Graph-based Selective Outlier Ensembles

Hamed Sarvari
George Mason University
Fairfax, VA
hsarvari@gmu.edu

Carlotta Domeniconi
George Mason University
Fairfax, VA
cdomenic@gmu.edu

Giovanni Stilo
Sapienza Università di Roma
Rome, Italy
stilo@di.uniroma1.it

ABSTRACT

An ensemble technique is characterized by the mechanism that generates the components and by the mechanism that combines them. A common way to achieve the consensus is to enable each component to equally participate in the aggregation process. A problem with this approach is that poor components are likely to negatively affect the quality of the consensus result. To address this issue, alternatives have been explored in the literature to build selective classifier and cluster ensembles, where only a subset of the components contributes to the computation of the consensus. Of the family of ensemble methods, outlier ensembles are the least studied. Only recently, the selection problem for outlier ensembles has been discussed. In this work we define a new graph-based class of ranking selection methods. A method in this class is characterized by two main steps: (1) Mapping the rankings onto a graph structure; and (2) Mining the resulting graph to identify a subset of rankings. We define a specific instance of the graph-based ranking selection class. Specifically, we map the problem of selecting ensemble components onto a mining problem in a graph. An extensive evaluation was conducted on a variety of heterogeneous data and methods. Our empirical results show that our approach outperforms state-of-the-art selective outlier ensemble techniques.

KEYWORDS

Ensemble methods, Graph Algorithms, Outlier detection, Ranking Selection

ACM Reference Format:
Hamed Sarvari, Carlotta Domeniconi, and Giovanni Stilo. 2019. Graph-based Selective Outlier Ensembles. In *The 34th ACM/SIGAPP Symposium on Applied Computing (SAC '19), April 8–12, 2019, Limassol, Cyprus.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3297280.3297329

1 INTRODUCTION

Ensemble techniques combine the outputs of multiple components to achieve robust and accurate results. One of the challenges in building effective ensembles is the design of the consensus function. In fact, the output of poor components may have a negative impact on the ensemble response. To avoid this drawback, selective classifier and cluster ensembles have been explored in the literature, where only a selected subset of ensemble components contributes to the computation of the consensus. Typically, in classifier ensembles the selection is driven by the trade-off between accuracy and diversity [24]. Boosting, perhaps the most well-known example, achieves the consensus by weighing the components based on their accuracy. In an unsupervised scenario, such as clustering and outlier detection, defining the selection mechanism is more challenging due to the lack of ground truth. Quality and diversity have been used as measure to drive the selection of components for cluster ensembles [16].

Of the family of ensemble methods, outlier ensembles are the least studied [1, 2, 25, 28, 31, 33, 36, 37]. In particular, only recently the selection problem for outlier ensembles has been discussed [31, 33], and its potential positive effect on event detection has been shown [31]. In this work, we further explore the selection issue for outlier ensembles, and define a new *graph-based class* of ranking selection methods, of which we detail specific instances.

To better understand the nature of the problem we want to tackle, let's consider Figure 1. Plots (a)-(f) show six ranking components generated from the WDBC data using LOF (Local Outlier Factor) [9] algorithm under different conditions (see Section 5 for details). Each row corresponds to a ranking. The horizontal axis captures the data points (in a fixed order across all six rows), and the vertical axis measures the LOF scores assigned to each point. The 10 leftmost points are the actual outliers, and the red vertical bars highlight the top-10 LOF score values, in the respective rankings. The four rankings (a)-(d) identify many of the outliers among the top-10 ranked points, while the rankings (e) and (f) have at most one outlier among the top-10 ranked points. Figure 1(g) shows the area under the precision-recall curve (AUCPR) for an ensemble of 20 rankings, of which six are the ones illustrated. Rankings (a)-(d) correspond to the most accurate ones, and (e)-(f) are the two least accurate. As also observed in [31], the best rankings tend to agree on the high scored points, but the actual scores change. As a consequence, their aggregation may produce an improved ranking. On the other hand, rankings (e) and (f) largely rank non-outliers as the highest, and including them in the aggregation process may affect the consensus ranking negatively. We aim at identifying such poor rankings and remove them from further consideration. Our technique, called Core (described in Section 4), was able to select the five top rankings from the ensemble of 20 components given in Figure 1. The consensus ranking achieved via averaging the selected five components gives an AUCPR of 0.8, while the consensus ranking achieved via averaging the entire 20 components gives an AUCPR of 0.2. This result is indicative of the great potential our graph-based approach to selective outlier ensembles has to offer.

The paper is organized as follows. Section 2 discusses related work. We introduce our framework and methodology in Sections 3 and 4, and in Section 5 we present our experiments, results, and analysis. A complexity analysis of the proposed method and of the

SAC '19, April 8–12, 2019, Limassol, Cyprus

ACM ISBN 978-1-4503-5933-7/19/04…$15.00
https://doi.org/10.1145/3297280.3297329

other baselines is provided in Section 6. Section 7 concludes the paper with thoughts for future work.

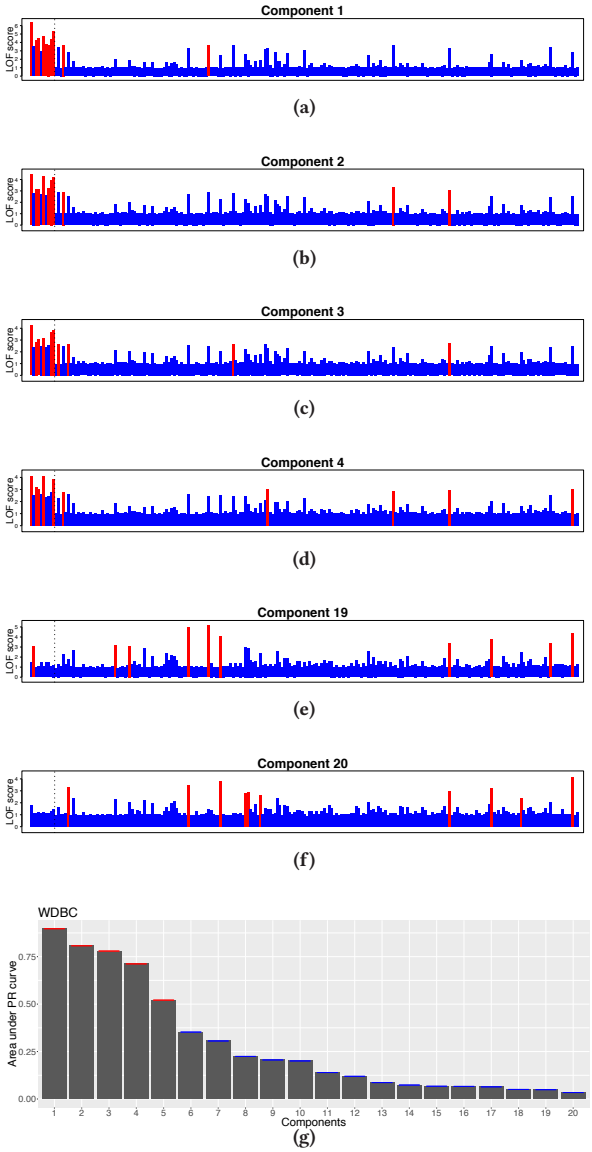


Figure 1: WDBC data set. (a)-(f): Outlier scores generated using LOF [9] for components 1, 2, 3, 4, 19, and 20, respectively; (g): AUCPR for 20 components.

2 RELATED WORK

Ensemble methods have been exploited in the literature to boost the performance of classifiers, e.g. [7, 8, 13], and more recently clustering ensembles have emerged as a technique for overcoming problems with clustering algorithms, e.g. [5, 27, 34]. It is well known that off-the-shelf clustering methods may discover different patterns in a given set of data. This is because each clustering algorithm has its own bias resulting from the optimization of different criteria. Furthermore, there is no ground truth against which the clustering result can be validated. Thus, no cross-validation technique can be carried out to tune input parameters involved in the clustering process. Clustering ensembles offer a solution to challenges inherent to clustering arising from its ill-posed nature: they can provide more robust and stable solutions by making use of the consensus across multiple clustering results, while averaging out emergent spurious structures that arise due to the various biases to which each participating algorithm is tuned, or to the variance induced by different data samples.

Anomaly (or outlier) detection is another unsupervised problem that suffers from many of the same challenges as clustering. As such, many different anomaly detection techniques (e.g., density-based and distance-based, global vs. local), and multiple variations of each have been studied [9, 18, 19, 29, 30]. A comprehensive survey of these methods can be found in [11]. Invariably, all anomaly detection algorithms involve parameters that are problematic to set. Ensemble techniques can provide a framework to address these issues for anomaly detection algorithms, in a way similar to clustering ensembles. Nevertheless, the discussion on outlier ensembles has started only recently [1, 2, 25, 28, 31, 33, 36, 37], and the avenue remains largely unexplored.

In this paper we focus on the problem of component selection for outlier ensembles. As discussed above with the example shown in Figure 1, poor components can negatively affect the consensus ranking. Only recently the selection problem for outlier ensembles has been discussed [31, 33], and its potential positive effect on event detection has been shown [31]. The selection models presented in [33] (DivE) and in [31] (SelectV) are both greedy selective strategies based upon a target ranking treated as pseudo ground-truth. Rankings are selected in sequence if they increase the weighted Pearson correlation of the current ensemble prediction with the target vector. SelectH [31] is also based on a pseudo ground-truth, for anomalies this time. Components that do not rank the estimated anomalies sufficiently high, are candidate to be discarded. To compute the pseudo ground-truth, a mixture modeling approach is used to convert each component ranking into a binary vector. A majority vote applied to these binary lists identifies the anomalies.

In this work, we take a different approach. We tackle the problem of selecting outlier ensemble components by mapping it onto a graph mining problem, which does not use the concept of pseudo ground-truth. The main idea is to capture high quality rankings as nodes forming patterns in a graph. We advocate that such transformation can lead to a family of new effective approaches for the selective outlier ensembles problem. The presentation of our framework follows.

3 SELECTIVE OUTLIER ENSEMBLES FRAMEWORK

Let $X = \{\mathbf{x}_i\}_{i=1}^n$, $\mathbf{x}_i \in \mathbb{R}^d$, be a collection of data. From the data, a collection of outlier score rankings $\{r_j\}_{j=1}^m$ is generated. Each ranking is a sequence of n outlier scores, one for each data point, sorted in non-decreasing order: $r_{j1} \geq r_{j2} \geq \cdots \geq r_{jn}$. The collection $\{r_j\}_{j=1}^m$ constitutes the ensemble.

In Algorithm 1, the **S**elective **Ou**tlier Ensembl**e**s framework (Soul) is presented. The Soul framework takes in input the collection of rankings $\{r_j\}_{j=1}^m$ (however generated), and applies a two-phase algorithm. The first step is the *Ranking Selection* phase, which allows to plug in *any* selection function that specifies which rankings to retain. The *Ranking Aggregation* phase enables the use of a variety of aggregators to compute a consensus ranking r^*. The Soul framework does *not* require access to the original features of the data, and is transparent to the process that generates the ensemble. Soul can therefore be used with any outlier detection algorithm that produces a ranking, and any combination thereof.

The two steps *Ranking Selection* and *Ranking Aggregation* can also be merged to produce the consensus ranking r^*. In this work, though, we focus on the design of an effective *Ranking Selection* function of the components. As such, we make a distinction between the two phases, and apply only commonly used consensus functions, i.e. Maximum, Average, and Minimum [9, 23, 29], for *Ranking Aggregation*.

Algorithm 1 The *Soul* Framework

Require: A collection of rankings $\{r_j\}_{j=1}^m$.
Ensure: A consensus ranking r^*.
1: *Ranking Selection*;
2: *Ranking Aggregation*;
3: **return** r^*;

Algorithm 2 *Core* Ranking Selection

Require: A collection of rankings $\{r_j\}_{j=1}^m$.
Ensure: A subset of rankings $\{r_j^*\}_{j=1}^l$.
1: Construct the complete weighted graph G_c;
2: Derive the pruned graph G;
3: Compute the k-core subgraph of G with largest k;
4: **return** Vertices (rankings) with largest coreness values;

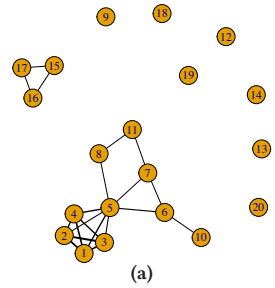
Algorithm 3 *Cull* Ranking Selection

Require: A collection of rankings $\{r_j\}_{j=1}^m$.
Ensure: A subset of rankings $\{r_j^*\}_{j=1}^l$.
1: Construct the complete weighted graph G_c;
2: Compute weighted degrees of each node;
3: Discard nodes with lowest weighted degree;
4: **return** Remaining vertices (rankings);

4 RANKING SELECTION

We define a *graph-based* class of ranking selection methods. A method in this class is characterized by two major steps:

(1) Mapping the rankings onto a graph structure



(a)

	1	2	3	4	...	19	20
1	–	0.92	0.91	0.91	...	0.55	0.38
2	0.92	–	0.93	0.89	...	0.52	0.38
3	0.91	0.93	–	0.90	...	0.53	0.39
4	0.91	0.89	0.90	–	...	0.55	0.41
...	...	...	...	...	–	...	...
19	0.55	0.52	0.53	0.55	...	–	0.58
20	0.38	0.38	0.39	0.41	...	0.58	–

(b)

Figure 2: (a) Core ranking selection: Graph G obtained for the 20 rankings of Figure 1(g). (b) Weighted Kendall tau similarity values between the six rankings given in Figure 1.

(2) Mining the resulting graph to identify a subset of rankings.

$$\{r_j\}_{j=1}^m \xrightarrow{\text{mapping to a graph}} G \xrightarrow{\text{graph mining}} \{r_j^*\}_{j=1}^l$$

Several approaches in the literature formulate the ensemble consensus function as a graph partitioning problem [14, 15, 34]. Our aim here is different, since we are not concerned with the aggregation step. Instead, we organize the components (rankings) in a graph, with the goal of removing poor rankings from further consideration. The challenge is to relate the quality of the rankings with the structure of the graph, in absence of supervision.

We define two instances of the graph-based ranking selection class, called *Core* and *Cull*. In the Core approach, we map the problem of selecting ensemble components onto a community detection problem in a graph. To this end, given the ranking ensemble $\{r_j\}_{j=1}^m$, we construct a complete weighted graph $G_c = (V, W^c)$, where the vertices in V correspond to the rankings, and $|V| = m$. In connecting the vertices with one another, we want the good rankings to form strongly connected components, so that they can emerge as a dense community. Looking at the rankings in Figure 1, we observe that pairs of good rankings are similar in their top ranked objects, while a good and a poor rankings will be dissimilar in the way they rank objects at the top. As such, a similarity measure that emphasizes the top ranked points will in general consider two good rankings as more similar than a good and a poor rankings. The weighted Kendal tau correlation measure [35] is a measure of similarity that satisfies this property. Consider, as an example, Figure 2b, which shows the weighted Kendall tau similarity values between all pairs of rankings

given in Figure 1. We observe high values for all pairs between 1 and 4 (the good rankings), and significantly smaller values for any ranking 1 through 4 and rankings 19 and 20 (the poor rankings). This suggests the following definition of W^c.

For every two distinct rankings r_i and r_j, $W^c_{ij} = K_w(r_i, r_j)$, where K_w is the weighted Kendall tau measure. $W^c_{ii} = 0$, for all i. In order to enable the strongly connected components to emerge, we then prune the edges in G_c by retaining only the edges corresponding to the largest m W^c_{ij} values, where m is equal to the number of vertices. We observe that, a connected graph with m vertices, has at least $m-1$ edges. In pruning the edges we wanted to enable graph connectivity, thus the choice of m as threshold on the number of edges was made. Figure 2a shows the graph G obtained for the ensemble of 20 rankings of Figure 1. Notably, the five best components form a 5-clique in this case, which is also the k-core subgraph for the resulting G, with the largest k ($k = 4$). A k-core subgraph is the maximal connected subgraph of G in which all vertices have degree at least k. Nodes 19 and 20 (the poorest rankings) end up being isolated nodes.

We also observe that the three poor rankings 15, 16, and 17 in Figure 2a form a 3-clique, revealing pair-wise correlations superior to the threshold. Under the assumption that the ranking components are affected by *diverse errors*, cliques of "poor" rankings will stay small, and the k-core subgraph, with the largest k, can identify the subset of rankings of good quality to provide as input to the aggregation function. We call this ranking selection algorithm *Core*, and summarize its steps in Algorithm 2.

The Cull ranking selection technique takes a different approach to prune the complete weighted graph G_c. The Core technique typically retains a minority of the ensemble components (25% on average in our experiments). In an effort of selecting a larger number of (good) components, rather than keeping the components in the largest k-core, we discard the ones deemed as poor, and keep the remaining. To estimate the poor components we proceed as follows. For each vertex v_i in G_c, we compute its weighted degree $d_i = \sum_{j=1}^{m} W^c_{ij}$. Under the assumption that rankings make diverse errors, we expect poor components to have small weights associated to the incident edges, and therefore a low weighted degree. For example, considering the adjacency matrix in Figure 2b, the weighted degrees of the vertices are: $d_1 = 3.67$, $d_2 = 3.64$, $d_3 = 3.66$, $d_4 = 3.66$, $d_{19} = 2.73$, $d_{20} = 2.14$; hence, the poor components (19 and 20) have the lowest weighted degrees. In our experiments we discard 20% of the total number of vertices with the lowest d_i values. We call the resulting algorithm *Cull*. Cull strikes to preserve a larger pool of components in comparison to Core. A summary of the steps is given in Algorithm 3.

Hierarchical versions of both Core and Cull can also be adopted. One can run Core on independent ensembles, and then aggregate all the selected components in a new ensemble, and run Core again on it. We can proceed similarly for Cull. If poor components are sifted at each level, improvements upon the Core (Cull) technique are expected. The depth of the hierarchy can be extended beyond two as well. In our experiments, we test the two-level hierarchy, and call the respective techniques Core^2 and Cull^2.

5 EXPERIMENTS

5.1 Datasets

To evaluate outlier methods, typically, data for classification is used and adapted to the task of anomaly detection. The majority class, or a combination of different large classes, is considered as the inliers. The rest of the data, mostly downsampled, plays the role of outliers. For our experiments, we used datasets from two publicly available repositories. In particular, Lymphography, Shuttle, SpamBase, Waveform, WDBC, Wilt, and WPBC were generated as described in [10][1]; Ecoli4, Pima, Segment0, Yeast2v4, and Yeast05679v4 were generated as described in [3, 4][2]. SatImage is taken from the UCI repository [6]: the majority class is used as inliers, and 0.01 of the rest of the data is subsampled to derive the outliers. A summary of the datasets is available in Table 1.

5.2 Ensemble construction

To evaluate ranking selection algorithms, we first need to construct an ensemble of outlier rankings. We construct homogeneous ensembles, where all components are generated using the LOF algorithm [9] as the base detector. In order to generate diverse components, we perform subsampling. It has been shown that subsampling can create diverse outlier components, and under specific conditions can improve the overall performance, compared to using the entire data [37]. To encourage diversity, for each component, we randomly select the subsampling rate from the (%) values $\{10, 15, 20, 25, 30\}$. We also select the value of the *MinPts* parameter of LOF in the set $\{3, 5, 7, 9\}$. Once the subsample is selected, for each point in the dataset, the nearest neighbors, their distances, and the relative densities required in the LOF algorithm are calculated only with respect to the points in the subsample and using the selected *MinPts* value. In our experiments, we fix the ensemble size to 20.

5.3 Consensus functions

Various methods exist in the literature to unify outlier scores obtained from different base detectors [17, 22], and to merge different rank lists [20, 21]. However, since in our setting each ensemble component is generated by the LOF algorithm, no unification is needed.

Our methods focus on the design of the selection mechanism of the ensemble components. As such, we use simple consensus functions across all approaches being compared, i.e. *Maximum*, *Average*, and *Minimum* functions. The use of more sophisticated aggregation functions is out of scope for the current study, and will be considered in the future. The *Maximum* function assigns to each point the largest score among those received by the various rankings. *Average* and *Minimum* work accordingly in the same fashion. *Maximum* and *Average* are among the most commonly used consensus functions to aggregate rankings [9, 23, 29].

5.4 Methods and Evaluation

We compare our techniques against state-of-the-art selective outlier ensemble methods, namely *SelectV* and *SelectH* (we used the code available from the author's website) [31], and DivE [33] (we used

[1]Data available at: http://www.dbs.ifi.lmu.de/research/outlier-evaluation/DAMI/
[2]Data available at: http://sci2s.ugr.es/keel/imbalanced.php#sub2A

	Ecoli4	Glass	Lymphography	PageBlocks	Pima	SatImage	Segment0	Shuttle	SpamBase	Stamps	Waveform	WDBC	Wilt	WPBC	Yeast05679v4	Yeast2v4
Instances	314	214	148	5,473	510	1,072	2,308	1,013	2,528	340	3,433	357	4,839	198	528	514
Attributes	8	7	47	10	8	37	20	9	59	9	21	32	5	35	8	8
Outliers %	6	4	4	10	2	3	14	1	2	9	3	3	5	23	10	10

Table 1: Characteristics of the datasets for outlier detection used in the experiments.

the implementation provided by the authors of [31]). The code of Core and Cull is publicly available[3]. In experiments, we also include the baseline that selects all the components (called *All*). Moreover, to assess the effectiveness of the ensemble, we apply the simple LOF algorithm [9] on the whole data. We ran all methods (Core, Cull, SelectV, SelectH, DivE, and All) on multiple independent ensembles, and report the average performance of each method. The LOF baseline was applied the same number of times on each data set, with a random choice of the *MinPts* parameter from the set $\{3, 5, 7, 9\}$. Performance is measured using the area under the Precision-Recall curve, namely *average precision* [32]. This measure was also used by the authors of *SelectV* and *SelectH* to assess their methods [31]. We observe that, although the area under ROC curve is widely used to evaluate outlier detection methods [10], it has been shown that the Precision-Recall curve is more informative than ROC plots when evaluating imbalanced datasets [32].

To run the hierarchical version of Core and Cull (Core^2 and Cull^2, respectively), we consider batches of 20 ensembles. For each batch, we run Core (Cull) on each ensemble; we then assemble the outputs of 20 selections in a new ensemble, and run Core (Cull) again on it. The process is repeated for multiple independent batches of 20 ensembles, and average performance is reported for each method. We also run a variant in which, for each batch of 20 ensembles, we just assemble the 20 outputs of Core (Cull) in a new ensemble and directly apply the consensus function. These variants are called Core.U and Cull.U.

For a fair comparison, we also set up runs of SelectV, SelectH, DivE, and All, where we enable the techniques to have access to all the ensembles in each batch. That is, we generate a single ensemble of $20 \times 20 = 400$ components from a given batch, and run each competitor method on it. The techniques in this setting are denoted as SelectV.U, SelectH.U, DivE.U, and All.U, respectively.

5.5 Results

Table 2 gives the average performances (areas under the PR curve) of Core, Cull, DivE, SelectV, SelectH, and All across all datasets and for the three consensus functions. For WDBC, WPBC, Pima, Yeast05679v4, Ecoli4, Shuttle, and SpamBase averages are computed over 400 independent ensembles. For the remaining datasets averages are computed over 200 independent ensembles.

Table 3 gives the average performances (areas under the PR curve) of Core^2, Cull^2, Core.U, Cull.U, DivE.U, SelectV.U, SelectH.U, and All.U across all datasets and for the three consensus functions. For WDBC, WPBC, Pima,Yeast05679v4, Ecoli4, Shuttle, and SpamBase averages are computed over 20 batches (of 20 ensembles each). For the remaining datasets, averages are computed over 10 batches.

For both tables, statistical significance is assessed using a one-way ANOVA with a post-hoc Tukey HSD test with a p-value threshold equal to 0.01. For each dataset, boldface indicates the technique with the best performance score, and any technique which is *not* statistically significantly inferior to it. For each dataset, the best performance score is also underlined.

5.6 Analysis

Table 2 shows that, out of the 16 datasets, Core and Cull are ranked among the top performers in 12 datasets; SelectV and SelectH in 6; DivE and All in 9, and LOF in 3. Overall, our selective techniques are superior against the state-of-the-art approaches for selective outlier ensembles (SelectV, SelectH, and DivE), and against All. In particular, Core and Cull give the *best performance scores* (underlined values) in 10 datasets; DivE in 1; SelectH and SelectV in 2; All in 3; and LOF in 3. Core and Cull win by a large margin.

It's known that the All technique, especially when combined with average, is a strong baseline and hard to defeat [12]. Our results confirm this fact. In particular, SelectV and SelectH are not competitive against All on the wide range of problems considered in our experiments. We also observe that DivE often selects all the components, and therefore reduces to All. Core and Cull emerge as the strongest competitors against All. Single LOF is among the top performers in only three cases; this supports the overall effectiveness of the constructed ensemble. It's interesting to observe that in two out of these three cases (Glass and Segment0), LOF is the only top performer, indicating that the constructed ensemble did not work well for these two problems, regardless of the selective or consensus techniques used.

The results reveal an interesting fact about Core and Cull: they manifest their best behavior under different scenarios. They are among the best performing methods in 7 and 11 datasets, respectively, of which only 6 are in common. On the other hand, SelectH and SelectV seem highly correlated. They both become competitive in the same 6 datasets. The complementary nature of Core and Cull enables them to succeed in a wide range of problems, since they seem to induce a different learning bias. This opens a new research path to investigate characteristics of the datasets to which each of these methods is tuned, and suggests the potential for a hybrid approach that leverages their diversity.

Table 3 shows that Core^2 and Cull^2 are ranked among the top performers in 15 datasets; Core.U and Cull.U in 15 as well; DivE.U in 4; SelectV.U and SelectH.U in 12; and All.U in 12. Again, the

[3]Code available at: https://github.com/HamedSarvari/Graph-Based-Selective-Outlier-Ensembles

DataSet \ Method	Core			Cull			Dive			SelectH			SelectV			All			Lof
	avg.	max.	min.	avg.	max.	min.	avg.	max.	min.	avg.	max.	min.	avg.	max.	min.	avg.	max.	min.	avg.
Ecoli4	**0.133**	**0.135**	0.125	**0.127**	0.124	0.108	0.123	0.115	0.094	0.123	0.114	0.093	0.121	0.112	0.094	0.123	0.114	0.093	0.053
Glass	0.115	0.115	0.117	0.125	0.123	0.107	0.131	0.113	0.098	0.098	0.086	0.072	0.098	0.085	0.077	0.131	0.114	0.097	**0.155**
Lymphography	0.311	0.294	0.321	0.585	0.379	**0.651**	0.567	0.368	0.627	0.291	0.266	0.32	0.322	0.288	0.331	0.348	0.3	0.365	0.475
PageBlocks	0.413	0.392	0.366	**0.428**	0.383	0.313	**0.428**	0.375	0.272	**0.426**	0.373	0.272	**0.423**	0.373	0.276	**0.428**	0.375	0.272	0.252
Pima	**0.028**	0.028	0.027	**0.027**	**0.028**	0.026	**0.027**	**0.027**	0.024	0.026	0.026	0.024	0.026	0.025	0.024	**0.027**	0.027	0.024	0.02
SatImage	**0.514**	**0.521**	0.472	0.505	**0.526**	0.36	0.479	**0.523**	0.231	0.464	0.502	0.23	0.46	0.5	0.232	0.486	**0.526**	0.232	0.21
Segment0	0.104	0.105	0.108	0.103	0.106	0.111	0.103	0.108	0.113	0.104	0.108	0.113	0.104	0.109	0.113	0.103	0.108	0.113	**0.118**
Shuttle	0.157	0.157	0.139	**0.168**	0.16	0.129	**0.17**	0.152	0.134	0.151	0.138	0.131	0.126	0.123	0.124	**0.17**	0.152	0.134	0.113
SpamBase	0.088	0.092	0.076	0.091	0.095	0.068	0.094	0.096	0.064	0.126	**0.129**	0.076	0.125	**0.128**	0.079	0.094	0.096	0.064	0.072
Stamps	0.081	0.077	0.088	0.087	0.077	**0.104**	0.089	0.072	**0.109**	0.076	0.061	0.093	0.077	0.062	0.089	0.089	0.073	**0.11**	0.098
Waveform	0.115	**0.13**	0.098	0.105	0.123	0.081	0.099	0.115	0.071	0.105	0.117	0.082	0.101	0.117	0.073	0.099	0.115	0.071	0.062
WDBC	**0.815**	**0.8**	0.798	**0.815**	**0.8**	0.798	**0.814**	0.796	0.747	0.752	0.732	0.688	0.749	0.735	0.716	**0.814**	0.796	0.748	0.514
Wilt	0.075	0.063	**0.084**	0.076	0.059	**0.085**	0.078	0.058	**0.083**	0.076	0.058	**0.083**	0.078	0.058	**0.084**	0.078	0.058	**0.084**	0.071
WPBC	**0.226**	**0.226**	**0.224**	**0.225**	**0.225**	0.222	**0.224**	**0.225**	0.219	0.224	**0.224**	0.222	0.223	**0.224**	0.219	**0.224**	**0.225**	0.219	0.21
Yeast05679v4	0.132	0.131	0.132	0.133	0.132	**0.134**	**0.134**	0.133	**0.136**	0.133	0.132	**0.136**	0.133	0.132	**0.135**	**0.134**	0.133	**0.136**	**0.137**
Yeast2v4	0.207	0.22	0.187	0.19	0.219	0.158	0.186	0.206	0.153	0.228	**0.239**	0.188	0.224	**0.234**	0.188	0.186	0.208	0.154	0.147

Table 2: Average performance (area under the PR curve) for all methods and datasets (no hierarchy).

hierarchical versions of our techniques emerge as the strongest competitors. In particular, Core^2 and Cull^2 give the *best performance scores* (underlined values) in 7 datasets; Core.U and Cull.U in 5; DivE.U in 1; SelectH.U and SelectV.U in 2; and All.U in 5.

The two-level pruning mechanisms of Core^2 and Cull^2 effectively prune poor components among a large pool of rankings. On the other hand, All.U deals with large ensembles, which are likely to contain a fair number of poor components, thus hurting the relative performance against the competitors. Overall, the behavior of SelectV.U and SelectH.U is comparable to All.U, while DivE.U gives the worst performance.

An insightful observation from the results in Table 3 is the strong performance of Core.U. It's among the best performers in 14 (out of 16) datasets, and its overall performance is superior to Core^2. In a way, Core.U achieves the best-of-both-worlds: it first uses Core to discard poor components across different ensembles; then it aggregates all selected rankings, acting like All, but on a "boosted" pool of components. Core.U is superior to (or tied with) All.U in almost all scenarios (15 out of 16), and therefore a very promising candidate for outlier ensemble selection.

We finally observe that the best performing consensus functions depend on the dataset, and to a less extent on the method. A deeper understanding of this behavior is in our agenda for future work.

6 COMPLEXITY ANALYSIS

We analyze the theoretical complexity for all the considered methods. For simplicity, we omit the cost of sampling, the cost needed to compute the anomaly scores, and the cost of running the aggregation function. These steps are common to all methods. Let n be the size of each ranking and m the size of the ensemble.

- *All*: The cost of selecting all the components is simply equal to the size of the ensemble: $\mathbf{m}$.
- *Core and Cull*: The cost of the graph construction is $m \cdot m \cdot n \cdot log(n)$, obtained by multiplying the number of edges with the cost of computing the weighted tau, which is $n \cdot log(n)$ as reported in [35]. The rest of the computation is linear in the number of edges, which is $m \cdot m$. So the total cost is: $(\mathbf{m} \cdot \mathbf{m} \cdot \mathbf{n} \cdot \mathbf{log(n)}) + (m \cdot m)$, which is dominated by the factor $\mathbf{n} \cdot \mathbf{log(n)} \cdot \mathbf{m^2}$.
- *DivE*: As reported in [33], the first operation performed by DivE is the *Union* of the top-k outliers, with a cost of $n \cdot m$ when $k = n$. The next step consists in sorting the converted rankings using the weighted Pearson correlation, with a cost of $m \cdot n + m \cdot log(m)$. This includes the cost of m Pearson's coefficients ($m \cdot n$), and the cost of sorting the m rankings according to the elaborated coefficients. The computation of sorting the converted rankings using the weighted Pearson correlation is repeated two times before the loop that contain the m rankings. As a result, the overall cost of DivE is: $n \cdot m + ((\mathbf{m} + \mathbf{2}) \cdot (\mathbf{m} \cdot \mathbf{n} + \mathbf{m} \cdot \mathbf{log(m)}))$, which it is dominated by the factor $\mathbf{n} \cdot \mathbf{m^2}$.
- *Select-V*: As reported in [31], the first operation performed by SelectV is *Unification*, which converts scores to probability estimates. Even if we consider as constant the cost of Unification, the total cost of performing Unification for all rankings is $n \cdot m$. The cost of rank sorting is $n \cdot log(n)$. The next step consists in sorting the converted rankings using the weighted Pearson correlation, with a cost of $m \cdot n + m \cdot log(m)$, exactly as in DivE. The computation of sorting the converted rankings using the weighted Pearson correlation is repeated $m + 1$ times, and the final running cost to perform SelectV is: $n \cdot m + n \cdot log(n) + ((\mathbf{m} + \mathbf{1}) \cdot (\mathbf{m} \cdot \mathbf{n} + \mathbf{m} \cdot \mathbf{log(m)}))$, which it is dominated by the factor $\mathbf{n} \cdot \mathbf{m^2}$.
- *Select-H*: As reported in [31], the first expensive procedure performed is the computation of *MixtureModel*. Its cost depends on the number of iterations i, which was set to 100 as suggested in [31], on the size m of the ensemble, and on the length n of the score vectors; the resulting cost is $i \cdot n \cdot m$. The second expensive procedure is *RobustRankAggregation*, which costs $n \cdot m$. The subsequent loop is dominated by the number of estimated outliers, and in the worst case its cost is $n^2 \cdot log(n)$. The algorithm concludes with a clustering phase (k-means [26] is the used algorithm), which costs $2 \cdot m \cdot n$. The final running cost to perform SelectH is: $(i \cdot n \cdot m) + (n \cdot m) + (\mathbf{n^2} \cdot \mathbf{log(n)}) + (2 \cdot m \cdot n)$, which is dominated by the factor $\mathbf{n^2} \cdot \mathbf{log(n)}$.

Method / DataSet	Core^2 avg.	Core^2 max.	Core^2 min.	Cull^2 avg.	Cull^2 max.	Cull^2 min.	Core.U avg.	Core.U max.	Core.U min.	Cull.U avg.	Cull.U max.	Cull.U min.
Ecoli4	**0.14**	**0.141**	**0.131**	**0.13**	**0.131**	**0.116**	**0.135**	**0.139**	**0.128**	**0.129**	**0.123**	0.089
Glass	**0.109**	**0.115**	**0.116**	**0.119**	**0.136**	**0.102**	**0.117**	**0.105**	**0.118**	**0.126**	**0.111**	0.076
Lymphography	0.28	0.269	0.289	0.527	0.282	**0.692**	0.31	0.27	0.336	**0.621**	0.293	**0.69**
PageBlocks	**0.416**	0.391	**0.399**	**0.437**	0.356	0.303	**0.436**	0.368	0.323	**0.439**	0.354	0.261
Pima	**0.03**	**0.032**	**0.03**	0.027	0.029	0.025	0.028	**0.03**	0.026	0.028	0.028	0.022
SatImage	**0.524**	**0.532**	**0.486**	**0.519**	**0.535**	0.332	**0.522**	**0.535**	0.413	**0.514**	**0.551**	0.224
Segment0	0.103	0.104	**0.108**	0.102	0.103	**0.11**	0.102	0.102	**0.11**	0.102	0.104	**0.114**
Shuttle	**0.158**	**0.158**	0.141	**0.17**	**0.171**	0.138	**0.165**	**0.172**	0.143	**0.172**	**0.158**	0.135
SpamBase	0.084	0.096	0.071	0.09	0.098	0.059	0.089	0.094	0.064	0.091	0.106	0.055
Stamps	0.071	0.076	0.088	0.08	0.076	**0.1**	0.077	0.068	**0.108**	0.093	0.078	**0.119**
Waveform	0.108	0.121	0.096	0.11	**0.146**	0.079	0.114	**0.159**	0.087	0.105	0.13	0.067
WDBC	**0.815**	**0.806**	**0.803**	**0.815**	**0.806**	**0.803**	**0.815**	**0.798**	**0.768**	**0.815**	**0.798**	**0.768**
Wilt	**0.079**	0.065	**0.091**	**0.077**	0.054	**0.089**	**0.075**	0.053	**0.092**	**0.077**	0.053	**0.089**
WPBC	**0.23**	**0.232**	**0.228**	**0.226**	0.224	0.224	**0.227**	**0.227**	**0.225**	**0.225**	**0.228**	0.224
Yeast05679v4	**0.131**	**0.131**	**0.132**	**0.132**	**0.133**	**0.131**	**0.132**	**0.133**	**0.132**	**0.133**	**0.136**	**0.13**
Yeast2v4	**0.255**	**0.268**	**0.246**	0.195	**0.235**	0.157	0.207	**0.241**	0.171	0.191	**0.22**	0.147

Method / DataSet	Dive.U avg.	Dive.U max.	Dive.U min.	SelectH.U avg.	SelectH.U max.	SelectH.U min.	SelectV.U avg.	SelectV.U max.	SelectV.U min.	All.U avg.	All.U max.	All.U min.
Ecoli4	0.097	0.097	0.097	**0.125**	0.107	0.076	**0.126**	**0.115**	0.08	**0.125**	0.107	0.076
Glass	0.106	0.106	0.106	0.1	0.077	0.053	0.1	0.083	0.052	**0.134**	0.094	0.07
Lymphography	0.444	0.444	0.444	0.288	0.256	0.336	0.325	0.261	0.314	0.353	0.277	0.362
PageBlocks	0.318	0.318	0.318	**0.435**	0.32	0.214	**0.432**	0.327	0.217	**0.437**	0.322	0.214
Pima	0.024	0.024	0.024	0.027	0.026	0.02	0.027	0.026	0.022	0.028	0.027	0.02
SatImage	0.429	0.429	0.429	**0.469**	**0.526**	0.156	0.453	**0.508**	0.176	**0.492**	**0.551**	0.156
Segment0	**0.115**	**0.115**	**0.115**	0.102	**0.108**	**0.112**	0.103	**0.108**	**0.114**	0.102	**0.108**	**0.112**
Shuttle	0.118	0.118	0.118	**0.157**	0.127	0.135	0.124	0.123	0.122	**0.173**	0.135	0.143
SpamBase	0.081	0.081	0.081	**0.132**	**0.138**	0.057	**0.132**	**0.131**	0.059	0.095	0.113	0.05
Stamps	0.079	0.079	0.079	0.085	0.055	**0.114**	0.082	0.057	**0.099**	**0.099**	0.066	**0.137**
Waveform	0.093	0.093	0.093	0.11	**0.141**	0.069	0.1	**0.14**	0.063	0.099	**0.136**	0.06
WDBC	**0.773**	**0.773**	**0.773**	0.755	0.726	0.655	0.751	0.737	0.708	**0.815**	**0.791**	0.731
Wilt	0.072	0.065	**0.077**	**0.078**	0.051	**0.089**	**0.079**	0.052	**0.09**	**0.079**	0.051	**0.09**
WPBC	0.219	**0.226**	0.203	**0.226**	**0.227**	0.223	0.224	**0.228**	0.218	**0.225**	0.224	0.216
Yeast05679v4	0.126	0.126	0.126	**0.133**	**0.136**	**0.133**	**0.132**	**0.137**	**0.131**	**0.134**	**0.137**	**0.133**
Yeast2v4	0.186	0.186	0.186	**0.233**	**0.221**	0.161	**0.232**	**0.234**	0.166	0.187	0.197	0.137

Table 3: Average performance (area under the PR curve) for all methods and datasets (with hierarchy).

The theoretical analysis shows that SelectH is dominated by n^2, our methods Core and Cull by $n \cdot log(n)$, and DivE and SelectV by n. In real world scenarios, as the number of data grows large, SelectH may become prohibitively expensive.

We have also computed the empirical running times. For each dataset, the average running time of all the runs for each method is recorded in Table 4. Experiments were run on a laptop with an Intel Core i7 Processor @2.80GHz and 16GB RAM. The empirical running times are consistent with the complexity analysis given above. In particular, we observe that the running times of DivE and SelectV have the same order of magnitude. Core and Cull are faster then SelectH but slower than DivE and SelectV. When the number of components increases, the running times of Core.U, Cull.U, Dive.U, and SelectV.U are almost identical; in contrast, the runnning time of SelectH.U increases much more rapidly with respect to the other methods.

7 CONCLUSION

We have introduced a new graph-based class of ranking selection methods for outlier ensembles. In particular, we have defined two

	Core/Cull	Dive	SelectH	SelectV	Core-U/Cull-U	DiveU	SelectHU	SelectVU
Ecoli4	0.1955	0.0407	0.3507	0.0139	3.7343	2.4337	9833.1550	1.8934
Glass	0.1202	0.0310	0.2313	0.0124	5.1370	2.2313	6139.0900	1.6363
Lymphography	0.0918	0.0228	0.2686	0.0107	37.3748	2.1586	1798.3125	0.7797
PageBlocks	3.5285	0.1021	7.7770	0.1160	98.0259	18.2131	134998.5714	11.9919
Pima	0.3028	0.0286	0.9487	0.0205	5.3593	2.8979	14137.0000	2.5817
SatImage	0.7446	0.0323	1.2203	0.0353	7.4759	4.0789	30714.4000	3.4625
Segment0	1.5537	0.0691	3.3306	0.0481	20.7469	6.0357	64577.3000	5.9603
Shuttle	0.6247	0.0346	1.1076	0.0214	9.6138	3.9340	27573.0500	3.0115
SpamBase	1.9090	0.0456	4.8094	0.0448	14.6854	7.1720	71212.8333	6.4726
Stamps	0.1834	0.0265	0.3725	0.0136	1.5201	2.3445	9135.5000	1.8553
Waveform	2.3860	0.0553	6.5074	0.0693	73.8695	9.1355	103188.4444	9.5774
WDBC	0.2162	0.0330	0.3878	0.0138	1.4657	2.5056	10659.8000	1.8951
Wilt	3.3298	0.0648	4.6988	0.0887	29.2481	13.7465	117418.8889	10.7646
WPBC	0.1116	0.0647	1.9563	0.0175	1.7298	2.6575	3197.7000	1.0396
Yeast2v4	0.3031	0.0373	0.5683	0.0152	3.7115	2.6427	14770.1000	2.2619
Yeast05679v4	0.3110	0.0538	0.5941	0.0154	3.8485	2.8501	15323.2500	2.3146
Average	0.2532	0.0473	0.4724	0.0146	3.7914	2.6419	12578.2025	2.1040

Table 4: Running times of experiments in Table 3 expressed in seconds.

specific approaches, Core and Cull, and hierarchical extensions of the same. Our extensive evaluation on a variety of heterogeneous data and methods shows that our approach outperforms state-of-the-art selective outlier ensemble techniques in a number of cases. Interesting and challenging questions are open for future investigation, including a characterization of the scenarios when Core outperforms Cull, or vice versa; studying how our selective techniques affect the accuracy/diversity tradeoffs; exploring hybrid methods, different outlier detection techniques and alternative consensus functions, and analyze their effects in more depth.

ACKNOWLEDGMENT

This work was supported in part by the MIUR under grant "Dipartimenti di eccellenza 2018-2022" of the Department of Computer Science of Sapienza University.

REFERENCES

[1] Charu C. Aggarwal. 2013. Outlier Ensembles: Position Paper. *SIGKDD Explor. Newsl.* 14, 2 (April 2013), 49–58. https://doi.org/10.1145/2481244.2481252

[2] Charu C. Aggarwal and Saket Sathe. 2015. Theoretical Foundations and Algorithms for Outlier Ensembles. *SIGKDD Explor. Newsl.* 17, 1 (Sept. 2015), 24–47. https://doi.org/10.1145/2830544.2830549

[3] Jesús Alcalá-Fdez, Alberto Fernández, Julián Luengo, Joaquín Derrac, Salvador García, Luciano Sánchez, and Francisco Herrera. 2011. Keel data-mining software tool: data set repository, integration of algorithms and experimental analysis framework. *Journal of Multiple-Valued Logic & Soft Computing* 17 (2011).

[4] Jesús Alcalá-Fdez, Luciano Sanchez, Salvador Garcia, Maria Jose del Jesus, Sebastian Ventura, Josep Maria Garrell, Jose Otero, Cristóbal Romero, Jaume Bacardit, Victor M Rivas, et al. 2009. KEEL: a software tool to assess evolutionary algorithms for data mining problems. *Soft Computing-A Fusion of Foundations, Methodologies and Applications* 13, 3 (2009), 307–318.

[5] Steffen Bickel and Tobias Scheffer. 2004. Multi-view clustering.. In *ICDM*, Vol. 4. 19–26.

[6] Catherine L Blake and Christopher J Merz. 1998. UCI Repository of machine learning databases [http://www. ics. uci. edu/˜ mlearn/MLRepository. html]. Irvine, CA: University of California. *Department of Information and Computer Science* 55 (1998).

[7] Leo Breiman. 1996. Bagging predictors. *Machine learning* 24, 2 (1996), 123–140.

[8] Leo Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5–32.

[9] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. 2000. LOF: identifying density-based local outliers. In *ACM sigmod record*, Vol. 29. ACM, 93–104.

[10] Guilherme O Campos, Arthur Zimek, Jörg Sander, Ricardo JGB Campello, Barbora Micenková, Erich Schubert, Ira Assent, and Michael E Houle. 2016. On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study. *Data Mining and Knowledge Discovery* 30, 4 (2016), 891–927.

[11] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2007. Outlier detection: A survey. *Comput. Surveys* (2007).

[12] Alvin Chiang and Yi-Ren Yeh. 2015. Anomaly detection ensembles: In defense of the average. In *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2015 IEEE/WIC/ACM International Conference on*, Vol. 3. IEEE, 207–210.

[13] Thomas G Dietterich et al. 2000. Ensemble methods in machine learning. *Multiple classifier systems* 1857 (2000), 1–15.

[14] Carlotta Domeniconi and Muna Al-Razgan. 2009. Weighted Cluster Ensembles: Methods and Analysis. *ACM Trans. Knowl. Discov. Data* 2, 4, Article 17 (Jan. 2009), 40 pages. https://doi.org/10.1145/1460797.1460800

[15] Xiaoli Zhang Fern and Carla E. Brodley. 2004. Solving Cluster Ensemble Problems by Bipartite Graph Partitioning. In *Proceedings of the Twenty-first International Conference on Machine Learning (ICML '04)*. ACM, New York, NY, USA, 36–. https://doi.org/10.1145/1015330.1015414

[16] Xiaoli Z. Fern and Wei Lin. 2008. Cluster Ensemble Selection. *Stat. Anal. Data Min.* 1, 3 (Nov. 2008), 128–141. https://doi.org/10.1002/sam.v1:3

[17] Jing Gao and Pang-Ning Tan. 2006. Converting output scores from outlier detection algorithms into probability estimates. In *Data Mining, 2006. ICDM'06. Sixth International Conference on*. IEEE, 212–221.

[18] Wen Jin, Anthony KH Tung, and Jiawei Han. 2001. Mining top-n local outliers in large databases. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 293–298.

[19] Wen Jin, Anthony KH Tung, Jiawei Han, and Wei Wang. 2006. Ranking Outliers Using Symmetric Neighborhood Relationship.. In *PAKDD*, Vol. 6. Springer, 577–593.

[20] John G Kemeny. 1959. Mathematics without numbers. *Daedalus* 88, 4 (1959), 577–591.

[21] Raivo Kolde, Sven Laur, Priit Adler, and Jaak Vilo. 2012. Robust rank aggregation for gene list integration and meta-analysis. *Bioinformatics* 28, 4 (2012), 573–580.

[22] Hans-Peter Kriegel, Peer Kroger, Erich Schubert, and Arthur Zimek. 2011. Interpreting and unifying outlier scores. In *Proceedings of the 2011 SIAM International Conference on Data Mining*. SIAM, 13–24.

[23] Aleksandar Lazarevic and Vipin Kumar. 2005. Feature bagging for outlier detection. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. ACM, 157–166.

[24] Nan Li and Zhi-Hua Zhou. 2013. Selective Ensemble of Classifier Chains. In *Proceedings of the International Workshop on Multiple Classifier Systems*. Springer, 146–156.

[25] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation Forest. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining (ICDM '08)*. IEEE Computer Society, Washington, DC, USA, 413–422. https://doi.org/10.1109/ICDM.2008.17

[26] S. Lloyd. 2006. Least Squares Quantization in PCM. *IEEE Trans. Inf. Theor.* 28, 2 (Sept. 2006), 129–137. https://doi.org/10.1109/TIT.1982.1056489

[27] Emmanuel Muller, Stephan Gunnemann, Ines Farber, and Thomas Seidl. 2012. Discovering multiple clustering solutions: Grouping objects in different views of the data. In *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*. IEEE, 1207–1210.

[28] Hoang Vu Nguyen, Hock Hee Ang, and Vivekanand Gopalkrishnan. 2010. Mining Outliers with Ensemble of Heterogeneous Detectors on Random Subspaces. In *Proceedings of the 15th International Conference on Database Systems for Advanced Applications - Volume Part I (DASFAA'10)*. Springer-Verlag, Berlin, Heidelberg, 368–383. https://doi.org/10.1007/978-3-642-12026-8_29

[29] Spiros Papadimitriou, Hiroyuki Kitagawa, Phillip B Gibbons, and Christos Faloutsos. 2003. Loci: Fast outlier detection using the local correlation integral. In *Data Engineering, 2003. Proceedings. 19th International Conference on*. IEEE, 315–326.

[30] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. 2000. Efficient algorithms for mining outliers from large data sets. In *ACM Sigmod Record*, Vol. 29. ACM, 427–438.

[31] Shebuti Rayana and Leman Akoglu. 2015. Less is more: Building selective anomaly ensembles with application to event detection in temporal graphs. In *Proceedings of the 2015 SIAM International Conference on Data Mining*. SIAM, 622–630.

[32] Takaya Saito and Marc Rehmsmeier. 2015. The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. *PloS one* 10, 3 (2015), e0118432.

[33] R. Schubert, R. Wojdanowski, A. Zimek, and H.-P. Kriegel. 2012. On evaluation of outlier rankings and outlier scores. In *Proceedings of the SIAM International Conference on Data Mining*. SIAM, 1047–1058.

[34] Alexander Strehl and Joydeep Ghosh. 2002. Cluster ensembles—a knowledge reuse framework for combining multiple partitions. *Journal of machine learning research* 3, Dec (2002), 583–617.

[35] S. Vigna. 2015. A weighted correlation index for rankings with ties. In *Proceedings of the 24th international conference on World Wide Web*. ACM, 1166–1176.

[36] Arthur Zimek, Ricardo JGB Campello, and Jörg Sander. 2014. Ensembles for unsupervised outlier detection: challenges and research questions a position paper. *ACM Sigkdd Explorations Newsletter* 15, 1 (2014), 11–22.

[37] Arthur Zimek, Matthew Gaudet, Ricardo JGB Campello, and Jörg Sander. 2013. Subsampling for efficient and effective unsupervised outlier detection ensembles. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 428–436.