Autonomous Anomaly Detection in Games

Master Thesis

Geert Beuneker

A thesis presented for the degree of
Master of Science

Faculty of Science
Utrecht University
Netherlands

Autonomous Anomaly Detection in Games

Master Thesis

Geert Beuneker

Abstract

In game development, a lot of time is spent on making sure the product runs smoothly and without too many bugs. This makes testing a very important part of the game development cycle. Delivering a high quality product is critical to maintaining customer satisfaction. Generally, testing games is done by a manual process which can be very time consuming and expensive for developers. In this paper we explore techniques to enable autonomous bug detection and improve the testing pipeline. To achieve this, we employ state-of-the-art anomaly detection techniques. We developed a generic framework for using anomaly detection to analyze the relations between different variables in games. Further, we perform a case study comparing different state-of-the-art anomaly detection algorithms in a wide range of scenarios.

With our framework it is possible to autonomously detect 15%-20% of the inserted anomalies with an accuracy of about 90% without the need for any training data. This research lays the groundwork for easy integration of autonomous testing in games. However, improvements can still be made with future research.

Acknowledgements

First and foremost I would like to thank Dr. Wishnu Prasetya for all his great feedback and advice and the overall pleasant collaboration throughout the process. I would also like to thank Prof. Dr. Frank Dignum for being the second supervisor of this thesis.

I would also like to thank the people at SOEDESCO for allowing me to scout at their location, to see how they work and helping me develop the ideas for this thesis.

Finally, a special thanks to my girlfriend for her mental support throughout the process as well as my friends and family.

Contents

1 Introduction

Throughout the development process of video games, bug testing and quality assurance are very important parts of delivering a quality product. Generally, unit tests are used in software development to test isolated behaviour and ensure those parts of the program are still working correctly after implementing new features or changing the code. Such tests are useful for software development, however they are often not applicable to game development. While the separate entities in games usually implement simple behaviour which could be encapsulated in unit tests, the primary focus of testing games is testing the entire game experience. The complexity of performing such tests mainly comes from the combined behaviour of multiple entities which make up the game experience. Testing the game experience would require testing the system as a whole instead of isolated parts. The reason for this is that game entities usually aren't isolated and have many complex, context sensitive dependencies between them. Because this complex combined behaviour is very hard to test automatically, a lot of time is spent on manually testing games. Developers want to make sure their games are working properly so that a high quality product can be delivered. This means a lot of time is spent on testing and Quality Assurance (QA). Therefore, enabling the use of automated testing to assist this manual testing can save a lot of time and money as well as increasing the likelihood of finding bugs in games. We will be researching the possibilities of using machine learning to perform autonomous testing. We would like to autonomously detect whether the behaviour of the game is in line with the previously observed behaviour or if the behaviour is incorrect, anomalous. Games are systems with a high level of interaction, a high number of mutual interacting entities and sometimes multiple users. This makes it unfeasible to describe the game state using classical predictors, such as "Properties" in [8], the pre/post-conditions mentioned in [32] or linear temporal logic predictions as seen in [31]. To be able to perform autonomous testing we need to look at *anomalies*, observations that are significantly different from previous observations as to arouse suspicion that they were generated by a faulty mechanism. To be able to detect these *anomalies* we will look into the field of *Anomaly Detection*. Performing *Anomaly Detection* for autonomous testing will allow us to create a generic framework which can observe a game state and report whether that state is behaving correctly, regardless of the specific implementation of the game. This may allow game developers to more easily perform autonomous testing.

Currently anomaly detection is scarcely used in game development. Most applications of anomaly detection can be found in cheat-detection applications [26] [40]. While these papers focus on cheat-detection rather than bug testing, the fundamental idea is the same. Similar to bug detection, cheat-detection tries to detect whether observed behaviour was generated by a different underlying mechanism. The main difference is whether that different underlying mechanism was used intentionally by the user (cheating) or was unintentionally hidden in the program (bug)[1]. [26] shows a nice comparison of different anomaly detection algorithms. However, their research focuses more on tracking specific, isolated behaviour such as valid transactions and detecting whether that behaves suspiciously. Their approach is more similar to log analysis. They observe logs for isolated variables, but do not observe the relations between different variables. Additionally,[40] shows cheat-detection using dynamic Bayesian networks. However, this requires training the algorithm beforehand making it more troublesome to use. The algorithm also works with the assumption that predictors can be found for the variables observed. This makes the approach much more difficult to use for a generic anomaly detector in games, as such predictors often cannot deal with the complex mutual interactions between game entities. Some work has been done in automated bug testing in games as well [30] [35] [1], but these applications are either focused on exploring the game state space or are designed for a specific game or application. Additionally, these existing applications for automated testing can only detect system failures such as crashes or exceptions. These errors however, are usually trivial to detect and can easily be identified. Moreover, errors such as exceptions are usually found relatively easily by developers or testers. However, finding faults which do not crash or halt the application but do exhibit unwanted behaviour are usually much more resource-intensive to find, if they're found at all. These bugs are often detected by users after the release of the product, which is undesirable for most developers. This research will focus on detecting these kinds of faults in the program. Detecting these requires specifying what is "correct" and what is "unwanted" behaviour. Determining "unwanted behaviour" is often non-trivial as formal software specifications which describe the correct behaviour of the program are rarely

[1]The distribution of data can be different for a cheating mechanism as users might use the mechanism constantly whereas bugs would occur occasionally, but this is currently beyond our scope.

defined for games. In this research we will explore ways of specifying unwanted behaviour of games using machine learning techniques and using that to perform automatic anomaly detection. We will test the effectiveness of various algorithms as well as compare their accuracy and performance. The **contribution** of our research is the exploration of methods to reduce the amount of resources required for testing. Secondly, we introduce a generic way to automate parts of the testing process which can be easily implemented by developers for many different types of games. Additionally, we explore a large solution space and find out what factors influence the accuracy of the results in order to lay the foundations for future research into automated anomaly detection for games.

1.1 Motivation

The motivation for this thesis was based on personal experience and observations done in the game development industry. A large part of developing games is testing them and making sure they run according to specifications. However, based on personal experience this can be very time consuming and usually has to be done manually every time a new feature is implemented. To find out if this problem also exists in the industry and to see how the game development industry deals with QA testing, some preemptive research was done. As part of this preemptive research a few days of scouting was conducted at SOEDESCO, a game dutch game publishing company which actively performs QA testing on games. From the observations done and the interviews conducted there, it was found that QA testing in the industry is mostly done manually and can be very time consuming. Thus, the motivation for this thesis comes not only from a scientific motivation but also from a business motivation.

1.1.1 Scientific Motivation

Not a lot of research has been done currently in the domain of automated anomaly detection in games. This makes it an interesting subject from a scientific perspective. Due to the interactive nature of games, every run of the game can be very different. This makes a lot of existing anomaly detection techniques incompatible. To conduct anomaly detection for this specific domain, some groundwork has to be laid down and some modifications of existing anomaly detection are required. We intend to thoroughly explore this relatively new domain by constructing a framework and comparing existing anomaly detection algorithms using various parameter settings in a series of different scenarios. By doing this, we intend to gain insight into the important parameters and settings to lay the proper groundwork for future research into this domain.

1.1.2 Business Motivation

As stated before, a lot of resources are spent on QA testing. This is mainly because it costs a lot of man hours to detect bugs in games. Additionally, QA testers aren't always available throughout the whole process due to time constraints. If some parts of this process can be automated it could save a company lots of resources. From a business perspective it is very important their product keeps working correctly to ensure their users remain satisfied. To this purpose, automated anomaly detection can be used to constantly monitor the software both during development and after deployment to catch possible bugs early. This could potentially detect issues or exploits before a lot of users are affected. This is important for user satisfaction and for possibly avoiding malicious exploitations in the product.

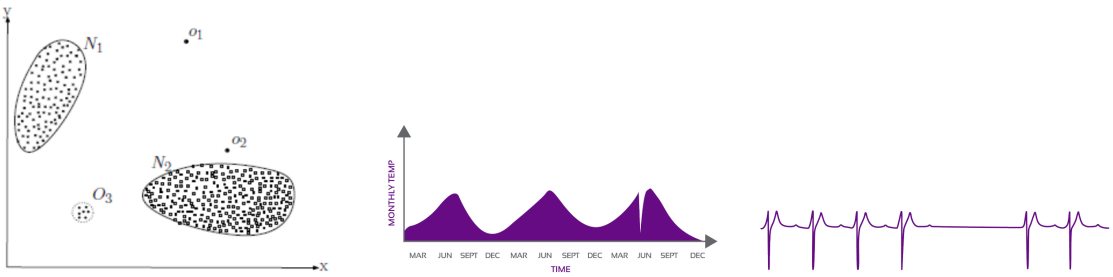
1.2 Background Information

An anomaly or outlier, can be defined as given by Hawkins[14]: "*An outlier is an observation that deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism*". Using algorithms to detect such anomalies in software is called *Anomaly Detection*. The field of *Anomaly Detection* has a very broad spectrum with applications in health care [11], fraud detection[7], intruder detection [29], industrial applications [27] and big sensor data systems [15] just to name a few. Generally, we can distinguish three different types of anomalies:

1. **Point anomalies** 1a anomalies where a single data point is considered anomalous in the global context of the program.

2. **Contextual anomalies** 1b anomalies where a data point is considered anomalous in its local context, while not necessarily being anomalous in the global context.

3. **Collective anomalies** 1c anomalies that contain a stream of data that is considered anomalous with respect to the complete dataset.



(a) Point Anomalies (b) Context Anomalies (c) Collective Anomalies

Figure 1: Different types of anomalies

Anomaly detection algorithms work by learning what behaviour is normal and what behaviour is anomalous for the program. This learning can be done in three different ways:

1. **Supervised learning** algorithms need to be trained with a complete dataset of both positive and negative training data to detect anomalies. This means the algorithm needs labeled data for both non-anomalous and anomalous behaviour. The advantage is that this technique has great accuracy and speed compared to the other techniques. The downside is that labeled training data for both positive and negative behaviour is often not available. This makes supervised learning techniques overall less generalizable than the other types of learning.

2. **Semi-Supervised learning** techniques only need to be trained using positive training data. Algorithms using semi-supervised learning can detect anomalies after being trained on just the normal behaviour of the program. While this technique is generally less accurate than supervised learning techniques, it has the advantage that you only need positive labeled training data which is much more widely available.

3. **Unsupervised learning** doesn't need any labeled training data. Because no labeled training data is required, unsupervised learning techniques are the most widely applicable. The trade-off for this very high generalizability is that unsupervised learning algorithms are often less accurate and slower than supervised or semi-supervised learning algorithms.

2 Related Work

We will begin by looking at the most commonly used algorithms for anomaly detection and what is currently considered as state-of-the-art. To get an overview of the currently existing algorithms for anomaly detection we looked at the works of [29][7][16][25] which are reviews and overviews of anomaly detection algorithms. [23] and [24] provide a more detailed in-depth look at specifically statistical and neural network approaches of anomaly detection. From these overviews and reviews we can derive the most prominent categories of anomaly detection. The types of anomaly detection that are commonly used are:

1. Classification Based
2. Clustering Based
3. Neural Network
4. Statistical
5. Support Vector Machines (SVM)

These works look at the broad applications of anomaly detection, showing methods for detecting *Point Anomalies*, *Context Anomalies* and *Collective Anomalies* as well as varying methods for *Supervised*, *Semi-Supervised* and *Unsupervised* learning techniques. Using these sources we get an overview of the advantages and disadvantages of each type of anomaly detection, on which we can base our choice of algorithms for our research. The next section will go over each of these categories and list the current state-of-the art research as well as their relevance to our domain.

2.1 State of the Art

This section will discuss the development and the latest trends of the anomaly detection categories mentioned above. We will look at the fundamentals of each category as well as current state-of-the-art algorithms. Additionally, we'll discuss their relevance to our domain, their strengths and their weaknesses. This should show us what algorithms are applicable for our domain and motivate our choice of algorithms for our research

2.1.1 Classification Algorithms

Classification techniques learn a classification model from a set of labeled data instances and then use that classifier to label new instances. For anomaly detection, such classifiers often make predictions about the data and then compare those predictions to the observed results. This works with the assumption that such a classifier exists for the data and that it can be used to distinguish anomalous data from normal data.

A common classification algorithm is the Bayesian Network [18]. A bayesian network can be viewed as a graphical model that shows the causal relationships between different sets of variables. It uses supervised learning to learn causal relationships between variables. Given $\mathbf{X}$ it can predict the chance of $\mathbf{Y}$. In anomaly detection this can be used by observing an event $\mathbf{Y}$ and comparing that to the predicted probability of $\mathbf{Y}$ by the bayesian network based on the current state $\mathbf{X}$. The more improbable the occurring event is based on the state $\mathbf{X}$, the more likely it is an anomaly.

Valdes et al. [37] developed an anomaly detection system using naive Bayesian networks. They showed that their use of Bayesian Networks can detect distributed intrusions in which the attack sessions individually were not anomalous enough to warrant an alert, but the sessions combined were. An example of the usage of Bayesian networks in anomaly detection closer to our domain of games can be found in [40]. Here, a bayesian model is used to detect cheats in online games. Their dynamic approach to the bayesian network provides an effective and scalable solution in the detection of anomalous, cheating behaviour.

Overall however, classification algorithms will likely be unsuitable for our domain. Reliable causal relationships between variables can be very difficult to detect in interactive games because many variables will depend on the player's input, which can vary greatly. While bayesian networks and other classification algorithms can be very good at expressing relationships between variables, they usually require *Supervised* learning and are not very generalizable. Since in most cases we won't have training data available and we need a highly generalizable algorithm for testing games, classification algorithms will not be considered for our research.

2.1.2 Clustering Algorithms

Clustering algorithms try to determine from a set of points in n-dimensional space where those points form clusters. It is used to group data points with similar properties into clusters. These algorithms work with the assumption that anomalous datapoints are significantly different from non-anomalous datapoints as given by Hawkins 1.2. Given this definition we can identify datapoints that fall outside of this non-anomalous cluster as anomalies. One of the earliest examples of solving this clustering problem is the K-Nearest Neighbour algorithm[34]. This algorithm is first trained by storing feature vectors with their corresponding labels in an n-dimensional space. Then, for classification, it checks for a datapoint its k nearest neighbours. These k neighbours determine what label is assigned to the datapoint. The label that is most common among these neighbours is what is assigned to the datapoint.

The KNN algorithm has a few drawbacks however. First, for every new datapoint all datapoints in the training set must be tested. This makes the algorithm slow for large datasets, especially if it is used in an unsupervised manner where the dataset keeps growing over time. Second, the

algorithm is quite sensitive to the chosen value of k, having imprecise results for lower values of k or losing the ability to properly detect outliers for higher values of k. Finally, it has weak performance for datasets of varying density. When some points form tight clusters and others form a loose cluster KNN loses accuracy. Another drawback of this property is that KNN only detects global outliers, meaning that it can only identify anomalies if the datapoint differs greatly from the entire dataset. Because of this, KNN deals poorly with datasets that have multiple clusters of varying density. However, it can still be a very powerful algorithm for anomaly detection as shown in [13] and[22].

The Local Outlier Factor (LOF)[5] attempts to solve the locality problem of clustering algorithms. Their first contribution is not assigning a binary property to a datapoint being an outlier or not. Instead, they assign a "Local Outlier Factor" to each object which determines to what degree that object is an outlier. For objects with a similar density as its neighbours the LOF value will be approximately 1, values higher than 1 indicate a higher density than the its neighbours while values lower than 1 indicate a lower density its neighbours. This was shown to deal much better with varying densities and the contextual information surrounding a datapoint. It has been shown to have great accuracy for anomaly detection in our domain of games as observed in [26]. However, it is still dependent on a user-defined k parameter similar to KNN. Second, the algorithm is very resource intensive for large datasets.

An algorithm less sensitive to k-values and which has much better performance on large datasets is the Local Correlation Integral (LOCI)[28]. Similarly to LOF, they introduce a value which represents the degree of outlierness to a datapoint called the Multi-granularity Deviation Factor (MDEF). However, unlike LOF their value does not just contain the outlierness of a point, it adds more information such as the data in the vicinity of the point, the determining clusters, micro-clusters, their diameters and their inter-cluster distances. All of this extra information is added while still being as fast as competing clustering algorithms such as the LOF. Because of this extra information, their algorithm handles both local context and varying densities very well. Unlike LOF and KNN, the implementation of LOCI leads to a fast approximation algorithm named aLOCI which performs much better on large datasets. While LOCI can be modified, most parameters have preset recommended values as mentioned in [28]. This makes LOCI much less sensitive to user-defined parameters.

The Local Anomaly Descriptor (LAD)[19] proposes a method which is also very robust against different values of k. This algorithm employs physics principles of heat distribution amongst data points to determine if points belong to a cluster. When compared to LOF, LOCI and a few other KNN-based algorithms LAD shows generally better accuracy and stability. The major advantages of LAD are its improved accuracy when classifying datapoints close to two clusters and its increased stability over LOF and other KNN-based algorithms with different values of k.

Most clustering algorithms are fully *unsupervised* making them highly generalizable. This makes clustering algorithms a good fit for our domain since we can generally make no assumptions about our dataset and usually don't have data available for training. The drawback is that they can be quite demanding on large datasets when classifying datapoints, since usually the algorithm needs to check all existing datapoints to properly classify a new datapoint.

2.1.3 Neural Network Algorithms

Neural networks have the ability to learn complex connections between a large amount of neurons. These neurons can represent values in our system, all the neurons in our network combined can then describe the current state our system is in. By training neural networks they can detect whether our system is behaving normally or anomalously. Most of the time neural networks have to be trained *Supervised* or *Semi-Supervised* meaning we would need labeled training data, making most neural network algorithms unsuitable.

[24] gives a detailed overview of a wide range of neural network functions. While it shows many applications, most neural networks shown require some form of training using labeled data. There do exist unsupervised learning algorithms however, the approach proposed by Kohonen [21] offers a neural network approach similar to clustering. The Self-Organizing Map (SOM) requires no dataset of labeled class data. It works by adjusting the nodes in the SOM for every new datapoint. For every datapoint that is added, the closest node is pulled towards that point and subsequently pulling the nodes connected to that closer as well. It converges to a state where its nodes(here

called: neurons) properly enclose the group of datapoints it has seen. After training anomalies can be detected by checking whether points fall inside or outside of the map created by the algorithm.

[3] uses features from the Self-Organizing Map to construct a highly adaptive algorithm for detecting anomalies. The SONDE algorithm proposed combines features from Adaptive Resonance Theory (ART) and Grow When Required (GWR) with the Self-Organizing Map. It shows the great adaptability of an SOM while maintaining decent accuracy for detecting anomalies.

Since it is difficult to get properly labeled training data and especially labeled anomalous training data most neural network algorithms cannot be used for our domain. The Self-Organizing Map however does not require labeled training data and can be trained fully unsupervised. Unfortunately, the SOM can have difficulty distinguishing between separate groups of datapoints which may reduce its accuracy in some cases. However, due to its similarity to clustering algorithms it is highly adaptable and generalizable which makes it an interesting algorithm to include in our research.

2.1.4 Statistical Algorithms

Generally statistical approaches require some knowledge about the distribution of the dataset in order to perform proper anomaly detection as seen in [23] and [29]. Moreover, statistical approaches are rarely unsupervised as they usually have to be trained on some known statistical distribution of data. [27] claims an unsupervised statistical approach to anomaly detection. However, their algorithm works mainly for analyzing a group ranked by anomaly scores and using statistics for determining which ones should really be considered anomalous compared to the rest of the group. This is less useful since it already requires instances ranked by anomaly scores. One more promising occasion of unsupervised statistical anomaly detection is SmartSifter(SS) presented by [39]. Their approach works by first using a probabilistic model to represent an underlying mechanism of data-generation. They're using a histogram to represent the *categorical* probability density of variables and a finite mixture model to represent the *continuous* probability density of variables. Next, for every new data input SS employs an online learning algorithm which updates the model representing the probability density, the amount a new data point changes this model represents that data point's anomaly score. The advantage is that their method is computationally inexpensive and can deal with both *categorical* and *continuous* anomalies. Finally, because it can be performed unsupervised the algorithm is much more generalizable than other statistical approaches.

While the algorithm can perform unsupervised anomaly detection in a very computationally inexpensive way, it would probably not work well for our domain. The data received from an interactive game may not show a consistent statistical mechanism for data-generation as most data is generated by the user's choices. Additionally, statistical approaches are often bad at representing correlations between a lot of variables and instead focus more on detecting anomalous behaviour of isolated variables. Moreover, statistical approaches are much less generalizable than clustering or machine learning approaches. Therefore, statistical approaches will not be considered for our research.

2.1.5 SVM Algorithms

Support Vector Machines [17] are supervised machine learning models that can classify data in binary categories. Based on labeled training data, a Support Vector is drawn which separates the two different labels. SVMs are similar to clustering techniques in the sense that they try to separate clusters of data and use that separation to label data points. A drawback of an SVM is its poor performance when the datapoints do not have a clear line of separation. Additionally, A Support Vector Machine requires supervised learning where both positive and negative datapoints are required for training.

To solve the problem of supervised learning one-class SVMs exist, which can determine whether a datapoint belongs to a class or doesn't belong to that class. For anomaly detection this class can represent the normal, non-anomalous behaviour. If a datapoint does not fall into that class, it can be marked anomalous. This property of one-class SVMs is leveraged by [4] and enhanced to allow for unsupervised anomaly detection using Support Vector Machines. Their experiments show that on some of their datasets the SVM based algorithms outperform clustering and statistical based unsupervised anomaly detection algorithms. They also introduce a method for calculating outlier score based on the distance to the decision boundary. Their experiments show that unsupervised SVMs generally perform at least as well and in some cases better than state-of-the-art clustering

and nearest-neighbour techniques.

It is shown by [4] that one-class Support Vector Machines can be used for unsupervised anomaly detection. An advantage of the SVMs is that they do not require a k value like in KNN-based algorithms. However, even though the SVMs sometimes outperformed clustering and nearest-neighbour techniques, their performance does depend on the dataset. Additionally, using these techniques we would only be able to use one-class SVMs meaning that it would be difficult to detect anomalies caused by correlations between different variables. Since we are looking for a very generalizable solution, and because we would like the ability to take correlations between variables into account clustering techniques seem more suitable.

2.2 Existing automated testing techniques

In the following section we will take a look at current automatic testing techniques. We will look at their strengths and weaknesses and identify their shortcomings.

2.2.1 Automatic Software testing

First we will take a look at regular software testing techniques as they are more common than games testing techniques. This should give us a good idea of what automatic testing pipelines are deployed.

We start by taking a look at the T3 [32] testing suite. T3 is designed for testing Java classes. It randomly generates a large amount of test sequences to trigger faulty behaviour. Faulty behaviour is only identified as behaviour where an exception is thrown. The advantage is that T3 is very quick with decent performance. It works mainly on exploration, meaning that it just tries to find combinations of functions which cause an exception to be thrown. This means that it does not focus on detecting deviations or anomalies, it only considers exceptions to be errors. As shown in [32], the T3 algorithm was generally outperformed by the Evosuite[12] which uses genetic algorithms to generate its test cases. Evosuite employs several mutation and evolutionary based techniques to maximize the coverage of their test cases with a minimal amount of test cases. While this increases the coverage and accuracy of finding errors, the only errors it is able to find are predefined errors such as exceptions or timeouts. TESTAR [10] is an open source tool for automated software testing that is continuously being updated and improved. It is used for testing the GUI of applications. In TESTAR Q-learning is used for finding the best test actions. This allows for deep exploration and exploitation of all the possible actions. Executing an action in a specific state provides the agent with a reward, the algorithm explores the tree of actions based on a maximum reward and discount parameters. These parameters influence the exploration and exploitation of the search space. By maximizing its total reward the algorithm can find the action space most likely to find errors in the program.

A different perspective on anomaly detection is known as invariant detection. Instead of trying to find anomalous behaviour, invariant detection attempts to find the statements and relations that always hold true for an application. For instance, a program where $x = 2y + 3z$ and $a \leq x \leq b$ always hold true. Detecting such invariants can be used to perform anomaly detection by observing when such invariants no longer hold true. The Daikon system [9] is one example of such invariant mining. Daikon can automatically discover likely invariants for a program as well as conditional invariants or implications. It discovers such invariants by first instrumenting the target program to trace certain variables. Then by observing these traced variables it deduces invariants over both the observed variables and the variables derived from those. The invariant patterns that Daikon finds are called *specifications* which indicate some aspects of the behaviour of the system. Trying to detect these specifications is known as *specification mining*. Other ways of specification mining exist, another example is defining artificial specifications using a neural network [33]. Here, a Feed forward Neural Network (FNN) is used to mine for artificial specifications. An advantage of using a neural network approach is that it can give continuous information about the state of the program. Additionally, it can handle a lot of input variables and detect connections and correlations between them. However, the challenge of using neural networks is generating enough properly labeled data for training. As mentioned in the paper the training can be done with back propagation requiring a set of sample inputs, or it can be done using an incremental learning mode which is easier, but might result in a weaker specification. Trying both of these training methods, it was shown to have promising performance with a True Positive rate of about 60-70% and a False Positive rate around

5%. This shows that it could be a very powerful method for generating artificial specifications for your program.

By doing specification mining as seen above, the specifications found can be used to do *runtime verification* [6]. This means checking the program at runtime against these formal specifications. Doing such runtime verification can tell you whether the program is behaving correctly or not. This is a common way of performing anomaly detection in software. However, these specifications aren't always clear and cannot always be found. In games it is usually more difficult to find such system specifications as many of the game's systems are influenced by players, making variable observations over different runs much more inconsistent. Therefore, automatic testing tools commonly used for software can be difficult to use on interactive games. Different systems have been researched and developed for automatic testing in games, which will be discussed in the next session.

2.2.2 Automatic Games Testing

ICARUS [30] shows a framework for unsupervised automated game testing. By narrowing down their scope and focusing on a specific genre of games, point-and-click adventure games, they managed to create a proper framework for automatically testing their games. Their framework can work fully unsupervised, meaning they do not need training data for their algorithm. Due to their choice of genre, they could easily generate a tree of actions and state transitions since in adventure games the players mainly have to find the correct combinations of items and correct sequences of actions to perform. This makes it a lot easier to explore the possible game states in the game. However, their technique is fairly limited. ICARUS is only able to detect common errors such as blocking or crashes and is not able to detect other, less clearly defined anomalous behaviour. Additionally, their state exploration is similar to a unit test which makes it much easier to detect anomalies. Simply by comparing a run to previous runs of the game, the program can observe if the runs behaved identical and reporting an anomaly if they behaved differently.

Another tool is Prowler [1] which demonstrates tools for AI-powered automated games testing. Their framework can train an agent to accomplish a goal in the game world. Their example shows an agent trying to reach a door in a room. By having an agent try to reach the same goal every time, developers can observe metrics such as CPU usage, GPU usage, framerates and completion time for each run. By comparing these metrics with previous runs developers can identify if problems occurred after changes in the program. This is more similar to some of the exploratory algorithms mentioned above such as T3 [32] and EvoSuite [12].

A final example of an automated game testing framework is Crushinator [35]. It promises to eliminate the need for a lot of beta testing in games. The framework was designed for performing server load testing and event-driven systems. The framework provides developers a way of simulating lots of clients and testing a program's limits. The crushinator framework focuses on performance testing, while this can detect certain systems failures we are more interested in functional testing of programs. Our research focuses more on finding anomalous behaviour under normal conditions, making this framework less relevant.

Overall, not a lot of generic automated game testing frameworks exist. Most frameworks seem to be intended for internal use within the companies that developed them for the type of games they're working on. Others focus more on exploring the game state, rather than detecting anomalies within that game state. We intend to explore this field and research possibilities for detecting anomalies other than trivial crashes or freezes in games.

3 Research Goals

3.1 Research Questions

In this paper we will research the application and accuracy of anomaly detection algorithms in games. We focus on the automatic detection of non-blocking, non-crashing bugs i.e. faults in the program which cause unexpected or abnormal behaviour but do not crash the game. For our research we seek to provide insight into the following research questions:

1. Can anomaly detection be performed for autonomously detecting anomalous behaviour of variables in games while limiting the number of false positives and false negatives?
   (a) Which algorithm achieves the best accuracy?
   (b) What degree of variable deviation can we detect?
   (c) What factors limit the number of false positives and false negatives?

These research questions can be answered by conducting experiments with the different algorithms we have chosen from our literature study. To determine whether anomaly detection algorithms can provide us with meaningful automatic bug detection methods, we will conduct the experiments on an open-source actively played video game. By injecting our own bugs/anomalies into the game we can determine if the anomaly detection algorithms are capable of detecting the bugs we inserted. These injected anomalies can also be used for measuring the accuracy of our algorithms by measuring when our injected anomalies are detected by the anomaly detection algorithms. We will be experimenting with several different parameters to find out what factors affect the accuracy of our algorithms.

Due to time constraints, we have decided to restrict our domain to a case study for a single game. However, as is further explained below, OpenTTD is a proper representation of commercial games and a good product to perform experiments on further motivated by [36]. OpenTTD has also been used as a case study before in other research such as [20]. Therefore, this case study should allow us to give insight into the answers of our research questions stated above.

3.2 Research Objective

The objective of our research is to provide insight into the possibilities of automating parts of the Quality Assurance(QA) process during game development. We would like to provide valuable, fundamental research for creating a generic, genre-independent framework for autonomous games testing. We aim to find methods to improve and speed up the development process of video games and to try and give the functional testing of games, which is quite unstructured by nature, a more structured approach. Ultimately, the goal of this research is to contribute to the game development industry as a whole by improving the game development cycle.

4 Algorithms

In this section we will give an explanation as well as a motivation of the algorithms chosen for our research. Given our domain and because we would like to have high generalizability, we are looking for unsupervised learning techniques. Therefore, the algorithms chosen are three clustering algorithms: KNN[34], LOF[5] and LOCI[28] and one clustering-like neural network algorithm: the Self-Organizing Map (SOM)[21]. Next follows the explanation and motivation of these algorithms.

4.1 KNN - K-Nearest Neighbours

K-Nearest Neighbours is a simple machine learning algorithm. The basic idea of KNN is that it checks for a given data point its k nearest neighbours and assigning it to the most frequent label among these k neighbours.

Training the KNN simply consists of storing feature vectors with their corresponding class labels in an n-dimensional space. Then, for classification, new data points are tested against this training set using a distance function (e.g. Euclidean distance, Manhattan distance, etc.). It only checks the k neighbours that are nearest based on this distance function. The classification of the new data point is based on these k neighbours, the label that is most frequent among these neighbours is the one that is assigned to our new data point. Pseudocode for this can be seen in Algorithm 1. A visual representation of the KNN algorithm can be found in Figure 2

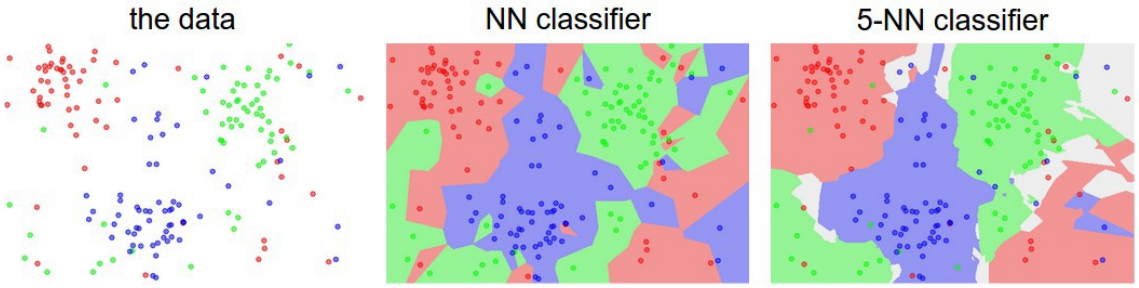


Figure 2: The KNN classification

Result: Classification of new datapoint x
// Initialization;
for All points in training set **do**
  Get k points closest to **x**;
end
for k closest points (excluding x) **do**
  Get most frequent label **l**;
end
x.label = **l**;

Algorithm 1: KNN classification

This can be altered to also calculate the longest distance from one point to its k neighbours. For anomaly detection this distance could tell you something about a point's outlierness. You can then set a threshold to determine whether a point can be considered an outlier or not, based on how far its k nearest neighbours are. While this algorithm isn't really state-of-the art anymore, due to the fact that it is fundamental to many clustering algorithms and because it still has decent performance we feel that it is important to include this in our research. It should provide a good baseline performance to compare against the other algorithms.

4.2 LOF - Local Outlier Factor

The Local Outlier Factor(LOF) is an iteration on the KNN algorithm. Calculating the LOF of a point p consists of a few parts. We will go over each of the individual parts first and then explain how they combine into the Local Outlier Factor.

We start with the *k-distance* of a point p. The *k-distance(p)* is defined as the distance to the k-th nearest neighbour of p, meaning that we want to find the distance to a point o where there are *k-1* points closer to p. For instance, a k value of 3 would mean we look for the distance to a point which has two points closer to p, making it the 3rd closest point to p. A k value of 1 would mean we would just return the distance to the point closest to p, since there are 0 points closer to p. Additionally, we define the *k-distance neighbourhood(p)* denoted as $N_k(p)$ as the collection of points whose distance from p is not greater than the *k-distance(p)*. For example for a k of 3 this collection would contain the first, second and 3rd closest points. It is important to note that the size of $N_k(p)$ can be greater than k when multiple points have the same distance to p. For example, if p has 10 points encircling it, each with a distance of 1 the size of $N_k(p)$ would be 10 even for a k of 1.

Next we define $distance(p, o)$ as some distance function between point p and o. Common distance functions are the Manhattan distance or the euclidean distance. In this example we will be using the euclidean distance as our distance function.

Now we can define the $reach\text{-}distance_k(p, o)$ of a point p with respect to point o. This is defined as the maximum between the $k\text{-}distance(o)$ and $distance(p, o)$. An illustration of this can be seen in 3 This means that if point p is far away from o then the reach distance between the two is just the actual distance, in this case their euclidean distance. Alternatively, if p and o are too close, then their euclidean distance is replaced with the $k\text{-}distance(o)$ meaning it will use the distance to the k-th nearest point instead. This acts as a smoothing effect for if there are a lot of p's close to o. The strength of this smoothing effect can be controlled by k.

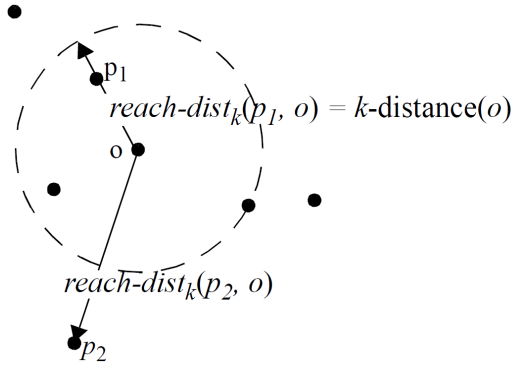


Figure 3: An illustration of the reach distance of a point o

The next component is the Local Reachability Density of p. This is defined as

$$lrd_k(p) = \frac{|N_k(p)|}{\sum_{o \in N_k(p)} reach\text{-}distance_k(p, o)} \tag{1}$$

This is the amount of points in the set of k nearest points to p divided by the sum of the *reach-distance(p,o)* of each of those k nearest points to p. The Local Reachability Density is the inverse of the average reachability distance based on the k nearest neighbours of p.

Finally, all these components are combined into the Local Outlier Factor (LOF). The LOF is defined as

$$LOF_k(p) = \frac{\sum_{o \in N_k(p)} \frac{lrd_k(o)}{lrd_k(p)}}{|N_k(p)|} \tag{2}$$

This captures the degree to which we can call p an outlier. The formula represents the average of the ratio of the local reachability density of p and its k-nearest neighbours. Generally the LOF value means:

- $LOF_k(p) \approx 1$ means p has a similar density as its neighbours
- $LOF_k(p) < 1$ means p has a higher density than its neighbours (inlier)
- $LOF_k(p) > 1$ means p has a lower density than its neighbours (outlier)

The value of the LOF will be approximately 1 for points in a cluster. The more outlying a point is, the higher its LOF value will be. The paper defines a general lower and upper bound for any value of $LOF(p)$.

- $direct_{min}(p) = min\{reach\text{-}dist(p, q) | q \in N_k(p)\}$
- $direct_{max}(p) = max\{reach\text{-}dist(p, q) | q \in N_k(p)\}$
- $indirect_{min}(p) = min\{reach\text{-}dist(q, o) | q \in N_k(p) \text{ and } o \in N_k(q)\}$
- $indirect_{max}(p) = max\{reach\text{-}dist(q, o) | q \in N_k(p) \text{ and } o \in N_k(q)\}$

Intuitively, $direct_{min}(p)$ is the minimum reach distance of all the k closest points to p, which are the direct k neighbours of p. Similarly, $direct_{max}(p)$ is the maximum reach distance of p's direct k neighbours. The indirect neighbours of p are the k nearest neighbours of all of p's direct neighbours. An illustrated example of this can be seen in4 Then, the general lower and upper boundaries of the LOF values of all the points p in the dataset are denoted as:

$$\frac{direct_{min}(p)}{indirect_{max}(p)} \leq LOF(p) \leq \frac{direct_{max}(p)}{indirect_{min}(p)} \tag{3}$$

Intuitively, the closer the LOF value is to 1, the more uniformly distributed the cluster is in which p resides. In a perfectly uniformly distributed dataset, where all distances between neighbouring points are the same we consider all points to be part of the same cluster. Since all distances are

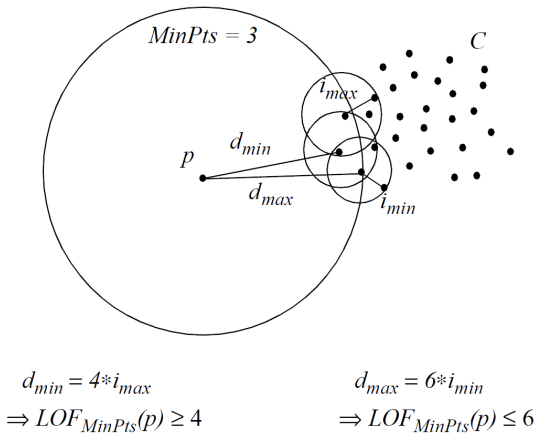


Figure 4: Illustration of the lower and upper bounds of a point p

the same, all *reach-dist*(p, q) will be the same for all points, meaning that the LOF value for all points will always be equal to 1. Only points with significantly different distances to the other points will deviate from 1.

The Local Outlier Factor is an interesting improvement over the traditional KNN algorithm. It can deal much better with the context around a single point and with distributions of varying densities. It would be interesting to see if this results in more accurate results. A closer look at LOF and a comparison of many variations of calculating the Local Outlier Factor can be found in [38].

4.3 LOCI - Local Correlation Integral

The Local Correlation Integral is another improvement over the KNN algorithm. Much like the LOF described above it is much better in dealing with the local context and multiple levels of granularity in the dataset. But one important improvement over LOF is that it is much less sensitive to a user-defined k-value. They do this by introducing the Multi-granularity deviation factor (MDEF). This factor consists of a few parts.

The first part is the number of *r-neighbours* of p denoted as $n(p, r)$. This is the number of neighbours within a range r from point p. The collection of points within range r of p is denoted as $\mathcal{N}(p, r)$. Note that this neighbourhood also always contains p itself.

Next we have $\hat{n}(p, r, k)$ which is the average of $n(p, kr)$ over the set of *r-neighbours*.

$$\hat{n}(p, r, k) = \frac{\sum_{o \in \mathcal{N}(p,r)} n(o, kr)}{n(p, r)} \tag{4}$$

This calculates the average over all data points o in the *r-neighbourhood* of p. The k is known as the sampling neighbourhood of p and is usually chosen to be $k = \frac{1}{2}$. A visualization of the neighbourhood and the sampling neighbourhood can be seen in 5. The value r is in the range of $r_{max} \approx k^{-1} \cdot R_{max}$ (where R_{max} corresponds to the maximum radius between any points in the dataset) and r_{min}, which the paper states is usually determined to be the minimum radius that contains 20 points. The use of two radii for o and p serves to decouple the neighbour size radius kr from the radius r over which we are averaging.

These combine into the MDEF function

$$MDEF(p, r, k) = 1 - \frac{n(p, kr)}{\hat{n}(p, r, k)} \tag{5}$$

Important to note is that the MDEF formula stated in [28] seems to contain an error. In the paper, the denominator in the formula is stated as $\hat{n}(p, \alpha, r)$ which should be changed to $\hat{n}(p, r, \alpha)$ to make their derivation of the MDEF formula correct. The main anomaly detection scheme relies on the standard deviation of the kr-neighbour count over the sampling neighbourhood of p. This is defined as

$$\sigma_{MDEF}(p, r, k) = \frac{\sigma_{\hat{n}}(p, r, k)}{\hat{n}(p, r, k)} \tag{6}$$

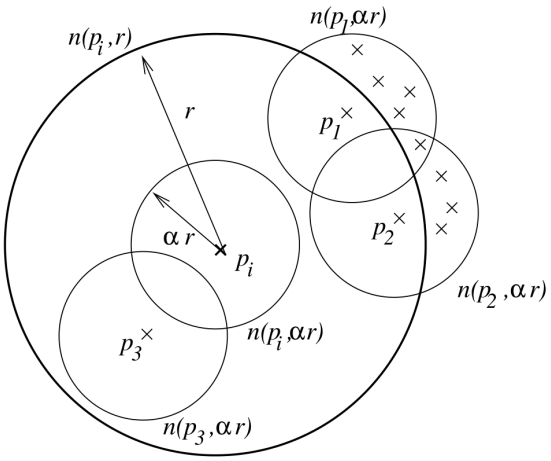


Figure 5: Visualization of the neighbourhood and sampling neighbourhood used in the LOCI algorithm

where

$$\sigma_{\hat{n}}(p,r,k) = \sqrt{\frac{\sum_{o\in\mathcal{N}(p,r)}(n(o,kr)-\hat{n}(p,r,k))^2}{n(p,r)}} \tag{7}$$

To flag a point as anomalous, its $MDEF$ score is compared with the σ_{MDEF}. More concretely, a point is flagged as anomalous if, for any $r \in [r_{min}, r_{max}] : MDEF(p,r,k) > l_\sigma \sigma_{MDEF}(p,r,k)$ In all experiments done by [28], they used $l_\sigma = 3$ and the chosen value for k was $k = \frac{1}{2}$. Usually the MDEF value is observed for a range of r values, in most experiments done by the paper the MDEF is observed over the full range of r values from r_{min} to r_{max}, if any of these values reach the threshold defined above, the point can be flagged as anomalous. By using the standard deviation the algorithm solves much of the problems of "magic cut-offs" nearest-neighbour algorithms often face, such as an arbitrarily chosen k-value for KNN or LOF.

In addition to the regular algorithm, the researchers also propose a fast, approximate LOCI algorithm: aLOCI. This works by performing a box count over a grid. Instead of precisely checking the distances to every point, it is possible to approximate that number by simply adding up the amount of points in adjacent cells in the grid. To obtain information at several scales, store cell counts can be efficiently stored in an n-dimensional quad-tree. Each cell of size $2kr$ is subdivided into 2^n subcells, each with size kr until the desired scale is reached (specified in terms of side length or cell count). This cell grid is used to quickly count the amount of surrounding elements around a point p and can give us a really quick approximation of the MDEF score.

The LOCI algorithm was chosen because it is much less dependent on a "magic cut-off" value of k. By taking into account multiple nearby neighbourhoods and their densities, it seems a good improvement over both KNN and LOF in terms of accuracy. In addition, the algorithm can be approximated and performed really quickly using their grid-based aLOCI approach which could be interesting in terms of speed concerns. Because of its improved accuracy and high applicability due to the option of a faster aLOCI approach, it makes it interesting to test against the other algorithms.

4.4 SOM - Self-Organizing Maps

The Self Organizing Map[21] is called a neural network. However, SOM is a special kind of neural network. As mentioned before, SOM is an unsupervised neural network. But its workings are more similar to a clustering-based approach. The "neurons" in the network act more like nodes. The Self-Organizing Map works in Euclidean space, meaning these nodes can be arranged in a single 2-dimensional or 3-dimensional grid. As data is processed, this grid of neurons will try and enclose the dataset. . The algorithm works in an iterative process, the steps are as follows:

- **Step 0:** Randomly position the grid's neurons in the data space

- **Step 1:** Select one data point, this can either be a random datapoint from the dataset or one chosen iteratively.
- **Step 2:** Find the neuron that is closest to the chosen datapoint. This neuron is called the Best Matching Unit (BMU)
- **Step 3:** Move the BMU closer to the datapoint. How far the BMU is moved is determined by the *learning rate*, which decreases after each iteration.
- **Step 4:** Move the BMU's neighbours closer to the data point as well. The closer the neighbours are to the BMU, the more they move. Neighbours are identified by their distance to the BMU, the threshold for this distance decreases as well after each iteration.
- **Step 5:** Update the learning rate and BMU radius.
- **Step 6:** Repeat steps 1-5 until the positions of the neurons have stabilized

The algorithm is fairly simple and will stabilize to a point where the neurons neatly enclose the dataset. The advantage is that, once trained, this algorithm can quickly determine if a point is an outlier or not based on the enclosure of the neurons. However, having just one grid of neurons might mean that this algorithm performs poorly on datasets with multiple clusters. It would be interesting to observe how well the Self Organizing Map performs compared to other clustering algorithms.

5 The Anomaly Detection Framework

In this chapter we introduce our framework to perform autonomous anomaly detection on games. This enables developers to easily incorporate automated testing into their games. We designed the framework to be independent of what game is used specifically. Therefore, it's designed to be agnostic of the data that is used for anomaly detection. The framework can be easily hooked up to games by simply flagging the variables that need to be tracked. Additionally, our modular design allows the user to use any preferred anomaly detection algorithm. Finally, by incorporating a sliding window approach the framework can deal with large datasets and allows it to be used for online anomaly detection. The framework has been designed to be highly flexible so that it can be applied in a broad range of different games across different genres.

	A	**B**	**C**	**D**
A	X	AB	AC	AD
B	X	X	BC	BD
C	X	X	X	CD
D	X	X	X	X

Table 1: Variable combinations

The first step is to make a selection of the variables we would like to observe. The collection of observed variables will be denoted as $\mathcal{V}$. The variables can then be added to our anomaly detection framework which will be used to detect anomalous behaviour in the game. Once all variables have been added, the framework will create charts using all possible combinations of the tracked variables. An example of how these combinations are constructed can be seen in Table 1. The collection of charts generated from these variables will be denoted as $\mathcal{G}$. The amount of charts generated from this step can be calculated as follows: $|\mathcal{G}| = \frac{n!}{k!(n-k)!}$. With n the amount of variables we are tracking and k the amount of variables we combine into one chart. For our framework $k = 2$, as we will only be combining two variables at a time in each chart. As can be observed from the formula, the amount of charts increases exponentially with the amount of variables tracked. During runtime, the framework reads the values from the variables and plots their values in each chart. Then, these values are evaluated using our anomaly detection algorithms.

The evaluation step marks whether the new points are an outlier or not and assigns them an *outlier score* from 0 to 1 which represents the certainty of the point being an outlier, 1 being 100% and 0 being 0% certain that the point is an outlier. Each chart is given an *outlier score* representing that chart's current "outlierness". Next, the *outlier scores* of all charts are added together to give an overall *anomaly score*. This *anomaly score* is used to determine whether the current gamestate is anomalous or not. The threshold for this *anomaly score* is determined by the amount of variables we are tracking. Every variable $v \in \mathcal{V}$ is combined with every other variable in $\mathcal{V}$ except itself $\{v_i \in \mathcal{V} | v_i \neq v\}$. Thus, every variable exists in $|\mathcal{V}| - 1$ charts. When a variable v changes, anomaly detection for that variable is performed. The *outlier score* of each chart containing v is added up and combined into a single *anomaly score*. We set our threshold such that if the *anomaly score* is at least half of the maximum score possible for a single variable, the current state is flagged as anomalous.

$$Anomalous = \begin{cases} true, & \text{if } anomaly\ score >= 0.5 \times (|\mathcal{V}| - 1) \\ false, & \text{otherwise} \end{cases}$$

The threshold is only exceeded when the charts containing v report on average an *outlier score* of at least 0.5, meaning that the charts are on average at least 50% certain the currently observed value of v is an outlier. Every game tick where this threshold is exceeded, the gamestate will be flagged as anomalous.

Our framework uses a sliding window approach. This means the anomaly score is based only on the last n observed points within our window. This approach allows us to drop points once they fall outside of our sliding window, which reduces the resources used for anomaly detection. This enables the analysis of large datasets and supports an online anomaly detection approach.

Currently, the four anomaly detection algorithms discussed in Chapter 4 have been selected as standard for our framework. However, our framework can be used with other anomaly detection algorithms as well. In principle, any anomaly detection algorithm could be used, as long as it can detect outlying points and assign them a certainty score from 0-1. An overview of the framework can be found in Figure 6

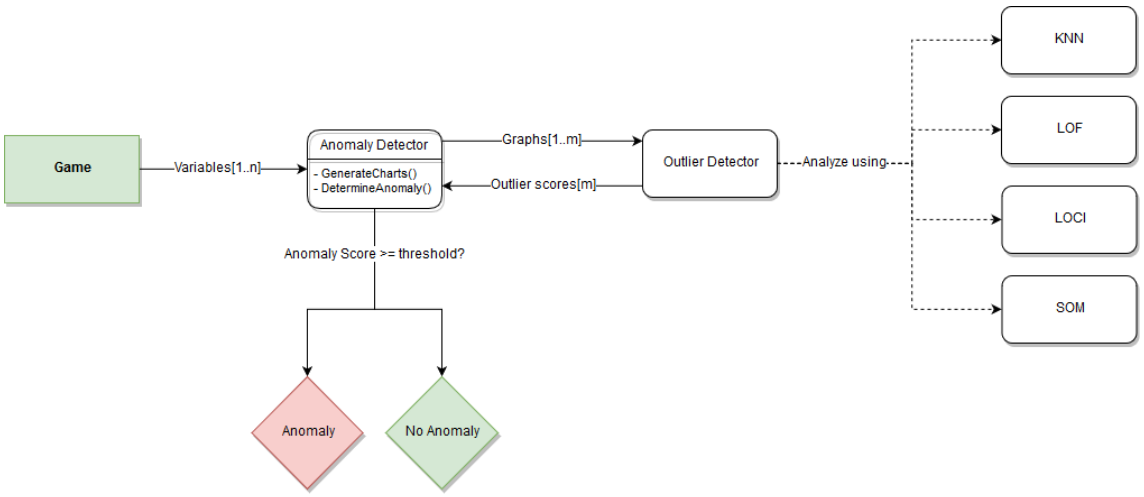


Figure 6: The Anomaly Detection Framework

5.1 Outlier Scores

Outlier detection algorithms commonly flag points in a binary manner as either anomalous or non-anomalous. However, for our anomaly detector we need to derive a point's "outlierness" to determine their outlier score. Similar to a neural network, our algorithms each calculate a point's "outlierness" and feed it into a sigmoid function to produce an outlier score from 0-1. The sigmoid function is defined as:

$$Sigmoid(x) = \frac{L}{1 + e^{-s(x - x_0)}} \tag{8}$$

where:

- **L**: curve's maximum value

- x_0: value of sigmoid's midpoint
- **s**: steepness of the curve

Using this sigmoid function makes sure that the input values can never exceed our boundaries of 0-1.

In the following sections we will discuss how a point's "outlierness" is determined and how outlier scores can be calculated for each of our algorithms.

5.1.1 KNN Outlier Score

For the K-Nearest Neighbours the radius of the k nearest neighbours, denoted as k_r, is calculated for every point p. Then, the average and standard deviation are calculated for the last n k-radii observed, denoted as avg_{k_r} and σ_{k_r} respectively. The size of n is determined by the size of our sliding window. To determine the "outlierness" of a point, its k-radius is compared to the average k-radius observed over the last n points. Its outlier score is determined by the amount of standard deviations the value is removed from the average. For our experiments we used an upper bound of 3 standard deviations, for a normal distribution this means that less than 1% of the values exceeds this threshold. Additionally, this was also the threshold chosen for the LOCI algorithm for the same reason. This value is then fed into a sigmoid function which will approach 1 for larger values and 0 for smaller values (including negative values). Notice that this means even when $k_r < avg_{k_r}$, the sigmoid function will still produce a value from 0-1. In summary, the outlier score of a point p is calculated as follows:

$$outlier\ score = sigmoid(\frac{k_r - avg_{k_r}}{3 * \sigma_{k_r}}) \tag{9}$$

5.1.2 LOF Outlier Score

Since there is no clear threshold stated in [5] our process will be very similar to the above section. For every point p the Local Outlier Factor(LOF) is calculated indicating that point's "outlierness". A higher LOF value means the point has a higher "outlierness". However, without any reference this value is hard to link directly to a uniform outlier score. Therefore, we once again observe the amount of standard deviations the LOF score is removed from the average LOF score accumulated over the last n points. Once more, an upper bound of 3 standard deviations is used and fed into our sigmoid function to get an outlier score. The outlier score is calculated as:

$$outlier\ score = sigmoid(\frac{LOF_p - avg_{LOF_p}}{3 * \sigma_{LOF}}) \tag{10}$$

5.1.3 LOCI Outlier Score

Unlike our previous examples, [28] states a very clear boundary for when a point can be flagged as outlier. The boundary mentioned in the paper is:

$$MDEF(p, r, k) > l_\sigma \sigma_{MDEF}(p, r, k) \tag{11}$$

With $l_\sigma = 3$ as stated in [28]. This gives us a very clear boundary for when a point can be flagged as outlier or not. However, this is a binary threshold flagging a point as anomalous if it exceeds this boundary. In our experiments we calculated the outlier score as the amount of standard deviations the MDEF value exceeds this boundary, which was then once again fed into our sigmoid function. Unlike previous examples, the LOCI algorithm already incorporates an outlier threshold using 3 standard deviations. However, we would still prefer some indication of certainty rather than a binary value. Therefore, our outlier score for the LOCI algorithm is calculated by checking how far this threshold is exceeded with a maximum of one standard deviation. The outlier score for the LOCI was calculated as follows:

$$outlier\ score = sigmoid(\frac{MDEF(p, r, k) - l_\sigma \sigma_{MDEF}(p, r, k)}{\sigma_{MDEF}(p, r, k)}) \tag{12}$$

5.1.4 SOM Outlier Score

The Self-Organizing Map(SOM) has no pre-determined way of defining points as outliers or not. The algorithm builds a cloud of points surrounding a given dataset. The Self-Organizing Map also has a more general purpose outside of outlier detection. Therefore, there is no clear definition stated in [21] for determining whether a point is outlier. Still, the Self-Organizing Map can be used for outlier detection. To determine whether a point is inside our outside of our Self-Organizing Map we construct a convex shape around the SOM nodes. If a point is inside the shape, it is considered an inlier otherwise it is considered an outlier. To determine a point's "outlierness" its distance to the closest edge of the convex shape is measured, illustrated in 7. Then, similar to KNN and LOF, this distance is compared to the average edge distances observed over the last n points. The outlier score is once again determined by the amount of standard deviations removed from the average, maxing out at 3 standard deviations. Once again, this is fed into a sigmoid function to get an outlier score. The outlier score for the SOM is calculated as:

$$outlier\ score = sigmoid(\frac{edgedist - avg_{edgedist}}{3 * \sigma_{edgedist}}) \tag{13}$$

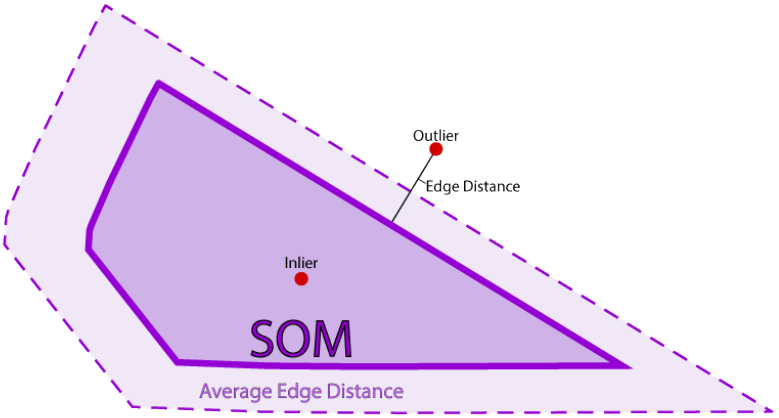


Figure 7: Classification with the Self-Organizing Map

5.2 Event Logging

For our Anomaly Detection framework we decided to make the algorithm event based instead of frame based. The reason for this is that with a frame based approach the baseline used for comparison becomes skewed. Most of the time, the variables we're tracking don't change. Including all frames where the data remains exactly the same introduces a bias against any changes in the data at all. Because we only want to detect if changes in the game are anomalous or not it becomes much more difficult to detect anomalies if all frames are considered. Changing this towards an event based system where we only log variables if their values change compared to the previous frame removes this bias and makes sure we only observe changes in the variables.

6 Experiment Setup

For our experiments we want to emulate realistic player behaviour. In a realistic scenario every play through will be different, this means we want to test our games in a stochastic setting. To avoid a lot of manual playthroughs we make use of AI agents in OpenTTD. For our experiments we use the SimpleAI scripts described in [2]. SimpleAI is compatible for running multiple instances of itself because it builds its routes randomly, this is needed because we want to run experiments with several players at a time. The AI also keeps managing its routes after they're built, selling unprofitable vehicles and replacing old ones. This is useful for our experiments because the AI will keep actively purchasing and selling vehicles. Using the SimpleAI script, we can let several

Figure 8: Open Transport Tycoon Deluxe (OpenTTD)

AI agents play the game and run our anomaly detection algorithms on those playthroughs. This should give us representable player data for our experiments and give us a good foundation to compare our algorithms with each other. By emulating stochastic player behaviour and injecting "random" system failures we should have a realistic setting in which we can research our research questions.

6.1 OpenTTD

For our experiments we will be using Open Transport Tycoon Deluxe (OpenTTD). OpenTTD is an online multiplayer real-time strategy game developed in C++. The game involves controlling a transportation company and earning money by building infrastructure and transportation vehicles. By constructing infrastructure and allowing transport between towns players can indirectly influence the growth of cities and towns placed on the map. The game's map is generated randomly every round, making every round different. The game is simulated in ticks, which means that the simulation of the game is not influenced by framerate. The simulation of the game is entirely dependent on the seed provided. This means we can run the same exact simulation using different Anomaly Detection algorithms uninfluenced by execution times.

OpenTTD is an open source project still actively being developed. It is currently a fully featured game played by many people. This means we have a realistic product similar to projects used in the commercial gaming industry to perform experiments on.

6.2 Anomaly injections

In order to properly measure the accuracy of our anomaly detection algorithms we will need to inject our own anomalies. All anomaly injections will be injected at random intervals based on a random seed we can set. If needed, this will allow us a reasonable amount of control over our experiments. Of course, we must ensure that these injected anomalies are still a realistic depiction of bugs in actual game production. While there is no real way of verifying whether injected anomalies are realistic or not, we can try and emulate "random" system failures. This should give us a realistic scenario in which we can perform experiments to answer our research questions. We can inject everal different types of anomalies into our program:

1. **Variable extrapolation** - On some variable changes, we can randomly increase/decrease the variable by much more than is intended

2. **Variable resets** - On some variable changes, we can randomly set the variable to some unexpected value such as 0, max value or some invalid value.

3. **Function Failures** - On some function calls, we can randomly make the function exit too early

6.3 Variable/Feature selection

It is very important to select the right variables(features) for our experiments. Observing the entire game state would give us too much data to work with, making Anomaly Detection unfeasible. Therefore we have to make a selection of the features which have the greatest impact on the game. The variable selection was done by hand. The variables selected for our experiments were:

- delta roads
- delta railways
- delta road vehicles (trucks, busses, etc.)
- delta rail vehicles (trains, trolleys, etc.)

The selected variables should give a decent representation of the most important elements in the game. These variables will be observed by our anomaly detector and should indicate whether an anomaly has occurred or not.

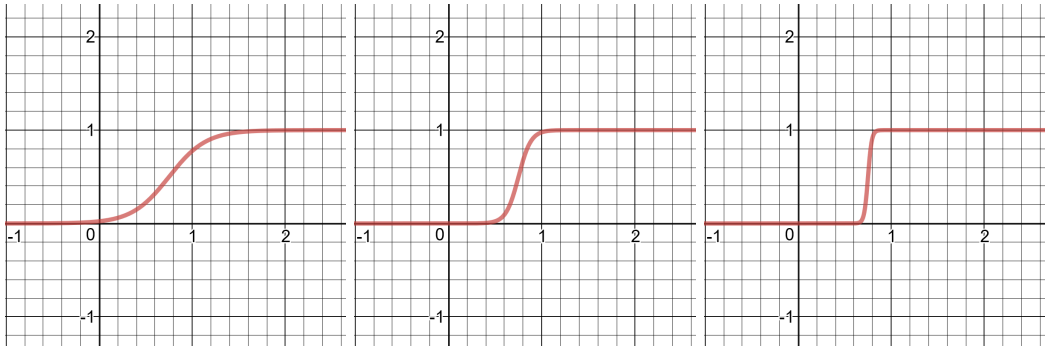
6.4 Experiment Parameters

A couple of important parameters can be identified for our experiments:

- Outlier detection Algorithm: KNN, LOF, LOCI, SOM
- Amount of players
- Type of anomaly insertion: variable reset, variable extrapolation, function failures
- Size of variable extrapolation for anomaly insertion
- Function for calculating outlier score
- Size of the sliding window
- K-size

One important parameter is testing all of the different outlier detection algorithms. Thus, our tests will include comparisons between all the chosen outlier detection algorithms. Since OpenTTD is a multiplayer game it can be played between several players. For consistency, the players in all of our experiments will be using the same SimpleAI scripts (see [2]) to control their behaviour. We will experiment with different amounts of players as we might observe very different results between different amounts of players. We will be testing a couple of different scenario's:

- **1 player:** a scenario with one player is synonymous with single-player games where we can only observe a single player's behaviour. This scenario is more sensitive to anomalies, but it might also be more difficult to establish a baseline describing "normal" behaviour.
- **4 players:** this scenario includes tracking the variables of multiple players at the same time and using their aggregated values for anomaly detection. For companies with massive multiplayer games it can be very beneficial to be able to detect anomalies using the aggregated values of all players. This would make for a very lightweight way to detect anomalous behaviour amongst all players. Accumulating the values of all the players provides us with a decent baseline for normal behaviour. However, if just one player has anomalous behaviour it might be more difficult to detect.



(a) $e^{-}5$ (b) $e^{-}15$ (c) $e^{-}50$

Figure 9: Sigmoid functions

- **1 vs n players:** in this scenario every player's behaviour is compared to the aggregated behaviour of all n players. This tries to combine the benefits of both scenarios by establishing a proper baseline of all n players while maintaining the sensitivity of a single player's anomalous behaviour. It is however, much more resource intensive than the other scenarios as each player's behaviour has to be evaluated against the average.

We have chosen to test just one type of anomaly for our experiments. We will be testing our Anomaly Detector using *variable extrapolation*, meaning we will be increasing or decreasing the amount a variable changes in its function call. This was deemed similar to a variable reset, because resetting a variable to a certain value is in principle an exaggerated version of the variable extrapolation. Additionally, function failures are very unlikely to be detected as it does not affect the values of variables directly. Therefore, those will also not be considered for our experiments. The *Anomaly Size* parameter we are using for our experiments indicates the percentage with which we increase the variable in its function call. For example an *Anomaly Size* of 1000% means we increase the variable with 10 times its value when it is changed in its function.

Threshold

As explained earlier, our outlier scores are all determined by using a sigmoid function. This is commonly used in neural networks, but also very useful in our application. The sigmoid function is defined as:

$$Sigmoid(x) = \frac{L}{1 + e^{-s(x-x_0)}} \tag{14}$$

where:

- **L**: curve's maximum value
- x_0: value of sigmoid's midpoint
- **s**: steepness of the curve

For our experiments we choose an L value of 1, to make sure the function always maps from 0 to 1. The x_0 (midpoint) parameter represents the point at which the function is exactly 0.5. For all experiments we set x_0 to 0.75. Finally, the steepness of the function is determined by the **s** parameter. A higher value represents a greater steepness as indicated by 9. For our experiments we will be comparing threshold values of $s = 5$, $s = 15$ and $s = 50$, in our parameters we indicate this steepness as: e^{-5}, e^{-15} and e^{-50}.

Window Size

Our framework uses a sliding window approach to reduce resources used. The size of this sliding window is one of our parameters. The window size represents the amount of events we consider for our anomaly detection. Meaning that a window size of n uses the last n events for anomaly detection. For our experiments three different window sizes are tested: 20, 75 and MAX. The first two parameters evaluate the last 20 and 75 events respectively. The last parameter of MAX means that the window size is infinite, causing all events that occurred in the entire run to be evaluated and used for anomaly detection.

K

Finally, both LOF and KNN are sensitive to their k parameter. Usually, this parameter is chosen manually per dataset. However, in our case we do not know beforehand what our dataset is going to look like thus it becomes very difficult to select a k value manually. Therefore, we chose to define our k relative to the size of the dataset instead of an absolute value. This way, the k parameter is always consistent in size compared to the amount of data. This is similar to choosing the parameter manually for a given dataset, as the relative size of k for a static dataset is consistent. In our experiments we define our k value in terms of percentage of the total amount of data in the current dataset. We run experiments using a k-value of 25%, 50% and 75%.

7 Results

Based on when our anomaly injections occur, we can extract the amount of true positives(tp), false positives(fp), true negatives(tn) and false negatives(fn) (See Table 2).

	Condition True	**Condition False**
Predicted True	True Positive (tp)	False Positive (fp)
Predicted False	False Negative (fn)	True Negative (tn)

Table 2: Confusion Matrix

Using this information, we will compare our algorithms by looking at several different accuracy measures. First, we look at their precision and recall:

$$Precision = \frac{tp}{tp + fp}$$

$$Recall(TPR) = \frac{tp}{tp + fn}$$

Precision is the amount of correct positive results divided by the number of all positive results. In other words, the precision represents the percentage of correctly predicted events from all predictions made. *Recall*, also known as the True Positive Rate (TPR), is the amount of correct positive results divided by the amount of all samples that should have been identified as positive. This shows what percentage of all positive conditions were actually predicted to be positive. These combine into an F-score which is the harmonic average of the precision and recall:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Next, we calculate the true negative rate (TNR):

$$TNR = \frac{tn}{tn + fp}$$

The *True Negative Rate* represents the amount of correct negative results divided by all results that should have been identified as negative. Finally, we will calculate the overall accuracy of the algorithm:

$$Accuracy = \frac{tp + tn}{tp + tn + fp + fn}$$

The *Accuracy* represents the amount of correctly flagged events divided by all occurred events. This represents from all data, the percentage of data that was labeled correctly.

For our experiments we use the same list of 10 unique seeds to run 10 different simulations of the game. These seeds are used by both the AI and the generation of the game world itself. This means that every seed produces a unique simulation of the game, but using the same seed also makes sure the game produces the same behaviour as long as the same number of players is used. This makes sure that for 1, 4 and 10 players we run the same 10 unique simulations to properly compare the average behaviour of the different parameters.

The values below are represented by the average value over 10 different simulations with their standard deviation over these 10 runs. A single run of the game takes one in-game year which constitutes of 27000 ticks. It takes approximately 10 minutes to go through a single run. The table below shows the average number of events that occurred for the amount of players used in our experiments:

	Number of Events (average ± stdev)
1 player	91 ± 30
4 players	337 ± 47
10 players	896 ± 66

Table 3: Average number of events

7.1 Overview

We will comparing the results of the following parameters:

- **Players:** 1, 4, 10
- **Algorithms:** KNN, LOF, LOCI, SOM
- **Anomaly Size:** 100%, 200%, 1000%
- **Threshold:** $e^{-}5$, $e^{-}15$, $e^{-}50$
- **Window Size:** 20, 75, MAX
- **K-size:** 25%, 50%, 75%

For more explanation of what these paramaters mean exactly you can refer to section Experiment Parameters.

In sections 7.2-7.5 we want to investigate what degree of anomalies can be detected. Additionally, we will investigate whether there is a noticeable difference between evaluating all the aggregated data at once or evaluating the data of a single player against the aggregated data of all players makes a difference.

In section 7.6 we investigate the influence of the threshold parameter for the sigmoid function. Section 7.7 compares different sizes of the sliding window and observes whether that has a large influence on the ability to detect anomalies. Section 7.8 investigates the influence of different K-values on the performance of KNN and LOF, since those are the only algorithms that use a K-value. Finally, in section 7.9 we look at combining the results of different algorithms. We take unions and intersections of different algorithms and investigate whether that can have a positive influence on our ability to detect anomalies. The values in the cells of the tables below are shown as: *average* ± *standard deviation*.

7.2 1 Player

7.2.1 Results

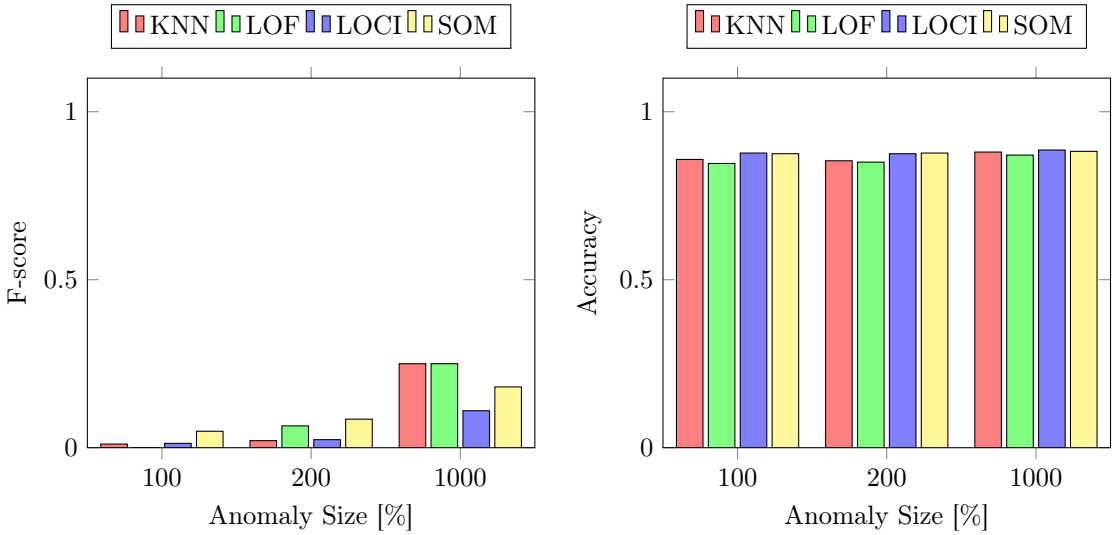


Figure 10: F-score and Accuracy for 1 player

TOTAL (10 runs)	**Precision**	**Recall, TPR**	**F-Score**	**TNR**	**Accuracy**
KNN	0.014 ± 0.045	0.009 ± 0.029	0.011 ± 0.035	0.960 ± 0.010	0.858 ± 0.035
LOF	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.947 ± 0.025	0.846 ± 0.050
LOCI	0.020 ± 0.063	0.009 ± 0.029	0.013 ± 0.040	0.982 ± 0.012	0.877 ± 0.034
SOM	0.125 ± 0.212	0.031 ± 0.054	0.049 ± 0.084	0.976 ± 0.016	0.875 ± 0.047

Table 4: Players: 1 | Anomaly Size: 100% | Threshold: $e^{-}15$ | Window Size: MAX | K: 50% (Experiment Parameters)

TOTAL (10 runs)	**Precision**	**Recall, TPR**	**F-Score**	**TNR**	**Accuracy**
KNN	0.031 ± 0.065	0.016 ± 0.035	0.021 ± 0.045	0.955 ± 0.014	0.854 ± 0.040
LOF	0.107 ± 0.167	0.052 ± 0.070	0.065 ± 0.085	0.945 ± 0.029	0.850 ± 0.050
LOCI	0.033 ± 0.105	0.018 ± 0.057	0.024 ± 0.074	0.979 ± 0.009	0.875 ± 0.035
SOM	0.198 ± 0.221	0.056 ± 0.068	0.085 ± 0.100	0.975 ± 0.017	0.877 ± 0.048

Table 5: Players: 1 | Anomaly Size: 200% | Threshold: $e^{-}15$ | Window Size: MAX | K: 50% (Experiment Parameters)

TOTAL (10 runs)	**Precision**	**Recall, TPR**	**F-Score**	**TNR**	**Accuracy**
KNN	0.310 ± 0.228	0.228 ± 0.263	0.249 ± 0.237	0.956 ± 0.009	0.880 ± 0.040
LOF	0.313 ± 0.201	0.211 ± 0.151	0.248 ± 0.170	0.950 ± 0.017	0.871 ± 0.044
LOCI	0.275 ± 0.309	0.068 ± 0.088	0.108 ± 0.136	0.983 ± 0.009	0.886 ± 0.034
SOM	0.270 ± 0.262	0.144 ± 0.202	0.181 ± 0.222	0.969 ± 0.022	0.882 ± 0.053

Table 6: Players: 1 | Anomaly Size: 1000% | Threshold: $e^{-}15$ | Window Size: MAX | K: 50% (Experiment Parameters)

7.2.2 Evaluation

We can see that for KNN, LOF and LOCI the F-scores are almost zero for an anomaly size of 100%. This is likely because the anomaly size is too small and thus deviates too little from the baseline behaviour of one player. These tiny fluctuations also occur for normal behaviour when a

player decides to buy multiple tracks or vehicles in one frame. Thus, an anomaly size of just 100% is hard to differentiate from normal behaviour by observing just one player. This results in little to no true positives for KNN, LOF and LOCI which explains the extremely low F-scores. The SOM has a slightly better detection rate, with quite some standard deviation. This is likely due to the fact that it can quickly establish a rough baseline behaviour by constructing maps around the observed behaviour of the player in the first stages of the game. Depending on how good this initial rough baseline fits the actual behaviour the SOM can have either a decent or a very low detection rate. From the results we can see that some runs showed decent enough detection rates to increase the average F-score. For all algorithms observed, The rate of true negatives is quite high, roughly around 95% and higher. This means that the algorithms tested have few false positives relative to the total amount of events that occurred. The accuracy for the algorithms is also similar around 85%.

For an anomaly size of 200% the algorithms tested start detecting some anomalies. KNN and LOCI still have very low amounts of true positives, resulting in very low f-scores. LOF and SOM seem to be performing a bit better, even though both still have very low True Positive Rates. Overall, almost no anomalies are detected, which causes larger differences in F-scores with just a few true positives. The TNR and Accuracy scores of all algorithms remain roughly the same.

Once an anomaly size of 1000% is reached, the anomaly detection algorithms start showing acceptable detection rates. Best performing are KNN and LOF with F-scores around 0.25. Both show a precision scores around 0.31, showing that around 30% of all positive predictions made were anomalies. Both also show a recall of >20%, which means that around 20% of all anomalies inserted were detected by KNN and LOF. LOCI shows the worst F-score here. This seems to be mainly caused by its low recall score compared to the other algorithms. It appears that for this case, the LOCI algorithm is only able to detect a small amount of the inserted anomalies. For this anomaly size SOM was outperformed by both KNN and LOF. It appears that once the "outlierness" of the anomalies becomes large enough, the rough baseline created by SOM is quickly outperformed by other outlier detection algorithms. Overall, the accuracy and true negative rate is consistent for all anomaly sizes, with only a slight improvement in accuracy for the larger anomaly sizes due to the increase in true positives.

7.3 4 players

7.3.1 Results

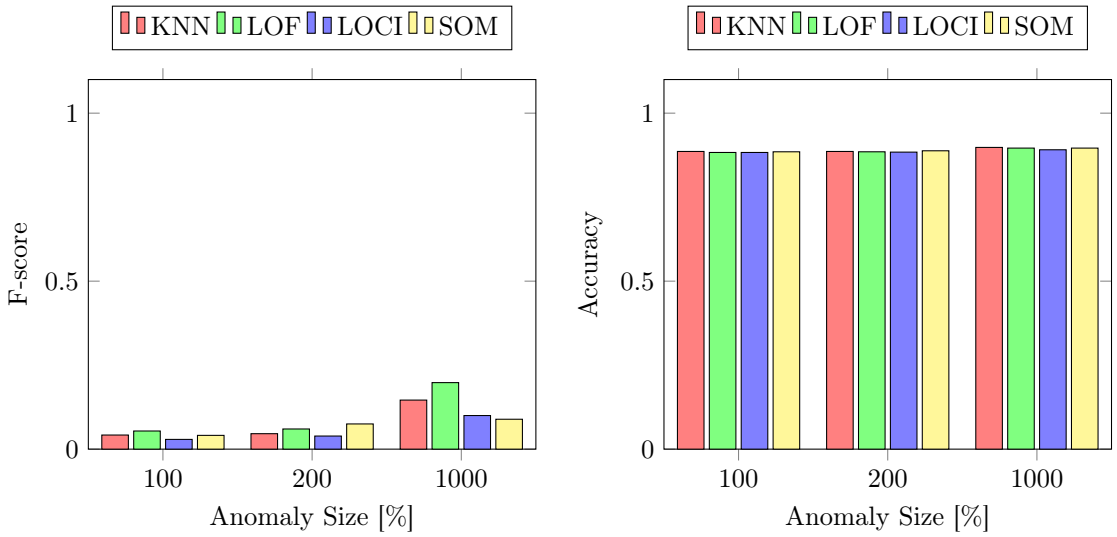


Figure 11: F-score and Accuracy for 4 players

TOTAL (10 runs)	**Precision**	**Recall, TPR**	**F-Score**	**TNR**	**Accuracy**
KNN	0.138 ± 0.183	0.025 ± 0.029	0.042 ± 0.048	0.975 ± 0.014	0.886 ± 0.018
LOF	0.135 ± 0.151	0.035 ± 0.030	0.054 ± 0.050	0.971 ± 0.014	0.883 ± 0.017
LOCI	0.080 ± 0.141	0.019 ± 0.030	0.029 ± 0.047	0.972 ± 0.008	0.883 ± 0.012
SOM	0.086 ± 0.122	0.028 ± 0.042	0.041 ± 0.061	0.974 ± 0.017	0.885 ± 0.015

Table 7: Players: 4 | Anomaly Size: 100% | Threshold: $e^{-}15$ | Window Size: MAX | K: 50% (Experiment Parameters)

TOTAL (10 runs)	**Precision**	**Recall, TPR**	**F-Score**	**TNR**	**Accuracy**
KNN	0.148 ± 0.201	0.028 ± 0.033	0.046 ± 0.056	0.975 ± 0.013	0.886 ± 0.017
LOF	0.158 ± 0.171	0.038 ± 0.028	0.060 ± 0.047	0.972 ± 0.013	0.885 ± 0.016
LOCI	0.101 ± 0.139	0.025 ± 0.029	0.039 ± 0.046	0.972 ± 0.009	0.884 ± 0.011
SOM	0.186 ± 0.136	0.049 ± 0.047	0.075 ± 0.068	0.975 ± 0.015	0.888 ± 0.015

Table 8: Players: 4 | Anomaly Size: 200% | Threshold: $e^{-}15$ | Window Size: MAX | K: 50% (Experiment Parameters)

TOTAL (10 runs)	**Precision**	**Recall, TPR**	**F-Score**	**TNR**	**Accuracy**
KNN	0.324 ± 0.178	0.097 ± 0.057	0.146 ± 0.076	0.980 ± 0.007	0.898 ± 0.011
LOF	0.349 ± 0.111	0.141 ± 0.057	0.198 ± 0.067	0.973 ± 0.007	0.896 ± 0.010
LOCI	0.210 ± 0.078	0.068 ± 0.036	0.100 ± 0.045	0.975 ± 0.007	0.891 ± 0.012
SOM	0.235 ± 0.181	0.057 ± 0.045	0.089 ± 0.068	0.982 ± 0.009	0.896 ± 0.015

Table 9: Players: 4 | Anomaly Size: 1000% | Threshold: $e^{-}15$ | Window Size: MAX | K: 50% (Experiment Parameters)

7.3.2 Evaluation

For both an anomaly size of 100% and an anomaly size of 200% we see similar results. The F-scores are very low for all algorithms tested, meaning that both precision and recall scores are low. This is likely explained by the fact that anomalies occur for a single player at a time. This makes it much harder to detect anomalies when observing the aggregated value of all 4 players. When compared to the single player tests, we see slightly higher F-scores. However, this is likely due to the fact that much more events occur with 4 players, which means there are much more opportunities to label anomalies correctly. Because of this increase in events, the precision and recall scores are pushed above 0 for most runs resulting in slightly higher F-scores.

Once again, the algorithms start properly detecting anomalies at an anomaly size of 1000%. When compared to the experiment with 1 player, the F-scores are lower for all algorithms. This can be explained by reduced impact of a single player on the aggregated value of all 4 players. When an anomaly occurs in one player, it is much harder to detect when the values of all 4 players combined are analyzed. This is supported when we look at the values of the precision and recall. While the precision shows similar results, the recall is much lower for all algorithms which means that of all anomalies that occurred much less are detected when compared to the single player test. The highest F-score of 0.198 is achieved by LOF, followed by KNN with an F-score of 0.146. Once again, LOF and KNN achieve the highest F-scores. True negative rate and accuracy have risen slightly, with TNR roughly around 97% and accuracy around 89%.

7.4 1v4 players

7.4.1 Results

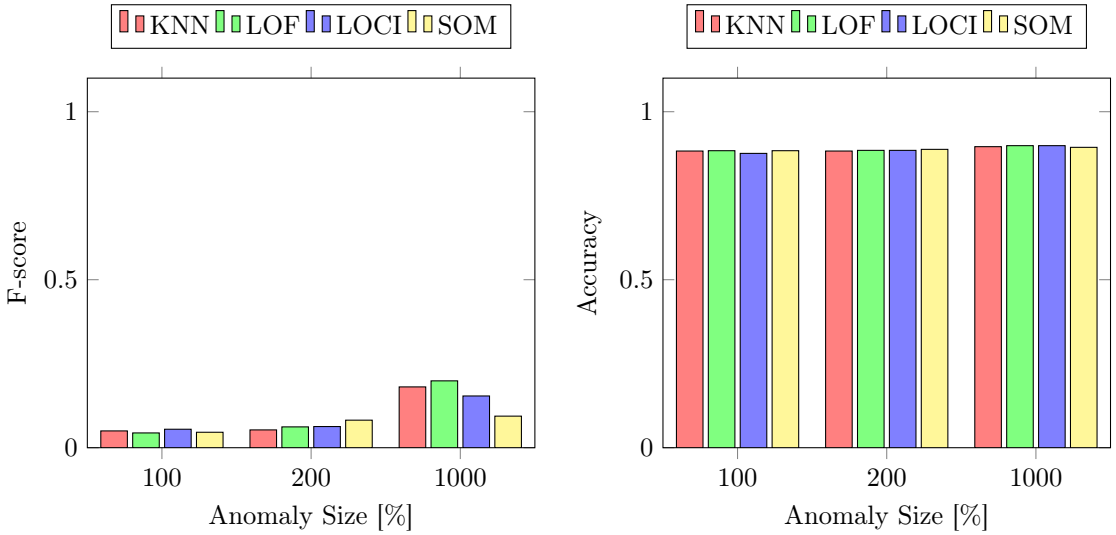


Figure 12: F-score and Accuracy for 1v4 players

TOTAL (10 runs)	**Precision**	**Recall, TPR**	**F-Score**	**TNR**	**Accuracy**
KNN	0.143 ± 0.185	0.031 ± 0.032	0.050 ± 0.053	0.973 ± 0.012	0.883 ± 0.016
LOF	0.117 ± 0.164	0.028 ± 0.033	0.044 ± 0.052	0.974 ± 0.008	0.884 ± 0.014
LOCI	0.100 ± 0.109	0.039 ± 0.037	0.055 ± 0.054	0.965 ± 0.008	0.876 ± 0.013
SOM	0.124 ± 0.172	0.031 ± 0.043	0.046 ± 0.061	0.974 ± 0.012	0.884 ± 0.014

Table 10: Players: 1v4 | Anomaly Size: 100% | Threshold: $e^{-}15$ | Window Size: MAX | K: 50% (Experiment Parameters)

TOTAL (10 runs)	**Precision**	**Recall, TPR**	**F-Score**	**TNR**	**Accuracy**
KNN	0.151 ± 0.183	0.033 ± 0.032	0.053 ± 0.053	0.973 ± 0.012	0.883 ± 0.016
LOF	0.155 ± 0.154	0.040 ± 0.031	0.062 ± 0.049	0.974 ± 0.009	0.885 ± 0.013
LOCI	0.114 ± 0.108	0.044 ± 0.037	0.063 ± 0.055	0.965 ± 0.007	0.877 ± 0.011
SOM	0.217 ± 0.192	0.052 ± 0.038	0.082 ± 0.058	0.976 ± 0.009	0.888 ± 0.015

Table 11: Players: 1v4 | Anomaly Size: 200% | Threshold: $e^{-}15$ | Window Size: MAX | K: 50% (Experiment Parameters)

TOTAL (10 runs)	**Precision**	**Recall, TPR**	**F-Score**	**TNR**	**Accuracy**
KNN	0.355 ± 0.149	0.127 ± 0.069	0.181 ± 0.083	0.976 ± 0.009	0.896 ± 0.013
LOF	0.381 ± 0.120	0.138 ± 0.067	0.199 ± 0.083	0.978 ± 0.006	0.899 ± 0.012
LOCI	0.277 ± 0.137	0.109 ± 0.054	0.154 ± 0.072	0.971 ± 0.009	0.890 ± 0.015
SOM	0.260 ± 0.145	0.060 ± 0.040	0.094 ± 0.059	0.981 ± 0.009	0.894 ± 0.016

Table 12: Players: 1v4 | Anomaly Size: 1000% | Threshold: $e^{-}15$ | Window Size: MAX | K: 50% (Experiment Parameters)

7.4.2 Evaluation

This experiment attempts to create a good baseline by using the average of multiple players while comparing every player individually to this baseline to maintain sensitivity. For smaller anomaly sizes, this doesn't seem to impact the results by much. The F-scores are only slightly higher for an

anomaly size of 100% and 200%. There is a slight, but noticeable difference in performance for the LOCI algorithm. With 4 players, LOCI achieved an F-score of 0.029 for an anomaly size of 100%. Now, by comparing each player to the baseline of 4 players, LOCI achieves an F-score of 0.055.

For the larger anomaly size of 1000% we see something interesting. When compared to the run with just 4 players KNN and LOCI show significantly better results. However, both LOF and SOM remain roughly the same. It appears the LOF algorithm is still performing effectively when observing the aggregated values of all 4 players. True negative rate and accuracy are similar to what we observed before with an accuracy of roughly 89% and TNR of around 97%.

7.5 1v10 players

7.5.1 Results

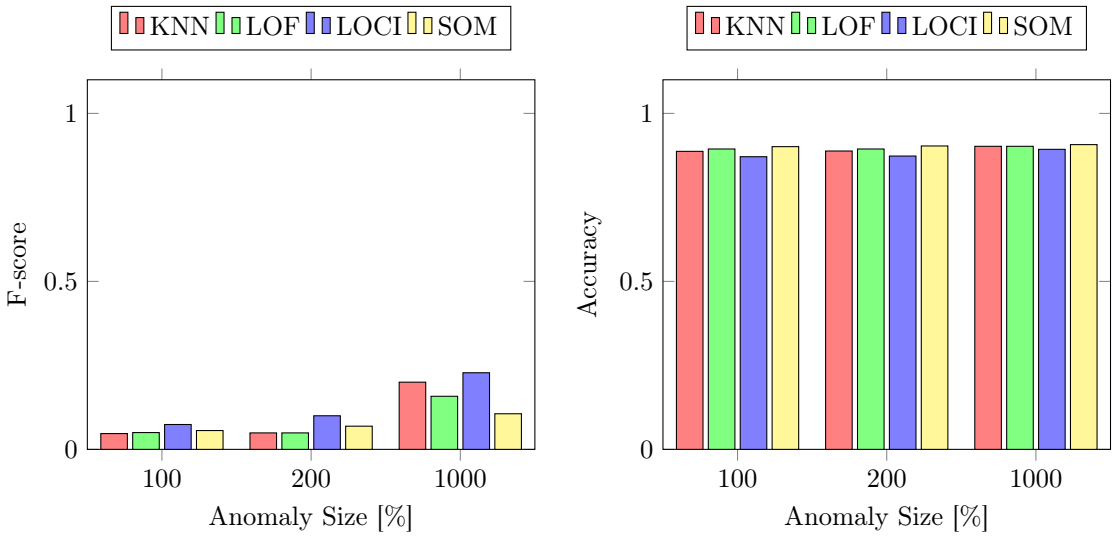


Figure 13: F-score and Accuracy for 1v10 players

TOTAL (10 runs)	**Precision**	**Recall, TPR**	**F-Score**	**TNR**	**Accuracy**
KNN	0.097 ± 0.072	0.031 ± 0.022	0.047 ± 0.033	0.971 ± 0.008	0.887 ± 0.014
LOF	0.121 ± 0.085	0.032 ± 0.025	0.050 ± 0.038	0.979 ± 0.005	0.894 ± 0.012
LOCI	0.103 ± 0.053	0.058 ± 0.031	0.074 ± 0.039	0.951 ± 0.006	0.871 ± 0.013
SOM	0.181 ± 0.124	0.034 ± 0.029	0.056 ± 0.044	0.987 ± 0.005	0.901 ± 0.011

Table 13: Players: 1v10 | Anomaly Size: 100% | Threshold: $e^{-}15$ | Window Size: MAX | K: 50% (Experiment Parameters)

TOTAL (10 runs)	**Precision**	**Recall, TPR**	**F-Score**	**TNR**	**Accuracy**
KNN	0.101 ± 0.073	0.033 ± 0.023	0.049 ± 0.035	0.972 ± 0.008	0.888 ± 0.014
LOF	0.119 ± 0.116	0.032 ± 0.028	0.049 ± 0.044	0.979 ± 0.008	0.894 ± 0.014
LOCI	0.138 ± 0.064	0.078 ± 0.039	0.100 ± 0.048	0.952 ± 0.006	0.873 ± 0.014
SOM	0.219 ± 0.141	0.042 ± 0.035	0.069 ± 0.053	0.988 ± 0.005	0.903 ± 0.010

Table 14: Players: 1v10 | Anomaly Size: 200% | Threshold: $e^{-}15$ | Window Size: MAX | K: 50% (Experiment Parameters)

TOTAL (10 runs)	**Precision**	**Recall, TPR**	**F-Score**	**TNR**	**Accuracy**
KNN	0.366 ± 0.081	0.139 ± 0.029	0.200 ± 0.038	0.976 ± 0.006	0.902 ± 0.012
LOF	0.376 ± 0.097	0.101 ± 0.019	0.158 ± 0.030	0.983 ± 0.006	0.905 ± 0.013
LOCI	0.306 ± 0.081	0.185 ± 0.075	0.228 ± 0.078	0.961 ± 0.007	0.893 ± 0.012
SOM	0.324 ± 0.211	0.065 ± 0.043	0.106 ± 0.067	0.989 ± 0.006	0.907 ± 0.010

Table 15: Players: 1v10 | Anomaly Size: 1000% | Threshold: $e^{-}15$ | Window Size: MAX | K: 50% (Experiment Parameters)

7.5.2 Evaluation

By observing more players we should be able to establish a better baseline. Comparing individual players to this baseline could give us better detection rates. For the smaller anomaly sizes of 100% and 200% we see that mostly the LOCI algorithm is affected. This is likely because LOCI uses the most contextual information of all algorithms. Using this information, LOCI is the most capable to take advantage of the stronger baseline. KNN, LOF and SOM show similar results to the 1v4 experiment.

When looking at an anomaly size of 1000% we see a big jump in F-score for KNN, LOF and LOCI. It appears KNN can only take advantage of the improved baseline with a larger anomaly size. When compared to 1v4 players, KNN shows slightly better performance with an F-score of 0.18 for 1v4 players and an F-score of 0.20 for 1v10 players. Interestingly LOF does not take advantage of an increased baseline. It appears to have decreased in performance when compared to both the 1v4 players and the 4 players experiments. Previously, LOF achieved an F-score of 0.198 and 0.199 for 4 players and 1v4 players respectively. Now, its F-score decreased to 0.158 for an anomaly size of 1000%. It seems LOF does not take advantage of comparing individual players to a baseline established by multiple players. That could explain the worse performance for 1v10 players, because without the advantage of comparing individual values to the baseline it would be even harder to detect anomalies in a group of 10 players when compared to 4 players. SOM appears to have stable performance for all experiments. True Negative Rate and Accuracy also seem similar to the other experiments with TNR around 98% and Accuracy around 90%.

7.6 Anomaly score threshold

7.6.1 Results

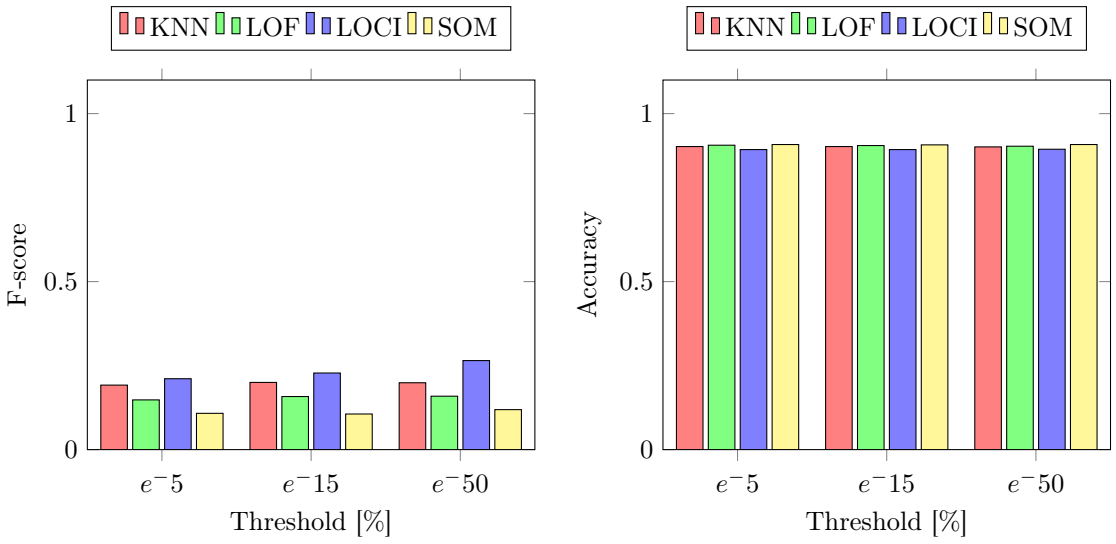


Figure 14: F-score and Accuracy for different threshold values

TOTAL (10 runs)	**Precision**	**Recall, TPR**	**F-Score**	**TNR**	**Accuracy**
KNN	0.355 ± 0.086	0.133 ± 0.036	0.192 ± 0.048	0.976 ± 0.006	0.902 ± 0.011
LOF	0.377 ± 0.095	0.093 ± 0.016	0.148 ± 0.025	0.984 ± 0.005	0.906 ± 0.012
LOCI	0.297 ± 0.079	0.166 ± 0.060	0.211 ± 0.067	0.963 ± 0.007	0.893 ± 0.012
SOM	0.329 ± 0.205	0.066 ± 0.041	0.108 ± 0.064	0.989 ± 0.006	0.908 ± 0.009

Table 16: Players: 1v4 | Anomaly Size: 1000% | Threshold: $e^{-}5$ | Window Size: MAX | K: 50% (Experiment Parameters)

TOTAL (10 runs)	**Precision**	**Recall, TPR**	**F-Score**	**TNR**	**Accuracy**
KNN	0.366 ± 0.081	0.139 ± 0.029	0.200 ± 0.038	0.976 ± 0.006	0.902 ± 0.012
LOF	0.376 ± 0.097	0.101 ± 0.019	0.158 ± 0.030	0.983 ± 0.006	0.905 ± 0.013
LOCI	0.306 ± 0.081	0.185 ± 0.075	0.228 ± 0.078	0.961 ± 0.007	0.893 ± 0.012
SOM	0.324 ± 0.211	0.065 ± 0.043	0.106 ± 0.067	0.989 ± 0.006	0.907 ± 0.010

Table 17: Players: 1v4 | Anomaly Size: 1000% | Threshold: $e^{-}15$ | Window Size: MAX | K: 50% (Experiment Parameters)

TOTAL (10 runs)	**Precision**	**Recall, TPR**	**F-Score**	**TNR**	**Accuracy**
KNN	0.357 ± 0.076	0.139 ± 0.027	0.199 ± 0.036	0.975 ± 0.007	0.901 ± 0.012
LOF	0.350 ± 0.078	0.104 ± 0.017	0.159 ± 0.025	0.981 ± 0.005	0.903 ± 0.012
LOCI	0.332 ± 0.079	0.224 ± 0.087	0.265 ± 0.085	0.959 ± 0.008	0.894 ± 0.012
SOM	0.342 ± 0.192	0.074 ± 0.048	0.119 ± 0.071	0.989 ± 0.006	0.908 ± 0.010

Table 18: Players: 1v4 | Anomaly Size: 1000% | Threshold: $e^{-}50$ | Window Size: MAX | K: 50% (Experiment Parameters)

7.6.2 Evaluation

We observed various different thresholds for our sigmoid function. A higher value means a steeper sigmoid function. We tested three different thresholds. For the varying thresholds only the LOCI algorithm has a slight increase in F-score for a threshold of $e^{-}50$. The other algorithms show little to no difference when tested with different threshold for the sigmoid funciton. It seems that the threshold chosen for the sigmoid function has little to no influence on the results. This is likely because most anomalies that are detected show rather far outlying behaviour. This would mean that the outlier scores fed into the sigmoid mostly fall into the tail end of the sigmoid function, translating the score to almost 1 for any of the thresholds chosen.

7.7 Window Size

7.7.1 Results

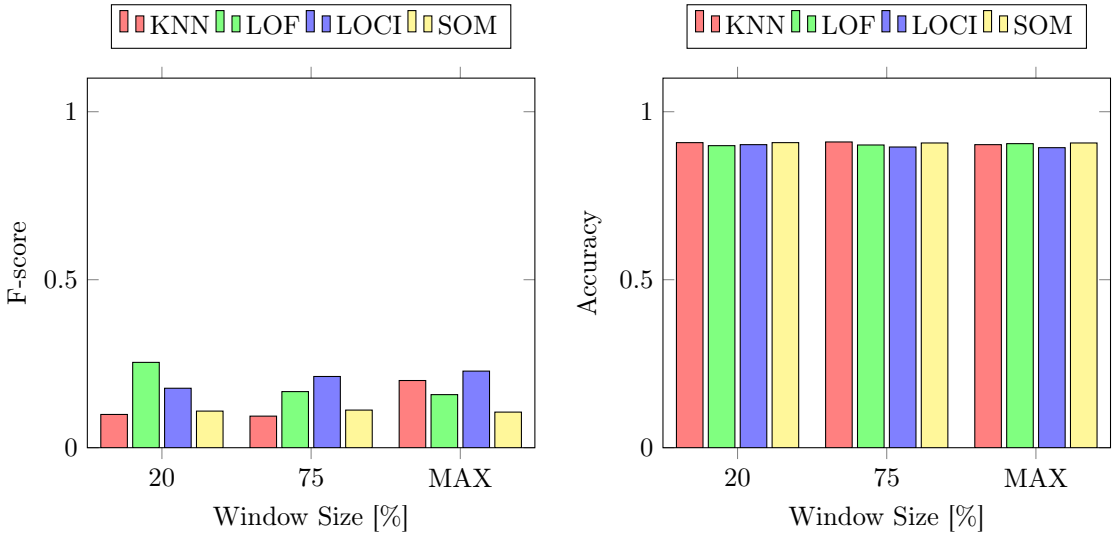


Figure 15: F-score and Accuracy for different window sizes

TOTAL (10 runs)	**Precision**	**Recall, TPR**	**F-Score**	**TNR**	**Accuracy**
KNN	0.370 ± 0.179	0.057 ± 0.031	0.099 ± 0.053	0.991 ± 0.003	0.908 ± 0.014
LOF	0.362 ± 0.066	0.197 ± 0.045	0.254 ± 0.052	0.967 ± 0.003	0.899 ± 0.010
LOCI	0.343 ± 0.066	0.121 ± 0.027	0.177 ± 0.034	0.977 ± 0.006	0.902 ± 0.013
SOM	0.369 ± 0.269	0.066 ± 0.041	0.109 ± 0.065	0.990 ± 0.007	0.908 ± 0.009

Table 19: Players: 1v4 | Anomaly Size: 1000% | Threshold: $e^{-}15$ | Window Size: 20 | K: 50% (Experiment Parameters)

TOTAL (10 runs)	**Precision**	**Recall, TPR**	**F-Score**	**TNR**	**Accuracy**
KNN	0.379 ± 0.154	0.054 ± 0.024	0.094 ± 0.041	0.992 ± 0.003	0.910 ± 0.010
LOF	0.325 ± 0.096	0.113 ± 0.040	0.167 ± 0.056	0.977 ± 0.005	0.901 ± 0.011
LOCI	0.313 ± 0.111	0.162 ± 0.061	0.212 ± 0.076	0.966 ± 0.006	0.895 ± 0.013
SOM	0.334 ± 0.204	0.069 ± 0.042	0.112 ± 0.064	0.989 ± 0.007	0.907 ± 0.009

Table 20: Players: 1v4 | Anomaly Size: 1000% | Threshold: $e^{-}15$ | Window Size: 75 | K: 50% (Experiment Parameters)

TOTAL (10 runs)	**Precision**	**Recall, TPR**	**F-Score**	**TNR**	**Accuracy**
KNN	0.366 ± 0.081	0.139 ± 0.029	0.200 ± 0.038	0.976 ± 0.006	0.902 ± 0.012
LOF	0.376 ± 0.097	0.101 ± 0.019	0.158 ± 0.030	0.983 ± 0.006	0.905 ± 0.013
LOCI	0.306 ± 0.081	0.185 ± 0.075	0.228 ± 0.078	0.961 ± 0.007	0.893 ± 0.012
SOM	0.324 ± 0.211	0.065 ± 0.043	0.106 ± 0.067	0.989 ± 0.006	0.907 ± 0.010

Table 21: Players: 1v4 | Anomaly Size: 1000% | Threshold: $e^{-}15$ | Window Size: MAX | K: 50% (Experiment Parameters)

7.7.2 Evaluation

We also tested different window sizes, meaning the algorithms observed different amounts of values from the total set of values when calculating the outlier score. For KNN we see little to no difference between a window size of 20 and a window size of 75. However, once we set the window size to

encompass all values (MAX) we see an increase in F-score. LOF seems to have an inverse effect, where a smaller window size results in a higher F-score. This can be explained by the fact that for a smaller window size, LOF has much less values to evaluate and the values have less context around them. This can make the algorithm more sensitive to small outliers which would otherwise not be flagged as anomalous when classified using the added context of a larger window size. This can be supported by looking at the recall of the LOF algorithm. We can see that LOF has a recall value of 0.197 for the smallest window size and a value of 0.101 for the maximum window size. This means that for the smaller window size is was able to detect a larger amount of the inserted anomalies. The LOCI algorithm improves slightly with larger window sizes. It has a larger context to consider with a maximum window size, but due to its implementation it will also consider subsets of smaller radii within this set op points which is similar to considering varying window sizes. SOM is mostly unaffected by window size as its mapping is updated with every new point that is observed. This is not dependent on window size. Accuracy and true negative rate are consistent across all window sizes.

7.8 K value

7.8.1 Results

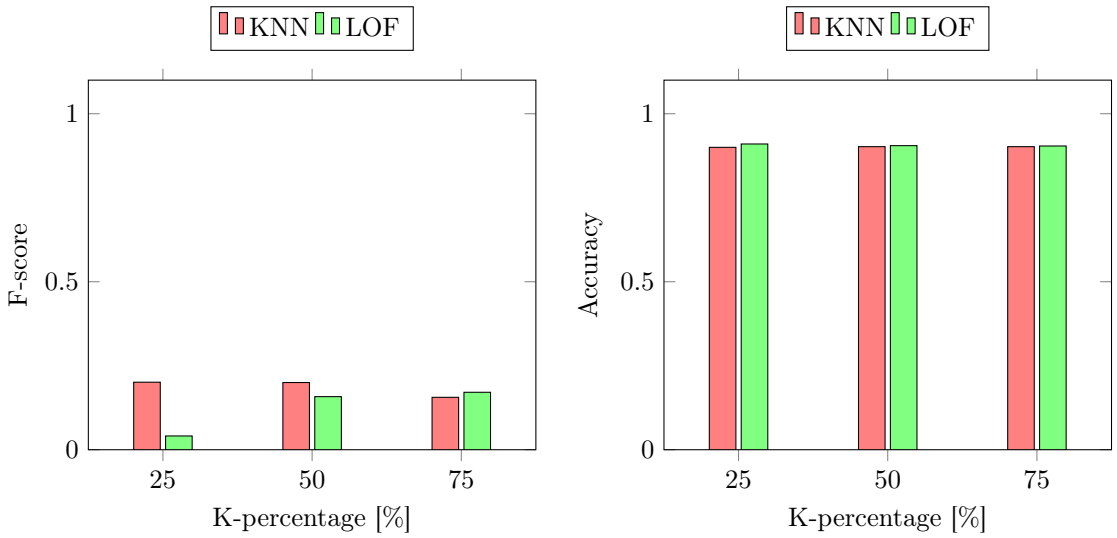


Figure 16: F-score and Accuracy for different k values

TOTAL (10 runs)	**Precision**	**Recall, TPR**	**F-Score**	**TNR**	**Accuracy**
KNN	0.346 ± 0.058	0.144 ± 0.029	0.201 ± 0.034	0.973 ± 0.006	0.900 ± 0.010
LOF	0.359 ± 0.326	0.023 ± 0.026	0.041 ± 0.043	0.995 ± 0.004	0.910 ± 0.011

Table 22: Players: 1v10 | Anomaly Size: 1000% | Threshold: $e^{-}15$ | Window Size: MAX | K: 25% (Experiment Parameters)

TOTAL (10 runs)	**Precision**	**Recall, TPR**	**F-Score**	**TNR**	**Accuracy**
KNN	0.366 ± 0.081	0.139 ± 0.029	0.200 ± 0.038	0.976 ± 0.006	0.902 ± 0.012
LOF	0.376 ± 0.097	0.101 ± 0.019	0.158 ± 0.030	0.983 ± 0.006	0.905 ± 0.013

Table 23: Players: 1v10 | Anomaly Size: 1000% | Threshold: $e^{-}15$ | Window Size: MAX | K: 50% (Experiment Parameters)

TOTAL (10 runs)	**Precision**	**Recall, TPR**	**F-Score**	**TNR**	**Accuracy**
KNN	0.328 ± 0.078	0.107 ± 0.050	0.156 ± 0.068	0.979 ± 0.009	0.902 ± 0.012
LOF	0.357 ± 0.062	0.113 ± 0.023	0.171 ± 0.032	0.980 ± 0.004	0.904 ± 0.011

Table 24: Players: 1v10 | Anomaly Size: 1000% | Threshold: $e^{-}15$ | Window Size: MAX | K: 75% (Experiment Parameters)

7.8.2 Evaluation

For this experiment only KNN and LOF are relevant since those are the only algorithms sensitive to a k-value. The k-value here is shown as a percentage. This represents what percentage of the total amount of values is used as the value for k. For further explanation of this k-percentage refer to section 6.4. We can observe that KNN performs roughly the same for a k-value of 25% and 50%. However, for a larger k-value of 75% its performance decreases. This can be explained by the fact that for a larger k, more context is considered. But if k becomes too large, then it becomes increasingly less sensitive to changes between individual points. That makes it harder to detect anomalies resulting in a decrease in recall values, which can be seen in our data. LOF performs better for larger K values. This is likely because LOF also considers the neighbourhood around a point for its classification. Meaning that for a larger k, a larger neighbourhood is used for classification which can result in better detection rates.

The accuracy and true negative rate are largely unaffected by the different k-percentages.

7.9 Combinations

In this section we will look at the intersections (∩) and unions (∪) of the different tested algorithms. Combining the results of different algorithms could result in a better detection rate or reduced false positives.

7.9.1 Results

TOTAL (10 runs)	**Precision**	**Recall, TPR**	**F-Score**	**TNR**	**Accuracy**
KNN	0.366 ± 0.081	0.139 ± 0.029	0.200 ± 0.038	0.976 ± 0.006	0.902 ± 0.012
LOF	0.376 ± 0.097	0.101 ± 0.019	0.158 ± 0.030	0.983 ± 0.006	0.905 ± 0.013
LOCI	0.306 ± 0.081	0.185 ± 0.075	0.228 ± 0.078	0.961 ± 0.007	0.893 ± 0.012
SOM	0.324 ± 0.211	0.065 ± 0.043	0.106 ± 0.067	0.989 ± 0.006	0.907 ± 0.010
KNN ∪ LOF	0.385 ± 0.065	0.159 ± 0.043	0.222 ± 0.046	0.975 ± 0.006	0.903 ± 0.012
KNN ∪ LOCI	0.341 ± 0.061	0.223 ± 0.058	0.267 ± 0.056	0.959 ± 0.008	0.894 ± 0.012
KNN ∪ SOM	0.360 ± 0.064	0.154 ± 0.031	0.214 ± 0.038	0.973 ± 0.005	0.901 ± 0.009
LOF ∪ LOCI	0.338 ± 0.054	0.214 ± 0.055	0.260 ± 0.053	0.960 ± 0.008	0.894 ± 0.011
LOF ∪ SOM	0.358 ± 0.052	0.124 ± 0.028	0.183 ± 0.037	0.979 ± 0.003	0.903 ± 0.009
LOCI ∪ SOM	0.305 ± 0.084	0.193 ± 0.077	0.234 ± 0.080	0.959 ± 0.006	0.892 ± 0.011
KNN ∪ LOF ∪ LOCI	0.343 ± 0.062	0.227 ± 0.056	0.271 ± 0.056	0.958 ± 0.008	0.894 ± 0.012
KNN ∪ LOF ∪ SOM	0.359 ± 0.062	0.159 ± 0.030	0.218 ± 0.037	0.973 ± 0.005	0.901 ± 0.009
KNN ∪ LOCI ∪ SOM	0.335 ± 0.062	0.227 ± 0.059	0.268 ± 0.058	0.957 ± 0.007	0.892 ± 0.011
LOF ∪ LOCI ∪ SOM	0.327 ± 0.066	0.215 ± 0.065	0.257 ± 0.064	0.958 ± 0.007	0.892 ± 0.010
KNN ∪ LOF ∪ LOCI ∪ SOM	0.337 ± 0.064	0.231 ± 0.057	0.272 ± 0.057	0.956 ± 0.008	0.892 ± 0.011

Table 25: Union(∪) | Players: 1v10 | Anomaly Size: 1000% | Threshold: $e^{-}15$ | Window Size: MAX | K: 50% (Experiment Parameters)

TOTAL	**Precision**	**Recall, TPR**	**F-Score**	**TNR**	**Accuracy**
KNN	0.366 ± 0.081	0.139 ± 0.029	0.200 ± 0.038	0.976 ± 0.006	0.902 ± 0.012
LOF	0.376 ± 0.097	0.101 ± 0.019	0.158 ± 0.030	0.983 ± 0.006	0.905 ± 0.013
LOCI	0.306 ± 0.081	0.185 ± 0.075	0.228 ± 0.078	0.961 ± 0.007	0.893 ± 0.012
SOM	0.324 ± 0.211	0.065 ± 0.043	0.106 ± 0.067	0.989 ± 0.006	0.907 ± 0.010
KNN ∩ LOF	0.373 ± 0.099	0.094 ± 0.017	0.148 ± 0.026	0.984 ± 0.006	0.905 ± 0.013
KNN ∩ LOCI	0.311 ± 0.077	0.101 ± 0.036	0.151 ± 0.048	0.979 ± 0.005	0.901 ± 0.011
KNN ∩ SOM	0.314 ± 0.191	0.050 ± 0.035	0.085 ± 0.057	0.991 ± 0.004	0.908 ± 0.011
LOF ∩ LOCI	0.345 ± 0.106	0.083 ± 0.027	0.133 ± 0.040	0.984 ± 0.005	0.905 ± 0.013
LOF ∩ SOM	0.308 ± 0.193	0.042 ± 0.031	0.073 ± 0.051	0.992 ± 0.004	0.908 ± 0.012
LOCI ∩ SOM	0.322 ± 0.199	0.057 ± 0.039	0.095 ± 0.062	0.991 ± 0.005	0.908 ± 0.011
KNN ∩ LOF ∩ LOCI	0.344 ± 0.105	0.081 ± 0.025	0.129 ± 0.037	0.984 ± 0.005	0.905 ± 0.013
KNN ∩ LOF ∩ SOM	0.298 ± 0.190	0.039 ± 0.028	0.068 ± 0.047	0.992 ± 0.004	0.908 ± 0.012
KNN ∩ LOCI ∩ SOM	0.297 ± 0.180	0.046 ± 0.035	0.078 ± 0.056	0.992 ± 0.004	0.908 ± 0.011
LOF ∩ LOCI ∩ SOM	0.308 ± 0.193	0.042 ± 0.031	0.073 ± 0.051	0.992 ± 0.004	0.908 ± 0.012
KNN ∩ LOF ∩ LOCI ∩ SOM	0.298 ± 0.190	0.039 ± 0.028	0.068 ± 0.047	0.992 ± 0.004	0.908 ± 0.012

Table 26: Intersection(∩) | Players: 1v10 | Anomaly Size: 1000% | Threshold: $e^{-}15$ | Window Size: MAX | K: 50% (Experiment Parameters)

7.9.2 Evaluation

The union(∪) of different algorithms means that for each event the maximum anomaly score is taken from each algorithm. This means that if any algorithm in the union detects an anomaly for that event, it is marked as anomalous. This should increase sensitivity and improve the recall rate, meaning it would detect more of the inserted anomalies. However, it would also mean it would detect more false positives which would decrease our precision. When compared to the singular algorithms we can see a significant increase in recall when algorithms are combined. Especially combinations which contain the LOCI algorithms greatly increase in recall. Notably **KNN∪LOCI** and **KNN∪LOF∪LOCI** show a significant increase in recall while only slightly decreasing in precision. This implies that most algorithms detect the same false positives, while not detecting the same true positives. We can also see a slight decrease in true negative rate and accuracy, but the difference is very small when compared to the singular algorithms. This makes the union effective at increasing the detection rate of true positives while limiting the increase in false positives.

Taking the intersection(∩) of different algorithms means that for each event the minimum anomaly score is taken from each algorithm. This means that the event is only marked anomalous if if all algorithms detect an anomaly for that event. This should result in a decrease in the amount of false positives, but also decrease the amount of true positives. When we look at the data we can see a significant decrease in the recall values when taking the intersection of different algorithms. However, we can also see a decrease in precision when more algorithms are intersected. This implies that most algorithms have more overlap in false positives than true positives. By taking the intersection of more algorithms less anomalies are detected while more false positives remain, resulting in a decrease in both precision and recall. As indicated by the increase in true negative rate we can see that the intersection does indeed reduce the amount of false positives, however, as indicated by the recall it also vastly reduces the amount of true positives. The intersection seems much less effective at increasing the anomaly detection rate because it seems to reduce the amount of true positives more than the amount of false positives.

8 Conclusions and Future Work

8.1 Scientific Evaluation

In our research we stated the following research question:

1. Can anomaly detection be performed for autonomously detecting anomalous behaviour of variables in games while limiting the number of false positives and false negatives?
   (a) Which algorithm achieves the best accuracy?
   (b) What degree of variable deviation can we detect?
   (c) What factors limit the number of false positives and false negatives?

In order to answer our main research questions we will evaluate our results in relation to our sub questions.

8.1.1 (a) Which algorithm achieves the best accuracy?

We have observed that LOF has very good performance when analyzing results aggregated by all players. For the 4 player and 1v4 player experiments LOF reached the highest f-score. However, it showed no improvement in performance when singular players were tested against the average of all players. LOCI achieved the highest f-score for the 1v10 players experiment. This implies that LOCI performs the best with a good enough baseline. Based on our results, when comparing singular players to a properly established baseline LOCI is the best choice. When classifying the aggregated values of all players as a whole, LOF is the best choice.

KNN showed decent performance overall, coming in as either second or a shared first place for our experiments. This shows the great versatility of KNN, having a decent performance in many scenarios. The Self Organizing Map often came in last with the worst f-score of all algorithms. Out of all algorithms, SOM was the most stable for all experiments. This would imply that it was good at establishing the same rough baseline for the relations between all the tracked variables. It just wasn't as effective at detecting if values deviated from this baseline.

8.1.2 (b) What degree of variable deviation can we detect?

Our experiments showed that anomaly sizes of 100% and 200% were often too small to detect. For the 1v10 experiment LOCI was able to detect some anomalies for an anomaly size of 200%, which shows that given a good enough baseline it is possible to detect small deviations to some degree. However, it was only once we increased the anomaly score to 1000% we were able to properly perform anomaly detection, commonly achieving an F-score of around 0.2.

8.1.3 (c) What factors limit the number of false positives and false negatives?

From our results we observed that there is a significant difference in detection rates for the number of players. For most algorithms tested, detection rates were much lower when observing the aggregated values of many players at a time. The more players observed at a time, the harder it gets to detect anomalies occurring in one player. Most algorithms were affected positively when comparing singular players to a baseline established by taking the average of all players. It seems that a strong baseline is an important factor in the number of false positives and false negatives.

We have also tested if the threshold of the sigmoid function influences the results. Our results imply that the threshold has little to no effect on the number of false positives and false negatives.

Window size does have an impact on detection rates. Especially KNN and LOF were affected, where KNN showed better performance with larger window sizes and LOF showed worse performance for larger window sizes. LOCI was shown to improve slightly with larger window sizes. SOM remained consistent regardless of window size.

KNN and LOF require the input of a k-value. We tested the influence of this parameter k on the results. Our results imply that KNN performs slightly worse for a k-value that becomes too large. LOF showed an opposite effect where a larger k-value resulted in better performance.

Finally we observed combinations of different algorithms. Taking the intersections of different algorithms influenced the detection rates negatively. From our results we can see that the intersections mostly increase the false negatives while having just a slight decrease in false positives.

We have shown that taking a union of KNN and LOCI and KNN, LOF and LOCI greatly reduces false negative rate while limiting the increase of false positives. This implies that taking the union positively influences the number of false positives and false negatives.

8.1.4 Main Research Question

Our experiments have shown that anomaly detection can autonomously detect anomalous behaviour of variables in OpenTTD. Our false positive rate was generally lower van 5% with an accuracy of around 90%, meaning that 90% of the events were correctly labeled. In most of our experiments about 15%-20% of the inserted anomalies could be detected. The best results were achieved by taking the union of all our algorithms with an f-score of 0.27 and an accuracy of around 90%.

We can conclude from our results that it is indeed possible to employ anomaly detection for autonomously detecting anomalous behaviour of variables in games.

8.2 Conclusion

In this thesis we have looked at 4 different anomaly detection algorithms testing them in several different scenarios. We evaluated their performance of autonomously detecting anomalies in variables in the game Open Transport Tycoon Deluxe (OpenTTD). We compared their performance using their f-score, accuracy score and various other performance measures. It was possible to autonomously detect about 15-20% of the inserted anomalies in the game with about a 90% accuracy rate without the need for training data. For situations where a single player could be evaluated against a baseline established by multiple players LOCI had the best performance, given that there were enough players to establish a proper baseline. For situations where the data of all players combined was evaluated LOF showed the best performance. KNN showed good performance overall, though it was never the single best performing algorithm for any of the scenarios tested. SOM was able to do some anomaly detection, but was shown to be the worst performing algorithm.

This thesis has made contributions to both the scientific and the business domains. The scientific contributions were the following:

- Compared both fundamental and state-of-the-art anomaly detection algorithms in a set different scenarios using various different parameter settings.
- Showed what parameters have the most influence on the results.
- Presented a framework for performing anomaly detection on a selection of variables from an interactive game

The contributions to the business domain were:

- Laid the foundation for autonomous anomaly detection in games.
- Implemented a generic, modular, flexible and easily integratable framework for autonomously detecting anomalies in various variables selected from a game with $<5\%$ false positive rate.

8.3 Discussion

Due to time constraints it was only possible to do a case-study for a single game. While we have shown that autonomous anomaly detection is possible for the game tested, it is desirable to test this for different games and genres as well. To improve generalizability, we have made as little assumptions as possible about the data and the relations of the variables. However, from a business perspective, it is important for the algorithm to be applicable to other games as well. Therefore, we would still need to test whether our results transfer to different games and genres. We can speculate that the framework will behave similarly for other strategy games as the data will likely have similar structures. However, the structure of the data will differ greatly between different games and genres. With more chaotic data it will be harder to detect anomalies. It would require further study to see if our framework performs similarly on other genres such as shooters or role-playing games.

To compare the different algorithms the F-score was used. This gives equal weight to the precision and recall scores of each algorithm. However, for the applicability of this framework it

might be more important to look at the false positive rate which is contained in the precision. In practice it would cost man-hours to check every report. Having little false positives would mean less man-hours wasted, making the precision of the algorithm an important part. Thus, it might be interesting to look at different accuracy measures which put more emphasis on the false positive rate.

Finally, we zoom in on the false positives that occurred. We noticed false positives often occurred when a player suddenly purchased a lot of roads or railway tracks at once. This is considered normal, expected behaviour in the game but for the anomaly detection algorithms such a large purchase is considered an outlier. With the variables we tracked it was very hard to distinguish this from anomalous behaviour. Theoretically, by tracking more variables such as the money spent or some relation between railway tracks earned and money spent such occurrences would not be marked as outlier. However, this requires the insight that such a problem exists and as such introduces a new challenge: how to determine which variables/relations should be tracked? This would require further study into the problem of feature selection.

8.4 Future Work

We explored a broad range of scenarios and parameter settings for various algorithms. However, while the false positive rate is quite low only about 15-20% of the anomalies could be detected. This obviously leaves room for improvement.

One possibility would be to employ a more neural network based approach to our framework. Currently, all charts containing variable combinations have the exact same weight. However, it is possible that some variable combinations have a stronger relation than others which would make their results more reliable than other charts. Assigning weights to different charts according to this reliability might improve our detection rates.

Another possibility for improving detection rates could be using training data. Future work might look into the improvements that can be made by using supervised or semi-supervised learning instead of completely unsupervised learning.

For our thesis we manually selected variables from the game representing the game state. It could be interesting to look into automatically detecting the most influential variables of a game. This would make integration even easier as the developers wouldn't even have to do a manual feature selection of their product.

As we have stated before, we only tested one game. Testing this framework on various different games could show more about the framework's versatility.

Finally, other outlier detection algorithms exist. Better results might be achieved by using different anomaly detection algorithms. Our research mainly contained clustering algorithms. Future work might focus on different types algorithms or perhaps test more advanced clustering algorithms.

References

[1] Prowler.io. https://www.prowler.io/blog/ai-tools-for-automated-game-testing.

[2] Simple ai - openttd. https://wiki.openttd.org/AI:SimpleAI.

[3] M. K. Albertini and R. F. de Mello. A self-organizing neural network for detecting novelties. pages 462–466, 2007.

[4] M. Amer, M. Goldstein, and S. Abdennadher. Enhancing one-class support vector machines for unsupervised anomaly detection. pages 8–15, 2013.

[5] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. LOF: identifying density-based local outliers. 29(2):93–104, 2000.

[6] T.-D. Cao, T.-T. Phan-Quang, P. Felix, and R. Castanet. Automated runtime verification for web services. pages 76–82, 2010.

[7] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. ACM computing surveys (CSUR), 41(3):15, 2009.

[8] K. Claessen and J. Hughes. QuickCheck: a lightweight tool for random testing of haskell programs. Acm sigplan notices, 46(4):53–64, 2011.

[9] M. D. Ernst, J. H. Perkins, P. J. Guo, S. McCamant, C. Pacheco, M. S. Tschantz, and C. Xiao. The daikon system for dynamic detection of likely invariants. Science of computer programming, 69(1-3):35–45, 2007.

[10] A. I. Esparcia-Alcázar, F. Almenar, M. Martínez, U. Rueda, and T. Vos. Q-learning strategies for action selection in the testar automated testing tool. 6th International Conferenrence on Metaheuristics and nature inspired computing (META 2016), pages 130–137, 2016.

[11] B. T. Fine. Unsupervised anomaly detection with minimal sensing. page 60, 2009.

[12] G. Fraser and A. Arcuri. Evosuite: automatic test suite generation for object-oriented software. pages 416–419, 2011.

[13] A. Grover. Anomaly detection for application log data. 2018.

[14] D. M. Hawkins. Identification of outliers. 11, 1980.

[15] M. A. Hayes and M. A. Capretz. Contextual anomaly detection framework for big sensor data. Journal of Big Data, 2(1):2, 2015.

[16] S. He, J. Zhu, P. He, and M. R. Lyu. Experience report: System log analysis for anomaly detection. pages 207–218, 2016.

[17] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf. Support vector machines. IEEE Intelligent Systems and their applications, 13(4):18–28, 1998.

[18] D. Heckerman. A tutorial on learning with bayesian networks. pages 301–354, 1998.

[19] H. Huang, H. Qin, S. Yoo, and D. Yu. Local anomaly descriptor: a robust unsupervised algorithm for anomaly detection based on diffusion space. pages 405–414, 2012.

[20] M. Jiang, O. W. Visser, I. Prasetya, and A. Iosup. A mirroring architecture for sophisticated mobile games using computation-offloading. Concurrency and Computation: Practice and Experience, 30(17):e4494, 2018.

[21] T. Kohonen. The self-organizing map. Proceedings of the IEEE, 78(9):1464–1480, 1990.

[22] P. Kostjens. Anomaly detection in application log data. 2016.

[23] M. Markou and S. Singh. Novelty detection: a review—part 1: statistical approaches. Signal processing, 83(12):2481–2497, 2003.

[24] M. Markou and S. Singh. Novelty detection: a review—part 2:: neural network based approaches. Signal processing, 83(12):2499–2521, 2003.

[25] D. Miljković. Review of novelty detection methods. pages 593–598, 2010.

[26] T. T. Nguyen, A. T. Nguyen, T. A. H. Nguyen, L. T. Vu, Q. U. Nguyen, and L. D. Hai. Unsupervised anomaly detection in online game. pages 4–10, 2015.

[27] T. Olsson and A. Holst. A probabilistic approach to aggregating anomalies for unsupervised anomaly detection with industrial applications. 2015.

[28] S. Papadimitriou, H. Kitagawa, P. B. Gibbons, and C. Faloutsos. Loci: Fast outlier detection using the local correlation integral. pages 315–326, 2003.

[29] A. Patcha and J.-M. Park. An overview of anomaly detection techniques: Existing solutions and latest technological trends. Computer networks, 51(12):3448–3470, 2007.

[30] J. Pfau, J. D. Smeddinck, and R. Malaka. Automated game testing with icarus: Intelligent completion of adventure riddles via unsupervised solving. pages 153–164, 2017.

[31] A. Pnueli. The temporal logic of programs. In 18th Annual Symposium on Foundations of Computer Science (sfcs 1977), pages 46–57. IEEE, 1977.

[32] I. W. B. Prasetya. T3, a combinator-based random testing tool for java: benchmarking. pages 101–110, 2013.

[33] I. W. B. Prasetya and M. A. Tran. Neural networks as artificial specifications. pages 135–141, 2018.

[34] S. Ramaswamy, R. Rastogi, and K. Shim. Efficient algorithms for mining outliers from large data sets. 29(2):427–438, 2000.

[35] C. Schaefer, H. Do, and B. M. Slator. Crushinator: A framework towards game-independent testing. pages 726–729, 2013.

[36] S. Shen, O. Visser, and A. Iosup. Rtsenv: An experimental environment for real-time strategy games. pages 1–6, 2011.

[37] A. Valdes and K. Skinner. Adaptive, model-based monitoring for cyber attack detection. pages 80–93, 2000.

[38] H. R. Vanda Vintrova, Tomas Vintr. Comparison of different calculations of the density-based local outlier factor. International Conference on Advances in Information Mining and Management, 2012.

[39] K. Yamanishi, J.-I. Takeuchi, G. Williams, and P. Milne. On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms. Data Mining and Knowledge Discovery, 8(3):275–300, 2004.

[40] S. F. Yeung and J. C. Lui. Dynamic bayesian approach for detecting cheats in multi-player online games. Multimedia Systems, 14(4):221–236, 2008.

A Results

A.1 Players: 1 | Anomaly Size: 100% | Threshold: e-15 | Window size: MAX | k: 50%

Seed: 100	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	0	5	12	114	0.000	0.000	0.000	0.958	0.870
LOF	0	5	12	114	0.000	0.000	0.000	0.958	0.870
LOCI	0	0	12	119	0.000	0.000	0.000	1.000	0.908
SOM	1	1	11	118	0.500	0.083	0.143	0.992	0.908

Seed: 101	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	0	1	9	67	0.000	0.000	0.000	0.985	0.870
LOF	0	1	9	67	0.000	0.000	0.000	0.985	0.870
LOCI	0	1	9	67	0.000	0.000	0.000	0.985	0.870
SOM	0	2	9	66	0.000	0.000	0.000	0.971	0.857

Seed: 102	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	1	6	10	120	0.143	0.091	0.111	0.952	0.883
LOF	0	9	11	117	0.000	0.000	0.000	0.929	0.854
LOCI	1	4	10	122	0.200	0.091	0.125	0.968	0.898
SOM	0	2	11	124	0.000	0.000	0.000	0.984	0.905

Seed: 103	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	0	4	4	79	0.000	0.000	0.000	0.952	0.908
LOF	0	4	4	79	0.000	0.000	0.000	0.952	0.908
LOCI	0	3	4	80	0.000	0.000	0.000	0.964	0.920
SOM	0	3	4	80	0.000	0.000	0.000	0.964	0.920

Seed: 104	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	0	2	8	50	0.000	0.000	0.000	0.962	0.833

LOF	0	2	8	50	0.000	0.000	0.000	0.962	0.833
LOCI	0	1	8	51	0.000	0.000	0.000	0.981	0.850
SOM	0	1	8	51	0.000		0.000	0.981	0.850

Seed: 105	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	0	2	8	50	0.000	0.000	0.000	0.962	0.833
LOF	0	7	15	87	0.000	0.000	0.000	0.926	0.798
LOCI	0	2	15	92	0.000	0.000	0.000	0.979	0.844
SOM	0	2	15	92	0.000	0.000	0.000	0.979	0.844

Seed: 106	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	0	4	13	95	0.000	0.000	0.000	0.960	0.848
LOF	0	5	13	94	0.000	0.000	0.000	0.949	0.839
LOCI	0	0	13	99	0.000	0.000	0.000	1.000	0.884
SOM	2	2	11	97	0.500	0.154	0.235	0.980	0.884

Seed: 107	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	0	2	10	48	0.000	0.000	0.000	0.960	0.800
LOF	0	5	10	45	0.000	0.000	0.000	0.900	0.750
LOCI	0	1	10	49	0.000	0.000	0.000	0.980	0.817
SOM	0	3	10	47	0.000	0.000	0.000	0.940	0.783

Seed: 108	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	0	5	14	93	0.000	0.000	0.000	0.949	0.830
LOF	0	6	14	92	0.000	0.000	0.000	0.939	0.821
LOCI	0	1	14	97	0.000	0.000	0.000	0.990	0.866
SOM	1	3	13	95	0.250	0.071	0.111	0.969	0.857

Seed: 109	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	0	3	4	65	0.000	0.000	0.000	0.956	0.903
LOF	0	2	4	66	0.000	0.000	0.000	0.971	0.917
LOCI	0	2	4	66	0.000	0.000	0.000	0.971	0.917
SOM	0	0	4	68	0.000	0.000	0.000	1.000	0.944

A.2 Players: 1 | Anomaly Size: 200% | Threshold: e-15 | Window size: MAX | k: 50%

Seed: 100	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	0	5	12	115	0.000	0.000	0.000	0.958	0.871
LOF	0	4	12	116	0.000	0.000	0.000	0.967	0.879
LOCI	0	1	12	119	0.000	0.000	0.000	0.992	0.902
SOM	1	1	11	119	0.500	0.083	0.143	0.992	0.909

Seed: 101	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	0	1	9	65	0.000	0.000	0.000	0.985	0.867
LOF	1	1	8	65	0.500	0.111	0.182	0.985	0.880
LOCI	0	1	9	65	0.000	0.000	0.000	0.985	0.867
SOM	0	2	9	64	0.000	0.000	0.000	0.970	0.853

Seed: 102	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	1	6	10	123	0.143	0.091	0.111	0.953	0.886

LOF	2	9	9	120	0.182	0.182	0.182	0.930	0.871
LOCI	2	4	9	125	0.333	0.182	0.235	0.969	0.907
SOM	2	3	9	126	0.400	0.182	0.250	0.977	0.914

Seed: 103	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	0	4	4	80	0.000	0.000	0.000	0.952	0.909
LOF	0	4	4	80	0.000	0.000	0.000	0.952	0.909
LOCI	0	3	4	81	0.000	0.000	0.000	0.964	0.920
SOM	0	3	4	81	0.000	0.000	0.000	0.964	0.920

Seed: 104	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	0	2	8	48	0.000	0.000	0.000	0.960	0.828
LOF	1	3	7	47	0.250	0.125	0.167	0.940	0.828
LOCI	0	1	8	49	0.000	0.000	0.000	0.980	0.845
SOM	0	1	8	49	0.000	0.000	0.000	0.980	0.845

Seed: 105	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	0	7	15	89	0.000	0.000	0.000	0.927	0.802
LOF	0	7	15	89	0.000	0.000	0.000	0.927	0.802
LOCI	0	2	15	94	0.000	0.000	0.000	0.979	0.847
SOM	1	2	14	94	0.333	0.067	0.111	0.979	0.856

Seed: 106	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	0	5	13	98	0.000	0.000	0.000	0.951	0.845
LOF	0	5	13	98	0.000	0.000	0.000	0.951	0.845
LOCI	0	2	13	101	0.000	0.000	0.000	0.981	0.871
SOM	2	2	11	101	0.500	0.154	0.235	0.981	0.888

Seed: 107	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	0	2	10	47	0.000	0.000	0.000	0.959	0.797
LOF	1	6	9	43	0.143	0.100	0.118	0.878	0.746
LOCI	0	1	10	48	0.000	0.000	0.000	0.980	0.814
SOM	0	3	10	46	0.000	0.000	0.000	0.939	0.780

Seed: 108	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	1	5	13	91	0.167	0.071	0.100	0.948	0.836
LOF	0	4	14	92	0.000	0.000	0.000	0.958	0.836
LOCI	0	1	14	95	0.000	0.000	0.000	0.990	0.864
SOM	1	3	13	93	0.250	0.071	0.111	0.969	0.855

Seed: 109	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	0	3	4	66	0.000	0.000	0.000	0.957	0.904
LOF	0	3	4	66	0.000	0.000	0.000	0.957	0.904
LOCI	0	2	4	67	0.000	0.000	0.000	0.971	0.918
SOM	0	0	4	68	0.000	0.000	0.000	1.000	0.944

A.3 Players: 1 | Anomaly Size: 1000% | Threshold: e-15 | Window size: MAX | k: 50%

Seed: 100	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	9	6	3	114	0.600	0.750	0.667	0.950	0.932

LOF	6	3	8	117	0.667	0.429	0.522	0.975	0.918
LOCI	3	1	9	119	0.750	0.250	0.375	0.992	0.924
SOM	8	2	4	118	0.800	0.667	0.727	0.983	0.955

Seed: 101	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	1	3	8	65	0.250	0.111	0.154	0.956	0.857
LOF	1	4	8	64	0.200	0.111	0.143	0.941	0.844
LOCI	0	1	9	67	0.000	0.000	0.000	0.985	0.870
SOM	1	4	8	64	0.200	0.111	0.143	0.941	0.844

Seed: 102	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	7	5	4	125	0.583	0.636	0.609	0.962	0.936
LOF	5	4	6	126	0.556	0.455	0.500	0.969	0.929
LOCI	2	1	9	129	0.667	0.182	0.286	0.992	0.929
SOM	3	3	8	127	0.500	0.273	0.353	0.977	0.922

Seed: 103	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	0	4	4	80	0.000	0.000	0.000	0.952	0.909
LOF	1	4	3	80	0.200	0.250	0.222	0.952	0.920
LOCI	0	3	4	81	0.000	0.000	0.000	0.964	0.920
SOM	0	3	4	81	0.000	0.000	0.000	0.964	0.920

Seed: 104	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	1	2	7	50	0.333	0.125	0.182	0.962	0.850
LOF	1	3	7	49	0.250	0.125	0.167	0.942	0.833
LOCI	0	1	8	51	0.000	0.000	0.000	0.981	0.850
SOM	0	1	8	51	0.000	0.000	0.000	0.981	0.850

Seed: 105	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	1	5	14	91	0.167	0.067	0.095	0.948	0.829
LOF	1	7	14	89	0.125	0.067	0.087	0.927	0.811
LOCI	0	1	15	95	0.000	0.000	0.000	0.990	0.856
SOM	1	3	14	93	0.250	0.067	0.105	0.969	0.847

Seed: 106	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	1	5	12	98	0.167	0.077	0.105	0.951	0.853
LOF	3	5	10	98	0.375	0.231	0.286	0.951	0.871
LOCI	1	1	12	102	0.500	0.077	0.133	0.990	0.888
SOM	2	2	11	101	0.500	0.154	0.235	0.981	0.888

Seed: 107	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	3	3	7	49	0.500	0.300	0.375	0.942	0.839
LOF	3	4	7	48	0.429	0.300	0.353	0.923	0.823
LOCI	1	1	9	51	0.500	0.100	0.167	0.981	0.839
SOM	1	4	9	48	0.200	0.100	0.133	0.923	0.790

Seed: 108	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	3	3	11	95	0.500	0.214	0.300	0.969	0.875
LOF	2	4	12	94	0.333	0.143	0.200	0.959	0.857
LOCI	1	2	13	96	0.333	0.071	0.118	0.980	0.866
SOM	1	3	13	95	0.250	0.071	0.111	0.969	0.857

Seed: 109	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	0	2	4	67	0.000	0.000	0.000	0.971	0.918
LOF	0	3	4	66	0.000	0.000	0.000	0.957	0.904
LOCI	0	2	4	67	0.000	0.000	0.000	0.971	0.918
SOM	0	0	4	69	0.000	0.000	0.000	1.000	0.945

A.4 Players: 4 | Anomaly Size: 100% | Threshold: e-15 | Window size: MAX | k: 50%

Seed: 100	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	0	12	30	323	0.000	0.000	0.000	0.964	0.885
LOF	0	12	30	323	0.000	0.000	0.000	0.964	0.885
LOCI	0	8	30	327	0.000	0.000	0.000	0.976	0.896
SOM	4	7	26	328	0.364	0.133	0.195	0.979	0.910

Seed: 101	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	1	4	25	247	0.200	0.038	0.065	0.984	0.895
LOF	1	7	25	244	0.125	0.038	0.059	0.972	0.884
LOCI	0	9	26	242	0.000	0.000	0.000	0.964	0.874
SOM	0	8	26	243	0.000	0.000	0.000	0.968	0.877

Seed: 102	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	2	9	31	266	0.182	0.061	0.091	0.967	0.870
LOF	2	6	31	269	0.250	0.061	0.098	0.978	0.880
LOCI	2	9	31	266	0.182	0.061	0.091	0.967	0.870
SOM	0	7	33	268	0.000	0.000	0.000	0.975	0.870

Seed: 103	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	0	8	22	262	0.000	0.000	0.000	0.970	0.897
LOF	1	10	21	260	0.091	0.045	0.061	0.963	0.894
LOCI	0	11	22	259	0.000	0.000	0.000	0.959	0.887
SOM	0	6	22	264	0.000	0.000	0.000	0.978	0.904

Seed: 104	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	0	20	42	348	0.000	0.000	0.000	0.946	0.849
LOF	1	22	41	346	0.043	0.024	0.031	0.940	0.846
LOCI	0	11	42	357	0.000	0.000	0.000	0.970	0.871
SOM	0	4	42	364	0.000	0.000	0.000	0.989	0.888

Seed: 105	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	2	2	27	250	0.500	0.069	0.121	0.992	0.897
LOF	2	4	27	248	0.333	0.069	0.114	0.984	0.890
LOCI	2	7	27	245	0.222	0.069	0.105	0.972	0.879
SOM	0	2	29	250	0.000	0.000	0.000	0.992	0.890

Seed: 106	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	0	7	30	346	0.000	0.000	0.000	0.980	0.903
LOF	0	9	30	344	0.000	0.000	0.000	0.975	0.898
LOCI	0	10	30	343	0.000	0.000	0.000	0.972	0.896
SOM	1	25	29	328	0.038	0.033	0.036	0.929	0.859

Seed: 107	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	2	3	34	331	0.400	0.056	0.098	0.991	0.900
LOF	3	4	33	333	0.429	0.083	0.140	0.988	0.901
LOCI	2	3	34	331	0.400	0.056	0.098	0.991	0.900
SOM	1	6	35	328	0.143	0.028	0.047	0.982	0.889

Seed: 108	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	0	6	28	289	0.000	0.000	0.000	0.980	0.895
LOF	0	7	28	288	0.000	0.000	0.000	0.976	0.892
LOCI	0	7	28	288	0.000	0.000	0.000	0.976	0.892
SOM	1	8	27	287	0.111	0.036	0.054	0.973	0.892

Seed: 109	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	1	9	38	312	0.100	0.026	0.041	0.972	0.869
LOF	1	11	38	310	0.083	0.026	0.039	0.966	0.864
LOCI	0	9	39	312	0.000	0.000	0.000	0.972	0.867
SOM	2	8	37	313	0.200	0.051	0.082	0.975	0.875

A.5 Players: 4 | Anomaly Size: 200% | Threshold: e-15 | Window size: MAX | k: 50%

Seed: 100	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	0	12	30	320	0.000	0.000	0.000	0.964	0.884
LOF	0	10	30	322	0.000	0.000	0.000	0.970	0.890
LOCI	0	9	30	323	0.000	0.000	0.000	0.973	0.892
SOM	5	6	25	326	0.455	0.167	0.244	0.982	0.914

Seed: 101	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	1	4	25	246	0.200	0.038	0.065	0.984	0.895
LOF	1	8	25	242	0.111	0.038	0.057	0.968	0.880
LOCI	0	9	26	241	0.000	0.000	0.000	0.964	0.873
SOM	0	8	26	242	0.000	0.000	0.000	0.968	0.877

Seed: 102	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	2	9	31	261	0.182	0.061	0.091	0.967	0.868
LOF	2	6	31	264	0.250	0.061	0.098	0.978	0.878
LOCI	2	8	31	262	0.200	0.061	0.093	0.970	0.871
SOM	1	7	32	263	0.125	0.030	0.049	0.974	0.871

Seed: 103	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	0	8	22	257	0.000	0.000	0.000	0.970	0.895
LOF	1	10	21	255	0.091	0.045	0.061	0.962	0.892
LOCI	1	11	21	254	0.083	0.045	0.059	0.958	0.889
SOM	2	7	20	258	0.222	0.091	0.129	0.974	0.906

Seed: 104	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	0	18	42	353	0.000	0.000	0.000	0.951	0.855
LOF	1	20	41	351	0.048	0.024	0.032	0.946	0.852
LOCI	1	12	41	359	0.077	0.024	0.036	0.968	0.872
SOM	1	3	41	368	0.250	0.024	0.043	0.992	0.893

Seed: 105	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	2	2	27	248	0.500	0.069	0.121	0.992	0.896
LOF	2	3	27	247	0.400	0.069	0.118	0.988	0.892
LOCI	2	6	27	244	0.250	0.069	0.108	0.976	0.882
SOM	1	2	28	248	0.333	0.034	0.063	0.992	0.892

Seed: 106	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	0	7	30	347	0.000	0.000	0.000	0.980	0.904
LOF	1	9	29	345	0.100	0.033	0.050	0.975	0.901
LOCI	0	10	30	344	0.000	0.000	0.000	0.972	0.896
SOM	1	22	29	332	0.043	0.033	0.038	0.938	0.867

Seed: 107	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	3	3	33	333	0.500	0.083	0.143	0.991	0.903
LOF	3	3	33	333	0.500	0.083	0.143	0.991	0.903
LOCI	2	3	34	333	0.400	0.056	0.098	0.991	0.901
SOM	1	7	35	329	0.125	0.028	0.045	0.979	0.887

Seed: 108	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	0	6	28	289	0.000	0.000	0.000	0.980	0.895
LOF	0	7	28	288	0.000	0.000	0.000	0.976	0.892
LOCI	0	7	28	288	0.000	0.000	0.000	0.976	0.892
SOM	1	8	27	287	0.111	0.036	0.054	0.973	0.892

Seed: 109	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	1	9	38	314	0.100	0.026	0.041	0.972	0.870
LOF	1	11	38	312	0.083	0.026	0.039	0.966	0.865
LOCI	0	8	39	315	0.000	0.000	0.000	0.975	0.870
SOM	2	8	37	315	0.200	0.051	0.082	0.975	0.876

A.6 Players: 4 | Anomaly Size: 1000% | Threshold: e-15 | Window size: MAX | k: 50%

Seed: 100	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	3	7	27	328	0.300	0.100	0.150	0.979	0.907
LOF	8	12	22	323	0.400	0.267	0.320	0.964	0.907
LOCI	2	7	28	328	0.222	0.067	0.103	0.979	0.904
SOM	4	3	26	332	0.571	0.133	0.216	0.991	0.921

Seed: 101	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	2	5	24	251	0.286	0.077	0.121	0.980	0.897
LOF	4	7	22	249	0.364	0.154	0.216	0.973	0.897
LOCI	1	7	25	249	0.125	0.038	0.059	0.973	0.887
SOM	2	9	24	247	0.182	0.077	0.108	0.965	0.883

Seed: 102	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	3	6	30	271	0.333	0.091	0.143	0.978	0.884
LOF	3	7	30	270	0.300	0.091	0.140	0.975	0.881
LOCI	2	8	31	269	0.200	0.061	0.093	0.971	0.874
SOM	1	7	32	270	0.125	0.030	0.049	0.975	0.874

Seed: 103	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	4	8	18	262	0.333	0.182	0.235	0.970	0.911
LOF	2	8	20	262	0.200	0.091	0.125	0.970	0.904
LOCI	3	11	19	259	0.214	0.136	0.167	0.959	0.897
SOM	3	6	19	264	0.333	0.136	0.194	0.978	0.914

Seed: 104	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	8	11	34	363	0.421	0.190	0.262	0.971	0.892
LOF	7	11	35	363	0.389	0.167	0.233	0.971	0.889
LOCI	4	10	38	364	0.286	0.095	0.143	0.973	0.885
SOM	2	2	40	372	0.500	0.048	0.087	0.995	0.899

Seed: 105	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	3	2	26	256	0.600	0.103	0.176	0.992	0.902
LOF	3	4	26	254	0.429	0.103	0.167	0.984	0.895
LOCI	1	5	28	253	0.167	0.034	0.057	0.981	0.885
SOM	0	2	29	256	0.000	0.000	0.000	0.992	0.892

Seed: 106	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	2	8	28	349	0.200	0.067	0.100	0.978	0.907
LOF	3	10	27	347	0.231	0.100	0.140	0.972	0.904
LOCI	3	10	27	347	0.231	0.100	0.140	0.972	0.904
SOM	1	7	29	350	0.125	0.033	0.053	0.980	0.907

Seed: 107	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	4	3	32	337	0.571	0.111	0.186	0.991	0.907
LOF	7	5	29	335	0.583	0.194	0.292	0.985	0.910
LOCI	3	5	33	335	0.375	0.083	0.136	0.985	0.899
SOM	1	7	35	333	0.125	0.028	0.045	0.979	0.888

Seed: 108	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	0	6	28	294	0.000	0.000	0.000	0.980	0.896
LOF	4	10	24	290	0.286	0.143	0.190	0.967	0.896
LOCI	1	5	27	295	0.167	0.036	0.059	0.983	0.902
SOM	1	6	27	294	0.143	0.036	0.057	0.980	0.899

Seed: 109	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	2	8	37	321	0.200	0.051	0.082	0.976	0.878
LOF	4	9	35	320	0.308	0.103	0.154	0.973	0.880
LOCI	1	8	38	321	0.111	0.026	0.042	0.976	0.875
SOM	2	6	37	323	0.250	0.051	0.085	0.982	0.883

A.7 Players: 1v4 | Anomaly Size: 100% | Threshold: e-15 | Window size: MAX | k: 50%

Seed: 100	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	1	12	29	319	0.077	0.033	0.047	0.964	0.886
LOF	0	8	30	323	0.000	0.000	0.000	0.976	0.895
LOCI	1	13	29	318	0.071	0.033	0.045	0.961	0.884
SOM	4	9	26	322	0.308	0.133	0.186	0.973	0.903

Seed: 101	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	1	6	25	238	0.143	0.038	0.061	0.975	0.885
LOF	1	6	25	238	0.143	0.038	0.061	0.975	0.885
LOCI	0	10	26	234	0.000	0.000	0.000	0.959	0.867
SOM	0	9	26	235	0.000	0.000	0.000	0.963	0.870

Seed: 102	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	2	7	31	255	0.222	0.061	0.095	0.973	0.871
LOF	2	7	31	255	0.222	0.061	0.095	0.973	0.871
LOCI	3	10	30	252	0.231	0.091	0.130	0.962	0.864
SOM	0	6	33	256	0.000	0.000	0.000	0.977	0.868

Seed: 103	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	0	9	22	260	0.000	0.000	0.000	0.967	0.893
LOF	0	9	22	260	0.000	0.000	0.000	0.967	0.893
LOCI	1	12	21	257	0.077	0.045	0.057	0.955	0.887
SOM	0	6	22	263	0.000	0.000	0.000	0.978	0.904

Seed: 104	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	1	16	41	344	0.059	0.024	0.034	0.956	0.858
LOF	1	13	41	347	0.071	0.024	0.036	0.964	0.866
LOCI	1	12	41	348	0.077	0.024	0.036	0.967	0.868
SOM	0	6	42	354	0.000	0.000	0.000	0.983	0.881

Seed: 105	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	2	2	27	246	0.500	0.069	0.121	0.992	0.895
LOF	2	2	27	246	0.500	0.069	0.121	0.992	0.895
LOCI	3	6	26	242	0.333	0.103	0.158	0.976	0.884
SOM	1	1	28	247	0.500	0.034	0.065	0.996	0.895

Seed: 106	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	0	9	30	336	0.000	0.000	0.000	0.974	0.896
LOF	0	9	30	336	0.000	0.000	0.000	0.974	0.896
LOCI	1	13	29	332	0.071	0.033	0.045	0.962	0.888
SOM	1	16	29	329	0.059	0.033	0.043	0.954	0.880

Seed: 107	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	3	4	33	320	0.429	0.083	0.140	0.988	0.897
LOF	3	10	33	314	0.231	0.083	0.122	0.969	0.881
LOCI	2	12	34	312	0.143	0.056	0.080	0.963	0.872
SOM	2	7	34	317	0.222	0.056	0.089	0.978	0.886

Seed: 108	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	0	6	28	287	0.000	0.000	0.000	0.980	0.894
LOF	0	5	28	288	0.000	0.000	0.000	0.983	0.897
LOCI	0	6	28	287	0.000	0.000	0.000	0.980	0.894
SOM	0	8	28	285	0.000	0.000	0.000	0.973	0.888

Seed: 109	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	0	13	39	302	0.000	0.000	0.000	0.959	0.853
LOF	0	10	39	305	0.000	0.000	0.000	0.968	0.862
LOCI	0	12	39	303	0.000	0.000	0.000	0.962	0.856
SOM	2	11	37	304	0.154	0.051	0.077	0.965	0.864

A.8 Players: 1v4 | Anomaly Size: 200% | Threshold: e-15 | Window size: MAX | k: 50%

Seed: 100	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	1	12	29	316	0.077	0.033	0.047	0.963	0.885
LOF	2	11	28	317	0.154	0.067	0.093	0.966	0.891
LOCI	0	13	30	315	0.000	0.000	0.000	0.960	0.880
SOM	4	6	26	322	0.400	0.133	0.200	0.982	0.911

Seed: 101	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	1	5	25	237	0.167	0.038	0.063	0.979	0.888
LOF	1	5	25	237	0.167	0.038	0.063	0.979	0.888
LOCI	0	9	26	233	0.000	0.000	0.000	0.963	0.869
SOM	0	8	26	234	0.000	0.000	0.000	0.967	0.873

Seed: 102	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	2	7	31	251	0.222	0.061	0.095	0.973	0.869
LOF	2	6	31	252	0.250	0.061	0.098	0.977	0.873
LOCI	3	10	30	248	0.231	0.091	0.130	0.961	0.863
SOM	1	6	32	252	0.143	0.030	0.050	0.977	0.869

Seed: 103	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	0	9	22	255	0.000	0.000	0.000	0.966	0.892
LOF	0	9	22	255	0.000	0.000	0.000	0.966	0.892
LOCI	1	12	21	252	0.077	0.045	0.057	0.955	0.885
SOM	2	6	20	258	0.250	0.091	0.133	0.977	0.909

Seed: 104	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	2	16	40	348	0.111	0.048	0.067	0.956	0.862
LOF	2	13	40	351	0.133	0.048	0.070	0.964	0.869
LOCI	2	13	40	351	0.133	0.048	0.070	0.964	0.869
SOM	1	6	41	358	0.143	0.024	0.041	0.984	0.884

Seed: 105	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	2	2	27	244	0.500	0.069	0.121	0.992	0.895
LOF	2	2	27	244	0.500	0.069	0.121	0.992	0.895
LOCI	3	6	26	240	0.333	0.103	0.158	0.976	0.884
SOM	2	1	27	245	0.667	0.069	0.125	0.996	0.898

Seed: 106	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	0	9	30	337	0.000	0.000	0.000	0.974	0.896
LOF	1	9	29	337	0.100	0.033	0.050	0.974	0.899
LOCI	2	12	28	334	0.143	0.067	0.091	0.965	0.894
SOM	1	12	29	334	0.077	0.033	0.047	0.965	0.891

Seed: 107	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	3	4	33	322	0.429	0.083	0.140	0.988	0.898
LOF	3	9	33	317	0.250	0.083	0.125	0.972	0.884
LOCI	2	12	34	314	0.143	0.056	0.080	0.963	0.873
SOM	2	7	34	319	0.222	0.056	0.089	0.979	0.887

Seed: 108	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	0	6	28	287	0.000	0.000	0.000	0.980	0.894
LOF	0	5	28	288	0.000	0.000	0.000	0.983	0.897
LOCI	0	7	28	286	0.000	0.000	0.000	0.976	0.891
SOM	1	9	27	284	0.100	0.036	0.053	0.969	0.888

Seed: 109	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	0	13	39	304	0.000	0.000	0.000	0.959	0.854
LOF	0	10	39	307	0.000	0.000	0.000	0.968	0.862
LOCI	1	11	38	306	0.083	0.026	0.039	0.965	0.862

SOM	2	10	37	307	0.167	0.051	0.078	0.968	0.868

A.9 Players: 1v4 | Anomaly Size: 1000% | Threshold: e-15 | Window size: MAX | k: 50%

Seed: 100	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	8	10	22	321	0.444	0.267	0.333	0.970	0.911
LOF	8	9	22	322	0.471	0.267	0.340	0.973	0.914
LOCI	4	7	26	324	0.364	0.133	0.195	0.979	0.909
SOM	4	3	26	328	0.571	0.133	0.216	0.991	0.920

Seed: 101	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	3	7	23	241	0.300	0.115	0.167	0.972	0.891
LOF	2	5	24	243	0.286	0.077	0.121	0.980	0.894
LOCI	0	7	26	241	0.000	0.000	0.000	0.972	0.880
SOM	1	9	25	239	0.100	0.038	0.056	0.964	0.876

Seed: 102	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	3	7	30	257	0.300	0.091	0.140	0.973	0.875
LOF	3	6	30	258	0.333	0.091	0.143	0.977	0.879
LOCI	2	10	31	254	0.167	0.061	0.089	0.962	0.862
SOM	1	6	32	258	0.143	0.030	0.050	0.977	0.872

Seed: 103	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	4	9	18	260	0.308	0.182	0.229	0.967	0.907
LOF	2	9	20	260	0.182	0.091	0.121	0.967	0.900
LOCI	4	12	18	257	0.250	0.182	0.211	0.955	0.897
SOM	3	7	19	262	0.300	0.136	0.188	0.974	0.911

Seed: 104	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	8	10	34	358	0.444	0.190	0.267	0.973	0.893
LOF	6	8	36	360	0.429	0.143	0.214	0.978	0.893
LOCI	7	9	35	359	0.438	0.167	0.241	0.976	0.893
SOM	2	3	40	365	0.400	0.048	0.085	0.992	0.895

Seed: 105	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	3	2	26	252	0.600	0.103	0.176	0.992	0.901
LOF	3	3	26	251	0.500	0.103	0.171	0.988	0.898
LOCI	3	5	26	249	0.375	0.103	0.162	0.980	0.890
SOM	1	2	28	252	0.333	0.034	0.063	0.992	0.894

Seed: 106	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	2	8	28	341	0.200	0.067	0.100	0.977	0.905
LOF	2	7	28	342	0.222	0.067	0.103	0.980	0.908
LOCI	4	12	26	337	0.250	0.133	0.174	0.966	0.900
SOM	1	5	29	344	0.167	0.033	0.056	0.986	0.910

Seed: 107	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	5	4	31	326	0.556	0.139	0.222	0.988	0.904
LOF	7	7	29	323	0.500	0.194	0.280	0.979	0.902
LOCI	5	6	31	324	0.455	0.139	0.213	0.982	0.899

SOM	2	7	34	323	0.222	0.056	0.089	0.979	0.888

Seed: 108	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	1	5	27	293	0.167	0.036	0.059	0.983	0.902
LOF	6	6	22	292	0.500	0.214	0.300	0.980	0.914
LOCI	2	7	26	291	0.222	0.071	0.108	0.977	0.899
SOM	1	6	27	292	0.143	0.036	0.057	0.980	0.899

Seed: 109	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	3	10	36	313	0.231	0.077	0.115	0.969	0.873
LOF	5	8	34	315	0.385	0.128	0.192	0.975	0.884
LOCI	4	12	35	311	0.250	0.103	0.145	0.963	0.870
SOM	2	7	37	316	0.222	0.051	0.083	0.978	0.878

A.10 Players: 1v10 | Anomaly Size: 100% | Threshold: e-15 | Window size: MAX | k: 50%

Seed: 100	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	5	27	86	779	0.156	0.055	0.081	0.967	0.874
LOF	3	17	88	789	0.150	0.033	0.054	0.979	0.883
LOCI	8	43	83	763	0.157	0.088	0.113	0.947	0.860
SOM	2	10	89	796	0.167	0.022	0.039	0.988	0.890

Seed: 101	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	2	37	95	863	0.051	0.021	0.029	0.959	0.868
LOF	3	17	88	789	0.150	0.033	0.054	0.979	0.883
LOCI	2	46	95	854	0.042	0.021	0.028	0.949	0.859
SOM	0	7	97	893	0.000	0.000	0.000	0.992	0.896

Seed: 102	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	1	20	74	789	0.048	0.013	0.021	0.975	0.894
LOF	0	16	75	793	0.000	0.000	0.000	0.980	0.897
LOCI	8	36	67	773	0.182	0.107	0.134	0.956	0.883
SOM	4	17	71	792	0.190	0.053	0.083	0.979	0.900

Seed: 103	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	3	13	65	777	0.188	0.044	0.071	0.984	0.909
LOF	4	14	64	776	0.222	0.059	0.093	0.982	0.909
LOCI	4	33	64	757	0.108	0.059	0.076	0.958	0.887
SOM	3	14	65	776	0.176	0.044	0.071	0.982	0.908

Seed: 104	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	1	30	72	834	0.032	0.014	0.019	0.965	0.891
LOF	2	16	71	838	0.111	0.027	0.044	0.981	0.906
LOCI	3	50	70	814	0.057	0.041	0.048	0.942	0.872
SOM	1	7	72	857	0.125	0.014	0.025	0.992	0.916

Seed: 105	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	3	34	89	838	0.081	0.033	0.047	0.961	0.872
LOF	2	25	90	847	0.074	0.022	0.034	0.971	0.881
LOCI	8	47	84	825	0.145	0.087	0.109	0.946	0.864

SOM	9	18	83	854	0.333	0.098	0.151	0.979	0.895

Seed: 106	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	6	22	81	826	0.214	0.069	0.104	0.974	0.890
LOF	6	18	81	830	0.250	0.069	0.108	0.979	0.894
LOCI	6	39	81	809	0.133	0.069	0.091	0.954	0.872
SOM	3	8	84	840	0.273	0.034	0.061	0.991	0.902

Seed: 107	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	1	16	55	705	0.059	0.018	0.027	0.978	0.909
LOF	0	11	56	710	0.000	0.000	0.000	0.985	0.914
LOCI	2	28	54	693	0.067	0.036	0.047	0.961	0.894
SOM	2	10	54	711	0.167	0.036	0.059	0.986	0.918

Seed: 108	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	0	14	84	725	0.000	0.000	0.000	0.981	0.881
LOF	1	12	83	727	0.077	0.012	0.021	0.984	0.885
LOCI	1	36	83	703	0.027	0.012	0.017	0.951	0.855
SOM	0	11	84	728	0.000	0.000	0.000	0.985	0.885

Seed: 109	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	4	25	78	782	0.138	0.049	0.072	0.969	0.884
LOF	5	23	77	784	0.179	0.061	0.091	0.971	0.888
LOCI	5	41	77	766	0.109	0.061	0.078	0.949	0.867
SOM	3	5	79	802	0.375	0.037	0.067	0.994	0.906

A.11 Players: 1v10 | Anomaly Size: 200% | Threshold: e-15 | Window size: MAX | k: 50%

Seed: 100	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	3	23	88	784	0.115	0.033	0.051	0.971	0.876
LOF	2	14	89	793	0.125	0.022	0.037	0.983	0.885
LOCI	8	39	83	768	0.170	0.088	0.116	0.952	0.864
SOM	2	9	89	798	0.182	0.022	0.039	0.989	0.891

Seed: 101	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	3	37	94	865	0.075	0.031	0.044	0.959	0.869
LOF	3	29	94	873	0.094	0.031	0.047	0.968	0.877
LOCI	4	46	93	856	0.080	0.041	0.054	0.949	0.861
SOM	0	6	97	896	0.000	0.000	0.000	0.993	0.897

Seed: 102	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	1	20	74	798	0.048	0.013	0.021	0.976	0.895
LOF	0	15	75	803	0.000	0.000	0.000	0.982	0.899
LOCI	11	33	64	785	0.250	0.147	0.185	0.960	0.891
SOM	7	15	68	803	0.318	0.093	0.144	0.982	0.907

Seed: 103	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	3	13	65	783	0.188	0.044	0.071	0.984	0.910
LOF	4	8	64	788	0.333	0.059	0.100	0.990	0.917
LOCI	5	33	63	763	0.132	0.074	0.094	0.959	0.889

SOM	4	14	64	782	0.222	0.059	0.093	0.982	0.910

Seed: 104	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	1	30	72	829	0.032	0.014	0.019	0.965	0.891
LOF	3	25	70	834	0.107	0.041	0.059	0.971	0.898
LOCI	4	51	69	808	0.073	0.055	0.063	0.941	0.871
SOM	2	6	71	853	0.250	0.027	0.049	0.993	0.917

Seed: 105	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	4	33	88	841	0.108	0.043	0.062	0.962	0.875
LOF	2	25	90	849	0.074	0.022	0.034	0.971	0.881
LOCI	11	46	81	828	0.193	0.120	0.148	0.947	0.869
SOM	9	16	83	858	0.360	0.098	0.154	0.982	0.898

Seed: 106	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	7	22	80	835	0.241	0.080	0.121	0.974	0.892
LOF	7	18	80	839	0.280	0.080	0.125	0.979	0.896
LOCI	10	38	77	819	0.208	0.115	0.148	0.956	0.878
SOM	2	8	85	849	0.200	0.023	0.041	0.991	0.901

Seed: 107	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	1	14	55	706	0.067	0.018	0.028	0.981	0.911
LOF	0	9	56	711	0.000	0.000	0.000	0.988	0.916
LOCI	4	29	52	691	0.121	0.071	0.090	0.960	0.896
SOM	3	11	53	709	0.214	0.054	0.086	0.985	0.918

Seed: 108	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	0	14	84	725	0.000	0.000	0.000	0.981	0.881
LOF	0	12	84	727	0.000	0.000	0.000	0.984	0.883
LOCI	3	36	81	703	0.077	0.036	0.049	0.951	0.858
SOM	0	9	84	730	0.000	0.000	0.000	0.988	0.887

Seed: 109	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	4	25	78	784	0.138	0.049	0.072	0.969	0.884
LOF	5	23	77	786	0.179	0.061	0.091	0.972	0.888
LOCI	3	36	81	703	0.077	0.036	0.049	0.951	0.858
SOM	4	5	78	804	0.444	0.049	0.088	0.994	0.907

A.12 Players: 1v10 | Anomaly Size: 1000% | Threshold: e-15 | Window size: MAX | k: 50%

Seed: 100	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	9	17	82	803	0.346	0.099	0.154	0.979	0.891
LOF	9	16	82	804	0.360	0.099	0.155	0.980	0.892
LOCI	15	30	76	790	0.333	0.165	0.221	0.963	0.884
SOM	7	6	84	814	0.538	0.077	0.135	0.993	0.901

Seed: 101	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	13	28	84	889	0.317	0.134	0.188	0.969	0.890
LOF	9	18	88	899	0.333	0.093	0.145	0.980	0.895
LOCI	7	35	90	882	0.167	0.072	0.101	0.962	0.877

SOM	0	1	97	916	0.000	0.000	0.000	0.999	0.903
Seed: 102	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	11	19	64	806	0.367	0.147	0.210	0.977	0.908
LOF	9	13	66	812	0.409	0.120	0.186	0.984	0.912
LOCI	20	32	55	793	0.385	0.267	0.315	0.961	0.903
SOM	9	15	66	810	0.375	0.120	0.182	0.982	0.910
Seed: 103	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	11	9	57	796	0.550	0.162	0.250	0.989	0.924
LOF	7	6	61	799	0.538	0.103	0.173	0.993	0.923
LOCI	10	22	58	783	0.313	0.147	0.200	0.973	0.908
SOM	7	13	61	792	0.350	0.103	0.159	0.984	0.915
Seed: 104	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	9	29	64	846	0.237	0.123	0.162	0.967	0.902
LOF	6	22	67	853	0.214	0.082	0.119	0.975	0.906
LOCI	15	45	58	830	0.250	0.205	0.226	0.949	0.891
SOM	6	6	67	869	0.500	0.082	0.141	0.993	0.923
Seed: 105	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	15	26	77	860	0.366	0.163	0.226	0.971	0.895
LOF	10	17	82	869	0.370	0.109	0.168	0.981	0.899
LOCI	30	44	62	842	0.405	0.326	0.361	0.950	0.892
SOM	9	15	83	871	0.375	0.098	0.155	0.983	0.900
Seed: 106	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	16	22	71	840	0.421	0.184	0.256	0.974	0.902
LOF	11	15	76	847	0.423	0.126	0.195	0.983	0.904
LOCI	19	31	68	831	0.380	0.218	0.277	0.964	0.896
SOM	2	8	85	854	0.200	0.023	0.041	0.991	0.902
Seed: 107	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	9	17	47	714	0.346	0.161	0.220	0.977	0.919
LOF	6	6	50	725	0.500	0.107	0.176	0.992	0.929
LOCI	8	26	48	705	0.235	0.143	0.178	0.964	0.906
SOM	5	13	51	718	0.278	0.089	0.135	0.982	0.919
Seed: 108	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	9	14	75	739	0.391	0.107	0.168	0.981	0.894
LOF	5	13	79	740	0.278	0.060	0.098	0.983	0.890
LOCI	9	31	75	722	0.225	0.107	0.145	0.959	0.873
SOM	0	8	84	745	0.000	0.000	0.000	0.989	0.890
Seed: 109	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	9	19	73	799	0.321	0.110	0.164	0.977	0.898
LOF	9	18	73	800	0.333	0.110	0.165	0.978	0.899
LOCI	16	28	66	790	0.364	0.195	0.254	0.966	0.896
SOM	5	3	77	815	0.625	0.061	0.111	0.996	0.911

A.13 Players: 1v10 | Anomaly Size: 1000% | Threshold: e-5 | Window size: MAX | k: 50%

Seed: 100	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	8	14	83	806	0.364	0.088	0.142	0.983	0.894
LOF	9	13	82	807	0.409	0.099	0.159	0.984	0.896
LOCI	13	27	78	793	0.325	0.143	0.198	0.967	0.885
SOM	7	5	84	815	0.583	0.077	0.136	0.994	0.902

Seed: 101	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	10	27	87	890	0.270	0.103	0.149	0.971	0.888
LOF	9	17	88	900	0.346	0.093	0.146	0.981	0.896
LOCI	7	33	90	884	0.175	0.072	0.102	0.964	0.879
SOM	0	1	97	916	0.000	0.000	0.000	0.999	0.903

Seed: 102	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	12	20	63	805	0.375	0.160	0.224	0.976	0.908
LOF	6	11	69	814	0.353	0.080	0.130	0.987	0.911
LOCI	20	30	55	795	0.400	0.267	0.320	0.964	0.906
SOM	9	15	66	810	0.375	0.120	0.182	0.982	0.910

Seed: 103	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	12	11	56	794	0.522	0.176	0.264	0.986	0.923
LOF	7	6	61	799	0.538	0.103	0.173	0.993	0.923
LOCI	10	20	58	785	0.333	0.147	0.204	0.975	0.911
SOM	6	13	62	792	0.316	0.088	0.138	0.984	0.914

Seed: 104	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	8	29	65	846	0.216	0.110	0.145	0.967	0.901
LOF	6	21	67	854	0.222	0.082	0.120	0.976	0.907
LOCI	11	41	62	834	0.212	0.151	0.176	0.953	0.891
SOM	6	7	67	868	0.462	0.082	0.140	0.992	0.922

Seed: 105	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	14	26	78	860	0.350	0.152	0.212	0.971	0.894
LOF	10	15	82	871	0.400	0.109	0.171	0.983	0.901
LOCI	22	42	70	844	0.344	0.239	0.282	0.953	0.885
SOM	9	15	83	871	0.375	0.098	0.155	0.983	0.900

Seed: 106	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	17	22	70	840	0.436	0.195	0.270	0.974	0.903
LOF	10	15	77	847	0.400	0.115	0.179	0.983	0.903
LOCI	19	30	68	832	0.388	0.218	0.279	0.965	0.897
SOM	3	7	84	855	0.300	0.034	0.062	0.992	0.904

Seed: 107	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	7	16	49	715	0.304	0.125	0.177	0.978	0.917
LOF	5	5	51	726	0.500	0.089	0.152	0.993	0.929
LOCI	8	26	48	705	0.235	0.143	0.178	0.964	0.906
SOM	5	13	51	718	0.278	0.089	0.135	0.982	0.919

Seed: 108	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	9	14	75	739	0.391	0.107	0.168	0.981	0.894
LOF	5	13	79	740	0.278	0.060	0.098	0.983	0.890
LOCI	9	31	75	722	0.225	0.107	0.145	0.959	0.873
SOM	0	7	84	746	0.000	0.000	0.000	0.991	0.891

Seed: 109	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy

KNN	9	19	73	799	0.321	0.110	0.164	0.977	0.898
LOF	8	17	74	801	0.320	0.098	0.150	0.979	0.899
LOCI	14	28	68	790	0.333	0.171	0.226	0.966	0.893
SOM	6	4	76	814	0.600	0.073	0.130	0.995	0.911

A.14 Players: 1v10 | Anomaly Size: 1000% | Threshold: e-50 | Window size: MAX | k: 50%

Seed: 100	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	9	17	82	803	0.346	0.099	0.154	0.979	0.891
LOF	9	19	82	801	0.321	0.099	0.151	0.977	0.889
LOCI	24	37	67	783	0.393	0.264	0.316	0.955	0.886
SOM	8	7	83	813	0.533	0.088	0.151	0.991	0.901

Seed: 101	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	13	31	84	886	0.295	0.134	0.184	0.966	0.887
LOF	9	24	88	893	0.273	0.093	0.138	0.974	0.890
LOCI	9	36	88	881	0.200	0.093	0.127	0.961	0.878
SOM	0	1	97	916	0.000	0.000	0.000	0.999	0.903

Seed: 102	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	11	20	64	805	0.355	0.147	0.208	0.976	0.907
LOF	9	13	66	812	0.409	0.120	0.186	0.984	0.912
LOCI	20	32	55	793	0.385	0.267	0.315	0.961	0.903
SOM	9	15	66	810	0.375	0.120	0.182	0.982	0.910

Seed: 103	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	11	10	57	795	0.524	0.162	0.247	0.988	0.923
LOF	7	7	61	798	0.500	0.103	0.171	0.991	0.922
LOCI	10	22	58	783	0.313	0.147	0.200	0.973	0.908
SOM	7	13	61	792	0.350	0.103	0.159	0.984	0.915

Seed: 104	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	10	31	63	844	0.244	0.137	0.175	0.965	0.901
LOF	7	23	66	852	0.233	0.096	0.136	0.974	0.906
LOCI	25	48	48	827	0.342	0.342	0.342	0.945	0.899
SOM	6	6	67	869	0.500	0.082	0.141	0.993	0.923

Seed: 105	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	14	26	78	860	0.350	0.152	0.212	0.971	0.894
LOF	10	17	82	869	0.370	0.109	0.168	0.981	0.899
LOCI	30	44	62	842	0.405	0.326	0.361	0.950	0.892
SOM	9	13	83	873	0.409	0.098	0.158	0.985	0.902

Seed: 106	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	16	22	71	840	0.421	0.184	0.256	0.974	0.902
LOF	11	15	76	847	0.423	0.126	0.195	0.983	0.904
LOCI	23	37	64	825	0.383	0.264	0.313	0.957	0.894
SOM	2	7	85	855	0.222	0.023	0.042	0.992	0.903

Seed: 107	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy

KNN	9	17	47	714	0.346	0.161	0.220	0.977	0.919
LOF	7	13	49	718	0.350	0.125	0.184	0.982	0.921
LOCI	9	27	47	704	0.250	0.161	0.196	0.963	0.906
SOM	8	15	48	716	0.348	0.143	0.203	0.979	0.920

Seed: 108	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	9	14	75	739	0.391	0.107	0.168	0.981	0.894
LOF	6	13	78	740	0.316	0.071	0.117	0.983	0.891
LOCI	10	33	74	720	0.233	0.119	0.157	0.956	0.872
SOM	1	11	83	742	0.083	0.012	0.021	0.985	0.888

Seed: 109	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	9	20	73	798	0.310	0.110	0.162	0.976	0.897
LOF	8	18	74	800	0.308	0.098	0.148	0.978	0.898
LOCI	21	29	61	789	0.420	0.256	0.318	0.965	0.900
SOM	6	4	76	814	0.600	0.073	0.130	0.995	0.911

A.15 Players: 1v10 | Anomaly Size: 1000% | Threshold: e-15 | Window size: 20 | k: 50%

Seed: 100	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	3	11	88	809	0.214	0.033	0.057	0.987	0.891
LOF	20	27	71	793	0.426	0.220	0.290	0.967	0.892
LOCI	10	18	81	802	0.357	0.110	0.168	0.978	0.891
SOM	7	1	84	819	0.875	0.077	0.141	0.999	0.907

Seed: 101	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	1	8	96	909	0.111	0.010	0.019	0.991	0.897
LOF	17	32	80	885	0.347	0.175	0.233	0.965	0.890
LOCI	12	16	85	901	0.429	0.124	0.192	0.983	0.900
SOM	0	1	97	916	0.000	0.000	0.000	0.999	0.903

Seed: 102	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	5	7	70	818	0.417	0.067	0.115	0.992	0.914
LOF	16	32	59	793	0.333	0.213	0.260	0.961	0.899
LOCI	12	18	63	807	0.400	0.160	0.229	0.978	0.910
SOM	9	14	66	811	0.391	0.120	0.184	0.983	0.911

Seed: 103	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	8	4	60	801	0.667	0.118	0.200	0.995	0.927
LOF	16	26	52	779	0.381	0.235	0.291	0.968	0.911
LOCI	7	10	61	795	0.412	0.103	0.165	0.988	0.919
SOM	6	13	62	792	0.316	0.088	0.138	0.984	0.914

Seed: 104	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	3	9	70	866	0.250	0.041	0.071	0.990	0.917
LOF	15	27	58	848	0.357	0.205	0.261	0.969	0.910
LOCI	7	16	66	859	0.304	0.096	0.146	0.982	0.914
SOM	5	5	68	870	0.500	0.068	0.120	0.994	0.923

Seed: 105	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy

KNN	6	8	86	878	0.429	0.065	0.113	0.991	0.904
LOF	17	27	75	859	0.386	0.185	0.250	0.970	0.896
LOCI	14	27	78	859	0.341	0.152	0.211	0.970	0.893
SOM	9	11	83	875	0.450	0.098	0.161	0.988	0.904

Seed: 106	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	6	6	81	856	0.500	0.069	0.121	0.993	0.908
LOF	15	27	72	835	0.357	0.172	0.233	0.969	0.896
LOCI	11	21	76	841	0.344	0.126	0.185	0.976	0.898
SOM	3	8	84	854	0.273	0.034	0.061	0.991	0.903

Seed: 107	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	4	4	52	727	0.500	0.071	0.125	0.995	0.929
LOF	9	26	47	705	0.257	0.161	0.198	0.964	0.907
LOCI	8	19	48	712	0.296	0.143	0.193	0.974	0.915
SOM	5	15	51	716	0.250	0.089	0.132	0.979	0.916

Seed: 108	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	2	11	82	742	0.154	0.024	0.041	0.985	0.889
LOF	10	25	74	728	0.286	0.119	0.168	0.967	0.882
LOCI	6	24	78	729	0.200	0.071	0.105	0.968	0.878
SOM	0	8	84	745	0.000	0.000	0.000	0.989	0.890

Seed: 109	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	6	7	76	811	0.462	0.073	0.126	0.991	0.908
LOF	23	24	59	794	0.489	0.280	0.357	0.971	0.908
LOCI	10	19	72	799	0.345	0.122	0.180	0.977	0.899
SOM	7	4	75	814	0.636	0.085	0.151	0.995	0.912

A.16 Players: 1v10 | Anomaly Size: 1000% | Threshold: e-15 | Window size: 75 | k: 50%

Seed: 100	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	2	4	89	816	0.333	0.022	0.041	0.995	0.898
LOF	11	19	80	801	0.367	0.121	0.182	0.977	0.891
LOCI	16	25	75	795	0.390	0.176	0.242	0.970	0.890
SOM	7	6	84	814	0.538	0.077	0.135	0.993	0.901

Seed: 101	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	0	4	97	913	0.000	0.000	0.000	0.996	0.900
LOF	8	20	89	897	0.286	0.082	0.128	0.978	0.893
LOCI	2	36	95	866	0.053	0.021	0.030	0.960	0.869
SOM	0	1	97	916	0.000	0.000	0.000	0.999	0.903

Seed: 102	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	5	8	70	817	0.385	0.067	0.114	0.990	0.913
LOF	12	19	63	806	0.387	0.160	0.226	0.977	0.909
LOCI	14	32	61	793	0.304	0.187	0.231	0.961	0.897
SOM	9	14	66	811	0.391	0.120	0.184	0.983	0.911

Seed: 103	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy

KNN	4	6	64	799	0.400	0.059	0.103	0.993	0.920
LOF	8	14	60	791	0.364	0.118	0.178	0.983	0.915
LOCI	11	17	57	788	0.393	0.162	0.229	0.979	0.915
SOM	6	13	62	792	0.316	0.088	0.138	0.984	0.914

Seed: 104	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	4	9	69	866	0.308	0.055	0.093	0.990	0.918
LOF	4	29	69	846	0.121	0.055	0.075	0.967	0.897
LOCI	15	31	58	844	0.326	0.205	0.252	0.965	0.906
SOM	6	7	67	868	0.462	0.082	0.140	0.992	0.922

Seed: 105	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	6	5	86	881	0.545	0.065	0.117	0.994	0.907
LOF	13	21	79	865	0.382	0.141	0.206	0.976	0.898
LOCI	22	34	70	852	0.393	0.239	0.297	0.962	0.894
SOM	9	11	83	875	0.450	0.098	0.161	0.988	0.904

Seed: 106	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	5	5	82	857	0.500	0.057	0.103	0.994	0.908
LOF	15	17	72	845	0.469	0.172	0.252	0.980	0.906
LOCI	16	26	71	836	0.381	0.184	0.248	0.970	0.898
SOM	4	8	83	854	0.333	0.046	0.081	0.991	0.904

Seed: 107	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	4	5	52	726	0.444	0.071	0.123	0.993	0.928
LOF	7	13	49	718	0.350	0.125	0.184	0.982	0.921
LOCI	8	29	48	702	0.216	0.143	0.172	0.960	0.902
SOM	6	18	50	713	0.250	0.107	0.150	0.975	0.914

Seed: 108	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	6	10	78	743	0.375	0.071	0.120	0.987	0.895
LOF	5	15	79	738	0.250	0.060	0.096	0.980	0.888
LOCI	9	24	75	729	0.273	0.107	0.154	0.968	0.882
SOM	0	8	84	745	0.000	0.000	0.000	0.989	0.890

Seed: 109	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	6	6	76	812	0.500	0.073	0.128	0.993	0.909
LOF	8	21	74	797	0.276	0.098	0.144	0.974	0.894
LOCI	16	24	66	794	0.400	0.195	0.262	0.971	0.900
SOM	6	4	76	814	0.600	0.073	0.130	0.995	0.911

A.17 Players: 1v10 | Anomaly Size: 1000% | Threshold: e-15 | Window size: MAX | k: 25%

Seed: 100	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	10	18	81	802	0.357	0.110	0.168	0.978	0.891
LOF	1	4	90	816	0.200	0.011	0.021	0.995	0.897

Seed: 101	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	14	32	83	885	0.304	0.144	0.196	0.965	0.887
LOF	1	5	96	912	0.167	0.010	0.019	0.995	0.900

Seed: 102	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	13	24	62	801	0.351	0.173	0.232	0.971	0.904
LOF	1	1	74	824	0.500	0.013	0.026	0.999	0.917

Seed: 103	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	10	13	58	792	0.435	0.147	0.220	0.984	0.919
LOF	2	0	66	805	1.000	0.029	0.057	1.000	0.924

Seed: 104	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	10	31	63	844	0.244	0.137	0.175	0.965	0.901
LOF	2	10	71	865	0.167	0.027	0.047	0.989	0.915

Seed: 105	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	14	26	78	860	0.350	0.152	0.212	0.971	0.894
LOF	8	9	84	877	0.471	0.087	0.147	0.990	0.905

Seed: 106	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	17	23	70	839	0.425	0.195	0.268	0.973	0.902
LOF	3	1	84	861	0.750	0.034	0.066	0.999	0.910

Seed: 107	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	9	21	47	710	0.300	0.161	0.209	0.971	0.914
LOF	0	1	56	730	0.000	0.000	0.000	0.999	0.928

Seed: 108	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	9	15	75	738	0.375	0.107	0.167	0.980	0.892
LOF	0	6	84	747	0.000	0.000	0.000	0.992	0.892

Seed: 109	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	9	19	73	799	0.321	0.110	0.164	0.977	0.898
LOF	1	2	81	816	0.333	0.012	0.024	0.998	0.908

A.18 Players: 1v10 | Anomaly Size: 1000% | Threshold: e-15 | Window size: MAX | k: 75%

Seed: 100	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	6	14	85	806	0.300	0.066	0.108	0.983	0.891
LOF	10	18	81	802	0.357	0.110	0.168	0.978	0.891

Seed: 101	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	7	27	90	890	0.206	0.072	0.107	0.971	0.885
LOF	11	19	86	898	0.367	0.113	0.173	0.979	0.896

Seed: 102	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	12	20	63	805	0.375	0.160	0.224	0.976	0.908
LOF	10	16	65	809	0.385	0.133	0.198	0.981	0.910

Seed: 103	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	8	11	60	794	0.421	0.118	0.184	0.986	0.919
LOF	9	12	59	793	0.429	0.132	0.202	0.985	0.919

Seed: 104	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	7	28	66	847	0.200	0.096	0.130	0.968	0.901
LOF	5	19	68	856	0.208	0.068	0.103	0.978	0.908

Seed: 105	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy

KNN	13	26	79	860	0.333	0.141	0.198	0.971	0.893
LOF	13	23	79	863	0.361	0.141	0.203	0.974	0.896

Seed: 106	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	16	21	71	841	0.432	0.184	0.258	0.976	0.903
LOF	11	16	76	846	0.407	0.126	0.193	0.981	0.903

Seed: 107	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	7	15	49	716	0.318	0.125	0.179	0.979	0.919
LOF	6	9	50	722	0.400	0.107	0.169	0.988	0.925

Seed: 108	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	8	14	76	739	0.364	0.095	0.151	0.981	0.892
LOF	7	13	77	740	0.350	0.083	0.135	0.983	0.892

Seed: 109	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	1	2	81	816	0.333	0.012	0.024	0.998	0.908
LOF	9	20	73	798	0.310	0.110	0.162	0.976	0.897

A.19 Players: 1v10 | Anomaly Size: 1000% | Threshold: e-15 | Window size: MAX | k: 50% | UNION

Seed: 100	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	9	17	82	803	0.346	0.099	0.154	0.979	0.891
LOF	9	16	82	804	0.360	0.099	0.155	0.980	0.892
LOCI	15	30	76	790	0.333	0.165	0.221	0.963	0.884
SOM	7	6	84	814	0.538	0.077	0.135	0.993	0.901
KNN ∪ LOF	10	21	81	799	0.323	0.110	0.164	0.974	0.888
KNN ∪ LOCI	16	33	75	787	0.327	0.176	0.229	0.960	0.881
KNN ∪ SOM	13	18	78	802	0.419	0.143	0.213	0.978	0.895
LOF ∪ LOCI	17	34	74	786	0.333	0.187	0.239	0.959	0.881
LOF ∪ SOM	13	18	78	802	0.419	0.143	0.213	0.978	0.895
LOCI ∪ SOM	15	31	76	789	0.326	0.165	0.219	0.962	0.883
KNN ∪ LOF ∪ LOCI	17	37	74	783	0.315	0.187	0.234	0.955	0.878
KNN ∪ LOF ∪ SOM	14	22	77	798	0.389	0.154	0.220	0.973	0.891
KNN ∪ LOCI ∪ SOM	16	34	75	786	0.320	0.176	0.227	0.959	0.880
LOF ∪ LOCI ∪ SOM	17	35	74	785	0.327	0.187	0.238	0.957	0.880
KNN ∪ LOF ∪ LOCI ∪ SOM	17	38	74	782	0.309	0.187	0.233	0.954	0.877

Seed: 101	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	13	28	84	889	0.317	0.134	0.188	0.969	0.890
LOF	9	18	88	899	0.333	0.093	0.145	0.980	0.895
LOCI	7	35	90	882	0.167	0.072	0.101	0.962	0.877
SOM	0	1	97	916	0.000	0.000	0.000	0.999	0.903
KNN ∪ LOF	14	28	83	889	0.333	0.144	0.201	0.969	0.891
KNN ∪ LOCI	15	39	82	878	0.278	0.155	0.199	0.957	0.881
KNN ∪ SOM	13	28	84	889	0.317	0.134	0.188	0.969	0.890
LOF ∪ LOCI	17	34	74	786	0.333	0.187	0.239	0.959	0.881
LOF ∪ SOM	9	18	88	899	0.333	0.093	0.145	0.980	0.895
LOCI ∪ SOM	7	35	90	882	0.167	0.072	0.101	0.962	0.877
KNN ∪ LOF ∪ LOCI	16	39	81	878	0.291	0.165	0.211	0.957	0.882
KNN ∪ LOF ∪ SOM	14	28	83	889	0.333	0.144	0.201	0.969	0.891
KNN ∪ LOCI ∪ SOM	15	39	82	878	0.278	0.155	0.199	0.957	0.881
LOF ∪ LOCI ∪ SOM	11	36	86	881	0.234	0.113	0.153	0.961	0.880
KNN ∪ LOF ∪ LOCI ∪ SOM	16	39	81	878	0.291	0.165	0.211	0.957	0.882

Seed: 102	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy

KNN	11	19	64	806	0.367	0.147	0.210	0.977	0.908
LOF	9	13	66	812	0.409	0.120	0.186	0.984	0.912
LOCI	20	32	55	793	0.385	0.267	0.315	0.961	0.903
SOM	9	15	66	810	0.375	0.120	0.182	0.982	0.910
KNN ∪ LOF	12	20	63	805	0.375	0.160	0.224	0.976	0.908
KNN ∪ LOCI	20	33	55	792	0.377	0.267	0.313	0.960	0.902
KNN ∪ SOM	13	23	62	802	0.361	0.173	0.234	0.972	0.906
LOF ∪ LOCI	20	32	55	793	0.385	0.267	0.315	0.961	0.903
LOF ∪ SOM	11	19	64	806	0.367	0.147	0.210	0.977	0.908
LOCI ∪ SOM	21	33	54	792	0.389	0.280	0.326	0.960	0.903
KNN ∪ LOF ∪ LOCI	20	33	55	792	0.377	0.267	0.313	0.960	0.902
KNN ∪ LOF ∪ SOM	13	24	62	801	0.351	0.173	0.232	0.971	0.904
KNN ∪ LOCI ∪ SOM	21	34	54	792	0.382	0.280	0.323	0.959	0.902
LOF ∪ LOCI ∪ SOM	21	33	54	792	0.389	0.280	0.326	0.960	0.903
KNN ∪ LOF ∪ LOCI ∪ SOM	21	34	54	791	0.382	0.280	0.323	0.959	0.902

Seed: 103	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	11	9	57	796	0.550	0.162	0.250	0.989	0.924
LOF	7	6	61	799	0.538	0.103	0.173	0.993	0.923
LOCI	10	22	58	783	0.313	0.147	0.200	0.973	0.908
SOM	7	13	61	792	0.350	0.103	0.159	0.984	0.915
KNN ∪ LOF	12	10	56	795	0.545	0.176	0.267	0.988	0.924
KNN ∪ LOCI	16	22	52	783	0.421	0.235	0.302	0.973	0.915
KNN ∪ SOM	14	17	54	788	0.452	0.206	0.283	0.979	0.919
LOF ∪ LOCI	11	22	57	783	0.333	0.162	0.218	0.973	0.910
LOF ∪ SOM	10	15	58	790	0.400	0.147	0.215	0.981	0.916
LOCI ∪ SOM	12	27	56	778	0.308	0.176	0.224	0.966	0.905
KNN ∪ LOF ∪ LOCI	16	22	52	783	0.421	0.235	0.302	0.973	0.915
KNN ∪ LOF ∪ SOM	14	18	54	787	0.438	0.206	0.280	0.978	0.918
KNN ∪ LOCI ∪ SOM	17	27	51	778	0.386	0.250	0.304	0.966	0.911
LOF ∪ LOCI ∪ SOM	13	27	55	778	0.325	0.191	0.241	0.966	0.906
KNN ∪ LOF ∪ LOCI ∪ SOM	17	27	51	778	0.386	0.250	0.304	0.966	0.911

Seed: 104	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	9	29	64	846	0.237	0.123	0.162	0.967	0.902
LOF	6	22	67	853	0.214	0.082	0.119	0.975	0.906
LOCI	15	45	58	830	0.250	0.205	0.226	0.949	0.891
SOM	6	6	67	869	0.500	0.082	0.141	0.993	0.923
KNN ∪ LOF	17	27	51	778	0.386	0.250	0.304	0.966	0.911
KNN ∪ LOCI	18	50	55	825	0.265	0.247	0.255	0.943	0.889
KNN ∪ SOM	10	30	63	845	0.250	0.137	0.177	0.966	0.902
LOF ∪ LOCI	16	48	57	827	0.250	0.219	0.234	0.945	0.889
LOF ∪ SOM	10	23	63	852	0.303	0.137	0.189	0.974	0.909
LOCI ∪ SOM	17	46	56	829	0.270	0.233	0.250	0.947	0.892
KNN ∪ LOF ∪ LOCI	18	51	55	824	0.261	0.247	0.254	0.942	0.888
KNN ∪ LOF ∪ SOM	10	31	63	844	0.244	0.137	0.175	0.965	0.901
KNN ∪ LOCI ∪ SOM	18	51	55	824	0.261	0.247	0.254	0.942	0.888
LOF ∪ LOCI ∪ SOM	18	49	55	826	0.269	0.247	0.257	0.944	0.890
KNN ∪ LOF ∪ LOCI ∪ SOM	18	52	55	823	0.257	0.247	0.252	0.941	0.887

Seed: 105	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	15	26	77	860	0.366	0.163	0.226	0.971	0.895
LOF	10	17	82	869	0.370	0.109	0.168	0.981	0.899
LOCI	30	44	62	842	0.405	0.326	0.361	0.950	0.892
SOM	9	15	83	871	0.375	0.098	0.155	0.983	0.900
KNN ∪ LOF	15	26	77	860	0.366	0.163	0.226	0.971	0.895
KNN ∪ LOCI	31	44	61	842	0.413	0.337	0.371	0.950	0.893
KNN ∪ SOM	16	28	76	858	0.364	0.174	0.235	0.968	0.894

LOF ∪ LOCI	31	44	61	842	0.413	0.337	0.371	0.950	0.893
LOF ∪ SOM	13	20	79	866	0.394	0.141	0.208	0.977	0.899
LOCI ∪ SOM	30	45	62	841	0.400	0.326	0.359	0.949	0.891
KNN ∪ LOF ∪ LOCI	31	44	61	842	0.413	0.337	0.371	0.950	0.893
KNN ∪ LOF ∪ SOM	16	28	76	858	0.364	0.174	0.235	0.968	0.894
KNN ∪ LOCI ∪ SOM	31	45	61	841	0.408	0.337	0.369	0.949	0.892
LOF ∪ LOCI ∪ SOM	31	45	61	841	0.408	0.337	0.369	0.949	0.892
KNN ∪ LOF ∪ LOCI ∪ SOM	31	45	61	841	0.408	0.337	0.369	0.949	0.892

Seed: 106	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	16	22	71	840	0.421	0.184	0.256	0.974	0.902
LOF	11	15	76	847	0.423	0.126	0.195	0.983	0.904
LOCI	19	31	68	831	0.380	0.218	0.277	0.964	0.896
SOM	2	8	85	854	0.200	0.023	0.041	0.991	0.902
KNN ∪ LOF	17	22	70	840	0.436	0.195	0.270	0.974	0.903
KNN ∪ LOCI	23	35	64	827	0.397	0.264	0.317	0.959	0.896
KNN ∪ SOM	16	22	71	840	0.421	0.184	0.256	0.974	0.902
LOF ∪ LOCI	21	33	66	829	0.389	0.241	0.298	0.962	0.896
LOF ∪ SOM	11	15	76	847	0.423	0.126	0.195	0.983	0.904
LOCI ∪ SOM	19	31	68	831	0.380	0.218	0.277	0.964	0.896
KNN ∪ LOF ∪ LOCI	24	35	63	827	0.407	0.276	0.329	0.959	0.897
KNN ∪ LOF ∪ SOM	17	22	70	840	0.436	0.195	0.270	0.974	0.903
KNN ∪ LOCI ∪ SOM	23	35	64	827	0.397	0.264	0.317	0.959	0.896
LOF ∪ LOCI ∪ SOM	21	33	66	829	0.389	0.241	0.298	0.962	0.896
KNN ∪ LOF ∪ LOCI ∪ SOM	24	35	63	827	0.407	0.276	0.329	0.959	0.897

Seed: 107	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	9	17	47	714	0.346	0.161	0.220	0.977	0.919
LOF	6	6	50	725	0.500	0.107	0.176	0.992	0.929
LOCI	8	26	48	705	0.235	0.143	0.178	0.964	0.906
SOM	5	13	51	718	0.278	0.089	0.135	0.982	0.919
KNN ∪ LOF	9	17	47	714	0.346	0.161	0.220	0.977	0.919
KNN ∪ LOCI	11	29	45	702	0.275	0.196	0.229	0.960	0.906
KNN ∪ SOM	9	23	47	708	0.281	0.161	0.205	0.969	0.911
LOF ∪ LOCI	10	27	46	704	0.270	0.179	0.215	0.963	0.907
LOF ∪ SOM	7	16	49	715	0.304	0.125	0.177	0.978	0.917
LOCI ∪ SOM	8	31	48	700	0.205	0.143	0.168	0.958	0.900
KNN ∪ LOF ∪ LOCI	11	29	45	702	0.275	0.196	0.229	0.960	0.906
KNN ∪ LOF ∪ SOM	9	23	47	708	0.281	0.161	0.205	0.969	0.911
KNN ∪ LOCI ∪ SOM	11	34	45	697	0.244	0.196	0.218	0.953	0.900
LOF ∪ LOCI ∪ SOM	10	32	46	699	0.238	0.179	0.204	0.956	0.901
KNN ∪ LOF ∪ LOCI ∪ SOM	11	34	45	697	0.244	0.196	0.218	0.953	0.900

Seed: 108	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	9	14	75	739	0.391	0.107	0.168	0.981	0.894
LOF	5	13	79	740	0.278	0.060	0.098	0.983	0.890
LOCI	9	31	75	722	0.225	0.107	0.145	0.959	0.873
SOM	0	8	84	745	0.000	0.000	0.000	0.989	0.890
KNN ∪ LOF	9	14	75	739	0.391	0.107	0.168	0.981	0.894
KNN ∪ LOCI	13	31	71	722	0.295	0.155	0.203	0.959	0.878
KNN ∪ SOM	9	14	75	739	0.391	0.107	0.168	0.981	0.894
LOF ∪ LOCI	13	31	71	722	0.295	0.155	0.203	0.959	0.878
LOF ∪ SOM	5	13	79	740	0.278	0.060	0.098	0.983	0.890
LOCI ∪ SOM	9	31	75	722	0.225	0.107	0.145	0.959	0.873
KNN ∪ LOF ∪ LOCI	13	31	71	722	0.295	0.155	0.203	0.959	0.878
KNN ∪ LOF ∪ SOM	9	14	75	739	0.391	0.107	0.168	0.981	0.894
KNN ∪ LOCI ∪ SOM	13	31	71	722	0.295	0.155	0.203	0.959	0.878
LOF ∪ LOCI ∪ SOM	13	31	71	722	0.295	0.155	0.203	0.959	0.878

KNN ∪ LOF ∪ LOCI ∪ SOM	13	31	71	722	0.295	0.155	0.203	0.959	0.878

Seed: 109	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	9	19	73	799	0.321	0.110	0.164	0.977	0.898
LOF	9	18	73	800	0.333	0.110	0.165	0.978	0.899
LOCI	16	28	66	790	0.364	0.195	0.254	0.966	0.896
SOM	5	3	77	815	0.625	0.061	0.111	0.996	0.911
KNN ∪ LOF	10	19	72	799	0.345	0.122	0.180	0.977	0.899
KNN ∪ LOCI	16	28	66	790	0.364	0.195	0.254	0.966	0.896
KNN ∪ SOM	10	19	72	799	0.345	0.122	0.180	0.977	0.899
LOF ∪ LOCI	17	28	65	790	0.378	0.207	0.268	0.966	0.897
LOF ∪ SOM	10	18	72	800	0.357	0.122	0.182	0.978	0.900
LOCI ∪ SOM	17	28	65	790	0.378	0.207	0.268	0.966	0.897
KNN ∪ LOF ∪ LOCI	17	28	65	790	0.378	0.207	0.268	0.966	0.897
KNN ∪ LOF ∪ SOM	11	19	71	799	0.367	0.134	0.196	0.977	0.900
KNN ∪ LOCI ∪ SOM	17	28	65	790	0.378	0.207	0.268	0.966	0.897
LOF ∪ LOCI ∪ SOM	18	28	64	790	0.391	0.220	0.281	0.966	0.898
KNN ∪ LOF ∪ LOCI ∪ SOM	18	28	64	790	0.391	0.220	0.281	0.966	0.898

A.20 Players: 1v10 | Anomaly Size: 1000% | Threshold: e-15 | Window size: MAX | k: 50% | INTERSECTION

Seed: 100	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	9	17	82	803	0.346	0.099	0.154	0.979	0.891
LOF	9	16	82	804	0.360	0.099	0.155	0.980	0.892
LOCI	15	30	76	790	0.333	0.165	0.221	0.963	0.884
SOM	7	6	84	814	0.538	0.077	0.135	0.993	0.901
KNN ∩ LOF	8	12	83	808	0.400	0.088	0.144	0.985	0.896
KNN ∩ LOCI	8	14	83	806	0.364	0.088	0.142	0.983	0.894
KNN ∩ SOM	3	5	88	815	0.375	0.033	0.061	0.994	0.898
LOF ∩ LOCI	8	13	83	807	0.381	0.088	0.143	0.984	0.895
LOF ∩ SOM	3	4	88	816	0.429	0.033	0.061	0.995	0.899
LOCI ∩ SOM	7	6	84	814	0.538	0.077	0.135	0.993	0.901
KNN ∩ LOF ∩ LOCI	8	13	83	807	0.381	0.088	0.143	0.984	0.895
KNN ∩ LOF ∩ SOM	3	4	88	816	0.429	0.033	0.061	0.995	0.899
KNN ∩ LOCI ∩ SOM	3	6	88	814	0.333	0.033	0.060	0.993	0.897
LOF ∩ LOCI ∩ SOM	3	4	88	816	0.429	0.033	0.061	0.995	0.899
KNN ∩ LOF ∩ LOCI ∩ SOM	3	4	88	816	0.429	0.033	0.061	0.995	0.899

Seed: 101	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	13	28	84	889	0.317	0.134	0.188	0.969	0.890
LOF	9	18	88	899	0.333	0.093	0.145	0.980	0.895
LOCI	7	35	90	882	0.167	0.072	0.101	0.962	0.877
SOM	0	1	97	916	0.000	0.000	0.000	0.999	0.903
KNN ∩ LOF	8	18	89	899	0.308	0.082	0.130	0.980	0.894
KNN ∩ LOCI	5	24	92	893	0.172	0.052	0.079	0.974	0.886
KNN ∩ SOM	0	1	97	916	0.000	0.000	0.000	0.999	0.903
LOF ∩ LOCI	5	17	92	900	0.227	0.052	0.084	0.981	0.893
LOF ∩ SOM	0	1	97	916	0.000	0.000	0.000	0.999	0.903
LOCI ∩ SOM	0	1	97	916	0.000	0.000	0.000	0.999	0.903
KNN ∩ LOF ∩ LOCI	5	17	92	900	0.227	0.052	0.084	0.981	0.893
KNN ∩ LOF ∩ SOM	0	1	97	916	0.000	0.000	0.000	0.999	0.903
KNN ∩ LOCI ∩ SOM	0	1	97	916	0.000	0.000	0.000	0.999	0.903
LOF ∩ LOCI ∩ SOM	0	1	97	916	0.000	0.000	0.000	0.999	0.903
KNN ∩ LOF ∩ LOCI ∩ SOM	0	1	97	916	0.000	0.000	0.000	0.999	0.903

Seed: 102	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy

KNN	11	19	64	806	0.367	0.147	0.210	0.977	0.908
LOF	9	13	66	812	0.409	0.120	0.186	0.984	0.912
LOCI	20	32	55	793	0.385	0.267	0.315	0.961	0.903
SOM	9	15	66	810	0.375	0.120	0.182	0.982	0.910
KNN ∩ LOF	8	12	67	813	0.400	0.107	0.168	0.985	0.912
KNN ∩ LOCI	11	18	64	807	0.379	0.147	0.212	0.978	0.909
KNN ∩ SOM	7	11	68	814	0.389	0.093	0.151	0.987	0.912
LOF ∩ LOCI	9	13	66	812	0.409	0.120	0.186	0.984	0.912
LOF ∩ SOM	7	10	68	815	0.412	0.093	0.152	0.988	0.913
LOCI ∩ SOM	8	14	67	811	0.364	0.107	0.165	0.983	0.910
KNN ∩ LOF ∩ LOCI	8	12	67	813	0.400	0.107	0.168	0.985	0.912
KNN ∩ LOF ∩ SOM	6	10	69	815	0.375	0.080	0.132	0.988	0.912
KNN ∩ LOCI ∩ SOM	7	11	68	814	0.389	0.093	0.151	0.987	0.912
LOF ∩ LOCI ∩ SOM	7	10	68	815	0.412	0.093	0.152	0.988	0.913
KNN ∩ LOF ∩ LOCI ∩ SOM	6	10	69	815	0.375	0.080	0.132	0.988	0.912

Seed: 103	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	11	9	57	796	0.550	0.162	0.250	0.989	0.924
LOF	7	6	61	799	0.538	0.103	0.173	0.993	0.923
LOCI	10	22	58	783	0.313	0.147	0.200	0.973	0.908
SOM	7	13	61	792	0.350	0.103	0.159	0.984	0.915
KNN ∩ LOF	6	5	62	800	0.545	0.088	0.152	0.994	0.923
KNN ∩ LOCI	5	9	63	796	0.357	0.074	0.122	0.989	0.918
KNN ∩ SOM	4	7	64	798	0.364	0.059	0.101	0.991	0.919
LOF ∩ LOCI	6	6	62	799	0.500	0.088	0.150	0.993	0.922
LOF ∩ SOM	4	7	64	798	0.364	0.059	0.101	0.991	0.919
LOCI ∩ SOM	5	9	63	796	0.357	0.074	0.122	0.989	0.918
KNN ∩ LOF ∩ LOCI	5	5	63	800	0.500	0.074	0.128	0.994	0.922
KNN ∩ LOF ∩ SOM	3	7	65	798	0.300	0.044	0.077	0.991	0.918
KNN ∩ LOCI ∩ SOM	3	6	65	799	0.333	0.044	0.078	0.993	0.919
LOF ∩ LOCI ∩ SOM	4	7	64	798	0.364	0.059	0.101	0.991	0.919
KNN ∩ LOF ∩ LOCI ∩ SOM	3	7	65	798	0.300	0.044	0.077	0.991	0.918

Seed: 104	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	9	29	64	846	0.237	0.123	0.162	0.967	0.902
LOF	6	22	67	853	0.214	0.082	0.119	0.975	0.906
LOCI	15	45	58	830	0.250	0.205	0.226	0.949	0.891
SOM	6	6	67	869	0.500	0.082	0.141	0.993	0.923
KNN ∩ LOF	6	21	67	854	0.222	0.082	0.120	0.976	0.907
KNN ∩ LOCI	6	24	67	851	0.200	0.082	0.117	0.973	0.904
KNN ∩ SOM	5	5	68	870	0.500	0.068	0.120	0.994	0.923
LOF ∩ LOCI	5	19	68	856	0.208	0.068	0.103	0.978	0.908
LOF ∩ SOM	2	5	71	870	0.286	0.027	0.050	0.994	0.920
LOCI ∩ SOM	4	5	69	870	0.444	0.055	0.098	0.994	0.922
KNN ∩ LOF ∩ LOCI	5	19	68	856	0.208	0.068	0.103	0.978	0.908
KNN ∩ LOF ∩ SOM	2	5	71	870	0.286	0.027	0.050	0.994	0.920
KNN ∩ LOCI ∩ SOM	3	5	70	870	0.375	0.041	0.074	0.994	0.921
LOF ∩ LOCI ∩ SOM	2	5	71	870	0.286	0.027	0.050	0.994	0.920
KNN ∩ LOF ∩ LOCI ∩ SOM	2	5	71	870	0.286	0.027	0.050	0.994	0.920

Seed: 105	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	15	26	77	860	0.366	0.163	0.226	0.971	0.895
LOF	10	17	82	869	0.370	0.109	0.168	0.981	0.899
LOCI	30	44	62	842	0.405	0.326	0.361	0.950	0.892
SOM	9	15	83	871	0.375	0.098	0.155	0.983	0.900
KNN ∩ LOF	10	17	82	869	0.370	0.109	0.168	0.981	0.899
KNN ∩ LOCI	14	26	78	860	0.350	0.152	0.212	0.971	0.894
KNN ∩ SOM	8	13	84	873	0.381	0.087	0.142	0.985	0.901

LOF ∩ LOCI	9	17	83	869	0.346	0.098	0.153	0.981	0.898
LOF ∩ SOM	6	13	86	873	0.316	0.065	0.108	0.985	0.899
LOCI ∩ SOM	9	14	83	872	0.391	0.098	0.157	0.984	0.901
KNN ∩ LOF ∩ LOCI	9	17	83	869	0.346	0.098	0.153	0.981	0.898
KNN ∩ LOF ∩ SOM	6	13	86	873	0.316	0.065	0.108	0.985	0.899
KNN ∩ LOCI ∩ SOM	8	13	84	873	0.381	0.087	0.142	0.985	0.901
LOF ∩ LOCI ∩ SOM	6	13	86	873	0.316	0.065	0.108	0.985	0.899
KNN ∩ LOF ∩ LOCI ∩ SOM	6	13	86	873	0.316	0.065	0.108	0.985	0.899

Seed: 106	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	16	22	71	840	0.421	0.184	0.256	0.974	0.902
LOF	11	15	76	847	0.423	0.126	0.195	0.983	0.904
LOCI	19	31	68	831	0.380	0.218	0.277	0.964	0.896
SOM	2	8	85	854	0.200	0.023	0.041	0.991	0.902
KNN ∩ LOF	10	15	77	847	0.400	0.115	0.179	0.983	0.903
KNN ∩ LOCI	12	18	75	844	0.400	0.138	0.205	0.979	0.902
KNN ∩ SOM	2	8	85	854	0.200	0.023	0.041	0.991	0.902
LOF ∩ LOCI	9	13	78	849	0.409	0.103	0.165	0.985	0.904
LOF ∩ SOM	2	8	85	854	0.200	0.023	0.041	0.991	0.902
LOCI ∩ SOM	2	8	85	854	0.200	0.023	0.041	0.991	0.902
KNN ∩ LOF ∩ LOCI	9	13	78	849	0.409	0.103	0.165	0.985	0.904
KNN ∩ LOF ∩ SOM	2	8	85	854	0.200	0.023	0.041	0.991	0.902
KNN ∩ LOCI ∩ SOM	2	8	85	854	0.200	0.023	0.041	0.991	0.902
LOF ∩ LOCI ∩ SOM	2	8	85	854	0.200	0.023	0.041	0.991	0.902
KNN ∩ LOF ∩ LOCI ∩ SOM	2	8	85	854	0.200	0.023	0.041	0.991	0.902

Seed: 107	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	9	17	47	714	0.346	0.161	0.220	0.977	0.919
LOF	6	6	50	725	0.500	0.107	0.176	0.992	0.929
LOCI	8	26	48	705	0.235	0.143	0.178	0.964	0.906
SOM	5	13	51	718	0.278	0.089	0.135	0.982	0.919
KNN ∩ LOF	6	6	50	725	0.500	0.107	0.176	0.992	0.929
KNN ∩ LOCI	6	14	50	717	0.300	0.107	0.158	0.981	0.919
KNN ∩ SOM	5	9	51	722	0.357	0.089	0.143	0.988	0.924
LOF ∩ LOCI	4	5	52	726	0.444	0.071	0.123	0.993	0.928
LOF ∩ SOM	4	4	52	727	0.500	0.071	0.125	0.995	0.929
LOCI ∩ SOM	5	9	51	722	0.357	0.089	0.143	0.988	0.924
KNN ∩ LOF ∩ LOCI	4	5	52	726	0.444	0.071	0.123	0.993	0.928
KNN ∩ LOF ∩ SOM	4	4	52	727	0.500	0.071	0.125	0.995	0.929
KNN ∩ LOCI ∩ SOM	5	8	51	723	0.385	0.089	0.145	0.989	0.925
LOF ∩ LOCI ∩ SOM	4	4	52	727	0.500	0.071	0.125	0.995	0.929
KNN ∩ LOF ∩ LOCI ∩ SOM	4	4	52	727	0.500	0.071	0.125	0.995	0.929

Seed: 108	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	9	14	75	739	0.391	0.107	0.168	0.981	0.894
LOF	5	13	79	740	0.278	0.060	0.098	0.983	0.890
LOCI	9	31	75	722	0.225	0.107	0.145	0.959	0.873
SOM	0	8	84	745	0.000	0.000	0.000	0.989	0.890
KNN ∩ LOF	5	13	79	740	0.278	0.060	0.098	0.983	0.890
KNN ∩ LOCI	5	14	79	739	0.263	0.060	0.097	0.981	0.889
KNN ∩ SOM	0	8	84	745	0.000	0.000	0.000	0.989	0.890
LOF ∩ LOCI	3	13	81	740	0.188	0.036	0.060	0.983	0.888
LOF ∩ SOM	0	8	84	745	0.000	0.000	0.000	0.989	0.890
LOCI ∩ SOM	0	8	84	745	0.000	0.000	0.000	0.989	0.890
KNN ∩ LOF ∩ LOCI	3	13	81	740	0.188	0.036	0.060	0.983	0.888
KNN ∩ LOF ∩ SOM	0	8	84	745	0.000	0.000	0.000	0.989	0.890
KNN ∩ LOCI ∩ SOM	0	8	84	745	0.000	0.000	0.000	0.989	0.890
LOF ∩ LOCI ∩ SOM	0	8	84	745	0.000	0.000	0.000	0.989	0.890

KNN ∩ LOF ∩ LOCI ∩ SOM	0	8	84	745	0.000	0.000	0.000	0.989	0.890

Seed: 109	TP	FP	FN	TN	Precision	Recall, TPR	F-Score	TNR	Accuracy
KNN	9	19	73	799	0.321	0.110	0.164	0.977	0.898
LOF	9	18	73	800	0.333	0.110	0.165	0.978	0.899
LOCI	16	28	66	790	0.364	0.195	0.254	0.966	0.896
SOM	5	3	77	815	0.625	0.061	0.111	0.996	0.911
KNN ∩ LOF	8	18	74	800	0.308	0.098	0.148	0.978	0.898
KNN ∩ LOCI	9	19	73	799	0.321	0.110	0.164	0.977	0.898
KNN ∩ SOM	4	3	78	815	0.571	0.049	0.090	0.996	0.910
LOF ∩ LOCI	9	18	73	800	0.333	0.110	0.165	0.978	0.899
LOF ∩ SOM	4	3	78	815	0.571	0.049	0.090	0.996	0.910
LOCI ∩ SOM	4	3	78	815	0.571	0.049	0.090	0.996	0.910
KNN ∩ LOF ∩ LOCI	9	18	73	800	0.333	0.110	0.165	0.978	0.899
KNN ∩ LOF ∩ SOM	4	3	78	815	0.571	0.049	0.090	0.996	0.910
KNN ∩ LOCI ∩ SOM	4	3	78	815	0.571	0.049	0.090	0.996	0.910
LOF ∩ LOCI ∩ SOM	4	3	78	815	0.571	0.049	0.090	0.996	0.910
KNN ∩ LOF ∩ LOCI ∩ SOM	4	3	78	815	0.571	0.049	0.090	0.996	0.910