Master Research Internship
Outlier detection on file server backup data

created by

Frederic Marvin Abraham
i6262598

RWTH:	Dr. Thomas Eifert
DKE:	Prof. Kurt Driessens

University of Maastricht
Data Science & Knowledge Engineering
Maastricht, January 24, 2022

Contents

Abbreviations

k-NN	k-nearest neighbors
SUOD	Scalable Unsupervised Outlier Detection
LOF	Local Outlier Factor
COF	Connectivity-based Outlier Factor
OCSVM	one-class support vector machines
CDF	cumulative distribution function

Abstract

Universities and organisations in education have increasingly become the target of cyberattacks due to their large attack vector. File servers that form the backbone of the different working processes are the primary target of these attacks. In the face of these dangers, data protection in the form of consecutive incremental backups is required. This internship's primary goal was to develop a machine learning tool capable of finding malicious activities that occurred on a secured file server by classifying the backups metadata into the respective normal mode and or being an outlier.

To archive this goal, different unsupervised outlier detection methods were researched, combined with the selection of two suitable techniques matching the requirements. The first approach uses state of the art outlier detection ensemble method SUOD on extracted and vectorised backup metadata, with the second approach addressing how path data of the individual files are parsed into multisets and used in One-Class Support Vector Machines. This internship further addressed a design concept for deploying the components of the ml-tool, including a web interface for visualising and interacting, a backend API and a database for persistent storing and continuously training the models on new data.

The evaluation of is these approaches on commit data of a large git project parsed into backup formats showed how the two methods classified the backups by their respective features and isolated different characteristics as outliers. The models further provide an outlier probability coupled with the model's certainty for the prediction.

1 Introduction

In recent years, universities have increasingly become the target of cyberattacks. According to a survey of 499 education organizations by Adam 2021, 44% of organizations in higher education were hit by a ransomware attack in the year 2020, of which 54% succeeded in encrypting the data. This relatively large amount of attacks could be due to several different reasons. One reason could be an often low amount of security, coupled with many employees and students that form a considerable attack vector. Additionally, as most cyberattacks are financially motivated, universities are more likely to pay the ransom. Of the universities that got their data encrypted, 35% paid the ransom to get their data back, while only 55% restored their data through prior backups.

The conducted survey focused on ransomware attacks, which encrypts the data to which it has read and write access. Attacks that infiltrate a system, leaving software behind that only modify specific files, could be more subtle to go undetected. Different attacks produce different access and modification patterns that deviate from the norm.

The underlying targets of all attacks are file servers that form the backbone of the working process in many companies and universities. Employees and students of the RWTH-Aachen use file servers to store work-related files and data daily. The main goal of this internship is a classification model capable of differentiating between a malicious action on a file server and the usual mode of operation.

1.1 Background Information

With the motivation of rising cybercrime in mind, developing an operating model is to establish cross-university data protection in and for NRW called the DaSi.NRW project. (Filla 2021) North Rhine-Westphalia has 14 Universities, 16 Universities of Applied Sciences and seven colleges of art and music. The project's primary goal is that 3-5 universities become able to provide backup infrastructure as a service provider, and the remaining entities become service users employing the backup capabilities offered.

The Commvault Data Management Solution is used to provide backup capabilities. It includes the central *CommServ* server component, which coordinates, manages, and monitors all activity in a *CommCell* environment. All software components that manage, move, store and protect data are grouped logically in a CommCell environment. A file server, or other server types referenced as clients, are protected through installed *MediaAgents* that provide an interface between the central management server and the client. The MediaAgent calculates incremental backups that are copied to the offside storage infrastructure by indexing the protected data. (Commvault Systems® 2019)

1.2 Problem Statement & Research Questions

This internship mainly focuses on these incremental backups. They are classified to decide if malicious activities occurred on any file server. Incremental backups are consecutive backups that only copy the data that has changed since the previous backup occurred. Due to the lack of labelled training data, unsupervised detection methods need to be employed.

By considering an incremental backup as a data point, the usual backups that do not contain any malicious behaviour represent the standard mode of operation of a file server and presumably form a cluster. Does a backup that contains malicious activities deviate enough from the standard cluster so that it can be identified as an outlier? Formally defined in the first research question:

Is it possible to detect anomalies in file modification patterns of a file server based on backup log data compared to a learned standard mode?

The field of research describes the problem of identifying rare data points that are deviant compared to the general data distribution as outlier detection, anomaly detection or novelty search. Therefore, a part of this internship includes researching different outlier detection methods resulting in chapter 2. It gives an overview of different outlier detection methods, their main intuition, advantages and disadvantages, and looks at the challenges of big data and high dimensionality in the field of outlier detection and how modern approaches handle these challenges.

Chapter 3 describes the used methods for outlier detection. Additionally, the chapter describes how an outlier probability can be calculated depending on a classification threshold and how a certainty in each prediction can be approximated. The adaptation of outlier detection methods characterized in the chapters 2 and 3 to the incremental backup data provided by the Media Agent is subsequently discussed in chapter 4.

The type of work done on each file server is subjected to change over time. Coming and going tasks, projects, or new staff working with the given file server could result in such a change. To differentiate between changing requirements on the file server or malicious behaviour on a file server is a non-trivial challenge. Therefore, the model needs to continuously change over time with the changing uses of the file server. This requirement is formally captured in the second research question:

Can the model continuously learn gradual changes in regular modification pattern's to accommodate new users or tasks?

A resulting requirement to this application is the ability to store its model components in a database with the capability to load a file server-specific model and continuously train it depending on the new backup data addressed in section 5.2. Including backups that contain malicious activities in the continued learning process could lead to problems. Reporting a malicious backup to an employee responsible for a file server and giving the said employee an interface to mark outliers as non-intended is covered by the developed web interface.

In the evaluation chapter 5 different experiments are described on created test data to assess the model's classification and adaptation capabilities, followed by a discussion of the findings in chapter 6.

2 Outlier Detection

One of the first definitions of an outlier can be credited to Grubbs 1969, as an example that appears to deviate significantly from other members of the sample in which it occurs. This definition is the basis for which future work typically makes the two assumptions: (1) outliers are different from the norm with respect to their features, and (2) outliers are rare in the dataset compared to the regular instances. (Goldstein and Uchida 2016)

Given the availability of the input data's labels, Boukerche, Zheng, and Alfandi 2020 categorizes outlier detection methods into three types: (1) supervised outlier detection, (2) semi-supervised outlier detection, and (3) unsupervised outlier detection. For their training process, Supervised methods require that each instance is either labelled as an outlier or, as an example, belonging to the expected behaviour. Semi-Supervised methods compared to the supervised ones either requires only examples of one class in their training process (Désir et al. 2013) or use a small amount of labelled data to enrich their model's classification, exemplary when including expert knowledge (Das et al. 2016). The last class created their model entirely on unlabeled data.

Searching malicious access patterns in backup data and most other outlier detection problems inherently has no labelled examples. The methods explored in this internship focus on unsupervised outlier detection, as also a majority of research has been done in this category.

2.1 Methods

At a high level, Boukerche, Zheng, and Alfandi 2020 classify the unsupervised outlier detection approaches in fundamental and advanced techniques, with the advanced approaches extending the fundamental ones to address new challenges. Challenges for outlier detection methods are, for example, **(1)** unbounded and dynamic data streams, **(2)** big data in a distributed setting with high-dimensional data and **(3)** in using expert knowledge to incorporate a limited amount of labelled data effectively.

Figure 2.1 shows the further classification of the fundamental approaches briefly reviewed in this section.

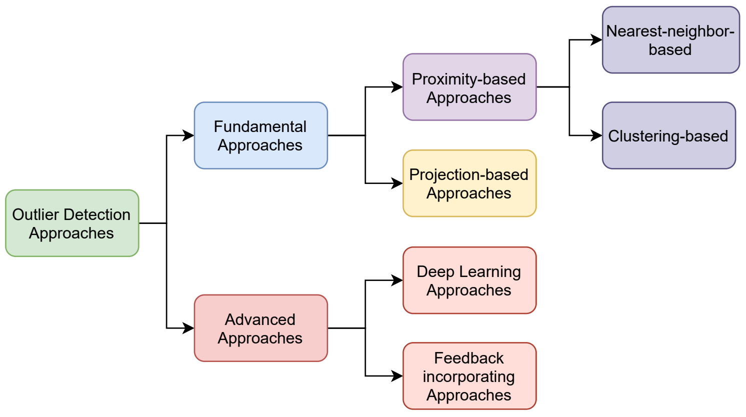

Outlier Detection Approaches
Fundamental Approaches
Advanced Approaches
Proximity-based Approaches
Projection-based Approaches
Deep Learning Approaches
Feedback incorporating Approaches
Nearest-neighbor-based
Clustering-based

Figure 2.1: Classification of outlier Detection techniques

They classify the Fundamental approaches into proximity-based and projection-based strategies. The proximity-based methods are classified into the nearest neighbour-based algorithms and approaches using clustering algorithms.

The methods classified under the proximity-based approach use the relation between nearby examples to identify outliers. In contrast, the projection-based methods translate the examples into a new space with reduced dimensionality and complexity while preserving as much information of the original data. (Boukerche, Zheng, and Alfandi 2020)

Generally, outlier detection algorithms assign each data point a score representing how anomalous the example is. Based on an a priori assumed contamination percentage, a threshold is used to turn the outlier score into the prediction.

2.1.1 Proximity Based Approaches

As the name implies, proximity-based methods function in calculating the proximity of each data point. Different techniques have a different meaning of proximity and, therefore, differentiating processes of calculating them. Generally, the approaches follow the assumption that outliers are located in sparse areas of the data and by calculating distances between each data point, the methods determine if a specific example is an outlier or not.

One of the most intuitive nearest-neighbour-based approach relies on the distance to the kth nearest neighbour by Ramaswamy, Rastogi, and Shim 2000. The average distance to their k-nearest neighbors (k-NN) is calculated for every instance to create an outlier ranking. This algorithm is often just referenced as k-NN. Elements further from the expected distribution result in a higher score as the distance to their neighbours is higher. Using the assumed contamination, they declare the top n points in this ranking to be outliers. The drawback of this approach is the indirect assumption of the same density of inliers over the whole dataset. Approaches that take varying densities between clusters into account outperform the simple k-NN technique in many cases.

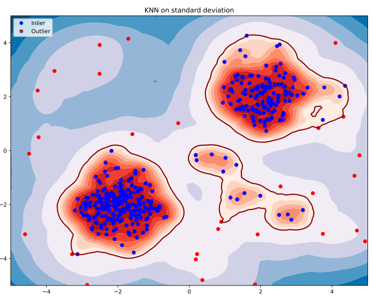


Figure 2.2: k-NN example on

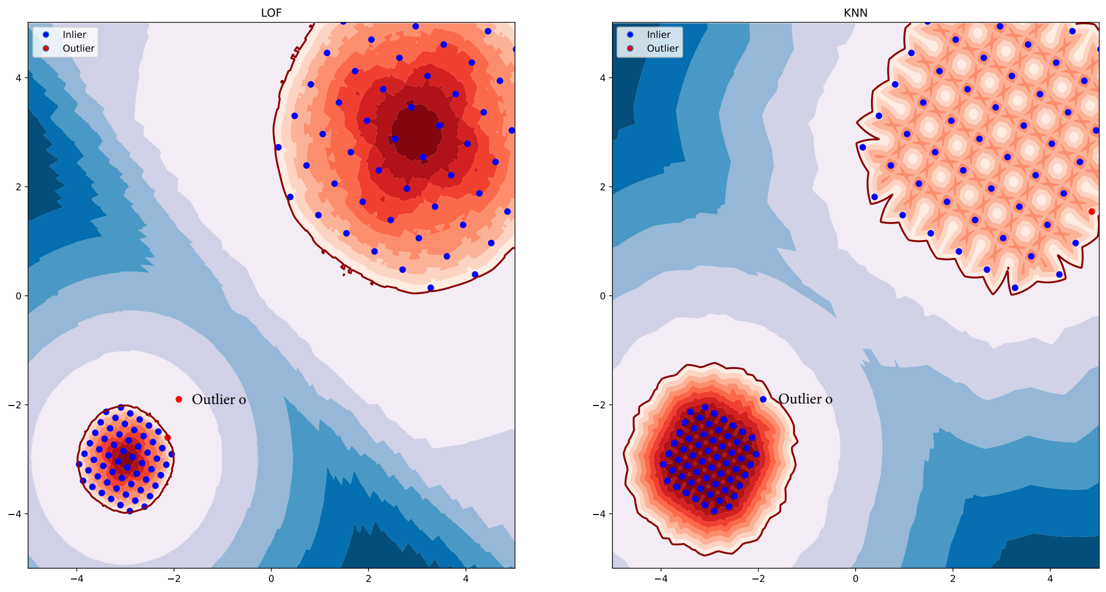


Figure 2.3: KNN compared to LOF

The Figure 2.3 shows a similar artificial dataset introduced by Breunig et al. 2000. Blue marked points marked as inliers and red points as detected outliers. The red line describes the decision boundary of each model. It underlines the shortcoming mentioned above of the k-NN approach by having two distributions with different densities. The distribution with more space between each datapoint compared to the outliers distance to its distribution results in the outlier, marked as o in the figure, being not detectable by the k-NN model as the outliers distance falls below the threshold.

The approach that takes this variation into account, considering local outliers, is introduced by Breunig et al. 2000 as the Local Outlier Factor (LOF) score. It defines the outlier score of a data point as the average ratio of its neighbour's density compared to its density. The main idea behind this approach is that outlier's density is usually smaller than that of their neighbours, resulting in a higher LOF score for outliers.

Calculating the density requires the euclidean distanced neighbourhood, which results in the assumption of a spherical distribution. This assumption gets problematic given the linear dataset, visible in figure 2.4, as the outlier is detected as belonging to the distribution. The Connectivity-based Outlier Factor (COF) introduced by Tang et al. 2002 addresses the shortcoming of LOF by introducing a "isolativty" for a data point that is calculated as the shortest path between itself and its k neighbours.

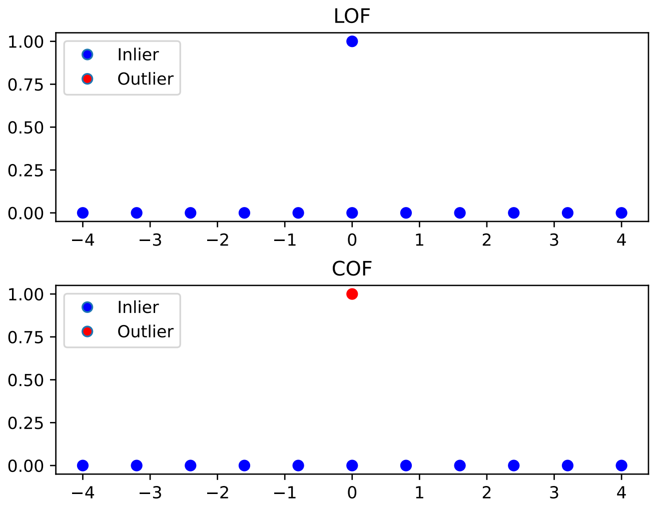


Figure 2.4: LOF compared to COF

Further research in the nearest neighbour-based approaches focuses on removing the outlier cutoff threshold that determines if a data point is an outlier given the scores of the whole dataset. Or speeding up the approach by subsampling and ensembling the neighbourhood of an example. (Boukerche, Zheng, and Alfandi 2020)

Clustering-based approaches follow a two-step approach: The first is to create clusters over the data, followed by checking the distances of each data point to its clusters. Most outlier cluster algorithms follow the assumptions that: 1. an outlier is further from a cluster centre than the average data points, and 2. that the cluster of outliers is sparse and small compared to large and dense of normal data points. A significant advantage of clustering approaches compared to the nearest-neighbour-based approach is its efficiency due to the pairwise distance computation in most nearest-neighbour-based approaches. On the other hand, nearest-neighbour-based approaches are usually more granular than clustering approaches. (Boukerche, Zheng, and Alfandi 2020)

2.1.2 Projection Based Approaches

Converting the data points into a new space and reducing their dimensionality is a technique that needs to be employed when facing larger datasets, The reduction is followed by a proximity-based approach in the projected space resulting in an improved runtime. The primary task in this dimensionality reduction is to preserve the proximity information of the original dataset.

One approach that uses LOF as underlying method is the Projection-indexed Nearest-neighbors (PINN) by Vries, Chawla, and Houle 2010. It uses Johnson and Lindenstrauss lemma (Johnson 1984) that states “that a set of points in a high-dimensional space can be embedded into a space of much lower dimension in such a way that distances between the points are nearly preserved” (Achlioptas 2003) visualized in figure 2.5. This approach results in a much faster euclidean distances calculation with a given probability of correctness, speeding up the most costly computation of the k nearest neighbours of each data point. They are determining the k-NN data points in the projected lower dimension to calculate the LOF score in the original dimension. The major advantage over PCA, which would result in a better representation in the lower space, is its speed while maintaining the required approximate Euclidean distances.

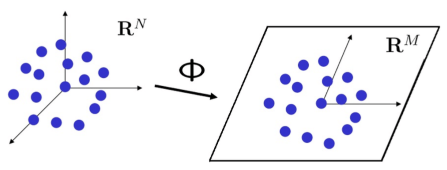


Figure 2.5: Johnson-Lindenstrauss lemma’s illustration (Source: Fejri 2017)

Another approach introduced by Wang, Parthasarathy, and Tatikonda 2011 employs a hashing algorithm that maps similar data points into one-dimensional hash buckets. More similar data points are mapped into the same bucket with a higher probability than less similar ones.

$$h(v) = \left\lfloor \frac{a \cdot v + b}{w} \right\rfloor \tag{2.1}$$

The equation 2.1 describes the locality-sensitive hashing function where v is the data point, a is a random vector from a p-stable distribution, b is a random real number chosen from a uniform distribution bounded by $[0, w]$ where w is a real number provided. A ranking over the data points is created by counting the number of elements in its correlated buckets, assuming that outliers are in buckets with small numbers.

A different approach for dimensionality reduction compared to random projections are space-filling curves. A space-filling curve is usually used to reduce a vector into one-dimensional space. Schubert, Zimek, and Kriegel 2015 uses an ensemble of n space-filling curves on which each data point is projected, resulting in a vector with a cardinality of n. He argues that space-filling curves are better in preserving the closeness of data points compared to hashing approaches or random projections.

Isolation Forests represent an entirely different approach to outlier detection than the regular dimensionality reduction and k-NN search. Liu, Ting, and Zhou 2009 introduced the unsupervised tree ensemble method. It stipulates that a binary tree is created by splitting the data repeatedly in two sections by recursively selecting a random attribute and value until a terminal height is reached. The main intuition visualized in figure 2.6 is that outliers are easier to separate from the remaining data with fewer cuts at a specific value, resulting in a higher avg position in the tree. The outlier score is calculated as the ratio between its average height and its expected height, defined as the average height of failed searches.

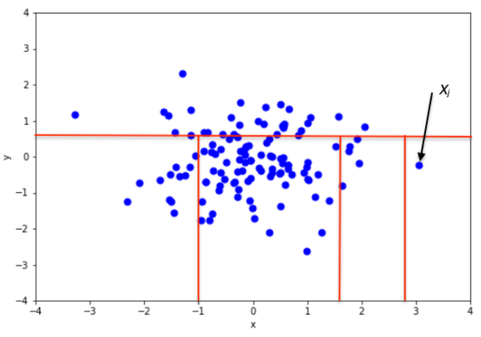


Figure 2.6: Isolating a data point in 4 cuts. (Source: Wikipedia Isolation forest)

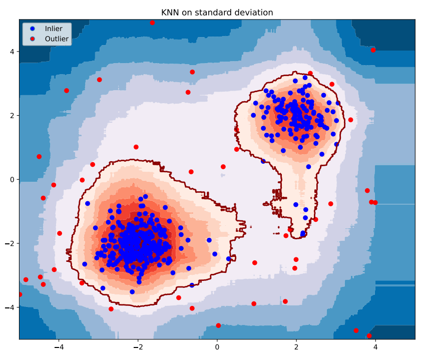


Figure 2.7: Isolation forest on two standard normal distributions.

Without calculating the pairwise distances, this approach archives a linear time complexity. Figure 2.7 shows Isolation Forests performing on the dataset with points from two normal distributions.

2.1.3 Advanced Approaches

In the face of big data and the non-linearity of data, neural networks with deep learning are increasingly used to model nonlinear representation. They lead to advancements in multiple fields such as speech recognition, natural language processing and object detection. (Boukerche, Zheng, and Alfandi 2020) While the initial idea of neural networks was firstly introduced by McCulloch and Pitts 1943, the advancements in recent years are enabled mainly through the progress of computation power.

Kramer 1991 introduced autoencoders as means for unsupervised dimensionality reduction. By having a symmetric structure with fewer nodes in the hidden layers, autoencoders are trained to reconstruct the input by expecting the network's input as its output. This results in a meaningful representation of the data in the latent space. The main assumption when using autoencoders for outlier detection is that data points that don't share many similarities with the remaining data would be harder to reconstruct. Levering this assumption, Hawkins et al. 2002 introduced the reconstruction error as an outlier score. One major drawback to the unsupervised training approach is that training data could be contaminated with outliers and the autoencoder overfitting to this contamination, affecting the model's accuracy. Combining the reconstruction error of an autoencoder ensemble, where each model has different network connections (as in using drop out layers), the model can be made less susceptible to overfitting. Another approach to archive regularization over the latent space is by encoding each input as distribution in the latent space. The decoding process of the Auto Encoder samples from the encoded distribution and trains through the standard reconstruction error. This method is known as Variational Auto Encoders, introduced by Kingma and Welling 2014. This has a similar result as using autoencoder ensemble by reducing the influence of outliers over the model, with the advantage of not training multiple autoencoders.

Generative Adversarial Net's (GAN), originally introduced for generating new content, consist of two adversarial components. (Goodfellow et al. 2014) A network that generates content by learning a mapping from a random vector into the data space. The generated data is given to the second discriminator network that tries to differentiate between generated data and the actual data. The Single-Objective Generative Adversarial Active Learning introduced by Liu et al. 2019 views outlier detection as a binary classification problem. The generator tries to generate possible outliers, while the discriminator creates a decision boundary dividing the generated data and the real data. In the early training stages, the generator doesn't create enough potential outliers, resulting in a poor decision boundary by the discriminator. By incorporating the feedback of the discriminator, the generator improves its ability to generate potential outliers that occur in or around the real data, leading to an improved decision boundary by the discriminator, enhancing its accuracy.

The significant advantage of using neural networks and deep learning for outlier detection is the ability to automatically learn features from high dimensional and complex data without any prior assumption over the data. On the other hand, deep learning requires a large data size, which is hard to obtain in some settings. Furthermore, most deep learning approaches are sensitive to their selected hyperparameters requiring tuning, which can be a time-consuming task (Boukerche, Zheng, and Alfandi 2020).

2.1.3.1 Semi-Supervised

Due to the nature that labelled outlier data is a rare occurrence, most research in this field is done on unsupervised methods. Further, labelled data is not even desired when searching for novelties in data.

Active Learning introduced by Settles 2009 describes the process of incorporating expert feedback into the learning process. This is based on the assumption that enhancing the unsupervised model with limited labelled data can significantly improve its accuracy. It initially builds a model base with a given unsupervised method, then queries an expert selected example to acquire the labelled information.

The support vector data description by Tax and Duin 2004 builds a hypersphere around the normal data. The outlier score is calculated as distance to the hypersphere centre, while outliers are on the outside of the sphere. The training process incorporates limited labelled data by setting the requirement that the data labelled as normal to be inside the hypersphere and the as outlier marked data on the outside.

3 Concepts

This chapter addresses the concepts underlying the different approaches that form the outlier detection model over the backups. The underlying models of each approach take the parsed and scaled features of a backup and turn them into a prediction and the outlier score. Using these scores, the model can provide an outlier probability using a stretched exponential function and a confidence level in the prediction.

3.1 SUOD model ensemble - Metadata model

The metadata model is the first approach that collects features over the whole backup concerning its previous backups. The underlying model uses an ensemble of different outlier prediction models to archive more robust results and more reliable confidence levels in its predictions. Scalable Unsupervised Outlier Detection (SUOD) introduced by Zhao et al. 2021 is the used state of the art modular outlier ensemble method. The main challenge they address is the acceleration of unsupervised, heterogeneous outlier models by focusing on data reduction techniques, approximations of more costly models and task load imbalance in distributed settings.

Their implemented framework addresses these challenges in the following way: **(1)** Facing the problem proximity-based methods have with high dimensional data, they implemented the mentioned random projection method for dimensionality reduction. **(2)** Accelerating the prediction of new examples is archived by replacing the costly unsupervised models, such as k-NN and LOF, with less expensive and faster-supervised models like a random forest. The supervised model replacing the unsupervised ones takes the output or outlying score of the unsupervised model as a “pseudo ground truth” for training. “The goal is to approximate the output of the underlying unsupervised model” (Zhao et al. 2021). **(3)** The last speedup is archived by executing the outlier prediction in a distributed manner Balanced Parallel Scheduling (BPS) reduces task load imbalances between the used models. By forecasting the expected cost of each model, the workload can be efficiently split between the employed workers.

3.2 One-class Support Vector Machines

The second approach previously mentioned uses the path of each file in the backups to build a model. Support Vector Machines (Cortes and Vapnik 1995) are a popular supervised prediction methods for classification and regression. They are trained by searching a hyperplane that separates the labelled data. The hyperplane that archives the largest separation by having the biggest distance between the closest data points is called the maximum margin hyperplane. Data points that are not linearly separable can be implicitly mapped into a higher dimension in which they again become linear separability. Figure 3.1 shows an example of how mapping a non-linear one-dimensional separable data into the second dimension by using the function $f(x) = x^2$ leads to again linear separable data. In the case of non-linear separable data, allowing data points in the maximizing margin is called a soft-margin hyperplane.

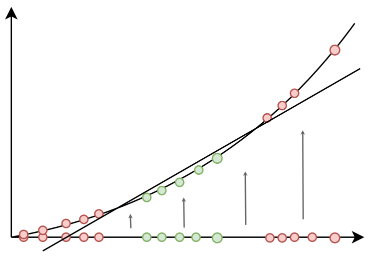


Figure 3.1: Mapping data with, $f(x) = x^2$

Schölkopf et al. 2001 describes one-class support vector machines (OCSVM) as a natural extension to the support vector algorithm for the case of unlabeled data. They describe their strategy of finding the regions with the most data points as finding the hyperplane in the kernels feature space that separates its data from the origin. By introducing a parameter, v in the primal function, that allows for controlling the maximum number of margin errors and the minimum number of support vectors, a separation of the probability density and non-probable regions can be achieved.

As previously mentioned, the “path” approach creates multisets of path components. To use these multisets in the OCSVM, a distance measure between these multisets needs to be defined. A kernel that can handle sets and multisets can be defined as (Raedt 2022):

$$K_{Set}(X, Y) = \sum_{x_i \in X} \sum_{y_j \in Y} K_{el}(x_i, y_j) \tag{3.1}$$

K_{el} is the kernel defined on the element of the sets. In the case of the path approach, the elements of the sets are preprocessed only to be a component of the path. Therefore, with the element-kernel δ, that compares the two components:

$$\delta(x,y) = \begin{cases} 1, if x = y \\ 0, \texttt{otherwise} \end{cases} \tag{3.2}$$

The set kernel K_{Set} becomes the number of elements that are equal in each set:

$$K_{Set-\delta}(X,Y) = |X \cap Y| \tag{3.3}$$

The path component used in the path approach is the folder a specific file is saved into. This is motivated by the intuition that creating files in a new folder is a rarer occurrence than creating files in already existing folders. Established applications store their files in specific work directories. If two files are stored in the same folder, the folder occurs twice in the multiset, resulting in higher similarity with backups that kept files in the same folders. If the selected component is the specific file, only matching files would be counted, leading to less similarity between backups.

Compared to the metadata approach, this approach takes individual files more into account. For example, an update of the operating system stored on the file server results in a backup that is classified wrongly as an outlier in the beginning. Future updates of the OS match with previous ones, forming a cluster on their own, resulting in a smaller outlier score.

3.3 Interpretability

When regarding outlier detection only as a classification problem that gives each example a binary label, a lot of interpretability and explainability is lost. Looking at the anomalies score used to classify the examples can give some insight into a model's prediction but makes it hard to compare with other models. Adding an outlier probability relative to the models' outlier scores and a confidence value improves the interpretability. This section firstly looks at how a stretched sigmoid is used to receive the probability scores, followed by the method to calculate the confidence level of each prediction.

3.3.0.1 Score to Probability

Figure 3.2 shows how the OCSVM classifies data drawn from a normal distribution. Some outliers are introduced through uniformly distributed data points in the range of -5 and 5.

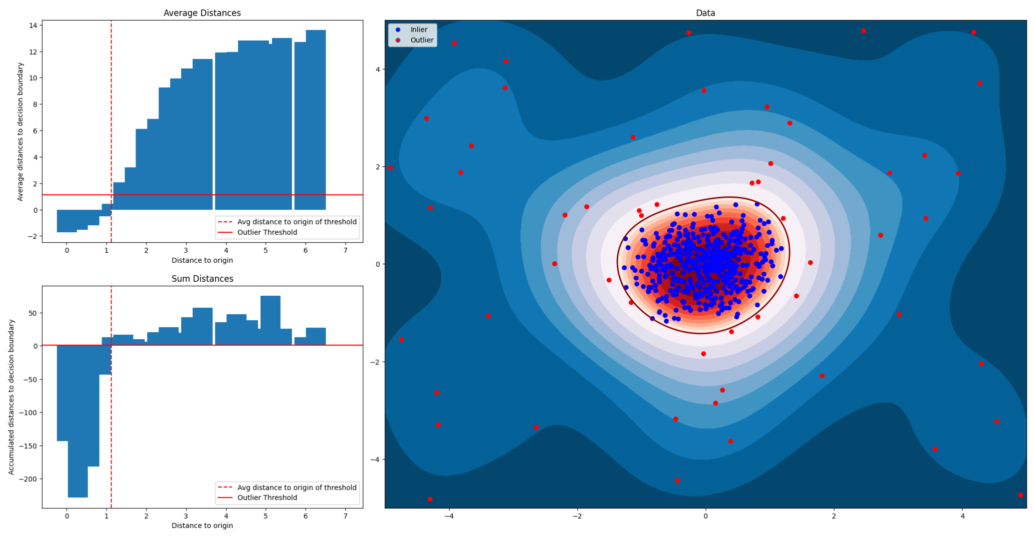


Figure 3.2: The process for each model.

The location of the normally distributed inliers is centred around the origin. Using this knowledge, we can correlate the norm, or distance to the origin, to the outlier score. The graphs on the left show on the x-axis the distance to the origin, the y-axis in the upper diagram the average outlier score and the bottom graph the accumulation of the outlier score. To calculate these y-axis values, the distance to the origin is discretized into 25 buckets, collecting all data points and their outlier scores in each bucket.

Different models use different outlier score scales that lead to problems in the interpretability of the scores. Assigning an outlier probability to each example that scales with the distance to the decision boundary is a non-trivial problem. The solution to this problem needs to address the following problem: (1) The example with the smallest outlier score should receive an outlier probability of 0. Figure 3.2 shows that examples closer to the origin receive a smaller outlier score. The smaller an outlier score gets, the less probable it's to be an outlier. (2) Examples classified as belonging to the distribution should receive a small probability below γ with $0 < \gamma < 1$, and (3) the example with the largest outlier score should obtain a probability close to 1.

A stretched exponential function can be used to model this behaviour. The cumulative distribution function (CDF) visible in figure 3.4 of the Weibull's probability density in figure 3.3 function can be adopted for this task.

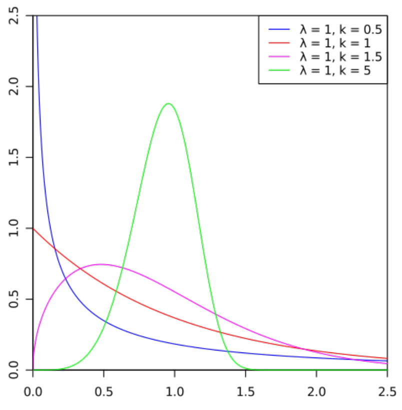


Figure 3.3: Weibull's probability density function (Source: Wikipedia Weibull distribution)

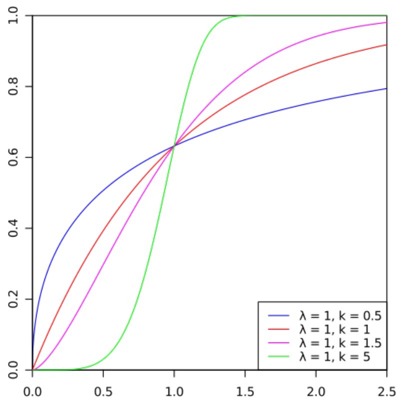


Figure 3.4: Weibull's Cumulative distribution function (Source: Wikipedia Weibull distribution)

The equation 3.4 describes the cumulative distribution function we want to adapt.

$$F(x, k, \beta) = 1 - e^{(-\frac{x}{\beta})^k} \tag{3.4}$$

To archive the second and third requirements mentioned above, we can solve the equation for its parameters with the following intuition: Let's define two points on the CDF as exemplarily shown in figure 3.5. The x_1 coordinate of the first point

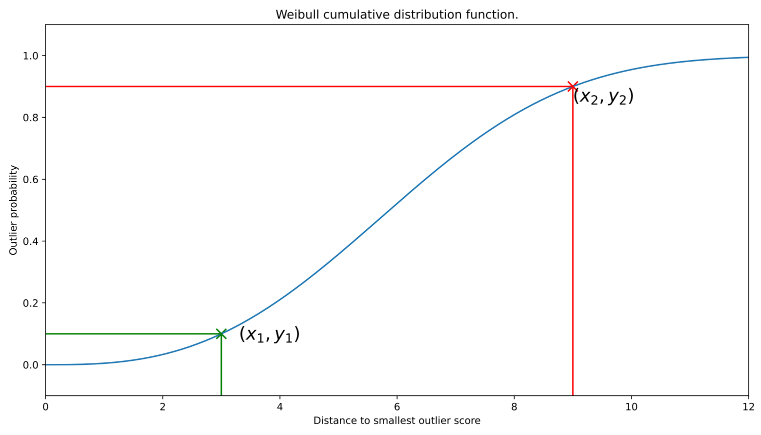


Figure 3.5: Two points on the CDF of the Weibull distribution for which needs to be solved.

p_1 represents the decision boundary threshold by which the model classifies. Setting its y_1 coordinate to the by (2) required parameter γ gives every value left of that point an outlier probability smaller than *gamma*. The second point p_2 represents an outlier with the highest outlier score. Its x_2 coordinate is the biggest predicted outlier score and y_2 is a value close to 1.

Solving the CDF for its parameters given the coordinates of the points p_1 and p_2 leads to the following parameter:

$$k = \frac{\ln(-\ln(1 - y_2)) - \ln(-\ln(1 - y_1))}{\ln(x_2) - \ln(x_1)} \tag{3.5}$$

$$\beta = x_1 \cdot (-\ln(1 - y_1)^{-1/k}) \tag{3.6}$$

Using these parameters in equation 3.4 with $y_1 = \gamma$ and y_2 a value close to 1 results in a probability function that gives meaningful results. To apply the CDF to the decision scores coming from the outlier model, we first need to find the smallest outlier score of the dataset, followed by calculating the absolute distance of each example to the smallest score. Applied to the outlier scores of the normal distribution in figure 3.2 results in the probability scores visible in figure 3.6.

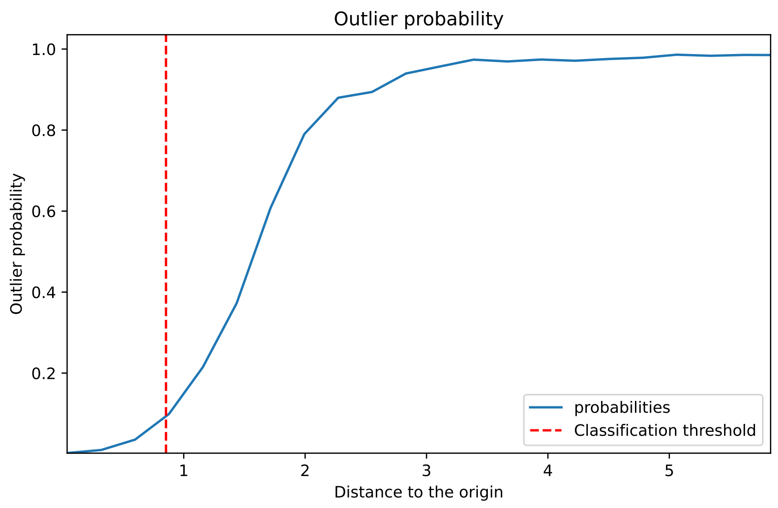


Figure 3.6: Applying the adapted CDF to the outlier scores.

The figure shows the aforementioned desired properties: The example with the smallest outlier score receives a probability of 0, as its distance to itself is also 0. Values below the classification threshold are smaller than the selected γ value of 0.1. The outlier probability rises quickly after the threshold, assigning the outlier with the highest score a probability close to 1.

One problem of this approach could be that one example with an exceptionally high outlier score could skew the probabilities of all other positive labels.

3.3.0.2 Confidence

Only providing probability scores has no expressiveness over the reliability of the prediction. Perini, Vercruyssen, and Davis 2021 introduced a method of calculating the confidence for each model prediction, assigning real-valued outlier scores.

As explained in chapter 2 for an outlier detection method to make a prediction, it usually calculates a threshold value based on a prior assumed contamination percentage and the outlier scores of the trained data. By perturbing the training data, the outlier scores change, affecting the picked threshold and changing the predicted class of an example. Intuitively, the method for calculating the confidence levels can be expressed as the probability that the prediction of an example would change if a different dataset were observed.

The definition of confidence is formally defined by Perini, Vercruyssen, and Davis 2021 as “the probability that an example x with score $s = f(x)$ and outlier probability p_s will be classified as an anomaly when randomly drawing a training set of n examples from the population of scores.” The outlier probabilities they use are calculated by a Bayesian approach to estimate the distribution of the outlier scores, allowing them to derive an examples’ outlier probability.

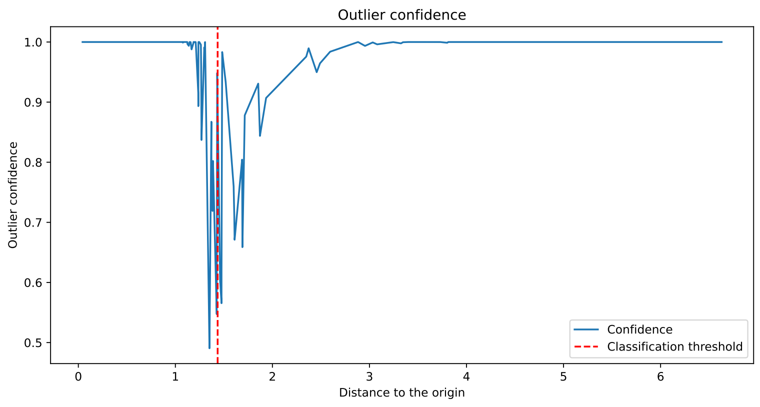


Figure 3.7: Applying Perini, Vercruyssen, and Davis 2021 calculation to the outlier scores of the data set from 3.2

Applied on the artificial dataset of figure 3.2, the figure 3.7 shows the confidence values fall around the decision threshold as expected. The examples with scores around the decision threshold would be suspected to change the most when randomly selecting different examples to calculate the decision boundary.

4 Approach

Each file server secured by the Commvault software receives its file server model trained on detecting abnormal behaviour, captured in a backup, as outliers. Generally, the model needs to face the two challenges of extracting relevant features from the backup data that represent the usual file server behaviour, alongside the classification of these features into outlier labels. The previous chapters covered how an outlier model can classify a data point, assigning outlier scores, outlier probabilities, and certainty values. The approach chapter looks at how the backup data is parsed into numerical vectors passed into the underlying outlier prediction models.

The feature extraction process is a significant component in applying outlier prediction concepts to the data. It tries to capture meaningful information. Usually, research on outlier detection methods does not address extracting features from the raw data and only describes the training process with numerical vectors. As the model is supposed to classify whole backups as outliers, one data point represents the data of one backup. The data of one backup consists of the list of files that were changed since the last backup.

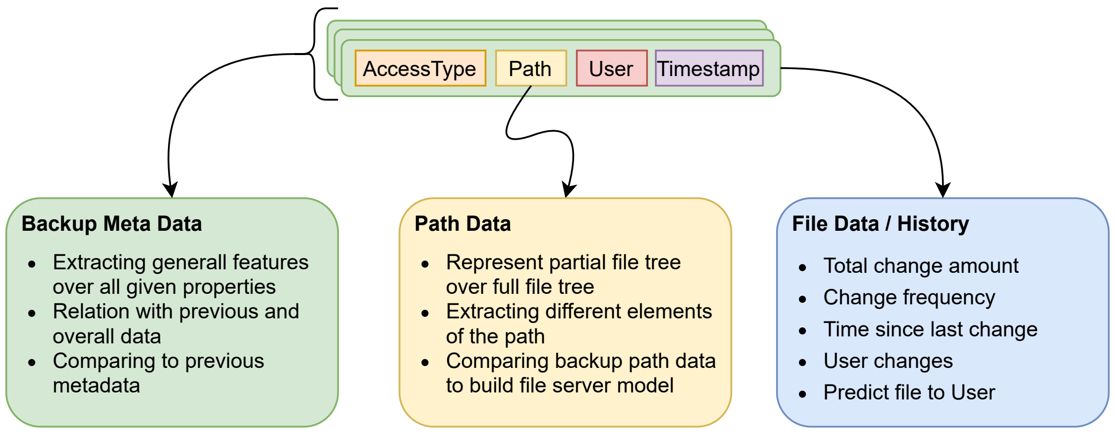

AccessType
Path
User
Timestamp
Backup Meta Data
Extracting generall features over all given properties
Relation with previous and overall data
Comparing to previous metadata
Path Data
Represent partial file tree over full file tree
Extracting different elements of the path
Comparing backup path data to build file server model
File Data / History
Total change amount
Change frequency
Time since last change
User changes
Predict file to User

Figure 4.1: The three main approaches to creating models from the given backup data.

Shown in figure 4.1, the models developed in this internship assumes that each file of the given backup has the following properties:

1. The last access type consisting of either: Added, Deleted, Modified, Renamed
2. Full path of each file that was stored in this backup
3. The user that accessed the file the last time
4. The time the file was last accessed

Also shown in figure 4.1 are three ways how features can be extracted. From which the first two have been implemented.

The first approach extracts its feature over the whole backup regarding previous backups. The second approach only uses the path component of the backup. By splitting the path into its parts, different path elements can be extracted, e.g. the file itself or the folder the file is saved into. By combining the selected component of the paths of one backup into a multiset and comparing each multiset, outlying backups can be determined.

The last, currently not implemented approach is considering each file as an example for the outlier detection. This approach is inspired by exiting log data outlier detection. For each log event, statistical features are extracted, such as the sequence of the log event, its frequency of occurrence, the seasonality of the event or how often it occurs compared to other events. (Wang et al. 2020) Similar statistical features can be extracted from the backup data for each file. For example, the sequence of files can be constructed through the given access dates, and a file frequency could be calculated over the previous backups. Given the features of each file in a backup and an outlier detection model trained on all the previous files, the outlier score of the backup could be calculated by averaging over the outlier scores of each file. One drawback of this approach could be the missing relation of outlier scores to the backup itself. The model learns the files that normally belong to the specific file server, but that doesn't necessarily correlate to files typically together in a backup. A further problem could be the computation time for calculating the individual features of each file.

4.1 File server model

The file server model is the component that implements each approach modularly. Figure 4.2 describe the three main aspects that each approach needs to support.

The model initialization describes the fitting process of the model. At least two initial backups are required to train the underlying outlier model. But any prediction with

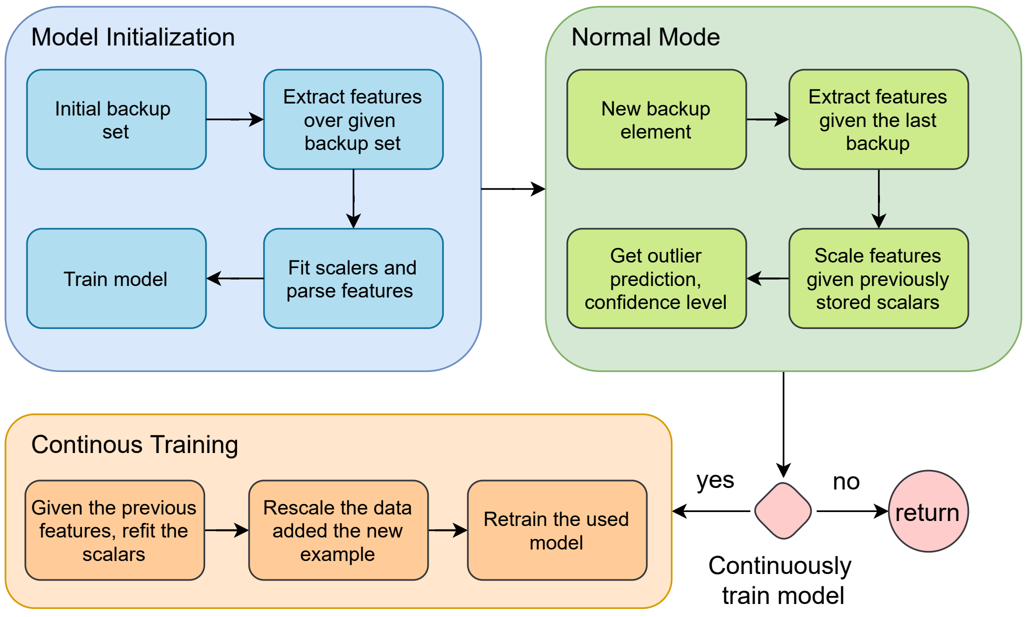

Model Initialization
Initial backup set
Extract features over given backup set
Fit scalers and parse features
Train model
Normal Mode
New backup element
Extract features given the last backup
Scale features given previously stored scalars
Get outlier prediction, confidence level
Continous Training
Given the previous features, refit the scalars
Rescale the data added the new example
Retrain the used model
yes
no
return
Continuously train model

Figure 4.2: The process for each model.

a small example size is unreliable. This should also result in a smaller confidence level for the prediction. The next step is to extract the features respectively for each used approach, given the backup data list, sorted by its date. Outlier prediction models function better with data values in a specific range therefore, each feature of the respective approach needs to be scaled accordingly. The scalers need to be saved to relate future extracted features to the features the model was trained on. Training the respective underlying model to represent the given backup list is the last step for the initialization process.

The normal mode of each approach describes the process of classifying each backup as an outlier or belonging to the expected data. The file server model gives each enabled approach the to be evaluated backup and its required data. Similar to the initialization process, the features are extracted and scaled given the previously-stored scalers. Finally, using the fitted model, the backup receives its classification and a confidence level of how probable the given classification is.

Each model also needs to support the continuous training modus. The model adds the previously predicted backup into its underlying model and data structures through which the features are extracted. A small efficiency increase is archived by taking the features that need to be added from the previous iteration. Adding the backup features to the model requires the same steps as the initialization process: Refitting the scalers to include the new backup, followed by scaling the data and training the used model.

The description of the different approaches corresponds to the method of parsing the backup elements into feature vectors, described in section 4.1 for each implemented approach. How the continuous training is implemented in each individual implemented approach, including the data structure of the elements that need to be stored. And finally, how the tool for detecting outliers in backup data could be deployed to support a changing number of file server backups.

4.2 Feature extraction

The metadata approach model implements a modular list of feature extractors that parse the data in the respective backup to compute the features. Figure 4.3 shows how the backup components are classified into their respective feature extractor.

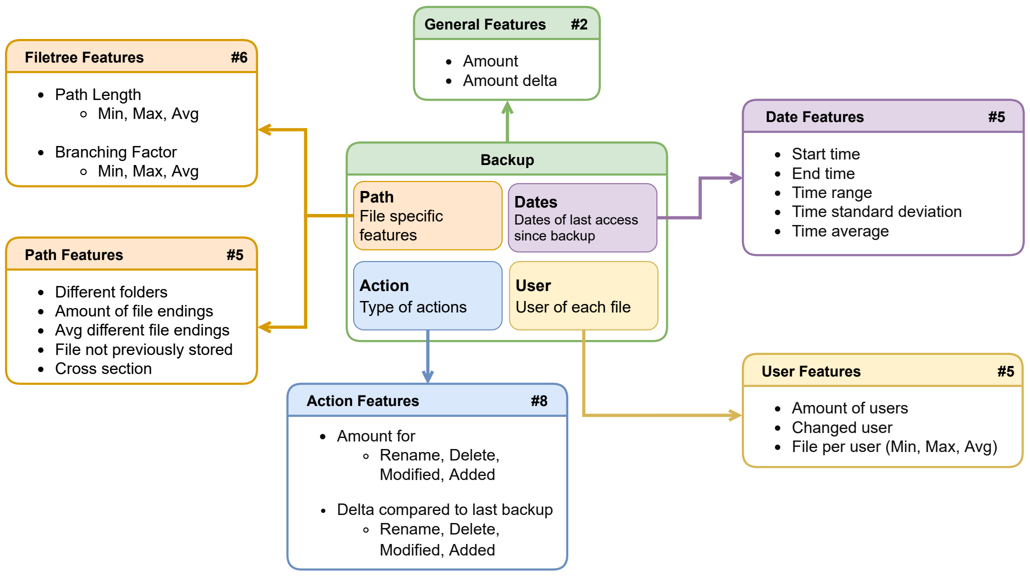


Figure 4.3: Overview of the selected metadata features.

Each feature extractor needs to implement an interface with the following functions: (1) *Get-Feature-List* that returns a mapping that maps feature names to a callable that returns the feature. (2) *Calc-Features* that parse the data and calculate the features. The feature extractors are classified into the following categories: (1) **General features** are features that do not use a specific backup component. (2) **Date features** use the time component of a file, at which it was last accessed. As time goes forward each day, we have to put the time component in relation to the backup date. To calculate the features, we parse the access time into Unix time stamps and subtract the timestamp of the current date, leaving us with the timestamps representing the time of day. Therefore, these features encapsulate different information over the timings of the day. (3) The access type is used in the **action**

features category, capturing numerical the distribution over these actions done on each stored file. Possible actions are renamed, added, modified or deleted. (4) The **user features** capture who has accessed each file the last. This includes the number of different users, if the user of a file changed, or how many files each user accessed. The last two categories use the path component of the backup and are extracted by the same feature extractor. (5) To calculate the **file tree features**, a tree based on the paths is created. Each folder of a path creates a node in this file tree, with the files forming the leaves. With the tree created, data over the branching factor and tree depth can be extracted into features. The last features categorized as **path features** compute different properties of each file in the backup. This includes what kind of files are stored, in how many distinct folders they are stored and also the cross-section to the previous backups and stored files. An extensive feature list with motivations can be found in the appendix A.1.

Features that require a file lookup, such as the cross-section feature, the previously stored feature, and the user changed feature, need a fast way to find the individual file. Keeping each file identified by its path in a simple list would have a $O(n)$ search complexity, with n being the length of the list. With many files in a backup, this is a bottleneck of the feature extraction. Keeping the list in alphabetical order could theoretically reduce the search complexity to $O(log(n))$ with a binary search, but comparing the path strings is still too slow. To speed up the file lookup, a file tree is built. This also reduces the theoretical lookup time complexity to $O(log(n))$, but if the path is not in the tree, we can often cancel earlier as the folder of the path is usually also not present. Also, the string compares are only between folders and not entire paths. Another speed-up could be archived by indexing each folder, leaving only having numerical comparisons.

The metadata model that encapsulates the feature extractor calls the *Calc-Feature* method of each extractor, followed by their getter methods provided by the *Get-Feature-List* function. As previously mentioned, the outlier detection algorithm is more reliable if the value ranges of each feature are roughly the same. Therefore, the next step is to create an array of standard scalers that scale the values of each feature independently to average around 0 and have a standard deviation of 1. The scaled feature vectors are passed to the underlying model described in the previous chapter.

On the other hand, the path approach that uses the one-class support vector machine as an underlying model only requires one feature to build the model. As explained in section 3.2, the OCSVM works with the pairwise distance measure between the sets of path elements. We can precompute all distances between each set, with the kernel defined as the set kernel 3.3, into the gram matrix. For the training case, the gram matrix is a positive, symmetric matrix with a size of $n \times n$ where n is the amount of examples in the train data. To make a prediction, we compute the set kernel of the example with every example in the training set, creating a $n \times m$ shaped matrix with m being the amount of examples that require a prediction. By using a vocabulary that indexes each element to a numerical value, we can speed up the precomputation of the gram matrix.

To use the method of turning the outlier scores into probabilities, we collect the distances to the decision boundary of each example after the fitting process in both approaches. The required min, max, and decision threshold values are saved to be used to calculate the outlier probability process.

The implementation of the underlying outlier models is provided by the PyOD library. (Zhao, Nasrullah, and Li 2019) It includes more than 30 detection algorithms. Following the SciPy model design, each approach encapsulating class and the file server model implements the fit method and predict method. A difference to the SciPy models is the need to call an initialization method. Providing this method with no parameter creates a new model ready to be fitted to any data. If a parameter is provided, the file server model will create the underlying models and data structures from the provided data. This is necessary for continuing the model's training process with a new prediction.

4.3 Continues Training

The continues training addresses the requirement for incorporating new backups into the model. Also addressed is the case that if the environment running the file server model is stopped for whatever reason, the configuration and state of the model is stored in a database. This comes with the bonus that the file server model doesn't need to be kept in the memory between each backup.

The naive approach for incorporating the new backup is to recall the fitting process, including the new instance. This would recalculate the features unnecessarily. As every backup that is supposed to be added into the system received a prediction from both models, we can use the calculated features from that step. In the case of the metadata model, the features are added to the whole feature list, followed by the recalibration of the data scalers. For the OCSVM, we need to extend the precomputed gram matrix. We can append the precomputed gram matrix used for predicting the example to the gram matrix used for training. The only extra work required is to compute the $m \times m$ gram matrix of the examples used in the prediction, with m being the amount. Having the gram matrix G_1 used for training $(n \times n)$, the gram matrix used for prediction G_2 $(n \times m)$ and the new computed gram matrix G_3 $(m \times m)$ we can create the new training gram matrix with G_{new} with $(n+m) \times (n+m)$ as shown in equation 4.1.

$$G_{new} = \begin{bmatrix} G_1 & G_2.T \\ G_2 & G_3 \end{bmatrix} \tag{4.1}$$

With the features integrated into the used data structures, the only thing left is to fit the underlying model to the new data and calculate the values for the outlier probability function.

Persistent storage is required to ensure that the model is not lost if the server needs to restart. A NoSQL database program was integrated into the outlier prediction process that stores the required data structures in JSON-like documents. The database is also used to report the results of each prediction.

Figure 4.4 shows the database structure created in the database.

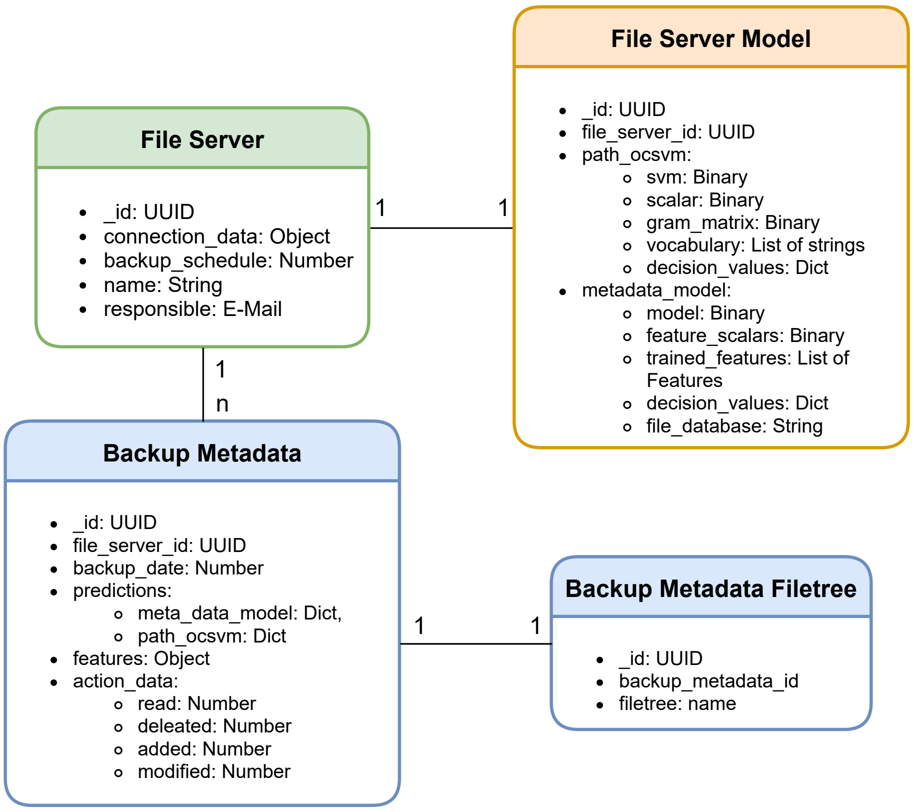


Figure 4.4: Database Structure

Each file server receives its outlier prediction model and is individually configurable. It stores the connection configuration used to pull the backup data from the backup software. Also stored is a responsible person or expert of that file server who is notified if an outlier is detected.

Each file server has a 1 : 1 connection to its stored file server model. The file server model calls for each implemented approach a *get storable components* method that serializes the components used for the outlier prediction and continues learning. The underlying model and the feature scalers are packed into binary strings in both approaches. The Path OCSVM adds the gram matrix also this way. The vocabulary used to index each path element is also required to map reoccurring elements to the same index. The metadata model stores the features the model was trained on and also the file tree of the file server that is required for some features. Also stored for both models are the values required for the outlier probability calculation.

The backup metadata is connected to the file server in a 1 : n relation. This stores the returned results for each implemented approach, the features extracted by the metadata model if enabled, and the different action types. Finally, the file tree stored in that backup is connected to each backup metadata. This is stored in an extra collection because the size of the serialized string would cause a relatively large overhead when loading the backups for visualization. This file tree could also aid the expert who needs to decide if a detected outlier is classified correctly.

4.4 Integration and Deployment

Due to difficulties with the backup software providing the backup data, no integration or deployment was possible. Therefore, this section discusses the design for possible integration into the server environment of the universities.

Figure 4.5 gives an overview of each component and how they work together. Each developed component runs in a separated docker container on the server environment. Docker uses virtualization to deliver each component in packages, so-called containers. This allows, theoretically, the horizontal scaling of the outlier detection model if the amount of file servers exceeds the handling capability of one classifier.

The first component is the Vue frontend. It is the component that interfaces with the user, visualizing the results of the outlier classification. The used technology is the modern JavaScript Framework Vue 3, allowing for dynamic and intractable websites. The Vuex storing technology further allows for web caching the communicated data, reducing the amount of traffic between the frontend and backend. Web sockets could

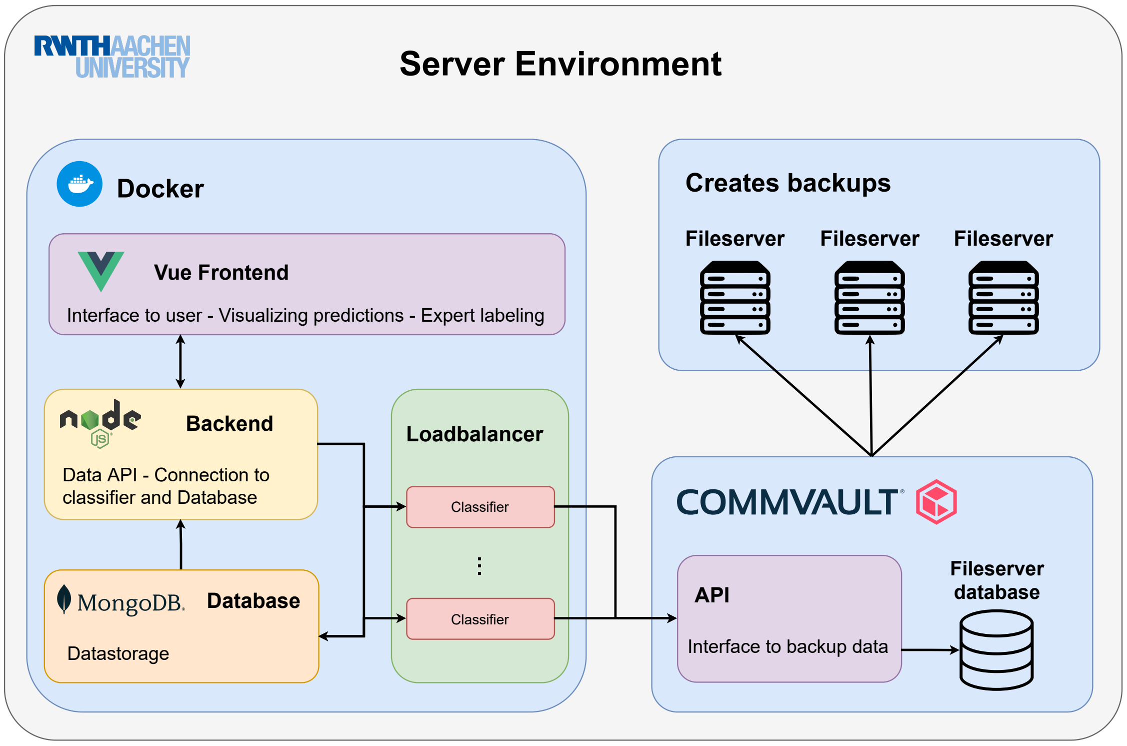


Figure 4.5: The general concept of how the developed components can be integrated into the existing server environment of the university.

be used to send real-time instructions, such as retraining the model with a selected set of backups.

The backend provides a rest interface to the database. Developed in the node runtime environment using the express *node.js* web application framework. By calling the root level URL, it returns the available collections from which to receive data. By appending the collection's name to the URL, it returns every entry in the respective collection. Further, adding the file server ID to the URL returns only elements with the specific ID. It provides further filter functionalities to the database if not the whole data is required by adding the required filter object as the request body. By choosing the express framework, we can extend the backend with state-of-the-art authentication functionalities if that requirement occurs.

The database, as previously mentioned, is the central storage unit for all the data collected in the classification process. It sends data to the backend and is the only component that is persistently storing data.

Using dockers load balancing abilities, we can start a dynamic amount of outlier classifiers, each receiving the file server ID they are supposed to classify for. Each model can independently load the model with the required data from the database for the specific file server. The backup software's API provides the interface for each classifier to pull the new backup data.

5 Evaluation

Without any integration, to test the outlier prediction on backup data, a mock test dataset is required. The test dataset was created by parsing commits from a relative large git repository into the required data format. Commits are similar to incremental backups as they add the local changes to a shared project. The used project is the Ludii project (Piette et al. 2020) with 28719 commits. The backup like data structures are created by grouping the commits by their date. A backup, therefore, contains all commits of a specific date, if there were any. This approach produces 415 backups with varying amounts of logged files.

Different experiments can be done using these backups for the outlier prediction of both models. The used models are evaluated using the predictions, probabilities, and outlier scores. Different training states are considered, comparing how the returned values of earlier backups change given a new training state. The assumed outlier contamination is 10% over the whole testing.

5.1 Results

As previously mentioned, the intended everyday use mode for the outlier prediction is the continued learning mode, where it continuously integrates the new backup points into the model. It therefore only predicts outliers based on the data it has previously seen. This mode is further called "Continues" mode. In this mode, the metadata model predicted 109 outliers, while the OCSVM predicted 54. The second mode, called full train mode, is if the model has been trained on the full dataset and then does the outlier prediction on the training set. This mode predicted 41 outliers by the OCSVM and 42 outliers by the metadata model. The difference in outliers between the two modes can be explained.

Figure 5.1 shows the predictions, confidence and outlier probability of the metadata model of the last 100 backups in the continuous mode. The amount of files per backup is also selected as it correlates with the majority of the other features, further discussed in section 5.3. Notable is that not every predicted outlier has a high outlier probability, expressing uncertainty in the model's prediction. This is also represented in the confidence in the predictions with less outlier probability.

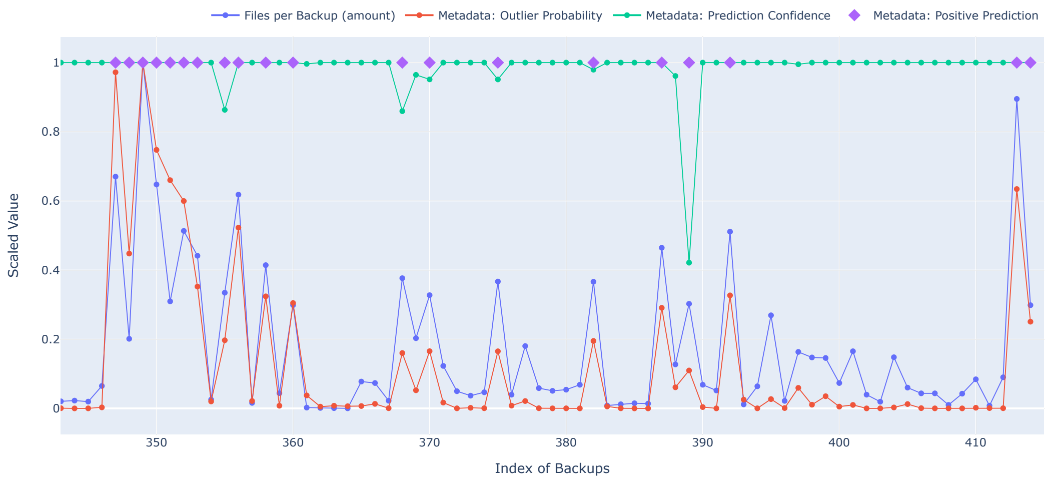


Figure 5.1: Visualization of the outlier probability, prediction confidence and positive prediction of the metadata model in the continuous mode with the amount of files in a backup.

A backup that was predicted as an outlier, with a smaller outlier probability, has compared to the outliers with a higher probability a smaller confidence in the prediction. Less confidence in the prediction usually occurs if the amount of files in the backup is also less than outliers with more files. Further, it is visible that the outlier probability is strongly correlated with the amount of files in a backup.

Comparing the predictions of the metadata model to the OCSVM. The table 5.1 shows that both models learned to classify different features.

	Same positive	Same negative	Different
Continues Mode	5	257	153
Full Trained Mode	1	332	82

Only five backups were classified equally for the continuous mode, while in the full trained mode, just one backup was classified as an outlier in both models. A bit of insight into the classification of the OCSVM could be archived by taking a look at the gram matrix. For example, by averaging the rows of the gram matrix by the respective classification: The average of all rows of positive predictions is 94.98 while the average over the as normal classified backups is 6566.21. As the used kernel counts the amount of equal folders between backups, these values represent the avg similarity between the backups. This indicates that the OCSVM learned to classify backups as outliers that share only a few folders with other backups.

Plotting the average similarity, given through the gram matrix with the amount of files per backup, shows that it is closely related to the amount of files in the backup, visible in figure 5.2.

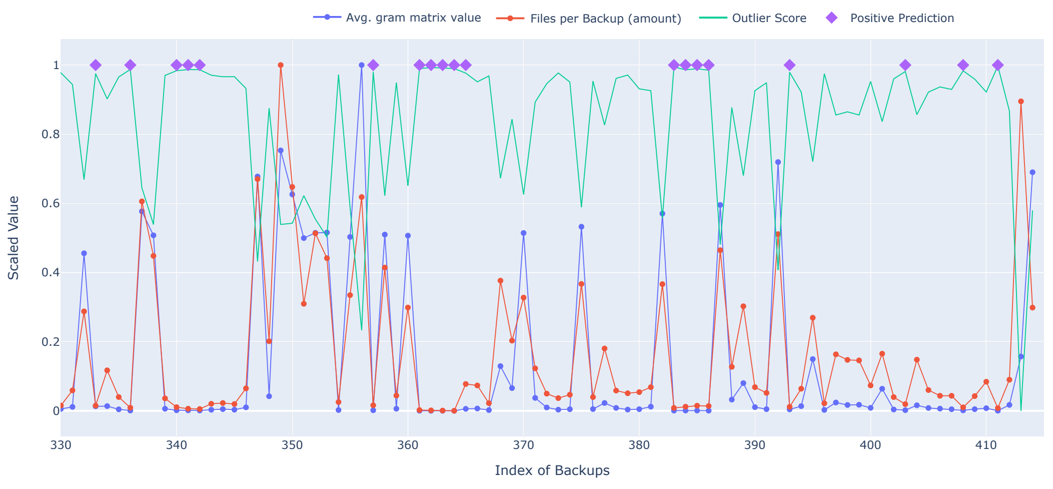


Figure 5.2: The Prediction and outlier score of the OCSVM in the continuous mode and average row value of the gram matrix with the amount of files in a backup.

It is also visible that the backup, labelled as an outlier, has fewer files in their backup compared to the other backups. Due to the correlation between the gram matrix and the file amount, this confirms the previous observation of the low average of the row of the gram matrix.

The close correlation between the kernel similarity and the amount of files in a backup could be due to the missing normalization of the kernel. Adding a normalization term to the kernel that divides each similarity measure by the squared product of the set's cardinality ($\frac{1}{\sqrt{|Set_A| \cdot |Set_B|}}$) could reduce the influence of the file amount.

Visualized in figure 5.3 is the outlier scores of both kernel variations in the full trained mode. Both models predicted 31 examples equally of the 42 predicted outliers. The outlier scores are closely correlated, and only predictions closer to the decision boundaries are classified differently. This is supported by averaging the confidence of the predictions' confidence that are different and the confidence of predictions that are the same in both models. The average confidence of different

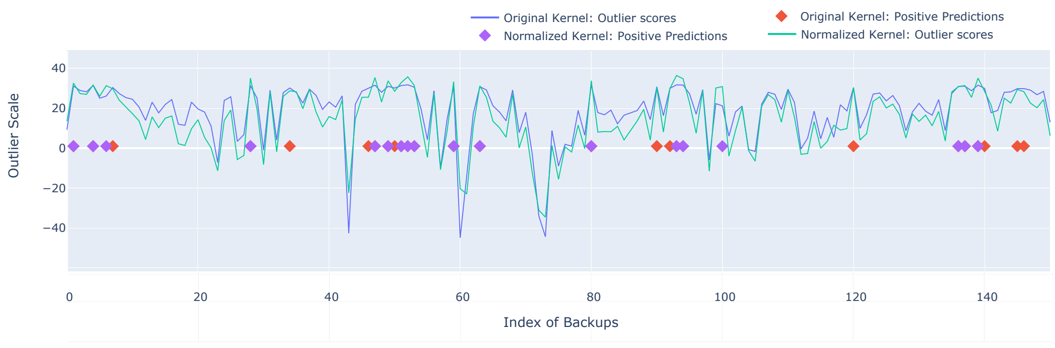


Figure 5.3: Normalized Kernel vs Absolute Kernel

predictions is around 84%, and for the predictions that both predicted as an outlier, it is 98%.

5.2 Continues mode vs Full trained mode

By comparing the performance of both models in the continuous mode, where they are only trained on the previous backups, to the full trained mode, we can evaluate the effectiveness of the continuous mode, as this is the mode supposed to be used in production.

In table 5.2, the predictions of the two modes are compared:

	Same positive	Same negative	Different
OCSVM	36	354	23
Metadata model	42	304	67

It is visible that compared to the total amount of predictions for both models, the amount of equal positive predictions is high. The metadata model predicted 42 outliers totally in the full trained mode, which were also predicted in the continuous mode. The relatively high number of different predictions is due to the high number of total predictions in the continuous mode. Of the 41 predicted outliers that the OCSVM did in the full trained mode, still 36 are matched in the continuous mode. This indicates that the continuous training mode finds all outliers with a high probability that are also found given the entire dataset.

Comparing the confidence levels of both modes and models, visible in figure 5.4, shows the initial uncertainty of both models in the continuous mode. The graph of figure 5.4 is created by using the Savitzky–Golay filter (Savitzky and Golay 1964)

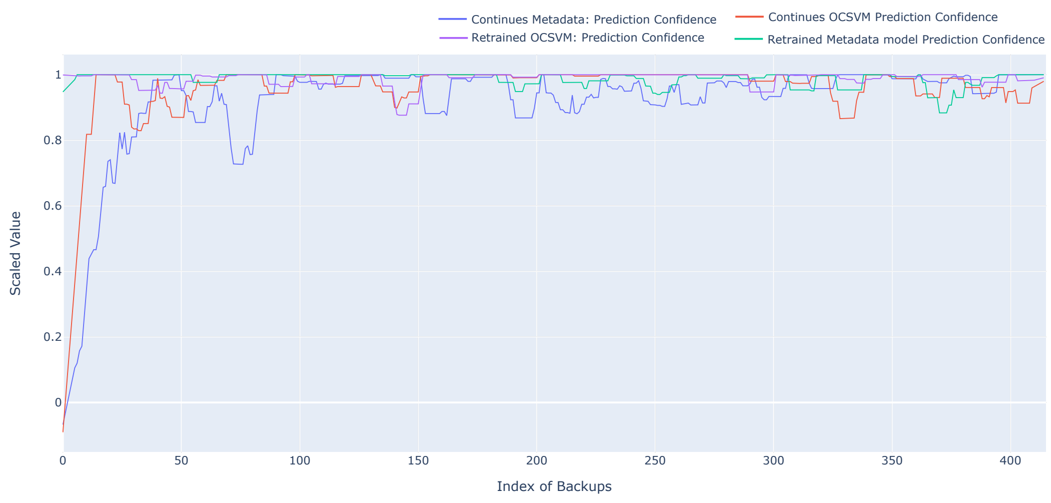


Figure 5.4: The overall confidence level of both models in both modes.

with a linear polynomial and a relatively small window size. Using a filter on the usually jagged graph shows the data trend. It can be seen that, on average, both models have lower confidence in the continuous mode. This could be explained by the smaller sample size used at the beginning of the continuous mode.

Another view on evaluating the continuous mode is by tracking the changes of model outputs of one specific backup. A good example is the initial backup, being the first in the sequence. Relative to its subsequent backups, it has more files per backup but compared to the later backups, it has fewer.

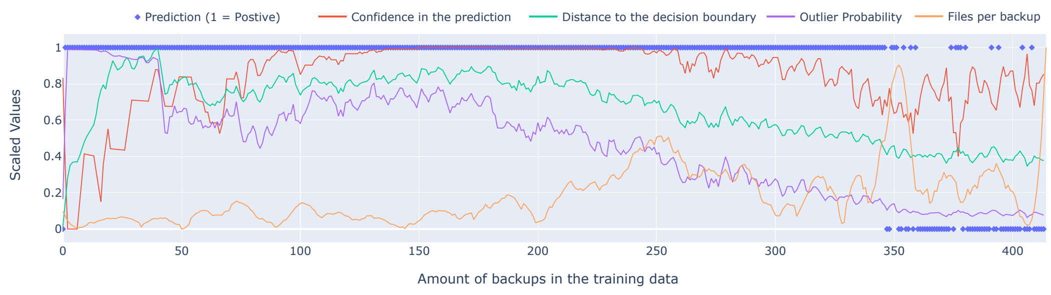


Figure 5.5: The output of the metadata model of the first backup over the whole continues mode.

Figure 5.5 shows how the output of the metadata model changes over time when more training data is available. In the beginning, it is classified as an outlier because compared to its following backups, it has more files in the backup. At around backup

200, the average size of a backup rises, and the outlier score and outlier probability starts to fall. At around 350, the amount of backups that are further away from the normal data than the first backup is higher than the contamination rate, and the prediction for the first backup starts to change. This is also represented by the outlier probability being around 10%, the designed decision boundary, as the classification changes.

5.3 Feature Influences

In the previous results, only the amount of files per backup was related to the results of the outlier prediction models. Different observations can be done by setting the features into relation by calculating their correlation.

Figure 5.6 shows the Pearson correlation matrix that calculates for each feature pair the Pearson correlation coefficient (Freedman, Pisani, and Purves 2007). It measures the linear correlation between two sets of data.

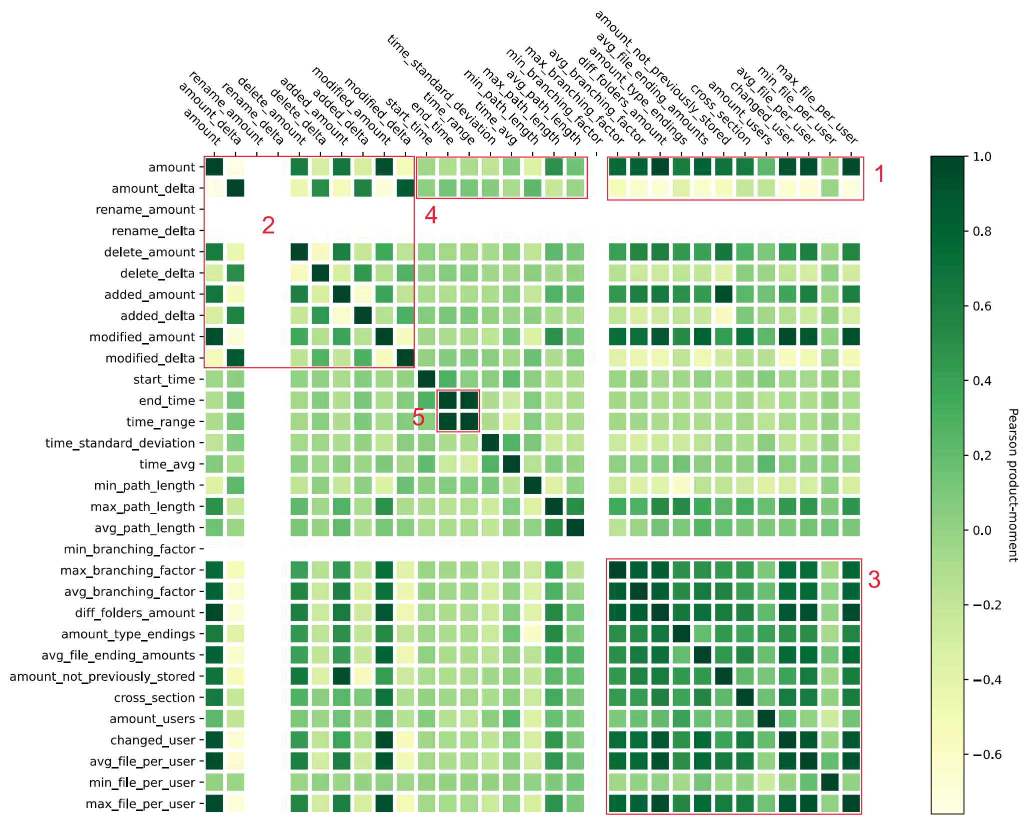


Figure 5.6: Pearson correlation matrix, visualizing how each feature correlates with one another.

The first observation, marked with a red 1 in the figure, is that the path features correlate positively with the amount of files in the data. In other words, if there are more files in the backup data, then there are, for example, more different folders in the backup. This could explain the close correlation between the metadata models output with the amount of files in a backup. The checker pattern between the action features and the general features, marked as second observation, can also be easily explained through the amount of files per backup. The delta features that compare to the previous backup are negatively correlated. The third observation marks the generally high correlation of path features. Since they are also mostly counting features, they are equally correlated with each other as they are correlated to the file's pre backup amount. The features representing the files' timings and the path length feature (red 4) don't show any particular high correlation, except the end time and time range. The fact that the end time and time range are positively correlated is in the nature of the range feature that grows if the end time is further away. The three features that do not correlate in this matrix are once the rename amount, as there were no files that got renamed, and the min branching factor that is almost always 1, due to the fact that just one file, that is in a folder chain, results to a path in the tree with a node that has the branching factor of 1.

Doing the same Pearson correlation coefficient between the features and the outlier score of both models in both modes gives insight into the importance of each feature in the prediction. Figure 5.7 shows this Pearson matrix.

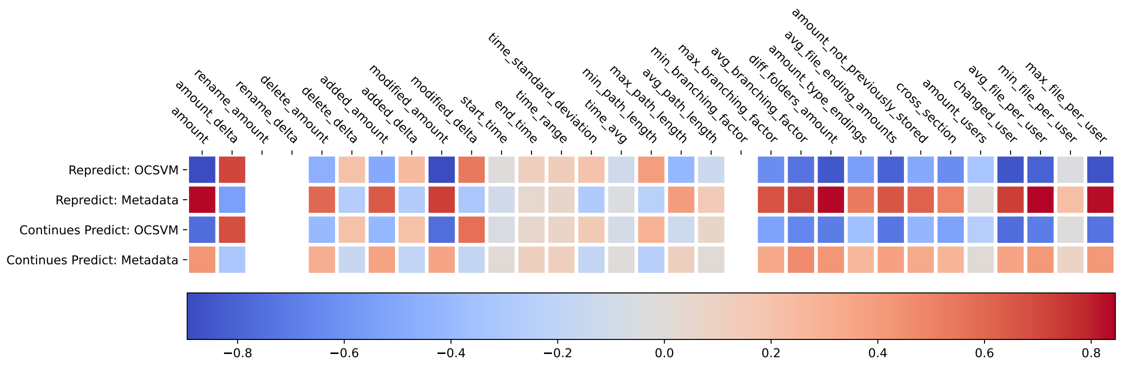


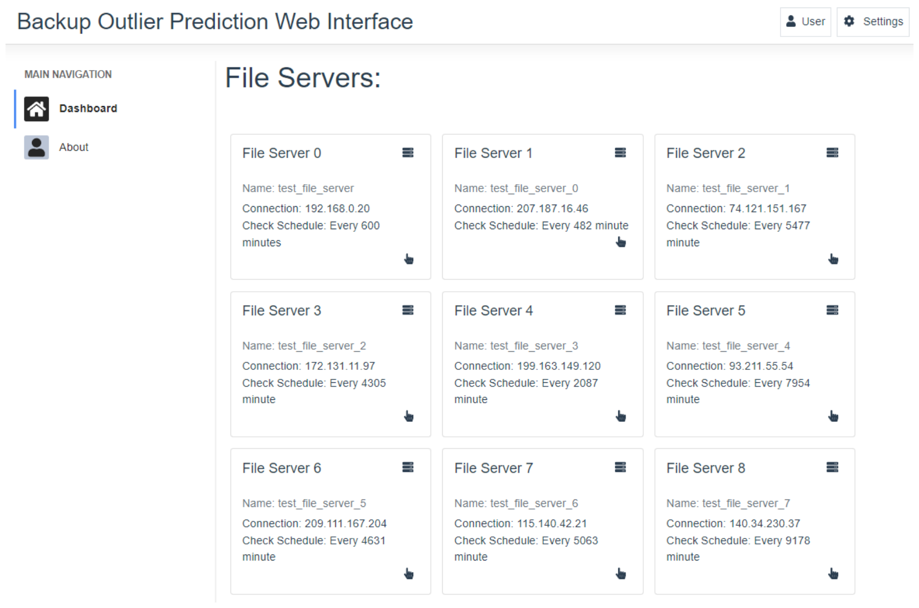
Figure 5.7: Correlation of features and outlier scores.

The figure shows that the correlation of the two metadata models is almost equal to the correlation between the amount of outliers per file and the remaining features. Both OCSVM models show a negative correlation to the features that are positive for the metadata model. The features that were less correlated with each other are also less correlated to the models' outlier scores.

5.4 Web Interface

A requirement of the internship was a web interface, mentioned in section 4.4, in order to interact with the results of the model.

On the homepage of the developed web interface, visual in figure 5.8, are the file servers listed that the user has access to.



File Servers:

File Server 0
Name: test_file_server
Connection: 192.168.0.20
Check Schedule: Every 600 minutes

File Server 1
Name: test_file_server_0
Connection: 207.187.16.46
Check Schedule: Every 482 minute

File Server 2
Name: test_file_server_1
Connection: 74.121.151.167
Check Schedule: Every 5477 minute

File Server 3
Name: test_file_server_2
Connection: 172.131.11.97
Check Schedule: Every 4305 minute

File Server 4
Name: test_file_server_3
Connection: 199.163.149.120
Check Schedule: Every 2087 minute

File Server 5
Name: test_file_server_4
Connection: 93.211.55.54
Check Schedule: Every 7954 minute

File Server 6
Name: test_file_server_5
Connection: 209.111.167.204
Check Schedule: Every 4631 minute

File Server 7
Name: test_file_server_6
Connection: 115.140.42.21
Check Schedule: Every 5063 minute

File Server 8
Name: test_file_server_7
Connection: 140.34.230.37
Check Schedule: Every 9178 minute

Figure 5.8: Homepage of the web interface

Each file server tile is a button that leads to its data. Visualized on the second page are the model's return values in a graph and the extracted features in a table. The figure A.1 shows the implemented intractable graph, visualizing the outlier probabilities and confidence levels. This graph element could be extended to also visualize selected features. The table of Figure A.2 shows every possible data available to each backup. This includes the extracted features and the model's return values.

6 Discussion & Conclusion

The previous chapter showed how the two outlier prediction models classified the backup data by their respective features. The two different approaches isolated different features as important to their outlier prediction.

One problem with the evaluation is the used data source. The fact that no actual backup data of file servers were used in the evaluation could have influenced the performance of the models. The strong fluctuation between the amount of files per backup is probably not representative of regular backups of file servers. Having a more consistent data set could increase the influence of the features that did not correlate with the model's outlier score on this data set. Therefore, evaluating the model's performance to detect outliers in backup data of file servers has to be taken with a grain of salt.

6.1 Different classifications

The metadata model and the OCSVM identified two different sets of backups as outliers is most-likely undesired. While the set for the metadata model contains the backups with many files, the OCSVM' group considers these backups with few files to be outliers. This could be explained by the fact that the metadata model classifies the backup globally by its features, while the OCSVM tries to create clusters by the similarity between backups. Backups with more files create clusters, while backups with almost no files represent more isolated changes and have practically no similarity to any other backup and therefore do not belong to any cluster. In the metadata model, the extracted feature vector of a smaller backup still looks similar to every other feature vector of a backup with few files. As there are relatively more backups with fewer files, these backups create the central cluster, from which the larger backups are further away and classified as outliers.

The original reason for this problem could be traced back to the test data source. A fundamental assumption for the developed outlier models is that backups stay relatively consistent. A day's work of a group of people working on the file server does not usually consist of an isolated change, while a commit to a project on a specific day could be an isolated fix or a small change. If the backups in the test data were more consistent, it is more likely that both models' predictions would align.

One could argue that the OCSVM ability to detect isolated changes is the desired behaviour by taking the mode of the file server into account. This file server mode would describe if changes usually consist of isolated changes or of a set of files that change regularly. Depending on the mode, different backups could be identified as outliers.

6.2 Feature significance

The strong correlation of the metadata model's outlier score with the amount of files in a backup could be undesired. Combing all features into one vector and making outlier predictions on that vector leads to the insignificance of some features, while the features that scale with the number of files per backup dominate the outlier prediction.

One possible solution could be to normalize the features that count the occurrence of a specific element by the amount of files in each backup. Another solution could be to split the metadata model classification into multiple models, one for each feature category. Reporting the results by combing outlier scores with the usual model ensemble approach, such as averaging or taking the max. This approach does not change the fact that features scale differently with the amount of files per backup. But increases the significance of features in a group that doesn't scale, such as the features using the date element.

6.3 Future Work

The next step in this project would be the integration into a Commvault test cell of the RWTH's server environment, fine-tuning the model based on expected outliers and evaluating its performance on a more consistent data set.

With the integration into an existing environment comes the integration of reporting outliers to the respective responsible person. Reporting every prediction only relying on the model's classification could lead to too many reports and numbness to such reports. Finding a way to combine the model's outputs of outlier prediction, confidence and outlier probability into a value that could be used to decide if a prediction should be reported is non-trivial. This value could be used only to report actual crucial outliers.

6.3.1 Prediction not by backup

Another set of possible future work could be to extend the outlier prediction by different classification approaches. The classification, as mentioned earlier, for each individual file is one possible extension.

Another approach could be to view the task of detecting abnormal behaviour on a file server not by classifying the overall backup but by classifying the individual behaviour of a specific user captured in a backup. This could be archived by creating user profiles for each user on the file server. In other words, for each user, a model is created. Clusters are created for the behaviour of one user and reported are activities behaviour not belonging to any user.

6.3.2 Expert Knowledge

One of the following problems could occur when only using unsupervised methods to train a model continuously. (1) More and more undesired outliers in the data are detected and form a cluster themselves. This could result in the model's inability to differentiate between normal and outlier data. In the case of the metadata model, for example, the backups with a lot of files per backup could start to form clusters themselves. (2) The second problem occurs if detected outliers are not integrated into the training process. This results in the model losing its ability to adapt to new situations on a secured file server.

One way to build a reliable training set for each file server is to utilize expert knowledge regarding the specific file server. For this, the web interface could be extended to approve of a detected outlier prediction. Using this information by either excluding it in the models training or including this example as a negative example as explained in the semi-supervised outlier methods section 2.1.3.1.

6.4 Conclusion

The task of developing an outlier detection tool that is supposed to be integrated into an existing file server backup solution that analysis made file server backups to search for malicious behaviour, including a web interface to interact with the tool and analyze the results, was an ambitious goal to archive in this internship. Partially, this internship could address these goals. For example, the goal of creating the outlier detection tool that classifies backups based on the extracted features and provides an outlier probability coupled with a certainty value could be archived. Further, a web interface that displays the stored data created by the detection tool through a backend was developed. Even so, no integration could be done, the design concept of a possible deployment plan capable of handling multiple file servers and a general software design infrastructure for each file server was developed.

Concluding the internship by addressing the research questions: *Is it possible to detect anomalies in file modification patterns of a file server based on backup log data compared to a learned normal mode?* Research on different outlier detection methods was conducted to answer this question, which resulted in the short survey of chapter 2. This was followed by selecting suitable methods explained in the concept's chapter 3 and their adaptation for backup data further investigated in chapter 4. The experiments show that the different models learned to detect different anomalies based on their extracted features.

To answer the second research question: *Can the model continuously learn gradual changes in regular modification pattern's to accommodate for new users or tasks?* It was shown by integrating a backup into the underlying models' data structures and integrating a database to store the model information persistently. The experiments of section 5.2 show how the outlier detection methods adapt to the changing underlying data and predict similar backups as an outlier when being only trained on previous examples.

6.4.1 Internship Learnings

This internship showed me how to apply general data science concepts to a real-world application. The broad research done on outlier detection methods helped me understand different unsupervised learning techniques that, compared to the usual curriculum, are not extensively covered. It further improved my skills in developing larger software projects independently. As most software projects done for the universities are small isolated projects, the internship taught me how to design proper data flow for a machine learning application that is supposed to be deployed in a production environment. It also taught me that things don't always go by the plan. The difficulty of missing communication and missing requirements.

List of Figures

Bibliography

Achlioptas, Dimitris. 2003. "Database-friendly random projections: Johnson-Lindenstrauss with binary coins." Special Issue on PODS 2001, *Journal of Computer and System Sciences* 66 (4): 671–687. ISSN: 0022-0000. https://doi.org/https://doi.org/10.1016/S0022-0000(03)00025-4.

Adam, Sally. 2021. *The State of Ransomware in Education 2021.* Technical report.

Boukerche, Azzedine, Lining Zheng, and Omar Alfandi. 2020. "Outlier Detection: Methods, Models, and Classification." *ACM Comput. Surv.* (New York, NY, USA) 53, no. 3 (June). ISSN: 0360-0300. https://doi.org/10.1145/3381028.

Breunig, Markus M., Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. 2000. "LOF: Identifying Density-Based Local Outliers." *SIGMOD Rec.* (New York, NY, USA) 29, no. 2 (May): 93–104. ISSN: 0163-5808. https://doi.org/10.1145/335191.335388.

Commvault Systems®. 2019.*,* November.

Cortes, Corinna, and Vladimir Vapnik. 1995. "Support-vector networks." *Machine Learning* 20, no. 3 (September): 273–297. https://doi.org/10.1007/bf00994018.

Das, Shubhomoy, Weng-Keen Wong, Thomas Dietterich, Alan Fern, and Andrew Emmott. 2016. "Incorporating Expert Feedback into Active Anomaly Discovery." In *2016 IEEE 16th International Conference on Data Mining (ICDM).* IEEE, December. https://doi.org/10.1109/icdm.2016.0102.

Désir, Chesner, Simon Bernard, Caroline Petitjean, and Laurent Heutte. 2013. "One class random forests." *Pattern Recognition* 46 (12): 3490–3506. ISSN: 0031-3203. https://doi.org/https://doi.org/10.1016/j.patcog.2013.05.022.

Fejri, Lotfi. 2017. "Matrix approximation methods and compressed sensing via random operators" (December). https://doi.org/10.13140/RG.2.2.19347.09760.

Filla, Nicole. 2021. *Datensicherung.NRW,* December.

Freedman, David, Robert Pisani, and Roger Purves. 2007. “Statistics (international student edition).” *Pisani, R. Purves, 4th edn. WW Norton & Company, New York.*

Goldstein, Markus, and Seiichi Uchida. 2016. “A Comparative Evaluation of Unsupervised Anomaly Detection Algorithms for Multivariate Data.” *PLOS ONE* 11, no. 4 (April): 1–31. https://doi.org/10.1371/journal.pone.0152173.

Goodfellow, Ian, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. “Generative Adversarial Nets.” In *Advances in Neural Information Processing Systems,* edited by Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, vol. 27. Curran Associates, Inc.

Grubbs, Frank E. 1969. “Procedures for Detecting Outlying Observations in Samples.” *Technometrics* 11 (1): 1–21. https://doi.org/10.1080/00401706.1969.10490657.

Hawkins, Simon, Hongxing He, Graham Williams, and Rohan Baxter. 2002. “Outlier Detection Using Replicator Neural Networks.” In *Data Warehousing and Knowledge Discovery,* edited by Yahiko Kambayashi, Werner Winiwarter, and Masatoshi Arikawa, 170–180. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-540-46145-6.

Johnson, William B. 1984. “Extensions of Lipschitz mappings into Hilbert space.” *Contemporary mathematics* 26:189–206.

Kingma, Diederik P, and Max Welling. 2014. *Auto-Encoding Variational Bayes.*

Kramer, Mark A. 1991. “Nonlinear principal component analysis using autoassociative neural networks.” *AIChE Journal* 37, no. 2 (February): 233–243. https://doi.org/10.1002/aic.690370209.

Liu, Fei Tony, Kai Ting, and Zhi-Hua Zhou. 2009. “Isolation Forest,” 413–422. January. https://doi.org/10.1109/ICDM.2008.17.

Liu, Yezheng, Zhe Li, Chong Zhou, Yuanchun Jiang, Jianshan Sun, Meng Wang, and Xiangnan He. 2019. *Generative Adversarial Active Learning for Unsupervised Outlier Detection.*

McCulloch, Warren S., and Walter Pitts. 1943. “A logical calculus of the ideas immanent in nervous activity.” *The Bulletin of Mathematical Biophysics* 5, no. 4 (December): 115–133. https://doi.org/10.1007/bf02478259.

Perini, Lorenzo, Vincent Vercruyssen, and Jesse Davis. 2021. “Quantifying the Confidence of Anomaly Detectors in Their Example-Wise Predictions.” In *Machine Learning and Knowledge Discovery in Databases,* 227–243. Springer International Publishing. https://doi.org/10.1007/978-3-030-67664-3_14.

Piette, É., D. J. N. J. Soemers, M. Stephenson, C. F. Sironi, M. H. M. Winands, and C. Browne. 2020. “Ludii – The Ludemic General Game System.” In *Proceedings of the 24th European Conference on Artificial Intelligence (ECAI 2020),* edited by G. De Giacomo, A. Catala, B. Dilkina, M. Milano, S. Barro, A. Bugarín, and J. Lang, 325:411–418. Frontiers in Artificial Intelligence and Applications. IOS Press.

Raedt, Luc De. 2022. “Kernels and Distances for Structured Data.” *Logical and Relational Learning,* 289–324. https://doi.org/10.1007/978-3-540-68856-3_9.

Ramaswamy, Sridhar, Rajeev Rastogi, and Kyuseok Shim. 2000. “Efficient Algorithms for Mining Outliers from Large Data Sets.” *SIGMOD Rec.* (New York, NY, USA) 29, no. 2 (May): 427–438. ISSN: 0163-5808. https://doi.org/10.1145/335191.335437.

Savitzky, Abraham., and M. J. E. Golay. 1964. “Smoothing and Differentiation of Data by Simplified Least Squares Procedures.” *Analytical Chemistry* 36, no. 8 (July): 1627–1639. https://doi.org/10.1021/ac60214a047.

Schölkopf, Bernhard, John Platt, John Shawe-Taylor, Alexander Smola, and Robert Williamson. 2001. “Estimating Support of a High-Dimensional Distribution.” *Neural Computation* 13 (July): 1443–1471. https://doi.org/10.1162/089976601750264965.

Schubert, Erich, Arthur Zimek, and Hans-Peter Kriegel. 2015. “Fast and Scalable Outlier Detection with Approximate Nearest Neighbor Ensembles.” In *Database Systems for Advanced Applications,* edited by Matthias Renz, Cyrus Shahabi, Xiaofang Zhou, and Muhammad Aamir Cheema, 19–36. Cham: Springer International Publishing. ISBN: 978-3-319-18123-3.

Settles, Burr. 2009. “Active Learning Literature Survey.”

Tang, Jian, Zhixiang Chen, Ada Wai-chee Fu, and David W. Cheung. 2002. “Enhancing Effectiveness of Outlier Detections for Low Density Patterns.” In *Advances in Knowledge Discovery and Data Mining,* edited by Ming-Syan Chen, Philip S. Yu, and Bing Liu, 535–548. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-540-47887-4.

Tax, David M.J., and Robert P.W. Duin. 2004. “Support Vector Data Description.” *Machine Learning* 54, no. 1 (January): 45–66. https://doi.org/10.1023/b:mach.0000008084.60811.49.

Vries, Timothy, Sanjay Chawla, and Michael Houle. 2010. “Finding Local Anomalies in Very High Dimensional Space,” 128–137. December. https://doi.org/10.1109/ICDM.2010.151.

Wang, Jin, Yangning Tang, Shiming He, Changqing Zhao, Pradip Kumar Sharma, Osama Alfarraj, and Amr Tolba. 2020. “LogEvent2vec: LogEvent-to-Vector Based Anomaly Detection for Large-Scale Logs in Internet of Things.” *Sensors* 20 (9). ISSN: 1424-8220. https://doi.org/10.3390/s20092451.

Wang, Ye, Srinivasan Parthasarathy, and Shirish Tatikonda. 2011. “Locality Sensitive Outlier Detection: A ranking driven approach.” In *2011 IEEE 27th International Conference on Data Engineering,* 410–421. https://doi.org/10.1109/ICDE.2011.5767852.

Zhao, Yue, Xiyang Hu, Cheng Cheng, Cong Wang, Changlin Wan, Wen Wang, Jianing Yang, Haoping Bai, Zheng Li, Cao Xiao, et al. 2021. “SUOD: Accelerating Large-scale Unsupervised Heterogeneous Outlier Detection.”

Zhao, Yue, Zain Nasrullah, and Zheng Li. 2019. “PyOD: A Python Toolbox for Scalable Outlier Detection.” *Journal of Machine Learning Research* 20 (96): 1–7.

A Appendix

A.1 Featuer List

The list of features that are extracted are the following:

General features over no specific component of the backup.

1. **Amount** is the numerical amount of files that were backed up in the incremental backup. The amount of files stored in each backup should stay approximately consistent for each file server. It could be an indicator for malicious activity if more files than usual are stored.
2. **Amount delta** is the difference in stored files compared to the last backup. A negative value means more data was stored than in the last backup.

Date features over the stored access time relative to the date of the backup.

1. **Start time** is the smallest timestamp of the day. It should be usually around the time a user of the file server starts to work. If someone modified files earlier than the normal work hours, this value becomes smaller.
2. **End time** represents the biggest timestamp of the day accordingly. The feature becomes bigger than normal if a user accessed a file after the normal working hours.
3. **Time range** is the distance between start time and end time and represent the usual length of a work day.
4. **Time average** uses a min max scalar to move the Unix times into a range of $0 - 1$. Followed by calculating the average of the timestamps. This could capture, if a bulk modification of files was done at a specific time, resulting in a shifted mean.
5. **Time standard deviation,** calculates the standard deviation on the scaled data. This represents the usual spread of modifications.

Action features over the stored action features: renamed, added, modified and deleted.

1. **Action amount (R, A, M, D)** is the numerical value counting the occurrence for each access type respectively. An obviously malicious indicator is, if the amount of deleted files becomes bigger than usual.

2. **Action amount delta (R, A, M, D)** is again the difference in the values to the previous backup, capturing specific trends in the data.

User features over the user that has accessed each file last.

1. **Amount of users** is the total amount of unique users in the backup.

2. **Changed users** calculates for each file if the user changed. A malicious action could be if a user changes a lot of files that do not belong to him. This would result in more files than usual with a new user.

3. **File per user (Min, Max, Avg)** counts the amount of files, each user modified, on each backup. Followed by calculating the minimum, maximum, and average. The min value represents the user that modified the least, while the max value accordingly the most. One user modifying more files than usual could be a malicious action.

Filetree features over the created file trees.

1. **Branching factor (Min, Max, Avg)** is the value how many folders or files are usually stored in one respective folder. A large branching factor corresponds to a flat folder structure with many files in a folder.

2. **Path length (Min, Max, Avg)** describes the depth of the tree.

File features capture different properties of files in the backup.

1. **Different folders** feature represent the amount of unique folders in the backup. A feature value of 1 would mean that files were only stored on the root level of the file server. The value scales with the distribution of how files are modified. Large distribution means a lot of files in different folders.

2. **Amount of file endings** isolates the file ending of the stored file. Motivated by the thought that a user group working with a file server would store approximately the same kind of files with roughly the same amount of types.

3. **Average different file endings**, being the average amount of files per file ending. A backup that contains a lot of files with the same file ending would result in shifted mean.

4. **File not previously stored** compare the files in the backup with the files that were previously stored on that file server, excluding the initial backup. This captures if a large amount of files that weren't touched previously is stored, hinting to a malicious action.
5. The **Cross-section** feature is the amount of files that are equal between the last two backups. If different parts of the file server are stored compared to before, the cross-section value becomes smaller.

A.2 Frontend images

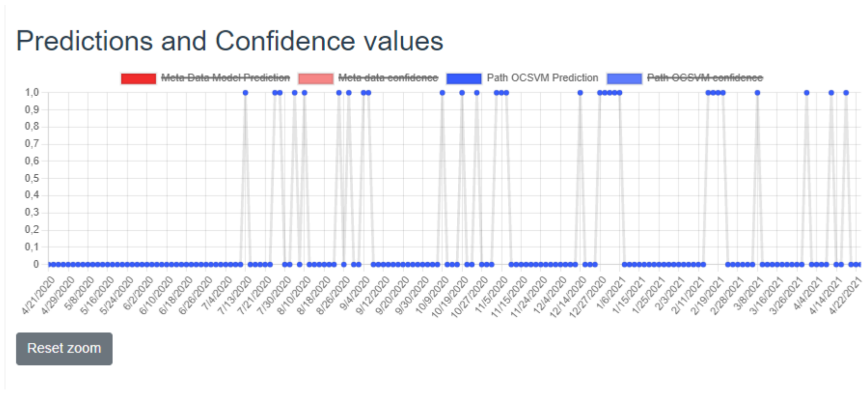
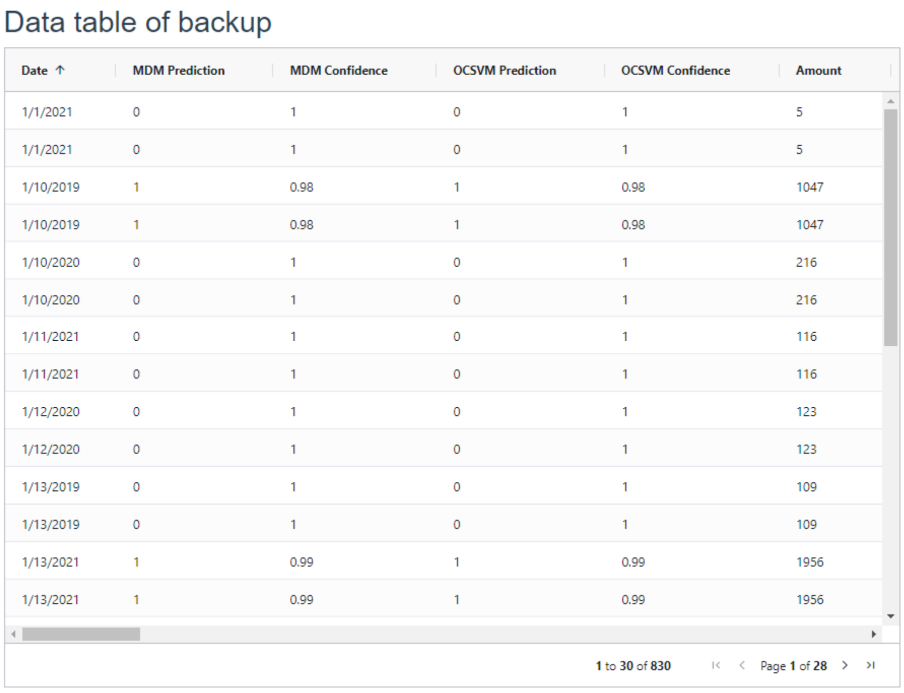


Figure A.1: Intractable visualization of the prediction and confidence values of both models. Extendable to also show different features of the selected file server.


Data table of backup

Date ↑	MDM Prediction	MDM Confidence	OCSVM Prediction	OCSVM Confidence	Amount
1/1/2021	0	1	0	1	5
1/1/2021	0	1	0	1	5
1/10/2019	1	0.98	1	0.98	1047
1/10/2019	1	0.98	1	0.98	1047
1/10/2020	0	1	0	1	216
1/10/2020	0	1	0	1	216
1/11/2021	0	1	0	1	116
1/11/2021	0	1	0	1	116
1/12/2020	0	1	0	1	123
1/12/2020	0	1	0	1	123
1/13/2019	0	1	0	1	109
1/13/2019	0	1	0	1	109
1/13/2021	1	0.99	1	0.99	1956
1/13/2021	1	0.99	1	0.99	1956

1 to 30 of 830 Page 1 of 28

Figure A.2: Intractable table showing the returned values of the model and the extracted features.