Designing a Stability Metric for Assessing the Robustness of Anomaly Rankings

Connor Galvin, Lorenzo Perini, and Vincent Vercruyssen

Dept. of Computer Science, KU Leuven, Belgium
`firstname.lastname@cs.kuleuven.be`

Abstract. The idea of stability has applications in many areas of machine learning. However, stability has not yet been applied in context of anomaly detection, where there is need for a metric that quantifies the robustness of anomaly score rankings to changes in training data. We propose such a metric and the methodology used in computing it. We then propose to use an algorithm with the goal of maximizing stability by learning training points that contribute most to high stability. Finally, we apply this stability metric and the proposed contribution update algorithm on several benchmark datasets. This evaluation is used to compare stability on different anomaly detection algorithms, and assess the contribution update algorithm's ability to increase stability.

Keywords: stability · anomaly detection.

1 Introduction

Anomaly detection seeks to find points that do not conform to normal behavior in a dataset. Anomaly detection has many applications in areas such as fraud detection, medical disease detection, and cyber security [2]. In this paper we are primarily concerned with unsupervised anomaly detection, which involves training an algorithm on an unlabeled training set. The performance of an unsupervised anomaly detection algorithm is only as good as the data that it was trained on. Since we often lack data from the full population, anomaly detection algorithms are typically executed on a random sample, leading to uncertainty in the selection of the training data. If small changes in the subsampled training data lead to drastic changes in the anomaly rankings of the test data, we become uncertain that the anomaly rankings and corresponding anomaly labels of the test data are truly representative. Ideally we would have a stable anomaly detection algorithm, where we can be confident that the generated anomaly rankings will be similar, independent of what subsample of data they were trained upon. The goal of this paper is to define a single metric that quantifies the stability of an anomaly detection algorithm on a given dataset.

The second contribution of this paper will be geared towards designing an algorithm that updates sample weights using multiple iterations with the goal of increasing stability. The idea is to selectively put more emphasis on training points that will yield a more stable anomaly detection model. In doing this, we can identify patterns in the training data that correspond to higher stability of anomaly detection algorithms.

2 Context and related work

The idea of stability has been applied to other areas of machine learning, but to our knowledge there has been no application of stability to assess the robustness of anomaly rankings. In particular [4] proposed point stability with respect to clustering. The idea proposed here is that clustering performed on subsets of data may not be representative of the clustering of the entire dataset. If two points belong to the same cluster, executed on some subset of data, it does not mean that they would belong to the same cluster if the full dataset was clustered. In a similar way we look into point stability for anomaly rankings. How much does the subset an anomaly detection algorithm was trained on effect the anomaly ranking of each test point?

First we introduce the setting for unsupervised anomaly detection. In this case, both the training and test sets do not contain labels. The anomaly detection model is fit on a training set. We need the contamination factor γ, which represents the percentage of anomalous examples in the dataset, to generate labels. The training points with the top γ anomaly scores will be labelled as anomalous while all others are labelled as normal .The anomaly threshold can then be computed as the lowest anomaly score at which a point will be in the top γ ranked training points. When assessing on a test set, we use the fit model to generate anomaly scores for every test point. The labelling of the test points is not determined by γ but rater by the anomaly threshold determined in the training phase. There are many anomaly detection algorithms that have been developed to rank points according to anomaly score. The focus here is not on any particular anomaly detection algorithm, but on designing a way to measure stability of the algorithm. Therefore we will use a few well-known algorithms (K-Nearest Neighbor Outlier, Local Outlier Factor, and Isolation Forest) to demonstrate the concept of stability. We briefly introduce these algorithms below. KNNO outlier detection is a distance based approach [6][7]. The model is fit on a training set and anomaly rankings are generated on a new, unseen test set. For each point in the test set, we find its K nearest neighbors in the training set. The anomaly score for the test point is equal to the mean distance to these k points. The idea is that outliers will be physically further away from non-outliers, and will therefore have a higher average distance to their nearest neighbors. Isolation Forest aims to separate datapoints with the idea that anomalous datapoints will be easier to separate than normal datapoints [5]. The algorithm is fit on a training set, where binary iTrees are built to isolate datapoints. The idea of an iTree is to randomly select a dimension of the data, then create a split along a randomly selected value in that dimension. This splits the data into two sets, and each additional split doubles the amount of sets. iTrees continue to be built until all datapoints are isolated, meaning they are the only point in their set. The points in the test set are then passed through the iTrees built in the training phase.

Scores are generated according to the number of splits that a point needs to go through in order to be isolated. Local Outlier Factor (LOF) is a density based approach that is similar to KNNO in implementation [2]. Instead of measuring a point's anomaly score based off of its distance to k neighbors, we measure a point's score by density compared to its nearest neighbors. For each point in the test set, we find its nearest neighbors in the training set. If a point has lower density compared to its neighbors, it receives a higher anomaly score and vice versa.

3 Methodology

3.1 Problem Statement

The problem we are trying to solve can be defined as:

Given: A dataset D containing points $\{x_1, x_2, \ldots, x_D\}$ suspected of containing outliers, the contamination factor of the data γ, and an anomaly detector model h.

Do: Design a stability metric $\mathcal{S}_h$ that measures the variation of predictions generated by the given anomaly detector h when trained on different subsets of data.

The goal of the stability metric is to quantify how much the ranking of points in the test set change with slight changes in the training data. In order to achieve this goal, we follow three steps. First, we draw multiple subsamples from the training data, train an instance of the model on each subsample, and generate rankings on unseen examples. Second we define and compute a stability metric for each point in the test set, using the rankings generated by the anomaly detectors to fit the Beta distribution. Finally, we compute the final stability metric for the model by averaging the stability of each point in the test set. The procedure can be seen in Algorithm 1.

3.2 Designing a stability metric for a point

Subsampling. In this setting we split our dataset D into a training set G and a test set T of cardinalities g and t respectively. In order to measure the effect of slight changes of training data on the anomaly rankings, we should theoretically draw subsamples from the distribution of data, assuming that the i.i.d. sample might replace the dataset that we get in practice. As only an entire dataset is commonly available in practice, the strategy consists of drawing examples from G instead of involving the actual distribution. We begin by drawing I random subsamples from G. We can consider both the case where subsample size is fixed, determined as a percentage of g, and the case where subsample size is variable, randomly selected from a range of possible sizes.

```
Name: Stability
input  : $D$, dataset containing outliers ;
         $\gamma$, contamination factor;
         $h$, anomaly detector model;
         $I$, number of subsample iterations
output: Stability score $\mathcal{S}_h$ for the anomaly detector $h$ on dataset $D$
Training set $G$, Test set $T$;
for $1 < i < I$ do
    $G_i$ = draw a random subsample from $G$ ;
    $h_i \leftarrow$ h.fit($G_i$);
    $Scores_i \leftarrow h_i$.decisionfunction($T$);
    Compose $Rankings_i$ from $Scores_i$ according to Figure 1;
end
Initialize Beta distribution;
Tune $\alpha$ and $\beta$ parameters using Eq. 1 and 2;
for $1 < t < |T|$ do
    Compose $r_i(x_t)$ from $Rankings_i$ for $i = 1, \dots, I$ according to Figure 1;
    Compute $S_t$ from $r_i(x_t)$ for $i = 1, \dots, I$ according to Eq. 3 and 4
end
Compute $S_h$ from $S_t$ for $t = 1, \dots, |T|$ according to Eq. 8;
return $S_h$
```

Algorithm 1: Stability

Generating rankings. The procedure for this part can be seen in figure 1. Our model h is trained I times, each time on a different subsample. Every trained model is then used to generate anomaly scores for each point in T. The result of repeating this for all I iterations is I lists of anomaly scores each of length t. Each list of anomaly scores is converted to a list of anomaly rankings, where points are sorted from lowest anomaly score (position 1) to highest anomaly score (position t). From these I lists of length t ranking the points at each iteration, we can compose t lists of length I, compiling a single point's rank at every iteration. We normalize these rankings, dividing each ranking by t. For any point $x_j \in T$, its normalized list of positions will be referred to as $r_i(x_j)$ for $i = 1, \dots, I$, showing for every point we obtain a list of rankings for iterations 1 to I.

Beta modelling. When assessing stability of a point we are interested in both the variation of a point's anomaly rankings as well as if the point frequently changes anomaly labels. The area under the beta distribution is a good choice for this task because we can penalize points for having large ranges of potential rankings and can give more weight to changes in certain areas.

The simplest way to capture a point's stability would be to take the variance over its anomaly rankings. However, this does not reflect that some changes in ranking are intuitively more important than others. In anomaly detection, the contamination factor γ is used during the training phase to determine the anomaly threshold. The anomaly threshold is determined by generating anomaly scores for each training point, and ranking from lowest to highest score. The

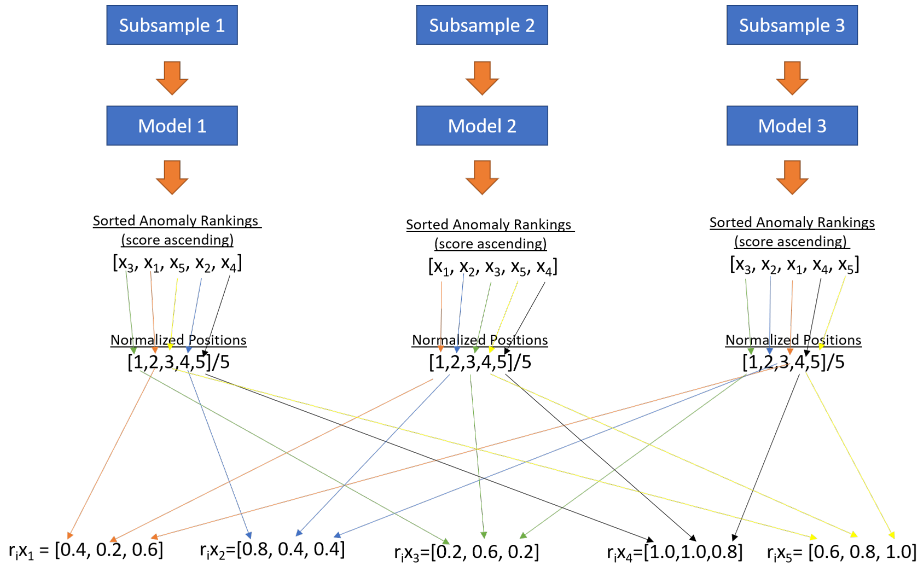

Fig. 1. Example procedure used to generate anomaly rankings and convert to a normalized list of positions for each point.

points with the γ highest scores are labelled as anomalies. We assume that the training set and test set have the same structure, i.e. the same distribution of data and contamination factor. With this information, we can assume that if the test set has sufficiently many elements, we can model the anomaly threshold using γ. This would not hold true if we work on small test sets, or test sets generated from a different distribution. So in our test set, we can estimate this anomaly threshold to be $1-\gamma$ in an anomaly score ascending ranking of points. When accounting for stability, we wish to assign greater weight to changes in ranking across this threshold, as this corresponds to a point changing from being labelled normal to being labelled anomalous.

How can we model assigning higher weights to changes in ranking near the anomaly threshold? Our proposed method involves taking the area under a Beta distribution. The reason we propose to take the area is to capture the uncertainty caused by the spread of all possible rankings of a point. We wish to penalize points for having a large potential range of rankings, as this corresponds to instability. By taking the area, we sum all penalizations in the sample space. If a point in T can be ranked as both the most anomalous and least anomalous point depending on the training data, we wish to assign the maximum penalty that we can, which is 1 (the area under the entire Beta distribution). With the right choice of α and β parameters, the shape of the Beta distribution can be tailored to our task. By setting the mode of the distribution equal to the anomaly threshold $1-\gamma$, we have an area of high density that reflects the importance of anomaly ranking changes in this region. Meanwhile, changes in anomaly rankings far away from this threshold are still captured, but are given much less importance. The general shape of our proposed Beta distribution can be seen in Figure 2. Equation 1 represents the relationship between the mode of the Beta distribution and the α and β parameters. Setting the mode equal to the anomaly threshold, we remove one degree of freedom and obtain Equation 2. For our task, we can tune the parameters to create a steeper or more spread out curve, as long as Equation 2 still holds. The steeper a curve is, the more we prioritize changes close to the anomaly threshold and give less weight to changes far away.

$$Mode = \frac{\alpha - 1}{\alpha + \beta - 2} \tag{1}$$

$$\alpha = \beta \left(\frac{1-\gamma}{\gamma} \right) + \frac{2\gamma - 1}{\gamma} \tag{2}$$

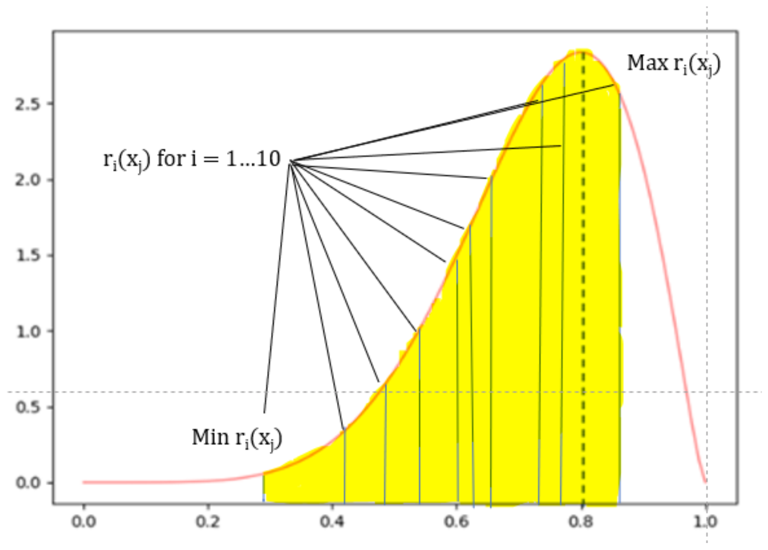

Fig. 2. Area under the Beta demonstration for a single point. Each line represents a ranking at a certain iteration. We penalize the entire range of the sample space

For each point x_j, we propose to use its normalized list of rankings $r_i(x_j)$ for $i = 1, \ldots, I$ to generate bounds with which to take the area under the Beta distribution. The minimum and maximum values of $r_i(x_j)$ for $i = 1, \ldots, I$ are taken as the lower and upper bounds of the integration of the Beta distribution. The area captured under the Beta distribution in this way corresponds to the stability of the anomaly rankings. A visualization of this logic can be seen in Figure 2. However, the area under the Beta does not entirely reflect a point's stability. Two points could have the same minimum and maximum anomaly rankings, but a large difference in the distribution of all other points. We propose to multiply the variance of $r_i(x_j)$ for $i = 1, \dots, I$ with this area under the Beta in order to take into account overall variation in the rankings.The result of this gives a single score representing stability.

How do we make sense of this score? We consider two scenarios, a maximally stable scenario and a random ranking scenario. In the maximally stable scenario, all of a point's anomaly ranking are the same, leading to zero variance and zero area under the Beta. In a completely random ranking scenario, over a large I the minimum and maximum rankings of 0 and 1 will appear, leading to an area under the Beta of 1. The variance of this random ranking scenario is shown in Theorem 1. Using t, we can compute the expected variance of a random ranking scenario, which will be referred to as σ^2_{rand}. Normalizing by σ^2_{rand}, we peg the maximally stable scenario to a score of 0 and the random ranking scenario to a score of 1. Thus we are not interested in ranking on a scale from maximally stable to maximally unstable, rather from maximally stable to randomly generated. Finally, we are ready to define a point instability score, $\mathcal{IS}_{x_j}$ and a point stability score $\mathcal{S}_{x_j}$. For each point x_j,

$$\mathcal{IS}_{x_j} = \frac{\sigma(r_i(x_j)) \times \int_{\min(r_i(x_j))}^{\max(r_i(x_j))} Beta}{\sigma^2_{rand}} \tag{3}$$

$$\mathcal{S}_{x_j} = 1 - \mathcal{IS}_{x_j} \tag{4}$$

Theorem 1. *Let X be a discrete random variable over the probability space $(\Omega, \mathcal{F}, \mathbb{P})$, where $\Omega = \{1, 2, \dots, t\}$, $t \in \mathbb{N}$, $\mathcal{F}$ a $\sigma-$algebra over Ω and $\mathbb{P}$ a probability measure. Assume that X follows a uniform distribution. Then the random variable $\frac{X}{t}$ still follows a uniform distribution with mean and variance, respectively equal to*

$$\mathbb{E}\left[\frac{X}{t}\right] = \frac{t+1}{2t}, \quad \mathrm{Var}\left[\frac{X}{t}\right] = \frac{(t+1)(t-1)}{12t^2} \tag{5}$$

Proof. We can directly compute the mean and the variance of $\frac{X}{t}$ by using the properties of random variables. First, we compute first and second moment of a discrete uniform random variables:

$$\begin{aligned} \mathbb{E}[X] &= \sum_{j=1}^{v} x_j \cdot \mathbb{P}(X = x_j) = \frac{1}{t}\sum_{j=1}^{t} j = \frac{t+1}{2} \\ \mathbb{E}\left[X^2\right] &= \sum_{j=1}^{t} x_j^2 \cdot \mathbb{P}(X = x_j) = \frac{1}{t}\sum_{j=1}^{t} i^2 = \frac{(t+1)(2t+1)}{6}. \end{aligned} \tag{6}$$

Second, we achieve the goal as follows:

$$\begin{aligned} \mathbb{E}\left[\frac{X}{t}\right] &= \frac{1}{t}\mathbb{E}[X] = \frac{t+1}{2t}; \\ \mathrm{Var}\left[\frac{X}{t}\right] &= \frac{1}{t^2}\mathrm{Var}\left[X\right] = \frac{1}{t^2}\left[\mathbb{E}\left[X^2\right] - \mathbb{E}\left[X\right]^2\right] = \frac{(t+1)(t-1)}{12t^2}. \end{aligned} \tag{7}$$

Note that, when $t \to +\infty$, the mean converges to $\frac{1}{2}$ and the variance to $\frac{1}{12}$. $\square$

3.3 Stability Score of a Model

Using the steps above, we obtain a point stability score for every example in T. However, we wish to obtain a stability score for a given anomaly detector h. The stability score of the model can simply be taken as the mean of all point stability scores in T:

$$\mathcal{S}_h = \frac{1}{t} \sum_{i=1}^{t} S_{x_i} \tag{8}$$

This stability score for a model h gives us a basis to compare the stability of different anomaly detection algorithms on the same dataset, or the stability of different datasets using the same anomaly detection algorithm. We can also use this score as a metric to maximize with the goal of achieving the most stable predictions.

3.4 Visualization of Stability

Here we visualize different stability scores as a heatmap. In these examples, the anomaly detection model is trained 100 times on different subsets of the training data. Each row represents one iteration, a model trained on one subset of training data. Each column corresponds to a single test point. The color of a coordinate on the heatmap represents a point's normalized anomaly ranking generated from a model trained on one subset of training data. In a perfectly stable scenario, each point has the same anomaly ranking at each iteration. In the heatmap, this corresponds to each vertical slice being consistent in color, as seen by Fig. 4. If anomaly rankings were generated randomly, a point would have a different ranking at each iteration. The heatmap in this scenario would look random noise, as seen by Fig. 3. We also show heatmaps for intermediate stability values that fit somewhere between the two extremes in Fig. 5 and Fig. 6.

4 Updating sample weights to increase stability

The problem we are trying to solve can be defined as:

Given: A dataset $D = \{x_1, x_2, \ldots, x_D\}$ suspected of containing outliers, the contamination factor of the data γ, and an anomaly detector model h.
Do: Increase the overall stability $\mathcal{S}_h$ of the model h on D and learn the training points that contribute the most to high stability.

In the previous section we have defined a new metric, which aims to measure the stability of an anomaly detection model upon a given dataset. Now we wish to use our stability metric to answer the following question: How can we select the most important examples from the training set in order to get very stable predictions? The basic idea of the algorithm involves making sample weight updates to training points based on their contribution to the model.

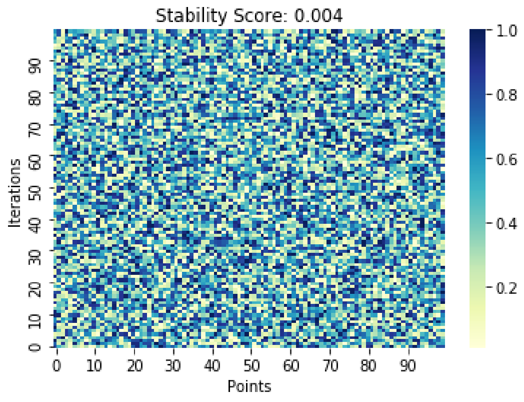

Fig. 3. Random rankings (Stability of 0.004)

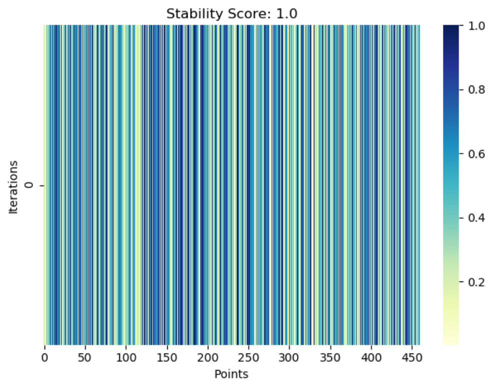

Fig. 4. Perfect Stability (Stability of 1.0)

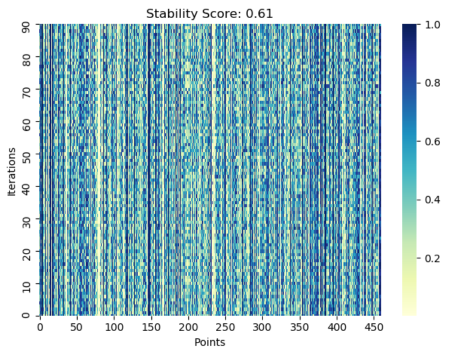

Fig. 5. Stability of 0.61

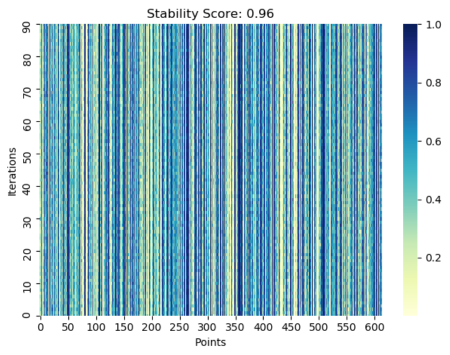

Fig. 6. Stability of 0.96

4.1 Sample weight updating

The idea of sample weight updating is that points with a high contribution to a high stability model are important and should have their sample weights increased. For any example $x_j \in G$, we define and estimate its contribution C_{x_j} as the proportion of times that x_j takes part of the subsamples to compute the stability score, as shown in equation 9. As defined, contribution is measured for training points and not validation or test points. We can make sample weight updates by comparing the contribution of a point and the overall stability of the model from two separate runs of algorithm 1. Inspiration for the sample weight update function came from [3].

$$C_{x_j} = \frac{|\{S \subseteq G \colon x_j \in S\}|}{I} \tag{9}$$

We make sample weight updates by comparison, so we need the contributions and overall stability from two separate runs of algorithm 1. So we use information from two previous runs, u-2 and u-1, to make sample weight updates for the current run, u. Contributions are normalized and multiplied by the size of the training set, meaning for uniform weights each point has a contribution of 1. First we calculate the stability change of the model between the previous 2 iterations. The stability of anomaly detector h at run u will be defined as $S_h(u)$. Likewise the contribution of point x_j at run u will be referred to as $C_{x_j}(u)$. Note that we do not calculate $\Delta S_h(u)$ or $\Delta C_{x_j}(u)$ for the first two runs.

$$\Delta S_h(u) = S_h(u-1) - S_h(u-2) \quad \forall u \geq 2 \tag{10}$$

Then calculate the difference in contribution for each training point x_j

$$\Delta C_{x_j}(u) = C_{x_j}(u-1) - C_{x_j}(u-2) \quad \forall i \geq 2 \tag{11}$$

Finally, $\forall u \geq 2$ the sample weight of each point x_j, for $1 \leq j \leq g$ is updated according to:

$$W_{x_j}(u) = \begin{cases} W_{x_j}(u-1) \cdot e^{\Delta C_{x_j}} & \text{if } \Delta S_h(u) \geq 0 \\ W_{x_j}(u-1) \cdot e^{-\Delta C_{x_j}} & \text{if } \Delta S_h(u) < 0 \end{cases} \tag{12}$$

The exponential function is useful in this case because it is symmetrical and centered at zero. Postive and negative changes in contribution are weighted equally. Additionally, updates are slow. We wish to keep sample weight updates slow, so random chance does not determine the outcome. The function produces the results that we would intuitively expect. If the contribution of a point and the model stability move in the same direction, i.e they both increase or they both decrease, then we increase the sample weight of the point. Likewise, if contribution of a point and model stability move in opposite directions, we decrease the sample weight of the point.

4.2 Practical algorithm for weight updating

The proposed method is summarized in algorithm 2. The algorithm can be summarized in three steps. First we run algorithm 1 two times, each time we compute the stability of the model on both a validation and test set. Second, we make our first sample weight update, using overall stability and point contributions from the first two runs to recalculate the subsample weights of all training points. Finally, we recompute stability using the new sample weights and repeat many times, alternating between sample weight updates and stability computations. The reason we have both a validation and a test set is to assess whether basing sample weights on one set will increase stability on new, unseen data.

```
Name: UpdateSampleWeights
input  : $D$, dataset containing outliers ;
         $\gamma$, contamination factor;
         $h$, anomaly detector model;
         $I$, number of comparison iterations;
         $U$, number of sample weight updates
output: $W_u$, Sample weights for training set at each $u$
Training set $G$, Validation set $V$, Test set $T$ ;
for $1 \leq u \leq 2$ do
    $W_u$ = Uniform;
    $S_h(V,u) \leftarrow Stability(V,h,\gamma,I,W_u)$;
    $S_h(T,u) \leftarrow Stability(T,h,\gamma,I,W_u)$;
    Compute $C_{x_j}(u)$ for $j = 1,2,\dots G$ according to Eq. 9
end
for $3 \leq u \leq U$ do
    Compute $\Delta S_h(u)$ according to Eq. 10;
    Compute $\Delta C_{x_j}(u)$ for $j = 1,2,\dots G$ according to Eq. 11;
    Update $W_{x_j}(u)$ for $j = 1,2,\dots G$ according to Eq. 12;
    $S_h(V,u) \leftarrow Stability(V,h,\gamma,I,W_u)$;
    $S_h(T,u) \leftarrow Stability(T,h,\gamma,I,W_u)$;
    Compute $C_{x_j}(u)$ for $j = 1,2,\dots G$ according to Eq. 9
end
return $W_u$ for $u = 1,2,\dots U$
```

Algorithm 2: UpdateSampleWeights

4.3 Implementing weight updating method into KNNO model

The proposed UpdateSampleWeights algorithm gives more weight to points that contribute to high stability. What if we also gave more weight to these points when actually generating the anomaly scores? Here we propose a method of using sample weights within the KNNO outlier detection algorithm to enhance the model's ability to increase stability. In KNNO anomaly detection, for each validation point we locate and calculate the distance to each of its K nearest neighbors in the training set. The anomaly score of a validation point can be taken as simply the mean of the K distances. However if our goal is to learn points contributing to high stability, we can give neighbors with high sample weights more power in determining the KNNO outlier score. We wish to give more weight to higher sample weight points because high sample weight points have been determined to be important to high stability. By giving more weight to these points in KNNO, we hope to generate more stable scores and therefore more stable anomaly rankings. In order to get this result, we follow the following method:

For any test point $x_j \in T$ we identify its K nearest neighbors x_{j_k} for $k = 1, 2 \ldots, K$ and compute Z, the sum of their sample weights:

$$\mathcal{Z}_{x_j} = \sum_{k=1}^{K} W(x_{j_k}) \tag{13}$$

where $W(x_{j_k})$ for $k = 1, 2 \ldots, K$ represents the weight of the k nearest neighbor of x_j.

We recompute the sample weights of a point's nearest neighbors, reducing the influence of very high sample weight points. We implement this step because we found that after many sample weight updates, a few points ended up with very stability and would dominate the computation of the KNNO scores. We again propose the exponential to help us with this task as it is effective in moderating extreme inputs. The weight $W(x_{j_k})$ of neighbor x_{j_k} will be recomputed as $W'(x_{j_k})$

$$W'(x_{j_k}) = e^{W(x_{j_k})/\mathcal{Z}_{x_j}} \tag{14}$$

Finally, the anomaly score of x_j is computed as the weighted mean of the distance to its nearest neighbors and their sample weights,

$$Score_{x_j} = \sum_{k=1}^{K} \frac{\|x_j - x_{j_k}\|_2 \times W'(x_{j_k})}{\mathcal{Z}_{x_j}}. \tag{15}$$

The expected result of this method is more stable anomaly rankings. By weighing the distances with sample weight of the points we reduce the effect of random variation. High sample weight points have been determined to be helpful in increasing stability, so we are just giving them an extra chance to boost their effect. Likewise, if a low sample weight point ends up in the training subsample, we reduce its effect because it has already been determined to be detrimental to stability.

5 Stability Experiments

In this section we seek to answer the following questions:

Q1: How does stability compare across different anomaly detector models?
Q2: What is the effect of the subsample size hyperparameter on stability?

Q3: What is the effect of the k hyperparameter on the stability of LOF and KNNO anomaly detector models?

For all experiments in this section, results are presented on seven datasets that are commonly used in anomaly detection [1]. Details about the datsets can be found in Table 1. These datasets will give us an idea of how our stability algorithm performs with different number of samples, dimensionality, and γ

Here we describe the experimental set-up for all experiments in this section. The contamination factor γ is computed as the percentage of anomalous points in the entire dataset. We utilize a random 2/3 training, 1/3 testing split in our evaluation and results are averaged across 10 different runs to reduce variability. For each run of the algorithm we use $I = 100$ comparison evaluations.

Q1: Model Stability Comparison. We compare the stability of anomaly rankings generated with Isolation Forest (IF) [5], Local Outlier (LOF) [2] with k=20 and K-Nearest Neighbors Outlier (KNNO) [6] with k=20 anomaly detection models. Results are displayed in Table 2. We also show the performance of the models as measured by area under the ROC curve in Table 3. We see that the stability of the anomaly detection models are much closer to being perfect than to randomly generated. KNNO had the highest average stability across the seven datasets, followed by Isolation Forest and LOF.

Table 1. Dimensionality, data size, and contamination factor of benchmark datasets

Dataset	Shuttle	Glass	WDBC	Stamps	Lympho	Iono	WBC
Dimensionality	9	7	30	9	19	32	9
Data Size	1013	214	367	340	148	351	454
Contamination	0.013	0.042	0.027	0.091	0.041	0.359	0.022

Table 2. Comparison of stability on IF, LOF, and KNNO anomaly detection models

Model	Shuttle	Glass	WDBC	Stamps	Lympho	Iono	WBC	**Average**
IF	0.987	0.974	0.975	0.967	0.901	0.966	0.991	**0.966**
KNNO (k=20)	0.991	0.984	0.994	0.974	0.975	0.985	0.999	**0.986**
LOF (k=20)	0.944	0.941	0.987	0.905	0.973	0.951	0.995	**0.957**

Q2: Effect of subsample size hyperparameter on stability. Here we investigate the effect that the subsample size hyperparameter has on stability results. When calculating stability we train on random subsamples of size s. In this experiment we use compare two different settings. In the first setting, we train on subsamples that always contain $\frac{1}{2}g$ points, where g is the cardinality of the

Table 3. Comparison of area under the ROC curve on IF, LOF, and KNNO anomaly detection models

Model	Shuttle	Glass	WDBC	Stamps	Lympho	Iono	WBC	**Average**
IF	0.626	0.481	0.742	0.622	0.921	0.759	0.926	**0.725**
KNNO (k=20)	0.497	0.479	0.730	0.586	0.889	0.819	0.968	**0.710**
LOF (k=20)	0.498	0.479	0.718	0.555	0.874	0.829	0.731	**0.669**

training set G. In the second setting, the size of the subsamples is randomly generated betwwen the bounds of $\frac{1}{4}g$ and $\frac{3}{4}g$. Results are averaged across all datasets and displayed in Table 4. For Isolation Forest, we see that using a variable subset size has no substantial effect on the stability. However for KNNO and LOF, the variable sample size noticeably decreases stability.

Table 4. Effect of subsample size hyperparameter on stability

Model	Subsample	**Average Stability**
IF	0.5 training set	**0.965**
IF	0.25-0.75 training set	**0.964**
KNNO (k=20)	0.5 training set	**0.986**
KNNO (k=20)	0.25-0.75 training set	**0.978**
LOF (k=20)	0.5 training set	**0.957**
LOF (k=20)	0.25-0.75 training set	**0.922**

Q3: Effect hyperparameter k on stability in KNNO and LOF. We examine the effect of the k hyperparameter on stability on LOF and KNNO anomaly detection. In these models, the k hyperparameter determines the number of neighbors of a point that are used to determine its anomaly score. Results are averaged across all dataets and displayed in Table 5. We see that generally larger values of k in these models produce more stable results. However the effect is highly dependent of the size and shape of the dataset. We also see that LOF seems to be more sensitive to the k parameter than KNNO.

Table 5. Effect of k hyperparameter on stability in KNNO and LOF

Model	k	**Average Stability**
KNNO	20	**0.986**
KNNO	10	**0.972**
KNNO	3	**0.924**
LOF	20	**0.957**
LOF	10	**0.882**
LOF	3	**0.686**

6 Experiments - UpdateSampleWeights Algorithm

In this section we wish to answer the following questions:

Q4: Does our UpdateSampleWeights algorithm actually increase stability?
Q5: Is the algorithm consistent in which points it updates to high sample weights?
Q6: Is our weighted KNNO model more successful in increasing stability than the original KNNO outlier detection when used with UpdateSampleWeights?

We evaluate the algorithm on the same datasets that we used in section 5. We only present results on one dataset for each experiment, the rest can be found in the appendix. Each experiment has a different set-up which will be described in its corresponding subsection.

Q4: UpdateSampleWeight's ability to increase stability To answer this question, we run Algorithm 2 on the given dataset. A random $\frac{1}{2}$ training, $\frac{1}{4}$ validation, and $\frac{1}{4}$ test split is used. In this experiment we use $I = 100$ subsample iterations to compute stability and $W = 25$ sample weight updates. Results will be presented for KNNO, LOF, and Isolation Forest anomaly detection models. We will be plotting stability and area under the ROC curve as a function of the sample weight update number.

Results on the Ionosphere dataset can be found in figure 7. We see that the algorithm run on all models was successful in improving stability. Additionally, the area under the ROC curve remained constant or saw slight improvement for all models. In the case of Isolation Forest and KNNO the visual increase in stability is modest, due to the models having high initial stabilities with uniform subsample weights. There is a larger visual increase in stability when run on the LOF model, as the model's initial stability is relatively low when compared to KNNO and Isolation Forest.

Q5: Algorithm consistency If we run the algorithm multiple times, will the sample weights converge to the same values every time? To answer this question, we run Algorithm 2 six times on the given dataset. A random $\frac{1}{2}$ training, $\frac{1}{4}$ validation, and $\frac{1}{4}$ test split is used. For each run we use $I = 100$ subsample iterations to compute and $W = 25$ sample weight updates. Results here are displayed for the Shuttle dataset.

The right side plots of figure 8 represent how many times each training point ended up with a high sample weight (above 1) out of 6 runs of the algorithm. The sample weights are measured after 25 updates. With all models, there was never a training point with high sample weight all six runs. The majority of points were inconsistent, being "selected" between 1 and 4 times out of the 6 runs. These plots show that there is limited consistency to the sample weight updates. However, since anomaly scores are generated based on proximity to neighbors we hypothesize that there is consistency on a neighborhood basis rather than an individual point basis. Our anomaly detection algorithms are dependent on the relationship amongst neighboring points. KNNO measures the average distance

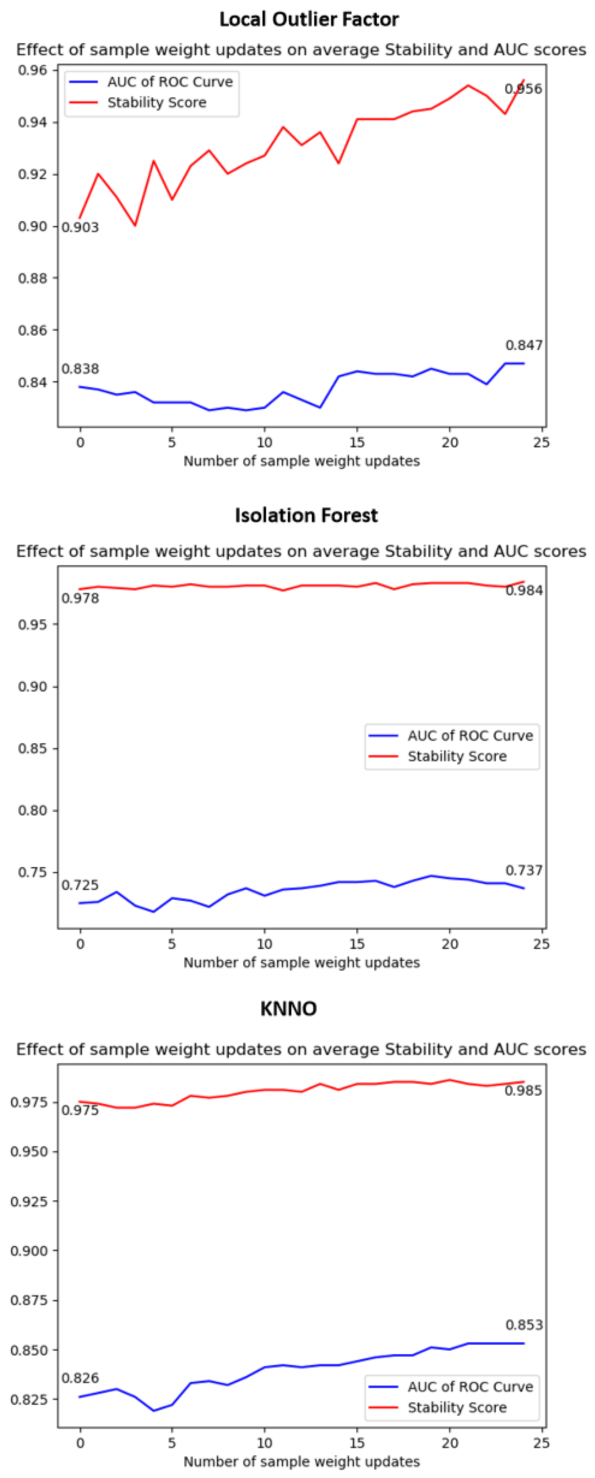

Fig. 7. Stability and AUC of LOF, KNNO, and IF on Ionosphere dataset plotted as a function of sample weight updates

of a point's nearest neighbors, LOF measures the local deviation of a given data point with respect to its neighbors, and IF aims to isolate points from their neighbors. One way that we can measure consistency on this neighborhood level is by taking the MSE of the sample weights of each point's neighbors as a function of sample weight updates. Results are shown in the left side plots of figure 8. For the MSE plots we used the mean sample weight of a points 10 nearest neighbors. While the results are not perfect, we see that for LOF and Isolation Forest there is a substantial leveling off of the MSE plots in the last 10 sample weight updates, indicating that the sample weights were moving towards convergence after 15 sample weight updates.

Q6: Comparison between weighted KNNO and original KNNO To answer this question, we run Algorithm 2 on both the original KNNO model and the proposed weighted KNNO model. A random $\frac{1}{2}$ training, $\frac{1}{4}$ validation, and $\frac{1}{4}$ test split is used. In this experiment we use $I = 100$ subsample iterations to compute stability and $W = 25$ sample weight updates. To asses performance, we will be plotting stability and area under the ROC curve as a function of the sample weight update number. Additionally we will be measuring consistency with the MSE of neighbor's sample weight plot introduced in the previous section.

Results for the WDBC dataset can be found in figure 9. The performance of both algorithms looks similar, however we do see a noticeable improvement of our weighted KNNO model over the original. The original KNNO model saw mild (less than 0.01) improvements in stability for most runs, while the weighted KNNO model consistently improved the stability from around 0.96 to between 0.98 and 0.99. The biggest advantage of the weighted KNNO model is its consistency, as shown by the MSE plots. The nearest neighbors MSE between different runs of the original KNNO model ranged between 0.25 and 0.35 and appeared to still be increasing after 25 sample weight update. By contrast, the weighted KNNO model saw nearest neighbor MSE values of between 0.10 and 0.15 and remained flat after 15 sample weight updates.

7 Conclusion and future work

Research on stability in machine learning so far has been applied to classification and clustering , but not on unsupervised anomaly rankings. In this paper, we design a stability metric quantifying how much the anomaly rankings of examples in the test set change with slight perturbations in the training set. With the newly defined stability metric, we designed an algorithm with the goal of increasing stability by learning which points in the training data contributed most to a stable anomaly detector. The stability computation was applied to several benchmark anomaly detection algorithms, comparing stability achieved on different anomaly detectors (KNNO, Isolation Forest, LOF) with different hyperparameters and subsampling methods. We also applied our subsample updating algorithm to the same datasets and anomaly detectors, showing consis-

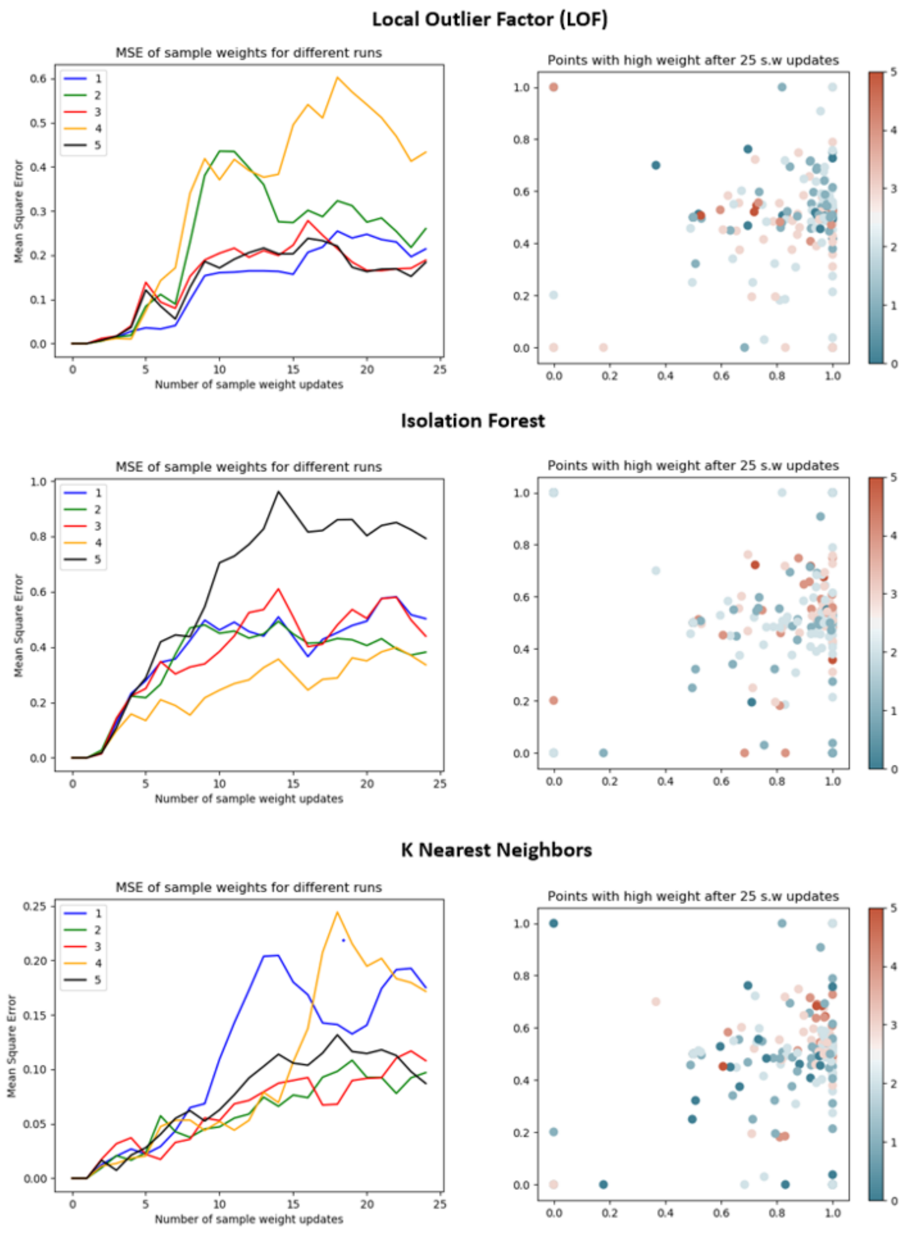

Fig. 8. Left plots measure the MSE between sample weights of different runs of the algorithm. Right plots show out of 6 runs, how many times each training point ends up with greater than uniform sample weight after 25 sample weight updates

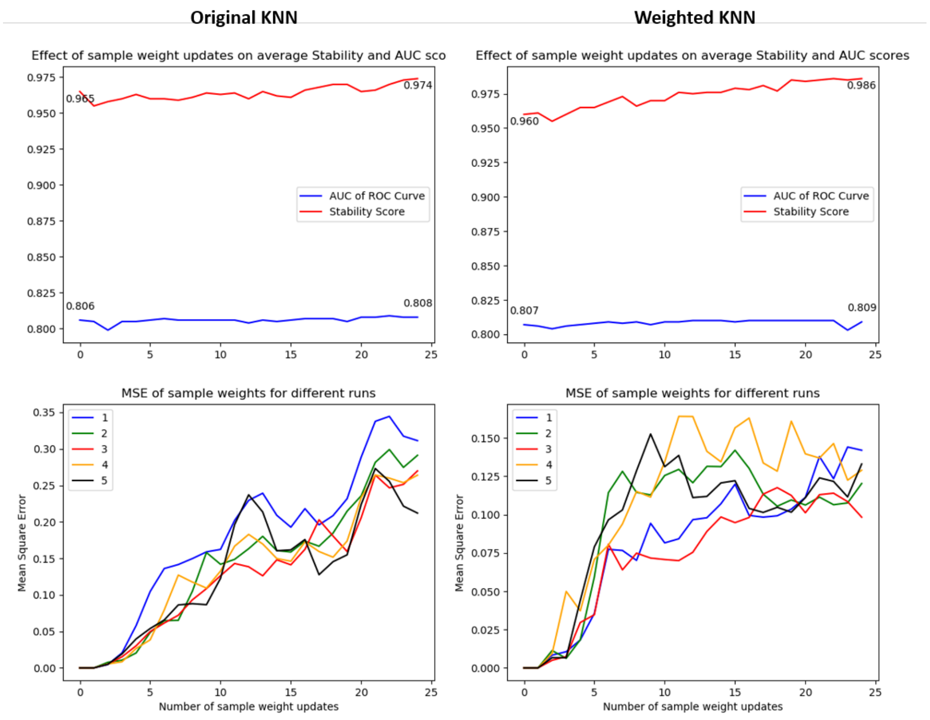

Fig. 9. Comparison of stability performance and consistency of original KNNO and weighted KNNO algorithms

tent increases in stability and revealing patterns about which training points contributed the most to high stability.

We see our stability metric as a promising method to measure the robustness of anomaly rankings, and could see the metric being useful in future applications and research in anomaly detection. Our subsample weight updating algorithm did reveal patterns about the points that contributed most to high stability, however the algorithm was not entirely consistent as shown by our experiments. We believe that we may be able to get more consistent results if we framed our goal as an optimization problem, instead of using iterations to make updates. With an optimization approach we could hope to obtain the same subsample weights and stability performance over many different runs of the algorithm.

References

1. Campos, G.O., Zimek, A., Sander, J., Campello, R.J., Micenková, B., Schubert, E., Assent, I., Houle, M.E.: On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study. Data Mining and Knowledge Discovery **30**(4), 891–927 (2016)
2. Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection: A survey. ACM computing surveys (CSUR) **41**(3), 1–58 (2009)
3. Freund, Y., Schapire, R., Abe, N.: A short introduction to boosting. Journal-Japanese Society For Artificial Intelligence **14**(771-780), 1612 (1999)
4. Höppner, F., Jahnke, M.: Holistic assessment of structure discovery capabilities of clustering algorithms. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. pp. 223–239. Springer (2019)
5. Liu, F.T., Ting, K.M., Zhou, Z.H.: Isolation-based anomaly detection. ACM Transactions on Knowledge Discovery from Data (TKDD) **6**(1), 1–39 (2012)
6. Otair, D., et al.: Approximate k-nearest neighbour based spatial clustering using kd tree. arXiv preprint arXiv:1303.1951 (2013)
7. Ramaswamy, S., Rastogi, R., Shim, K.: Efficient algorithms for mining outliers from large data sets. In: Proceedings of the 2000 ACM SIGMOD international conference on Management of data. pp. 427–438 (2000)

A Results on additional datasets

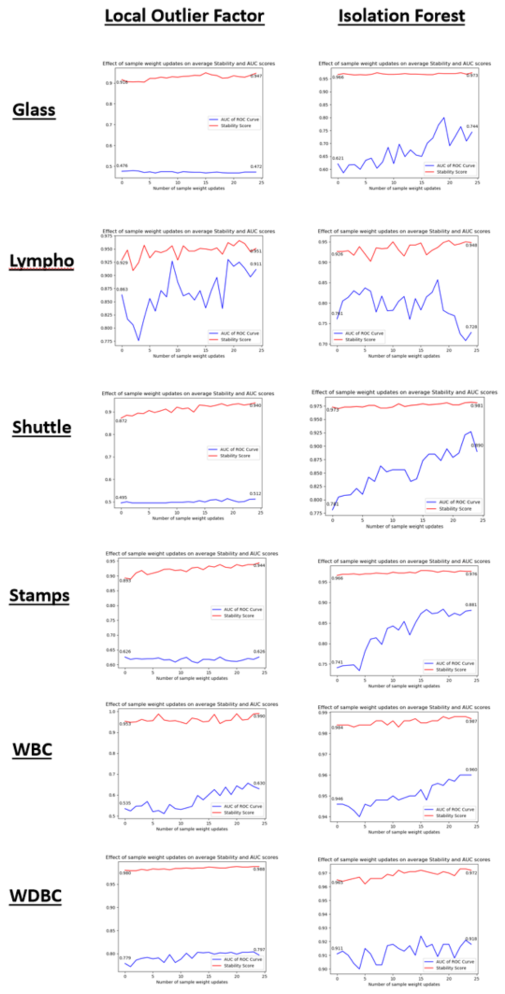

Fig. 10. Algorithm performance on LOF and Isolation Forest for additional datasets

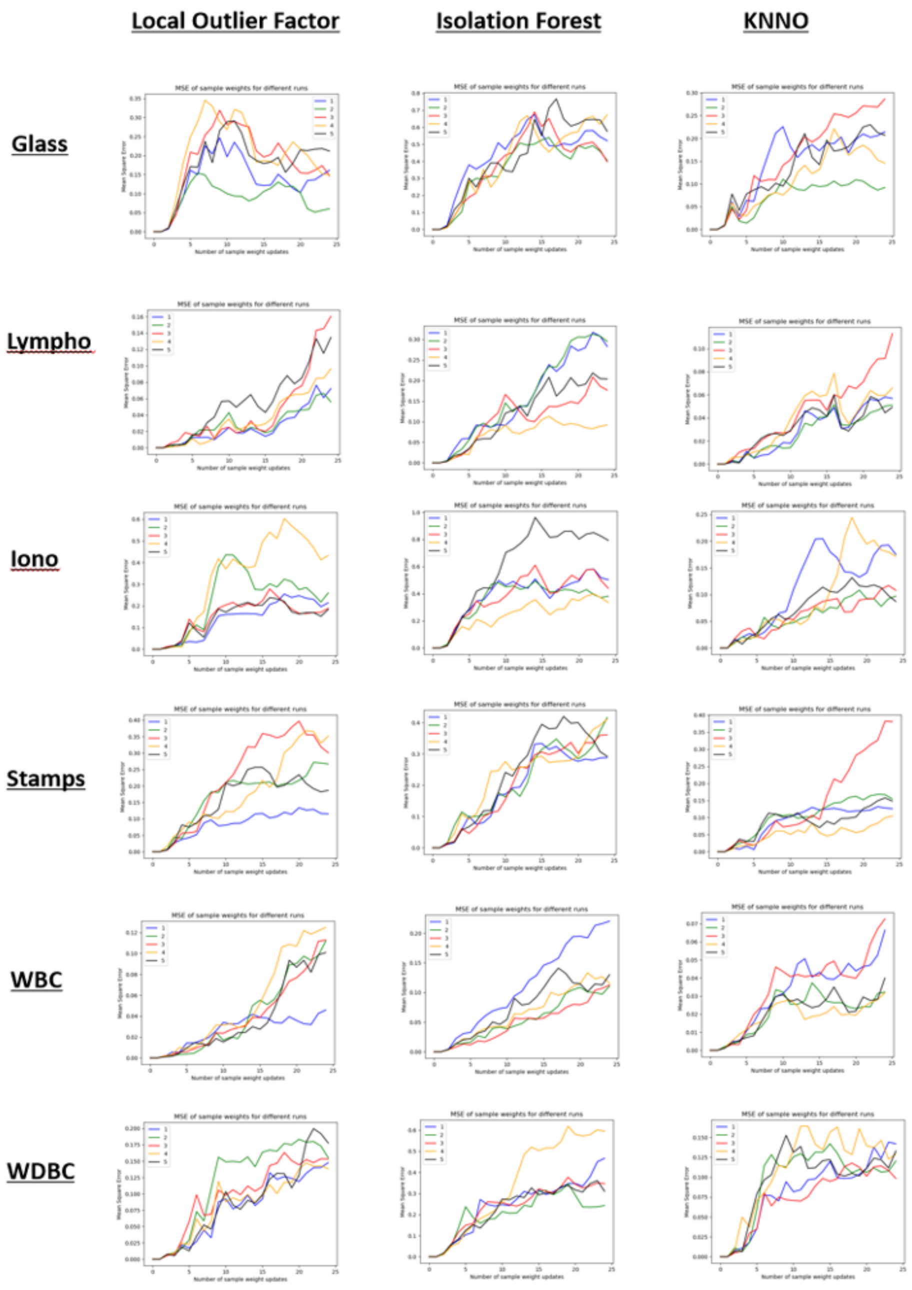

Fig. 11. Nearest Neighbor MSE plots for additional datasets

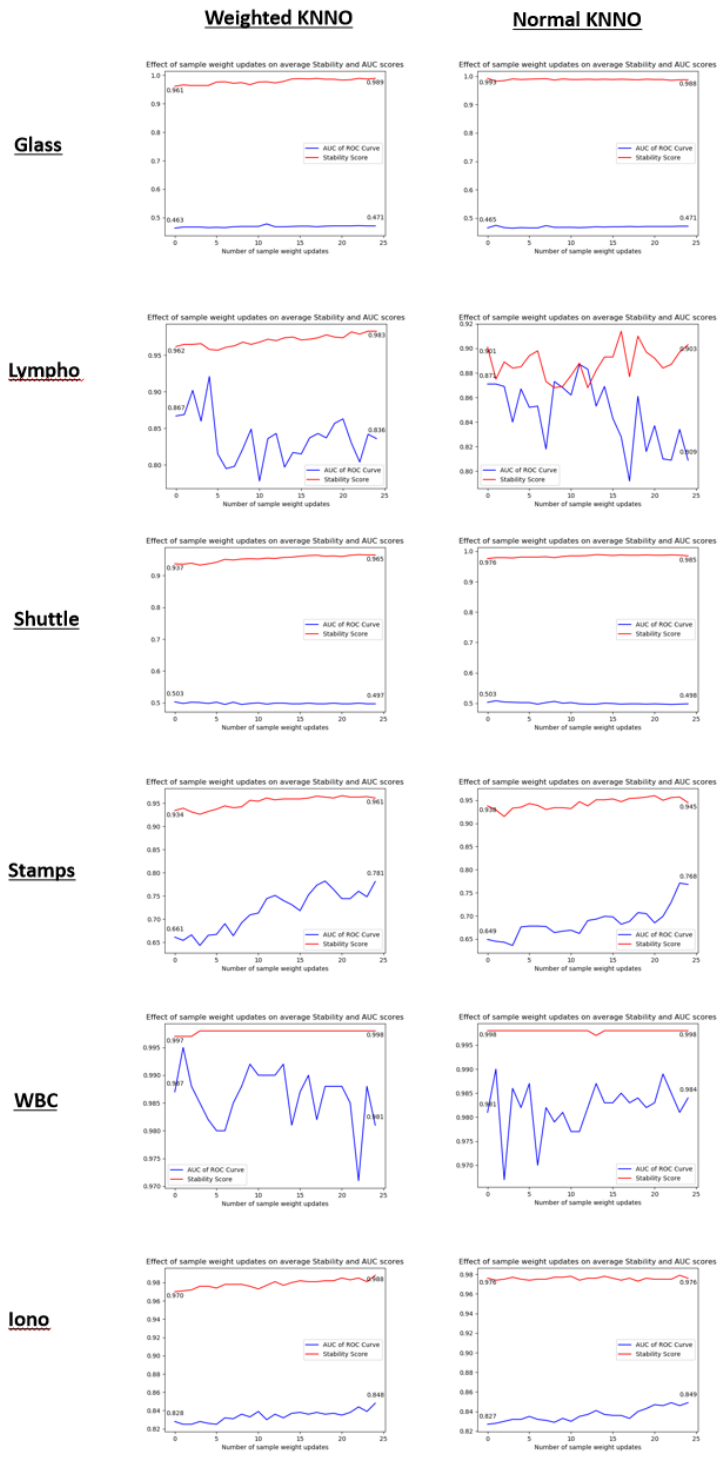

Fig. 12. Normal KNNO vs. Weighted KNNO for additional datasets