL) is different and falls under the latter. Table 1 shows a quick comparison between OJRANK and various active sampling and top-1 feedback methods. Detailed discussion follows.

Active Sampling: AL is the task of selecting a small budget of most informative instances to be queried for labels, such that a model trained on those labeled instances achieves high performance. AL for classification has employed various selection/sampling strategies such as uncertainty, query-by-committee, variance reduction, etc. for the

Table 1: Qualitative comparison between OJRANK and related methods.

Properties	Methods						
	AL [6–11]	RCD [12–14]	Ghani and Kumar [15]	Top-1 L2R [3, 4]	AAD [1, 5]	OMD [2]	**OJRANK**
Top-1 Feedback	✗	✗	✗	✓	✓	✓	✓
Online Model Updates	✗	✗	✗	✓	✗	✓	✓
Precision @ Budget	✗	✗	✓	✗	✓	✓	✓
Expert Effort	✗	✗	✓	✗	✗	✗	✓

details of which we refer to a survey by Settles [16]. Others studied active sampling for learning-to-rank [6–9] and for anomaly detection [10–12]. A key difference is in how the queries are selected: In active sampling they are strategically and carefully chosen; in OJL the query is always the top-most yet-unlabeled instance (i.e., there is no active selection). The goals also differ: active sampling aims to maximize the label-incorporated model's final performance on unlabeled examples, i.e. performance is measured *after* querying; in contrast OJL aims to maximize the number of anomalous instances presented to user *during* querying.

Active sampling techniques have also been studied for rare category discovery (RCD) [12–14], which is a setting where anomalies are assumed to form multiple micro-clusters (i.e. rare categories). The goal is also notably different from OJL's, where they aim to identify at least one example from each rare category by (strategically) querying the expert for as few labels as possible in total.

Ghani and Kumar [15] studied interactively detecting errors in insurance claims, while also aiming to reduce context switching costs for experts. They use a query selection heuristic that first clusters the top-scoring instances based on similarity and ranks the clusters based on a combination of measures. Instances from top-ranked cluster is then shown to the expert (helping reduce context switch) until precision falls below a threshold upon which instances from the next cluster are presented. Their model is never updated. In contrast, OJRANK boosts up instances similar to a true-positive feedback in an online fashion.

Top-1 Feedback: Compared to active sampling, there has been relatively limited work on learning-to-rank (L2R) problems where feedback is only given for the topmost instance of the ranked list. Chaudhuri et al. [3, 4] proposed algorithms for well-known L2R loss functions from the pointwise, pairwise and listwise families. However, they aim to maximize the *resulting* performance over the entire ranked list after all feedback is collected, unlike in our setting, where our goal is to maximize the overall number of anomalies presented to the expert *during* feedback.

On the anomaly ranking side, Das et al. [1] proposed AAD (Active[1] Anomaly Discovery) that partly share the same goal as OJRANK, that is to maximize the number of anomalies presented at the top (and partly not, as we also care about expert effort). After each feedback, AAD solves an optimization program involving constraints between *all* of the previous expert-labeled anomalies and nominals (i.e., updates are not online). As the number of pairs grows along with each feedback round, the all-pairs constraints increase the running time. The initial AAD method was intended to work with LODA [17], a projection-based ensemble detection algorithm, which is later extended [5] for tree-based ensembles like iForest [18] and HS-Trees [19]. AAD is also sped up by intelligently replacing the all-pairs constraints by those relative to the instance ranked at the τ^{th}-quantile.

Most recently, Siddiqui et al. [2] proposed to optimize pointwise loss functions in an online fashion upon each feedback via online mirror descent (OMD). AAD and OMD (and variants) all aim to maximize the number of true anomalies shown to the expert. However, they do not put any emphasis on expert effort, which takes into account the effort consumed while verifying instances that is likely to decrease if the instances shown consecutively are similar.

3 Preliminaries and Problem Definition

3.1 Learning on-the-job Setup: We are given a dataset containing n instances $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ in d dimensions, as well as an anomaly detection model M that provides scores for the input instances $\{s_1, \ldots, s_n\}$, the higher the more anomalous. The instances are ranked in descending order of these scores.

The procedure of learning on-the-job proceeds in rounds. In each round, (a) the expert verifies the top-1 instance with the highest score and reveals a label and (b) our OJRANK algorithm uses this feedback to update the detection model M and hence the ranking of the instances. In the next round, the expert is presented with the top-1 instance based on the *updated* ranking and so on. This procedure continues for b rounds, where b specifies the expert's budget (e.g., the number of invoices an auditor has the capacity to analyze within a day's work).

3.2 Family of Detection Models: Our proposed work can subsume any *ensemble* anomaly detection model M, where each ensemble component provides a separate score and the overall anomalousness score of an instance is the sum (or average) of those scores across components. Many state-of-the-art detectors fall into this category such as LOF with

[1]Here, their use of the term 'active' is misleading, as AAD does not employ any active sampling strategy for querying—always the top-1 instance is labeled. We find OJL to be a more suitable name, as the expert verifies the next top-1 instance sequentially as part of their job.

feature bagging [20], LODA [17] with 1-d projections, and various tree-based ensembles including iForest [21], HS-Trees [19], and RS-Forest [22].

Without loss of this generality, this paper adopts the iForest detector. We denote the number of components (i.e., iTrees) in the ensemble by m. Each iTree is constructed over a random subsample of the input data $\mathcal{D}$, by splitting the data at each internal node over a randomly selected feature and a threshold. As anomalies are fewer in number and isolated from nominal instances, they require fewer splits to reach a leaf node, hence are quickly isolated. Therefore, anomalous instances are located at a shorter depth than nominal instances on average over all the trees in the iForest.

We denote the number of leaves in tree t by L_t. Each instance is placed in exactly one of the L_t leaves in each tree. The score of an instance u by the t-th tree is given by

$$s_u^{(t)} = 1/[\ \text{pathlen}(l_u^{(t)}) + h(cnt_{l_u^{(t)}})\ ] \tag{3.1}$$

where $l_u^{(t)}$ denotes the leaf in t which u falls into; pathlen captures its depth from the root, $cnt_{l_u^{(t)}}$ is the total number of instances it contains, and h function returns the expected path length of unsuccessful searches in a Binary Search Tree (BST) constructed with the given number of samples.

In this paper we work with the leaves representation denoted by $\mathbf{s}_u = [\mathbf{s}_u^{(1)} \dots \mathbf{s}_u^{(m)}]$ where $\mathbf{s}_u^{(t)}$ is a vector with entries $i = 1 \dots L_t$ where

$$\mathbf{s}_u[i] = \begin{cases} s_u^{(t)}, & \text{if } i = l_u^{(t)} \\ 0, & \text{otherwise} \end{cases} \tag{3.2}$$

As such $\mathbf{s}_u$ is $\sum_{t=1}^{m} L_t = l$ dimensional with exactly m nonzeros. We denote by $\boldsymbol{S} \in \mathbb{R}^{n \times l}$ the scores matrix. As such, $\mathbf{s} = \boldsymbol{S} \cdot \mathbf{1}$ contains all the anomaly scores.

3.3 Metrics of Interest and Problem Statement:

In this work, we aim to improve two different metrics of interest.

First is the total number of true anomalies verified by the expert within their budget b. In auditing systems, this would correspond to the number of erroneous (tax, insurance, reimbursement) invoices caught among the ones they could analyze within a day's work—others that could not be analyzed need to be paid in full—as such, the more errors caught, the higher the savings could be. This metric is essentially the precision at the budget, denoted *precision@b*.

The second metric of interest is related to the cognitive burden the expert would have due to context switch. Intuitively, the more similar two instances analyzed in sequence are, the lower the context switching costs would be for the expert. Since expert effort is not as well-established a metric as precision, we define it as follows.

Definition 1 (***expert effort***). *: Given the sequence of b instances $\{\mathbf{s}_{\pi(1)}, \dots, \mathbf{s}_{\pi(b)}\}$, where $\pi(r)$ denotes the index of the instance ranked at the top in round r, we define:*

$$\textit{expert effort} = \sum_{r=1}^{b-1} 1 - \textit{sim}\big(\mathbf{s}_{\pi(r)}, \mathbf{s}_{\pi(r+1)}\big)\ , \tag{3.3}$$

which is the similarity between consecutive instances verified by the expert. Here we employ cosine similarity in the scoring space $\mathcal{S}$. If two instances fall in the same leaves with high scores (see Eq. (3.1)), these points are considered anomalous for the same reasons (in the same feature subspaces). Intuitively, analyzing invoices containing similar type of anomalies would reduce verification effort.

One can also argue for similarity in the input space $\mathcal{X}$. This captures the insight that two similar-looking invoices would be easier for the expert to process back to back. In the experiments we show that OJRANK outperforms the baselines w.r.t. both similarities.

Having outlined the preliminaries and goals, our problem statement can be given as follows:

Problem 1 (**On-the-Job Re-ranking**). *For rounds $1 \dots b$:*

- ***Obtain** label for the top-1 instance from expert*
- ***Update** the detection model based on the feedback and re-rank instances*

such that precision@b is maximized, total expert effort is minimized, and updates are fast.

We set up the model update problem as learning a ranking of the instances based on a *weighted* sum of leaf scores. That is, we replace the sum $\mathbf{s} = \boldsymbol{S} \cdot \mathbf{1}$ with

$$\mathbf{s} = \boldsymbol{S} \cdot \mathbf{w} \tag{3.4}$$

and aim to estimate $\mathbf{w}$ from expert feedback on-the-job.[2]

4 Proposed Approach: OJRANK

We formulate the on-the-job re-ranking problem as an online learning-to-rank task. To this end, we adopt a pairwise learning to rank objective with a convex cross entropy loss.

Given training examples $\langle (u, v), p_{uv} \rangle \in T$ where p_{uv} is the desired probability of instance u being ranked above instance v, we aim to find the weight vector that minimizes the cross entropy loss over all the training pairs:

$$\min_{\mathbf{w}} f = \sum_{(u,v) \in T} -p_{uv} \log(\widehat{p}_{uv}) - (1 - p_{uv}) \log(1 - \widehat{p}_{uv}) \tag{4.5}$$

where $\widehat{p}_{uv}$ is the estimated probability based on our current estimate of $\mathbf{w}$, and is acquired using the logistic function:

$$\widehat{p}_{uv} = \frac{e^{(s_u - s_v)}}{1 + e^{(s_u - s_v)}}\ , \text{ where } s_u = \mathbf{s}[u] = \boldsymbol{S}_u \cdot \mathbf{w} \tag{4.6}$$

[2] Using leaves representation provides us with the capacity to weight l different feature subspaces (that each leaf corresponds to) rather than m different trees, the former allowing a larger granularity as $l > m$.

Updating $\mathbf{w}$ leads to updating estimated probability $\widehat{p}_{uv}$ and moving it closer to the desired probability p_{uv}.

OJRANK re-ranks the anomalies after obtaining the feedback on top-1 instance u from the expert. It has two major components: (1) **Generating pairs** - as the cross entropy loss function is pairwise, we pair u for which we received feedback with other instances, which we choose by either sampling or using the historical instances labeled in the previous rounds, and (2) **Optimization** - which involves updating $\mathbf{w}$ via optimizing the loss function over the generated pairs, which is then used to re-compute the scores $\mathbf{s}$ in the following round. The top-1 instance from the updated ranking is presented to the expert for feedback. The steps of the algorithm are given in Algorithm 1.

4.1 Generating pairs: After each round of feedback, we obtain label from the expert for a single instance u - we pair this instance with other instances v to create pairs.

Using history: We pair u with each *previously labeled* instance v with an opposite label to that of u. This allows us to establish a clear ordering amongst paired instances - indicating which instance should be ranked higher. For example, if u is anomalous (nominal), we pair it with nominal (anomalous) instances v, hence we are certain that u should be ranked higher (lower) than v.

By sampling: During initial feedback rounds, it is possible that there are no instances in the history that have opposite label to that of u. To handle such cases, we sample v from unlabeled instances such that we maximize the probability of obtaining oppositely labeled instances. Specifically, we skew the probability of sampling from unlabeled instances such that chances of getting oppositely labeled instances increase. This is done by (i) truncating the sample space – if u is anomalous (nominal), we sample from bottom (top) half of the ranked list and (ii) sampling an instance with probability inversely (directly) proportional to its score. In particular, when u is anomalous we use the sampling probability proportional to $1/s_v$ for instance v. If u is nominal, we sample v (after normalizing the scores $\bar{s}_v \in [0,1]$) with probability proportional to $(c\bar{s}_v + 1)^{1/c}$ with $c = -0.99$ to increase the chances of sampling an anomalous v. The polynomial scaling for the latter is to account for the fewer number of anomalies in the data.

We give priority to generating pairs using history rather than sampling, as sampling could lead to pairing identically labeled instances. Therefore, we place an upper limit k on the number of sampled pairs and only when the number of pairs generated from history is less than (small) k, we sample the remaining pairs (lines 9, 14 in Algo. 1).

Besides the pairs (u, v), we also need to provide desired probability p_{uv} for each pair as input to the objective function in (4.5). For v that has been sampled from history and u anomalous (nominal), we set p_{uv} to be the maximum (minimum) of all the current estimated probability values among

Algorithm 1 Proposed OJRANK

Input Ensemble (in our implementation iForest) scores $\boldsymbol{S} \in \mathbb{R}^{n\times l}$; Initial weights $\mathbf{w} \in \mathbb{R}^l$; Budget b; Scale factor δ; Num. pairs to sample k

1: $\mathbf{s} = \boldsymbol{S} \cdot \mathbf{w}$; round $= 0$ ▷ Initialize
2: **while** round $< b$ **do**
3: $y_u \leftarrow$label from expert for $u := \arg\max(\mathbf{s})$ ▷ Top-1 feedback
/* Setting Up (Generating Pairs)*/
4: $a \leftarrow \arg\max(\mathbf{s}), z \leftarrow \arg\min(\mathbf{s})$
5: $P_H = \emptyset, P_S = \emptyset$ ▷ Historical and Sampled sets
6: **if** $y_u = 1$ **then** ▷ True anomaly (positive)
7: **for** $v \in \mathcal{H}_N$ **do**
8: add $\langle (u,v), \widehat{p}_{az} \rangle$ to P_H
9: **for** $v \in$ sample$(\mathbf{s}, y_u, (k - |\mathcal{H}_N|)_+)$ **do**
10: add $\langle (u,v), (1+\delta)\widehat{p}_{uv} \rangle$ to P_S
11: **else** ▷ False positive
12: **for** $v \in \mathcal{H}_A$ **do**
13: add $\langle (u,v), (1-\widehat{p}_{az}) \rangle$ to P_H
14: **for** $v \in$ sample$(\mathbf{s}, y_u, (k - |\mathcal{H}_A|)_+$ **do**
15: add $\langle (u,v), (1-\delta)\widehat{p}_{uv} \rangle$ to P_S
/* Optimization (Updating weights)*/
16: $t = 0$; $\mathbf{w}^t = \mathbf{w}^{t-1} \leftarrow \mathbf{w}$ ▷ Initialize with $\mathbf{w}$ from last round
17: $\eta = 0.1, \gamma = 0.75, \epsilon = 10^{-8}$, batch_size $= 100, T_{\max} = 1000$
18: **repeat**
19: $\widetilde{P}_H \leftarrow$ get_next_SGD_batch$(P_H,$ batch_size$)$
20: **for** $\langle (u,v), p_{uv} \rangle \in \widetilde{P}_H \cup P_S$ **do**
21: **if** $(u,v) \in P_S$ **then** $c \leftarrow \mathbb{1}(\boldsymbol{S}_u > 0)$ **else** $c \leftarrow [1 \dots l]$
22: $\mathbf{w}^{t+1}[c] \leftarrow \mathbf{w}^t[c] - \gamma(\mathbf{w}^t[c] - \mathbf{w}^{t-1}[c])$
$- \eta(\boldsymbol{S}_u[c] - \boldsymbol{S}_v[c])(\widehat{p}_{uv} - p_{uv})$
23: $t \leftarrow t + 1$
24: **until** $t \geq T_{\max}$ OR $f(\mathbf{w}^{t+1}) - f(\mathbf{w}^t) \leq \epsilon$
25: $\mathbf{w} := \mathbf{w}^{t+1}$, $\mathbf{s} = \boldsymbol{S} \cdot \mathbf{w}$ ▷ Rescore according to the updated $\mathbf{w}$
26: **if** $y_u == 1$ **then** $\mathcal{H}_A := \mathcal{H}_A \cup u$ **else** $\mathcal{H}_N := \mathcal{H}_N \cup u$
27: round $\leftarrow$ round $+ 1$

all pairs (lines 8 and 13). For example, if u is anomalous (v is nominal), we set $p_{uv} = \widehat{p}_{az}$ where a is the highest scored instance and z is the lowest scored instance (line 4). Here, we are certain about the ordering among the (u, v) instances, thus we set the desired probability to maximum so as to push u in the correct direction quickly. On the other hand, when v is obtained by sampling, there is still a chance that we might have generated identically labeled instances. In that case, we aim to avoid the mistake of pushing u in opposite direction by a high magnitude. Therefore, for sampled pairs we nudge the current estimate of probability between u and v by only a (small) factor of δ, i.e., $p_{uv} = (1 \pm \delta)\widehat{p}_{uv}$, the sign depending on whether u is anomalous or nominal (lines 10, 15).

4.2 Optimization: We now have training instances in the form of $\langle (u,v), p_{uv} \rangle \in P = \widetilde{P}_H \cup P_S$, generated using history and via sampling as explained in the previous subsection. Next, we are interested in solving the optimization problem in (4.5), i.e., find $\mathbf{w}$ such that the cross entropy loss over all pairs in the training set is minimized. The gradient update equation is written as

$$\mathbf{w}^t = \mathbf{w}^{t-1} - \eta \cdot \sum_{(u,v)\in P} (\widehat{p}_{uv} - p_{uv})(\boldsymbol{S}_u - \boldsymbol{S}_v) \,. \tag{4.7}$$

Relation of gradient updates to precision and effort: Importantly, these gradient updates are interpretable and have clear impact on the expert effort and the true positive rate. Consider the case where u is anomalous and v is nominal: by construction, we know that $p_{uv} > \widehat{p}_{uv}$. Each coordinate in $\mathbf{w}$ represents the relative importance of a leaf (i.e., subspace) in a tree. Looking at the update, we can observe that all the coordinates that are responsible for making u anomalous, i.e., those with a magnitude in $\boldsymbol{S}_u$ significantly higher than those in $\boldsymbol{S}_v$, will increase in weight. Therefore, the updated $\mathbf{w}$ will push u, as well as other anomalous instances similar to u (i.e., those that share the same high-scoring leaves with u) higher to the top; contributing to reduced effort and increased precision. In contrast, $p_{uv} < \widehat{p}_{uv}$ when u is nominal, in which case updates will tend to push u and other similar instances down in ranking, contributing to reduced false positive rate.

Coordinate selection: One caveat with the gradient updates is the set of *sampled* pairs, which may contain identically-labeled (u, v) pairs. In those cases, the updates will nevertheless enforce a (wrong) ordering between them. For example, if u is nominal and we sampled v nominal as well, updates will tend to increase weights on coordinates of v that have higher magnitude than those of u (e.g., different leaf in the same tree), causing v (and nominals similar to v) climb higher in the list, which is undesirable. We circumvent this issue by updating only the non-zero coordinates of u for sampled pairs (as shown in line 21).

Online updates and acceleration: Note that $\mathbf{w}^0$ is set to the latest $\mathbf{w}$ from the previous round (lines 16, 25), and the updates are only over the newly generated pairs P for the top-1 labeled instance u in the current round (line 20). As such, OJRANK stands on *online* gradient-based learning.

Specifically, we employ batch stochastic gradient descent (SGD). Each batch is selected from the pairs that are created using history (line 19) and combined with the (at most k) sampled pairs. We also use the momentum-based SGD to accelerate the descent (hence, the response time). The momentum-based update equation (line 22) is similar to Eq. (4.7), which is obtained by adding γ fraction (momentum factor) of the gradient from the previous step to the current gradient update vector. Finally, all relevant parameters are listed in line 17 of Algo 1 and our implementation is open-sourced at `https://ojrank.github.io`.

5 Evaluation

In this section, we evaluate our proposed OJRANK approach in comparison with five baselines as listed in §5.1. Experiments are conducted over two types of datasets, introduced in §5.2. We then present the results with respect to three performance metrics of interest in §5.3: *precision@b*, *expert effort* and speed (i.e., online response time).

Table 2: Summary statistics for two sets of data used in experiments: (left) BENCHMARK and (right) CLUSTERED.

BENCHMARK DATASETS			CLUSTERED DATASETS		
Name	n	**Anom. %**	**Name**	n	**Anom. %**
abalone	1920	1.51	vowels	2821	1.77
ann	3251	2.24	optdigits	592	4.22
cardio	1700	2.64	letters	2433	2.05
ecoli	336	2.67	sensor	16257	0.92
glass	214	4.20	segment	1090	4.58
mammography	11183	2.32	statlog	1665	3.00
shuttle	12345	7.02	vehicle	495	6.06
wbc	378	5.56	svmguide	544	9.19
yeast	1191	4.61			
lympho	148	4.05			
musk	3062	3.16			
thyroid	3772	2.46			
wine	129	7.76			
vertebral	240	12.5			

5.1 Baselines: We compare OJRANK with two state-of-the-art techniques that addressed the problem of online re-ranking of anomalies from top-1 feedback, as discussed in related work (§2). We also compare with the offline baseline as well as two variants of OJRANK explained below.

- **AAD [1]**: See §2 and Table 1.
- **OMD [2]**: See §2 and Table 1. We compare to both versions based on the type of loss used – (a) OMD-Lin (linear loss) and (b) OMD-LLH (log-likelihood loss).
- **Offline**: Static top-b instances based on the initial ranking by the detector, no re-ranking over rounds.
- **OJR-MO**: Mistake-Only variant; we run online model updates only when top-1 feedback is a false positive.
- **OJR-ALL**: All coordinates variant; we do not scale the sampling probabilities–increases the risk of identically-labeled pairs (`ilp`)–and perform no coordinate selection for sampled pairs–enforces a ranking among `ilp`.

All compared methods use the same underlying iForest detectors. We report performance results averaged over 10 different runs of iForest. AAD and OMD are run with the author-recommended parameters. We set budget b equal to the number of true anomalies in each dataset.

5.2 Datasets: We evaluate performance over two types of datasets as listed in Table 2; namely, (1) BENCHMARK DATASETS: a set of 14 real-world datasets and (2) CLUSTERED DATASETS: 8 datasets generated from multi-class classification datasets, as described below.

BENCHMARK DATASETS: The first data collection contains 14 real-world datasets from a publicly-available outlier detection dataset repository [23].

CLUSTERED DATASETS: OJRANK learns well from feedback when there are other points similar to the feedback instance, i.e., when instances form (micro)-clusters. Learning from an extreme outlier would be very limited, as there are no other points in the dataset that are similar to it, hence no other instances can benefit from the feedback. To create such a setting, we synthetically generate 8 datasets by modifying multi-class datasets from the UCI repository.

Table 3: ***precision@b* on BENCHMARK DATASETS.** Per dataset rank provided in parentheses (the lower the better). Average rank across datasets given in the last row. Symbols ▲ and △ denote the cases where OJRANK is significantly better than the baseline w.r.t. the Wilcoxon signed rank test, respectively at (p<0.01) and (p<0.05).

Dataset	OJRANK	OJR-MO	OJR-ALL	AAD	OMD-Lin	OMD-LLH	Offline
abalone	0.52 ± 0.00(5.0)	0.52 ± 0.00(5.0)	0.52 ± 0.00(5.0)	0.56 ± 0.02(1.0)	0.52 ± 0.01(3.0)	0.54 ± 0.02(2.0)	0.51 ± 0.02(7.0)
ann	0.75 ± 0.03(3.0)	0.73 ± 0.03(4.0)	0.52 ± 0.32(5.0)	0.39 ± 0.05(6.0)	0.78 ± 0.03(1.0)	0.76 ± 0.04(2.0)	0.19 ± 0.07(7.0)
cardio	0.64 ± 0.02(4.0)	0.65 ± 0.02(2.0)	0.60 ± 0.06(5.0)	0.55 ± 0.04(6.0)	0.65 ± 0.01(3.0)	0.69 ± 0.04(1.0)	0.38 ± 0.04(7.0)
ecoli	0.72 ± 0.06(1.0)	0.57 ± 0.08(3.0)	0.70 ± 0.10(2.0)	0.44 ± 0.05(6.0)	0.56 ± 0.09(4.0)	0.51 ± 0.10(5.0)	0.42 ± 0.04(7.0)
glass	0.11 ± 0.00(4.5)	0.11 ± 0.00(4.5)	0.17 ± 0.17(1.0)	0.11 ± 0.00(4.5)	0.11 ± 0.00(4.5)	0.11 ± 0.00(4.5)	0.11 ± 0.00(4.5)
mammography	0.58 ± 0.02(3.0)	0.56 ± 0.02(5.0)	0.56 ± 0.01(4.0)	0.41 ± 0.02(6.0)	0.60 ± 0.01(2.0)	0.62 ± 0.01(1.0)	0.25 ± 0.05(7.0)
shuttle	0.96 ± 0.04(5.0)	0.94 ± 0.03(6.0)	0.97 ± 0.01(4.0)	0.98 ± 0.00(1.0)	0.97 ± 0.01(3.0)	0.98 ± 0.00(2.0)	0.89 ± 0.03(7.0)
wbc	0.71 ± 0.06(1.0)	0.66 ± 0.03(3.0)	0.67 ± 0.05(2.0)	0.53 ± 0.05(6.0)	0.60 ± 0.03(4.0)	0.59 ± 0.05(5.0)	0.50 ± 0.03(7.0)
yeast	0.27 ± 0.05(5.0)	0.25 ± 0.05(6.0)	0.18 ± 0.04(7.0)	0.34 ± 0.02(3.0)	0.35 ± 0.01(2.0)	0.36 ± 0.04(1.0)	0.34 ± 0.01(4.0)
lympho	0.92 ± 0.08(6.0)	0.93 ± 0.08(3.0)	0.60 ± 0.11(7.0)	0.93 ± 0.08(3.0)	0.93 ± 0.08(3.0)	0.93 ± 0.08(3.0)	0.93 ± 0.08(3.0)
musk	1.00 ± 0.00(3.0)	0.99 ± 0.00(4.0)	0.99 ± 0.01(6.0)	0.99 ± 0.02(5.0)	1.00 ± 0.00(1.5)	1.00 ± 0.00(1.5)	0.97 ± 0.03(7.0)
thyroid	0.81 ± 0.02(4.0)	0.77 ± 0.01(5.0)	0.82 ± 0.02(2.0)	0.69 ± 0.03(6.0)	0.82 ± 0.02(3.0)	0.86 ± 0.01(1.0)	0.54 ± 0.03(7.0)
wine	0.42 ± 0.19(1.0)	0.27 ± 0.13(3.0)	0.28 ± 0.35(2.0)	0.09 ± 0.03(5.5)	0.09 ± 0.03(5.5)	0.09 ± 0.03(5.5)	0.09 ± 0.03(5.5)
vertebral	0.33 ± 0.04(1.0)	0.31 ± 0.06(2.0)	0.05 ± 0.05(4.0)	0.05 ± 0.02(3.0)	0.05 ± 0.02(5.5)	0.05 ± 0.02(5.5)	0.04 ± 0.02(7.0)
Avg. Rank	**3.32**	**3.96**▲	**4.00**▲	**4.43**△	**3.21**	**2.86**	**6.21**▲

Table 4: ***precision@b* on CLUSTERED DATASETS.** Per dataset rank provided in parentheses (lower is better). Average rank provided in the last row. Symbol ▲ denote the cases where OJRANK is significantly better than the corresponding baseline w.r.t. the Wilcoxon signed rank test at (p<0.01).

Dataset	OJRANK	OJR-MO	OJR-ALL	AAD	OMD-Lin	OMD-LLH	Offline
vowels	0.78 ± 0.09(2.0)	0.58 ± 0.09(4.0)	0.09 ± 0.26(7.0)	0.45 ± 0.04(5.0)	0.77 ± 0.06(3.0)	0.80 ± 0.05(1.0)	0.17 ± 0.06(6.0)
optdigits	0.08 ± 0.13(1.0)	0.07 ± 0.13(2.0)	0.01 ± 0.03(7.0)	0.04 ± 0.03(5.0)	0.06 ± 0.04(3.0)	0.05 ± 0.03(4.0)	0.03 ± 0.02(6.0)
letters	0.62 ± 0.12(1.0)	0.51 ± 0.12(3.0)	0.21 ± 0.28(5.0)	0.16 ± 0.06(6.0)	0.53 ± 0.12(2.0)	0.47 ± 0.17(4.0)	0.05 ± 0.01(7.0)
sensor	0.95 ± 0.04(1.0)	0.95 ± 0.03(2.0)	0.48 ± 0.38(6.0)	0.52 ± 0.12(5.0)	0.95 ± 0.03(4.0)	0.95 ± 0.03(3.0)	0.14 ± 0.08(7.0)
segment	0.48 ± 0.20(1.0)	0.40 ± 0.14(2.0)	0.00 ± 0.00(6.5)	0.02 ± 0.02(5.0)	0.25 ± 0.15(3.0)	0.04 ± 0.03(4.0)	0.00 ± 0.00(6.5)
statlog	0.93 ± 0.02(2.0)	0.91 ± 0.01(5.0)	0.92 ± 0.01(4.0)	0.90 ± 0.01(6.0)	0.93 ± 0.01(1.0)	0.92 ± 0.01(3.0)	0.87 ± 0.03(7.0)
vehicle	0.31 ± 0.14(1.0)	0.29 ± 0.07(2.0)	0.12 ± 0.03(4.0)	0.11 ± 0.03(6.0)	0.13 ± 0.04(3.0)	0.11 ± 0.03(5.0)	0.09 ± 0.02(7.0)
svmguide	0.12 ± 0.04(1.0)	0.11 ± 0.01(2.0)	0.10 ± 0.03(3.0)	0.10 ± 0.00(5.5)	0.10 ± 0.00(5.5)	0.10 ± 0.00(5.5)	0.10 ± 0.00(5.5)
Avg. Rank	**1.25**	**2.75**▲	**5.31**▲	**5.44**▲	**3.06**▲	**3.69**▲	**6.50**▲

From each multi-class dataset, we first select two classes at random, with the intuition that the instances within each class would be clustered. Instances from the remaining classes are designated as nominals. In case of too many remaining classes, 3 of them are randomly selected. We next downsample the selected two classes to equal number so that the percentage is consistent with the usual anomaly detection settings. We designate the downsampled instances from the first class as "anomalies" and those from the other as "rare nominals". The detector is likely to rank both as anomalous, yet from the expert's point of view, they would correspond to true and false positives, respectively. Here, "rare nominals" represent rare yet uninteresting group of instances. This setup allows us to directly test the ability of the methods in learning to boost/mute instances from these respective classes upon expert feedback.

Summary statistics for the CLUSTERED DATASETS are given in Table 2. Details for the mapping of classes to above categories are provided in Table 7 of Supplementary. We also share these generated datasets at our aforementioned URL.

5.3 Results : We analyze performance results over both datasets w.r.t. (a) *precision@b*, (b) *expert effort* and (c) runtime per update. We also present a sensitivity analysis of OJRANK w.r.t. two input parameters in Algo. 1; δ and k.

***Precision@b*:** Table 3 and Table 4 provide precision across BENCHMARK DATASETS and CLUSTERED DATASETS, respectively. On each dataset, we show the average precision and standard deviation over 10 different runs of iForest. Rank of each method per dataset is in parentheses (in case of ties, average of the ranks are assigned to each tied method). Finally, the last row gives the average rank per method across all datasets (lower is better).

The precision magnitudes differ quite a bit among datasets, therefore we perform a *rank* test to compare the methods statistically. Specifically, the Wilcoxon signed rank test between OJRANK and each baseline shows that OJRANK significantly outperforms its two variants and the offline baseline at p<0.01 on both BENCHMARK and CLUSTERED datasets. (Actual p-values can be found in Figure 2 (b) in Section 1.) In fact, notice that Offline is ranked at the bottom in both setups, demonstrating the value of learning on-the-job. OJRANK is also superior to AAD, respectively at p<0.05 and p<0.01. We find no significant difference (p=0.5) between OMD variants and OJRANK on BENCHMARK DATASETS. On the other hand, OJRANK significantly outperforms all baselines including OMD on CLUSTERED DATASETS, showcasing its ability to learn from feedback on clustered instances.

We illustrate how the number of true discovered anomalies change over rounds with the expert on several datasets from BENCHMARK DATASETS and CLUSTERED DATASETS in Figure 6 and Figure 7, respectively (See Supplementary).

Table 5: ***Expert effort* on BENCHMARK DATASETS.** Per dataset rank shown in parentheses (lower is better). Average rank is in the second last row (*effort* in $\mathcal{S}$ space). Average rank for *effort* in $\mathcal{X}$ space also given in last row. Symbols ▲(p<0.01) and △(p<0.05) denote the cases where OJRANK is significantly better than the baseline w.r.t. Wilcoxon signed rank test.

Dataset	OJRANK	OJR-MO	OJR-ALL	AAD	OMD-Lin	OMD-LLH	Offline
abalone	0.59 ± 0.02(3.0)	0.65 ± 0.02(7.0)	0.60 ± 0.03(5.0)	0.60 ± 0.08(4.0)	0.54 ± 0.04(1.0)	0.56 ± 0.03(2.0)	0.65 ± 0.03(6.0)
ann	0.49 ± 0.02(1.0)	0.70 ± 0.02(5.0)	0.65 ± 0.21(4.0)	0.92 ± 0.02(7.0)	0.60 ± 0.02(3.0)	0.60 ± 0.02(2.0)	0.92 ± 0.02(6.0)
cardio	0.69 ± 0.01(1.0)	0.83 ± 0.03(5.0)	0.72 ± 0.03(2.0)	0.89 ± 0.03(6.0)	0.77 ± 0.02(4.0)	0.73 ± 0.03(3.0)	0.90 ± 0.03(7.0)
ecoli	0.96 ± 0.02(1.0)	1.10 ± 0.02(7.0)	0.98 ± 0.02(2.0)	1.09 ± 0.02(5.0)	1.07 ± 0.02(4.0)	1.07 ± 0.04(3.0)	1.10 ± 0.02(6.0)
glass	1.12 ± 0.00(7.0)	1.12 ± 0.00(6.0)	1.09 ± 0.07(1.0)	1.12 ± 0.00(3.5)	1.12 ± 0.00(3.5)	1.12 ± 0.00(3.5)	1.12 ± 0.00(3.5)
mammography	0.57 ± 0.01(2.0)	0.70 ± 0.01(5.0)	0.60 ± 0.01(3.0)	0.82 ± 0.02(6.0)	0.60 ± 0.02(4.0)	0.57 ± 0.02(1.0)	0.83 ± 0.02(7.0)
shuttle	0.20 ± 0.03(2.0)	0.65 ± 0.04(5.0)	0.20 ± 0.01(1.0)	0.81 ± 0.02(7.0)	0.35 ± 0.01(4.0)	0.26 ± 0.01(3.0)	0.70 ± 0.03(6.0)
wbc	0.91 ± 0.02(1.0)	1.00 ± 0.01(5.0)	0.92 ± 0.02(2.0)	1.00 ± 0.01(6.0)	0.99 ± 0.01(4.0)	0.98 ± 0.01(3.0)	1.00 ± 0.01(7.0)
yeast	0.86 ± 0.03(4.0)	0.91 ± 0.03(6.0)	0.91 ± 0.02(7.0)	0.81 ± 0.03(3.0)	0.77 ± 0.01(2.0)	0.77 ± 0.02(1.0)	0.88 ± 0.01(5.0)
lympho	1.20 ± 0.00(2.0)	1.20 ± 0.00(5.0)	1.19 ± 0.00(1.0)	1.20 ± 0.00(5.0)	1.20 ± 0.00(5.0)	1.20 ± 0.00(5.0)	1.20 ± 0.00(5.0)
musk	0.20 ± 0.01(1.0)	0.47 ± 0.03(5.0)	0.22 ± 0.01(2.0)	0.48 ± 0.03(7.0)	0.26 ± 0.02(4.0)	0.24 ± 0.01(3.0)	0.48 ± 0.02(6.0)
thyroid	0.48 ± 0.02(2.0)	0.66 ± 0.04(5.0)	0.48 ± 0.02(1.0)	0.80 ± 0.03(7.0)	0.59 ± 0.03(4.0)	0.51 ± 0.03(3.0)	0.76 ± 0.02(6.0)
wine	1.06 ± 0.04(1.0)	1.10 ± 0.01(3.0)	1.06 ± 0.05(2.0)	1.11 ± 0.00(7.0)	1.11 ± 0.00(5.0)	1.11 ± 0.00(6.0)	1.10 ± 0.00(4.0)
vertebral	0.89 ± 0.06(1.0)	0.95 ± 0.02(2.0)	1.00 ± 0.02(3.0)	1.02 ± 0.00(5.0)	1.02 ± 0.00(7.0)	1.02 ± 0.00(6.0)	1.02 ± 0.00(4.0)
Avg. Rank	**2.07**	**5.07**▲	**2.57**△	**5.61**▲	**3.89**△	**3.18**△	**5.61**▲
Avg. Rank (Orig. Space)	**2.00**	**4.75**▲	**3.07**△	**5.57**▲	**3.86**△	**3.11**△	**5.64**▲

Table 6: ***Expert effort* on CLUSTERED DATASETS.** Per dataset rank provided in parentheses (lower is better). Average rank is in the second last row. Average rank for *effort* in $\mathcal{X}$ space also given in last row. Symbols ▲(p<0.01), △(p<0.05) and ▽(p<0.1) denote the cases where OJRANK is significantly better than the baseline w.r.t. Wilcoxon signed rank test.

Dataset	OJRANK	OJR-MO	OJR-ALL	AAD	OMD-Lin	OMD-LLH	Offline
vowels	0.63 ± 0.06(1.0)	0.85 ± 0.03(4.0)	0.97 ± 0.13(7.0)	0.90 ± 0.02(5.0)	0.69 ± 0.02(3.0)	0.67 ± 0.02(2.0)	0.94 ± 0.01(6.0)
optdigits	1.02 ± 0.03(2.0)	1.03 ± 0.01(7.0)	1.03 ± 0.01(4.0)	0.99 ± 0.02(1.0)	1.03 ± 0.00(6.0)	1.03 ± 0.00(5.0)	1.02 ± 0.01(3.0)
letters	0.72 ± 0.06(1.0)	0.89 ± 0.04(4.0)	0.91 ± 0.14(5.0)	0.92 ± 0.04(6.0)	0.84 ± 0.06(2.0)	0.86 ± 0.07(3.0)	0.98 ± 0.01(7.0)
sensor	0.45 ± 0.05(1.0)	0.65 ± 0.06(4.0)	0.72 ± 0.22(5.0)	0.92 ± 0.01(7.0)	0.55 ± 0.05(3.0)	0.52 ± 0.06(2.0)	0.88 ± 0.05(6.0)
segment	0.71 ± 0.14(1.0)	0.83 ± 0.07(3.0)	1.01 ± 0.01(7.0)	0.80 ± 0.03(2.0)	0.88 ± 0.07(4.0)	0.95 ± 0.02(6.0)	0.95 ± 0.03(5.0)
statlog	0.52 ± 0.01(1.0)	0.66 ± 0.03(5.0)	0.54 ± 0.02(2.0)	0.69 ± 0.03(6.0)	0.60 ± 0.02(4.0)	0.56 ± 0.02(3.0)	0.69 ± 0.03(7.0)
vehicle	0.88 ± 0.07(1.0)	0.95 ± 0.04(3.0)	0.99 ± 0.01(7.0)	0.94 ± 0.02(2.0)	0.97 ± 0.02(5.0)	0.97 ± 0.02(4.0)	0.98 ± 0.02(6.0)
svmguide	0.95 ± 0.04(5.0)	0.97 ± 0.01(7.0)	0.95 ± 0.02(6.0)	0.88 ± 0.01(1.0)	0.94 ± 0.01(4.0)	0.92 ± 0.01(3.0)	0.91 ± 0.02(2.0)
Avg. Rank	**1.62**	**4.62**▲	**5.38**▲	**3.75**△	**3.88**▲	**3.50**△	**5.25**▲
Avg. Rank (Orig. Space)	**2.25**	**4.12**▲	**4.38**▽	**4.00**▽	**4.12**▽	**3.88**▽	**5.25**△

***Expert effort*:** Next we analyze the results on *expert effort* on BENCHMARK DATASETS in Table 5 and on CLUSTERED DATASETS in Table 6. The differences between OJRANK and baselines become apparent especially on this metric. Notice that OJRANK yields significantly better *expert effort* than all of the baselines at p<0.05. (See Figure 2 (b) in Section 1 for the actual p-values.)

As given in Def.n 3.3, *expert effort* utilizes similarity in the anomaly scoring space $\mathcal{S}$. Recall that one could also argue for similarity in the original input space $\mathcal{X}$. To this end, we also report (only) the average rank (for brevity) per method across all datasets based on effort utilizing similarity in $\mathcal{X}$ space, shown in the last row of Tables 5 and 6. Here, we observe the same trends on BENCHMARK DATASETS. OJRANK also outperforms all the baselines on CLUSTERED DATASETS at (a slightly higher) p<0.1. The somewhat better effort the baselines achieve in this setup is because they show consecutive instances from the "rare nominal" cluster or from the same larger nominal clusters. This querying of similar instances achieves reduced effort, however at the expense of poor precision (as observed from Table 4).

Overall comparison: The ideal method for on-the-job re-ranking is the one that achieves high precision and enables low effort at the same time. To compare all the methods in both grounds, Figure 2 (a) presents a scatter plot of the avg. rank w.r.t. *precision@b* versus avg. rank w.r.t. *expert effort* for each setup. It is easy to see that OJRANK is closest to the top (denoted by the target symbol on the plots), especially on CLUSTERED DATASETS, where the differences are significant as discussed in the previous subsections.

Response time to update: An important requirement for the kind of applications considered in this work is fast response time; since the expert is to wait between feedbacks to be presented with the updated top-1 instance. In Figure 3, we show the distribution of per-round update time (avg.'ed over 10 iForests) over all rounds with boxplots. For brevity, results for a subset of BENCHMARK DATASETS are shown (others are similar and can be seen in Figure 5 of Supplementary). Moreover, only the state-of-the-art baselines (AAD and OMD) and OJRANK are compared.

The key take-aways are two: OJRANK takes less than one fifth of a second to provide a model update on average – which would be near instantaneous for a human expert. In addition, the update time has low variance from round to round and from dataset to dataset (unlike e.g., AAD).

5.4 Sensitivity Analysis: We conclude experiments with an analysis of OJRANK's sensitivity to its input parameters; scaling factor δ and number of pairs k to sample. We tested over small values of the parameters so as not to stray far away from the original ranked list upon a single feedback. As shown in Figure 4, performance remains nearly stable for most datasets. We use and recommend $\delta = 0.1$ and $k = 5$.

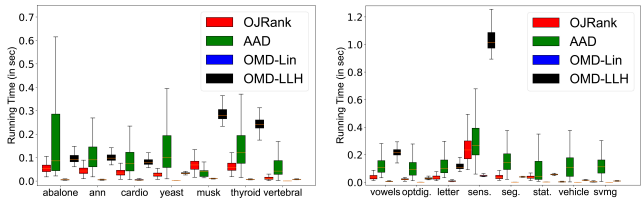


Figure 3: Avg. runtime per update on several (left) BENCHMARK & (right) CLUSTERED datasets. OJRANK's response time is less than one fifth of a second, with low variance.

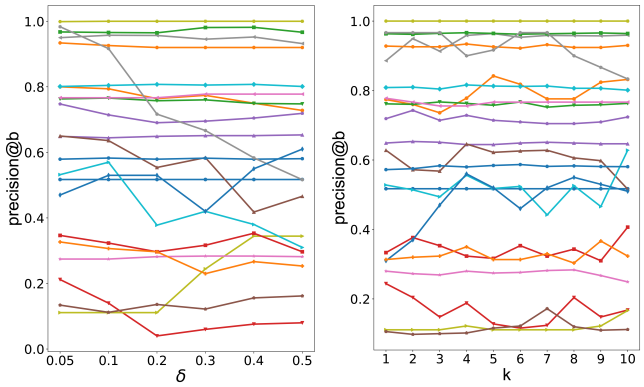


Figure 4: *precision@b* remains reasonably stable upon varying (left) δ and (right) k (2 input parameters to OJRANK). Each line corresponds to one of all 14+8 datasets in Table 2.

6 Conclusion

In this work we addressed the problem of how to leverage the label revealed by an expert on the top-1 instance to quickly re-rank the anomalies in an online fashion. The proposed approach OJRANK works alongside the expert and continues to learn on-the-job from every top-1 feedback. To this end, OJRANK leverages a cross entropy based pairwise learning to rank objective along with accelerated online gradient updates. These updates correspond to a 'more-like-this' strategy on true positive feedback – boosting other similar instances up the list, and a 'less-like-this' strategy on false positive feedback – muting other similar false positives. We show that OJRANK not only increases *precision* but also decreases *expert effort* over two different classes of datasets, and significantly outperforms the offline and state-of-the-art baselines over both metrics. Finally, OJRANK has constant time complexity with instantaneous response time to update, and linear space requirement on the number of instances.

Acknowledgments

This research is sponsored by NSF CAREER 1452425 and IIS1408287. Conclusions expressed in this material are of the authors and do not necessarily reflect the views, expressed or implied, of the funding parties.

References

[1] S. Das, W.-K. Wong, T. Dietterich, A. Fern, and A. Emmott. Incorporating expert feedback into active anomaly discovery. In *IEEE ICDM*, pages 853–858, 2016.

[2] M. A. Siddiqui, A. Fern, T. G. Dietterich, R. Wright, A. Theriault, and D. W. Archer. Feedback-guided anomaly discovery via online optimization. In *KDD*. ACM, 2018.

[3] S. Chaudhuri and A. Tewari. Online ranking with top-1 feedback. In *AISTATS*, pages 129–137, 2015.

[4] S. Chaudhuri and A. Tewari. Online learning to rank with feedback at the top. In *AISTATS*, pages 277–285, 2016.

[5] S. Das, W.-K. Wong, A. Fern, T. G. Dietterich, and M. A. Siddiqui. Incorporating feedback into tree-based anomaly detection. *arXiv preprint arXiv:1708.09441*, 2017.

[6] P. Donmez and J. G. Carbonell. Optimizing estimated loss reduction for active sampling in rank learning. In *ICML*, 2008.

[7] K. Hofmann, S. Whiteson, and M. d. Rijke. Balancing exploration and exploitation in listwise and pairwise online learning to rank for information retrieval. *Inf. Retr.*, 16(1):63–90, 2013.

[8] B. Long, J. Bian, O. Chapelle, Y. Zhang, Y. Inagaki, and Y. Chang. Active learning for ranking through expected loss optimization. *IEEE TKDE*, 27(5):1180–1191, 2015.

[9] R. M. Silva, G. Gomes, M. S. Alvim, and M. A. Gonçalves. Compression-based selective sampling for learning to rank. In *CIKM*. ACM, 2016.

[10] N. Görnitz, M. Kloft, K. Rieck, and U. Brefeld. Toward supervised anomaly detection. *J. Artif. Intell. Res.*, 46, 2013.

[11] N. Nissim, A. Cohen, R. Moskovitch, A. Shabtai, M. Edry, O. Bar-Ad, and Y. Elovici. ALPD: active learning framework for enhancing the detection of malicious pdf files. In *JISIC*, pages 91–98. IEEE, 2014.

[12] D. Pelleg and A. W. Moore. Active learning for anomaly and rare-category detection. In *NIPS*, pages 1073–1080, 2004.

[13] J. He and J. G. Carbonell. Nearest-neighbor-based active learning for rare category detection. In *NIPS*, 2007.

[14] J. He and J. G. Carbonell. Rare class discovery based on active learning. In *ISAIM*, 2008.

[15] R. Ghani and M. Kumar. Interactive learning for efficiently detecting errors in insurance claims. In *KDD*. ACM, 2011.

[16] B. Settles. Active learning. *Synthesis Lectures on AI and ML*, 6(1):1–114, 2012.

[17] T. Pevný. Loda: lightweight on-line detector of anomalies. *Machine Learning*, 102(2):275–304, 2016.

[18] F. T. Liu, K. M. Ting, and Z.-H. Zhou. Isolation forest. In *ICDM*, pages 413–422, 2008.

[19] S. C. Tan, K. M. Ting, and F. T. Liu. Fast anomaly detection for streaming data. In T. Walsh, editor, *IJCAI*. AAAI, 2011.

[20] A. Lazarevic and V. Kumar. Feature bagging for outlier detection. In *KDD*, pages 157–166. ACM, 2005.

[21] F. T. Liu, K. M. Ting, and Z.-H. Zhou. Isolation-based anomaly detection. *TKDD*, 6(1):3, 2012.

[22] K. Wu, K. Zhang, W. Fan, A. Edwards, and P. S. Yu. Rs-forest: a rapid density estimator for streaming anomaly detection. In *ICDM*, pages 600–609. IEEE, 2014.

[23] S. Rayana. ODDS library, 2016. URL: http://odds.cs.stonybrook.edu.